PIOTR WDOWIŃSKI

Liniowe modele ekonometryczne

Zaawansowane zastosowania języka R

Liniowe modele ekonometryczne

WYDAWNICTWO
UNIWERSYTETU
ŁÓDZKIEGO

PIOTR WDOWIŃSKI

Liniowe modele ekonometryczne

Zaawansowane zastosowania języka R

WYDAWNICTWO
UNIWERSYTETU
ŁÓDZKIEGO

Łódź 2025

Piotr Wdowiński (ORCID: 0000-0002-0355-5736) – Uniwersytet Łódzki
Wydział Ekonomiczno-Socjologiczny, Katedra Ekonometrii
90-214 Łódź, ul. Rewolucji 1905 r. 41/43

RECENZENCI
Ewa Majerowska, Michał Rubaszek

REDAKTOR INICJUJĄCY
Katarzyna Włodarczyk

REDAKCJA JĘZYKOWA I KOREKTA
Piotr Wdowiński

SKŁAD I ŁAMANIE
Piotr Wdowiński

PROJEKT OKŁADKI
Monika Rawska

Ilustracja wykorzystana na okładce: © Depositphotos.com/videoflow

Wydrukowano z gotowych materiałów dostarczonych do Wydawnictwa UŁ

https://doi.org/10.18778/8331-975-9

Wydane przez Wydawnictwo Uniwersytetu Łódzkiego
Wydanie I. W.11892.25.0.M
Ark. druk. 36,5

ISBN 978-83-8331-974-2
e-ISBN 978-83-8331-975-9

Wydawnictwo Uniwersytetu Łódzkiego
90-237 Łódź, ul. Matejki 34A
www.wydawnictwo.uni.lodz.pl
e-mail: ksiegarnia@uni.lodz.pl
tel. 42 635 55 77

Książkę dedykuję
moim Rodzicom
– Witoldowi i Natalii

Spis treści

Spis rysunków

Spis tablic

Przedmowa

Badania naukowe często wiążą się z koniecznością wyboru oprogramowania obliczeniowego. Dlatego każdy badacz powinien znać podstawowe zasady algorytmiki i programowania, będące kluczowym elementem planowania złożonych eksperymentów naukowych. Oprogramowanie komputerowe jest niezbędne zarówno przy realizacji badań symulacyjnych, jak i analiz empirycznych. Jeśli eksperyment wymaga utworzenia dużego zbioru danych (*big data*) z różnych źródeł statystycznych i uporządkowania wyników, konieczne jest użycie odpowiedniego skryptu obliczeniowego. Taki skrypt zapisuje się zwykle w języku programowania wysokiego poziomu (tzw. *meta*-języku), wyposażonym w typowe narzędzia programistyczne. Można do nich zaliczyć:

- deklarację stałych i zmiennych,
- wyrażenia logiczne,
- pętle i funkcje.

Różnorodne programy i języki programowania (takie jak: `Econometric Views`, `Matlab`, `R`, `Stata` itp.) funkcjonują również dzięki rozszerzeniom w postaci bibliotek, które umożliwiają realizację złożonych obliczeń. Do takich obliczeń można zaliczyć np. wykonanie szacunków parametrów wielorównaniowego modelu ekonometrycznego za pomocą podwójnej metody najmniejszych kwadratów (2MNK). Jeśli zachodzi potrzeba modyfikacji estymatora, można samodzielnie zaprogramować etapy obliczeniowe korzystając z wbudowanych narzędzi rachunku wektorowo-macierzowego. Jeśli program obliczeniowy nie zawiera odpowiednich metod wbudowanych, dane można przekazać do innego programu i pobrać wyniki do dalszego opracowania.

Automatyzacja obliczeń jest szczególnie ważna przy sporządzaniu raportów z badań, zarówno naukowych, jak i analitycznych. W praktyce każdy eksperyment obliczeniowy kończy się przygotowaniem raportu. Nie jest to zadanie proste, zwłaszcza, jeśli proces przygotowania raportu ma być czynnością powtarzalną, a próba statystyczna jest często aktualizowana lub na ostateczną decyzję o wyborze modelu składa się wiele podobnych prób obliczeniowych. Często raport obejmuje wiele formatów plików – `HTML`, `PDF` czy `DOCX`. Wówczas niezbędne jest właściwe narzędzie, nie tylko obliczeniowe. Należy stwierdzić, że program `R` łączy wszystkie wspomniane cechy.

Można w nim również zbudować aplikację internetową, która jest zasilana danymi i wykonuje obliczenia w czasie rzeczywistym. Do budowy takich aplikacji służy biblioteka `shiny` (Chang i in., 2024) i serwer `R/Shiny` (zob. rozdział 4).

Skorzystanie z wielu funkcji języka R i bibliotek towarzyszących wymaga od badacza umiejętności korzystania z zaawansowanego aparatu pojęciowego, który obejmuje znajomość:

- zasad algorytmiki,
- podstaw konstrukcji języków `LaTeX`, `HTML`, `PHP` i `JavaScript` oraz arkuszy stylów `CSS` (*kaskadowych arkuszy stylów*),
- zasad pracy w środowisku `Linux` i organizacji pracy serwerów:
  - zdalnych `VPS` (*Virtual Private Server*),
  - `Apache` do obsługi stron internetowych,
  - baz danych `MySQL` i programów towarzyszących, np. `phpMyAdmin`,
- systemów `CMS` (obsługi treści) do budowy stron internetowych:
  - `Joomla`,
  - `Wordpress`,
- konstrukcji domen internetowych i rekordów `DNS`,
- programów administracyjnych dla serwerów `Apache/PHP/HTML`, np. `Direct Admin`.

Przedstawiona powyżej lista umiejętności wymaga szerokiej wiedzy informatycznej. W tym zakresie w książce zostaną przedstawione wybrane zagadnienia. Jej celem jest zaprezentowanie zaawansowanych możliwości języka R w kontekście badań statystyczno-ekonometrycznych, ze szczególnym uwzględnieniem modelowania ekonometrycznego. Dlatego szczególną uwagę poświęcono procesom stochastycznym oraz liniowym modelom ekonometrycznym. Uzupełnieniem tych zagadnień są metody prognozowania oparte na modelach ekonometrycznych.

Książka ukazuje się w dwóch wersjach: drukowanej oraz elektronicznej (PDF). Mam nadzieję, że dzięki tak przyjętej formule wydania książka trafi do wielu odbiorców. Będę wdzięczny za uwagi, które pomogą przygotować kolejne, poprawione wydania oraz będą pomocne przy pracach nad następnymi książkami.

Pragnę podkreślić, że technologię przygotowania tekstu książki opracowano z użyciem następujących bibliotek R i programów[1]:

- `bookdown` (Xie, 2025a),
- `R/Markdown` (Allaire i in., 2025; R Core Team, 2025),
- `R/Shiny` i `LaTeX`[2],

[1] Technologia ta obejmuje systemy i formaty plików: `R/Markdown/LaTeX/PDF`.
[2] Por. https://www.latex-project.org.

- edytora MiKTeX[3],
- kompilatora LuaTeX[4],
- programu Pandoc[5].

Do składu tekstu zastosowano dwie czcionki:

- Noto Serif[6] oraz
- Noto Sans[7].

Dane statystyczne wykorzystane w książce znajdują się pod adresem:

- https://github.com/wdowp/r-book-2.

Styl CSL[8] (*Elsevier*), używany do cytowania pozycji literatury, pobrano ze strony internetowej:

- http://www.zotero.org/styles/econometrics-and-statistics.

Składam serdeczne podziękowania wszystkim twórcom systemów informatycznych, graficznych i statystycznych, bez których ta książka nie mogłaby powstać. Możliwości programu R są bardzo szerokie. Nowe biblioteki funkcji powstają w odpowiedzi na rosnące potrzeby obliczeniowe oraz rozwój badań naukowych. Język R jest powszechnie wykorzystywany w badaniach i analizach. Jest chętnie stosowany przez pracowników nauki i studentów. Zachęcam wszystkich statystyków i ekonometryków do korzystania z tego programu. Upowszechnienie wiedzy na temat języka R i jego zastosowań jest także jednym z celów tej książki. Zachęcam również innych autorów do korzystania z nowoczesnych systemów składu tekstu, w których można osadzać kod języka R i tworzyć interaktywne dokumenty.

W przypadku książek dotyczących oprogramowania szczególne znaczenie mają źródła inspiracji. Jestem przekonany, że książka nie powstałaby w obecnej formie, gdybym nie zainteresował się językiem R podczas pracy w *Narodowym Banku Polskim* i nie rozwijał tych umiejętności w *ING Hubs Poland*, w tym z wykorzystaniem narzędzi *sztucznej inteligencji*. Chciałbym tym samym podziękować bankowi centralnemu oraz *ING Hubs Poland* za umożliwienie wykorzystania języka R w pracy nad modelami polityki makroostrożnościowej i ryzyka kredytowego. Do kontynuowania prac nad wykorzystaniem języka R przyczyniły się również pierwsze kroki podsumowane

[3] Por. https://miktex.org/.
[4] Por. http://www.luatex.org/.
[5] Por. https://pandoc.org/.
[6] Por. https://fonts.google.com/noto/specimen/Noto+Serif.
[7] Por. https://fonts.google.com/noto/specimen/Noto+Sans.
[8] CSL (*Citation Style Language*) to język opisu stylów cytowań zapisany w XML (*Extensible Markup Language*), czyli języku znaczników służącym do strukturalnego zapisu danych tekstowych.

w książce pt. *Wstęp do programowania i analizy danych w języku* R (Wdowiński, 2020). W trakcie pracy nad obecną książką pomocne okazały się także narzędzia oparte na sztucznej inteligencji, takie jak `ChatGPT` oraz `Copilot`, wspierające redakcję tekstu. Jedna z ilustracji w książce została wygenerowana za pomocą modelu `AI Sora`, dostępnego pod adresem: https://openai.com/sora. Narzędzia te odegrały rolę nie tylko w przyspieszeniu pracy, ale także w inspirowaniu nowych ujęć problemów.

Książka zawiera liczne przykłady kodu w języku R, które ilustrują omawiane zagadnienia i ułatwiają zrozumienie poszczególnych etapów analizy. Przykłady przygotowano tak, aby można je było łatwo odtworzyć i dostosować do własnych badań.

Treść książki odzwierciedla doświadczenia zdobyte podczas wieloletnich kursów *Ekonometrii*, *Prognozowania i symulacji* oraz *Programowania w języku R* prowadzonych dla studentów kierunków: Finanse i Rachunkowość, Bankowość i Finanse Cyfrowe, Ekonometria i Analityka Danych oraz Informatyka Ekonomiczna. Pragnę wyrazić wdzięczność wszystkim osobom, które przyczyniły się do powstania tej publikacji – studentom za inspirujące pytania i dyskusje, a także współpracownikom za wsparcie merytoryczne. Składam również serdeczne podziękowania Recenzentom – dr hab. Ewie Majerowskiej, profesor Uniwersytetu Gdańskiego (Katedra Finansów Przedsiębiorstw, Wydział Zarządzania) oraz prof. dr. hab. Michałowi Rubaszkowi (Zakład Modelowania Rynków Finansowych, Instytut Ekonometrii, Szkoła Główna Handlowa w Warszawie) za wnikliwe uwagi, które pozwoliły nadać książce ostateczny kształt. Dziękuję również mojej rodzinie za udzielone wsparcie podczas pracy nad książką.

Piotr Wdowiński
Uniwersytet Łódzki
piotr.wdowinski@uni.lodz.pl

Wprowadzenie

Oprogramowanie R należy do rodziny programów przeznaczonych głównie do wykonywania obliczeń statystycznych (R Core Team, 2025). Do tej rodziny należą również m.in. następujące programy:

- Econometric Views (EViews)[1],
- MATLAB[2],
- Python[3],
- SAS[4],
- Stata[5].

Wspomniane programy wykorzystuje się w wielu dziedzinach nauki – nie tylko w statystyce i ekonometrii. Najważniejsze obszary zastosowań programu R obejmują:

- statystykę i ekonometrię (Cohen i Cohen, 2008; Cryer i Chan, 2008; Kleiber i Zeileis, 2008; Gatnar i Walesiak, 2009; Robert i Casella, 2010; Suess i Trumbo, 2010; Kamiński i Zawisza, 2012; Pace, 2012; Rubaszek, 2012; Biecek, 2013; Schumacker i Tomek, 2013; Marin i Robert, 2014; Tsay, 2014; Verzani, 2014; Gerrard i Johnson, 2015; Makhabel, 2015; Ugarte i in., 2015; Kopczewska i in., 2016; Croissant i Millo, 2018),
- algorytmikę i programowanie (Chambers, 2008; Spector, 2008; Teetor, 2011; Crawley, 2012; Grolemund, 2014; Gągolewski, 2016; Wickham i Grolemund, 2016; Wiley i Wiley, 2016; Gillespie i Lovelace, 2017; Mailund, 2017; Wdowiński, 2020),
- medycynę i biologię (Lewis, 2010),
- przetwarzanie dużych zbiorów danych (Munzert i in., 2015),
- finanse (Kaas i in., 2008; Arratia, 2014; Berlinger i in., 2015),
- lingwistykę (Gries, 2009).

[1] Por. https://www.eviews.com/.
[2] Por. https://www.mathworks.com/.
[3] Por. https://www.python.org/.
[4] Por. https://www.sas.com/.
[5] Por. https://www.stata.com/.

Program R jest oprogramowaniem typu Open Source, opartym na licencji *GNU General Public License*[6], co sprawia, że jest on powszechnie wykorzystywany. Program R jest językiem skryptowym, który pozwala na:

- tworzenie własnych funkcji i bibliotek,
- osadzanie w kodzie skryptu funkcji z innych bibliotek oraz
- rozbudowę istniejących bibliotek.

W Dodatku A przedstawiono listę książek poświęconych językowi R i jego zastosowaniom, wraz z krótką charakterystyką każdej z nich. Język R oparty jest na języku S, stworzonym przez J. Chambersa i innych (Becker i in., 1988; Chambers i Hastie, 1992; Chambers, 1998). Projekt powstał w *Bell Laboratories* (dawniej: *AT&T*, obecnie: *Lucent Technologies*). W ramach *Bell Laboratories* opracowano również system operacyjny Unix oraz języki programowania C i C++.

Program R pozwala na przedstawienie wyników badań i analiz w postaci graficznej. Pozwala również na budowę interaktywnych aplikacji internetowych opartych na kodzie języka R/Shiny, nieco odmiennego od klasycznego języka R. Takie aplikacje w połączeniu z serwerem R/Shiny stanowią generator kodu HTML do tworzenia interaktywnych stron internetowych.

Jak wspomniano, program R jest oparty na idei otwartego oprogramowania Open Source w ramach licencji *GNU General Public License*. Licencja ta otwiera możliwość tworzenia oraz modyfikacji bibliotek obliczeniowych napisanych dla programu R. W przypadku wprowadzania modyfikacji kod źródłowy jest traktowany jako rozwiązanie pierwotne. Możliwość samodzielnej analizy kodu źródłowego programu R i jego bibliotek ma dla użytkowników walory dydaktyczne i naukowe, a także dyscyplinujące dla samych autorów bibliotek, gdyż ich rozwiązania są poddane powszechnej weryfikacji. Prowadzone przez społeczność programu R statystyki popularności bibliotek nakładają na autorów dodatkową dyscyplinę. Główne repozytorium kodu źródłowego R znajduje się pod adresem:

- https://svn.r-project.org/R/.

Większość bibliotek w postaci kodu źródłowego jest zamieszczonych na platformie GitHub[7]. Lustrzana kopia kodu źródłowego R ze strony https://svn.r-project.org/R/, aktualizowana co godzinę, w trybie tylko do odczytu, znajduje się na stronie internetowej:

- https://github.com/wch/r-source.

Instrukcje kompilacji znajdują się na stronie internetowej wiki:

[6] Por. https://www.r-project.org/Licenses/.
[7] Por. https://github.com/.

- https://github.com/wch/r-source/wiki/.

Powszechny dostęp do programu R, bibliotek i dyskusji w Internecie stwarza unikalne warunki dla upowszechniania nauki, w tym w krajach o niskim dochodzie, gdzie dostęp do programów komercyjnych jest znacząco utrudniony. Dzięki dyskusji w Internecie dotyczącej szczegółowych rozwiązań algorytmicznych dla programu R na platformach takich jak `Stack Overflow`[8], użytkownicy z całego świata mogą zgłaszać problemy i poszukiwać rozwiązań. Taka sytuacja stwarza bodźce do efektywnej alokacji wiedzy i umiejętności.

Rozwojem programu R zajmuje się grupa statystyków skupionych wokół projektu *R Development Core Team* oraz fundacja *The R Foundation*[9]. Fundacja wydaje recenzowane czasopismo *The R Journal*[10] w otwartym dostępie (*open access*). Czasopismo to ukazuje się od 2009 roku w formie dwóch numerów rocznie[11]. Jest ono indeksowane w bazie *Web of Science* – w kolekcji *Core Collection Science Citation Index Expanded (SCIE)*. Do obszarów naukowych czasopisma zalicza się:

- informatykę (*Computer Science*),
- badania interdyscyplinarne (*Interdisciplinary Applications*),
- statystykę i rachunek prawdopodobieństwa (*Statistics & Probability*),
- matematykę (*Mathematics*).

Zewnętrzne biblioteki dla programu R są tworzone przez ogromną grupę entuzjastów z całego świata. Wśród nich są naukowcy o znaczącym dorobku, pracujący w uznanych uniwersytetach. Wszystkie ścieżki dla programu R prowadzą do zasobów zgromadzonych na stronie internetowej *The R Project for Statistical Computing* – https://www.r-project.org/. Na tej stronie znajdują się:

- program R,
- baza bibliotek,
- dokumentacja,
- materiały dodatkowe,
- artykuły czasopisma *The R Journal*.

Bardzo istotnym ogniwem środowiska systemu R jest oprogramowanie towarzyszące. Zaliczają się do niego cztery systemy:

- `RStudio`,
- `RStudio Server`,

[8] Por. https://stackoverflow.com/.
[9] Por. https://www.r-project.org/foundation/.
[10] Por. http://journal.r-project.org/.
[11] Wskaźnik cytowań (*impact factor*) tego czasopisma za rok 2023 wyniósł 2,3.

- `Shiny Server`,
- `shinyapps.io`.

Program `RStudio`[12] jest produktem organizacji o tej samej nazwie. Jest interfejsem graficznym GUI (*graphical user interface*) ułatwiającym obsługę programu R. Repozytorium organizacji `RStudio` można znaleźć również na stronie internetowej `GitHub`[13]. Program `RStudio` powstał przy wykorzystaniu następujących języków programowania:

- `Java` (35,1%),
- C++ (30,2%),
- `JavaScript` (19%),
- `TypeScript` (6,3%),
- R (3,3%),
- C (2,9%),
- pozostałe (3,2%).

Program `RStudio` zawiera kilka obszarów roboczych obejmujących:

- pozycje menu,
- kod skryptu,
- konsolę wyników,
- środowisko zmiennych,
- obszar prezentacji grafiki.

System obróbki interaktywnych dokumentów `R/Markdown`, rozwijany w ramach projektu `RStudio`, umożliwia tworzenie artykułów, prezentacji oraz innych form tekstowych, zawierających obliczenia wykonane w programie R. Jest to szczególnie istotne w pracy naukowej, ponieważ kod skryptowy programu R można osadzić bezpośrednio w treści dokumentu (`chunk`) i wykonać podczas generowania dokumentu wynikowego. W środowisku `RStudio` możliwa jest kompilacja treści wraz z wynikami obliczeń do wielu formatów, w tym:

- `HTML`,
- `DOCX`,
- `PDF`.

Podstawą budowy *dynamicznych*[14] dokumentów są:

[12] Por. https://rstudio.com/.

[13] Por. https://github.com/rstudio/.

[14] Dynamiczne dokumenty tworzone w `R/Markdown` łączą kod, wyniki obliczeń i opis w jednym pliku, dzięki czemu analizy są w pełni replikowalne i automatycznie aktualizują się po każdej zmianie danych lub kodu. Pozwala to na generowanie raportów, artykułów i prezentacji, w których wykresy, tablice oraz wnioski są zawsze zgodne z aktualnym stanem analizy.

- biblioteki `rmarkdown` i `knitr` (Xie, 2025b),
- program `Pandoc` (https://pandoc.org/),
- system składu tekstu `LaTeX`.

Na tej podstawie można przygotowywać artykuły naukowe, a także książki. Niniejsza książka powstała w programach `R` i `RStudio` z wykorzystaniem biblioteki `bookdown`[15] autorstwa Y. Xie (Xie, 2015, 2016; Xie i in., 2018). Źródłowy spis literatury przygotowano w formacie `BibTeX`[16]. W tym celu wykorzystano następujące narzędzia:

- konwerter `AnyStyle` do przekształcenia pozycji literatury z formatu `TXT` do formatu `BibTeX` (https://anystyle.io/),
- program `JabRef` do zarządzania spisem literatury (https://www.jabref.org/),
- ChatGPT (https://chatgpt.com/) (OpenAI, 2023).

Program `RStudio Server` jest – podobnie jak `RStudio` – interfejsem graficznym. Służy on do pracy zdalnej w programie `R`. Oznacza to, że instaluje się go na serwerze w środowisku `Linux`, na którym również jest zainstalowany program `R`. Można wówczas pracować w programie `R` z dostępem zdalnym, podobnie jak w oparciu o program `RStudio` instalowany lokalnie na komputerze. Praca zdalna daje ogromne możliwości współpracy zespołów z różnych części świata, ponieważ skrypt obliczeniowy oraz wspólny artykuł są umieszczone na serwerze, gdzie odbywa się praca całego zespołu. Możliwości oprogramowania `RStudio Server` są wykorzystywane również do celów dydaktycznych i szkoleniowych, gdyż studenci uzyskują dostęp do jednego serwera poprzez Internet z wielu terminali komputerowych. Niepotrzebne są wówczas lokalne instalacje programów `R` i `RStudio`. `RStudio Server` obejmuje dwie wersje: `Open Source` i komercyjną.

Program `Shiny Server`, rozwijany przez firmę `RStudio`, jest serwerem aplikacji internetowych, opartym na bibliotece `shiny` dla programu `R`, generującym kod `HTML`. Program ten instaluje się w środowisku `Linux` po pobraniu z adresu:

- https://www.rstudio.com/products/shiny/download-server/.

Serwer `Shiny` i biblioteka `shiny` pozwalają na budowanie i uruchamianie interaktywnych aplikacji internetowych. Odbywa się to poprzez umieszczenie standardowego kodu obliczeniowego `R` wewnątrz kodu napisanego poleceniami biblioteki `shiny`. Następnie taką aplikację (zob. podrozdział 4.3) uruchamia się na serwerze `Shiny`, który zwraca kod `HTML` do przeglądarki internetowej, w efekcie czego powstaje dynamiczny obraz strony internetowej. Taką aplikację można powiązać z domeną internetową, a komunikację zabezpieczyć za pomocą certyfikatu `SSL`.

[15] Por. https://bookdown.org/.
[16] Por. http://www.bibtex.org/.

W ten sposób powstaje zaawansowana aplikacja internetowa z osadzonym algorytmem obliczeniowym w programie R, w której parametry są przekazywane do kodu za pomocą standardowych elementów HTML stosowanych w formularzach stron internetowych. Tworzy się wówczas rozbudowane systemy eksperckie. W podobny sposób, za pomocą biblioteki shinyMobile (Granjon i in., 2024) i systemu Framework7 (https://framework7.io/), tworzy się aplikacje zoptymalizowane pod kątem urządzeń mobilnych, korzystające z funkcji ekranów dotykowych.

Strona internetowa shinyapps.io[17] zawiera projekty aplikacji utworzonych za pomocą biblioteki shiny w programie R. W tym systemie tworzy się różne rodzaje kont i umieszcza na serwerze kod aplikacji shiny. Platforma shinyapps.io jest miejscem, gdzie zainstalowany jest serwer Shiny. Jak wspomniano wcześniej, Shiny Server można zainstalować samodzielnie w systemie Linux na dowolnym serwerze zdalnym, np. VPS, udostępnianym, zwykle komercyjnie, przez wielu dostawców infrastruktury internetowej. W następnym rozdziale przedstawiono szczegóły instalacji programów R i RStudio oraz bibliotek w systemie Windows.

[17] Por. https://www.shinyapps.io/.

Rozdział 1

Instalacja

W tym rozdziale przedstawiono sposób instalacji programów R i RStudio oraz wybranych bibliotek dla programu R w systemie Windows. Programy te można zainstalować także w systemach Linux i macOS. Instalacja programów RStudio Server i Shiny Server wymaga dostępu administracyjnego (root) do serwera VPS z systemem Linux – np. w wersji Ubuntu 18.04.6 LTS (Bionic Beaver). W tym rozdziale wykorzystano zagadnienia omówione w książce Wdowiński (2020).

1.1. Program R

Program R składa się z programu głównego oraz bibliotek. Język R jest zaimplementowany w części głównej. Podstawowe biblioteki, które realizują funkcje obliczeniowe, są wbudowane w program główny. Większość bibliotek instaluje się dodatkowo w środowisku R. Program R znajduje się w repozytorium:

- https://cran.r-project.org/.

Z repozytorium należy wybrać wersję odpowiednią dla używanego systemu (Linux, macOS lub Windows). Dla użytkowników systemu Windows wystarczy pobranie wersji binarnej dla systemu 32/64-bitowego, znajdującej się pod adresem:

- https://cloud.r-project.org/bin/windows/base/.

Po pobraniu pliku *.exe należy go uruchomić i postępować zgodnie z instrukcjami programu instalacyjnego. Program R jest standardowo instalowany w podkatalogu o nazwie tożsamej z wersją programu. Kolejne wersje programu R są instalowane

w odpowiednich podkatalogach, a dotychczasowa instalacja nie jest nadpisywana. W przypadku zainstalowania wielu wersji programu R w jednym systemie Windows, odpowiednią wersję można wybrać w programie RStudio w pozycji menu:

`Tools/Global Options/R version/Choose a specific version of R`

lub pozostawić domyślną wersję, zapisaną w rejestrze systemu Windows. Należy również pamiętać, że podczas instalacji bibliotek zewnętrznych pobierane są dane aktualnej wersji programu R, zadeklarowanej w programie RStudio. Po instalacji nowszej wersji programu R, biblioteki trzeba zainstalować ponownie.

W systemie Windows program instalacyjny zwykle umieszcza na pulpicie ikonę skrótu do programu R. Wykonywalny program R (`Rgui.exe`) (np. w wersji 4.4.3) można znaleźć w katalogu:

`C:\Program Files\R\R-4.4.3\bin\x64\Rgui.exe`.

Uruchomienie tego programu powoduje wyświetlenie komunikatu:

R version 4.4.3 (2025-02-28 ucrt) – "Trophy Case". Copyright (C) 2025 The R Foundation for Statistical Computing. Platform: x86_64-w64-mingw32/x64. R jest oprogramowaniem darmowym i dostarczany jest BEZ JAKIEJKOLWIEK GWARANCJI. Możesz go rozpowszechniać pod pewnymi warunkami. Wpisz 'license()' lub 'licence()' aby uzyskać szczegóły dystrybucji. R jest projektem kolaboracyjnym z wieloma uczestnikami. Wpisz 'contributors()' aby uzyskać więcej informacji oraz 'citation()' aby dowiedzieć się jak cytować R lub biblioteki R w publikacjach. Wpisz 'demo()' aby zobaczyć demo, 'help()' aby uzyskać pomoc on-line, lub 'help.start()' aby uzyskać pomoc w przeglądarce HTML. Wpisz 'q()' aby wyjść z R.

Pojawienie się powyższego komunikatu oznacza, że program R został zainstalowany poprawnie. Wpisanie i zatwierdzenie polecenia `quit()` oznacza wyjście z programu R.

W kolejnym podrozdziale omówione zostaną podstawowe elementy programu RStudio oraz jego funkcje ułatwiające pracę z kodem i danymi.

1.2. Program RStudio

Program RStudio jest produktem organizacji o tej samej nazwie. Instalację RStudio w wersji Open Source dla systemu Windows należy rozpocząć od pobrania pliku `*.exe` z lokalizacji:

- https://www.rstudio.com/products/rstudio/download/#download.

Następnie należy uruchomić plik instalacyjny, np. w wersji *RStudio Desktop 2021.09.0+351*:

- https://download1.rstudio.org/electron/windows/RStudio-2024.12.1-563.exe.

oraz postępować zgodnie z instrukcjami. Program zwykle jest instalowany w katalogu:

`C:\Program Files\RStudio`

RStudio można uruchomić za pomocą pliku `rstudio.exe`. Po uruchomieniu programu ukazują się cztery główne obszary robocze:

- kod skryptu,
- środowisko zmiennych,
- konsola z wynikami obliczeń,
- podgląd: plików, rysunków, bibliotek i dokumentów pomocy.

W zależności od wybranego rodzaju dokumentu dostępne są różne opcje obsługi:

- `R Script`, `R Markdown`, `Shiny App`, `R Presentation` i inne,
- `Project` (`R Package`, `Shiny Web Application`, `Book Project` i inne).

Jeśli użytkownik zamierza utworzyć kod skryptowy R i uruchamiać go interaktywnie w RStudio, wystarczy, że wybierze kombinację klawiszy `Ctrl+Shift+N`. Wówczas otworzy do edycji w obszarze skryptów dokument o nazwie `Untitled` z kolejnym numerem. Można od razu utworzyć kod i go wykonać. Należy wówczas zaznaczyć kod naciskając `Ctrl+A` i wykonać polecenie Run (`Ctrl+Enter`). Jeśli skrypt źródłowy ma zostać wykonany, a wyniki przekształcone np. do formatu PDF, to należy wybrać format dokumentu `R/Markdown`. Ten format pozwala na przygotowanie dynamicznych dokumentów poprzez skojarzenie tekstu z kodem R, który jest wykonywany a wyniki są umieszczane w dokumencie. W tym celu należy zainstalować kompletne środowisko systemu LaTeX, np. korzystając z oprogramowania MiKTeX (https://miktex.org/). Dla systemu Windows zaleca się wykorzystanie instalatora sieciowego (*Net Installer*) w sekcji `All downloads` z lokalizacji:

- https://miktex.org/download.

Niektóre biblioteki stosowane w języku `R/Markdown` korzystają z języka `Perl`:

- https://www.perl.org/.

Dlatego konieczne jest zainstalowanie również tego języka, np. w wersji `Strawberry Perl`:

- https://strawberryperl.com/.

Format R/Markdown zawiera wiele opcji konfiguracyjnych, które dają dużą swobodę w organizowaniu dokumentu. Szczegółowe rozwiązania można znaleźć w opracowaniach Xie (2015); Xie (2025b); Allaire i in. (2025).

Na zakończenie tego podrozdziału warto podkreślić, że RStudio stanowi zintegrowane środowisko pracy, które znacząco ułatwia tworzenie, edycję i uruchamianie skryptów w języku R. Intuicyjny interfejs, liczne funkcje wspomagające analizę danych oraz możliwość integracji z różnymi formatami dokumentów sprawiają, że jest to podstawowe narzędzie pracy statystyków, ekonometryków i analityków danych.

W kolejnym podrozdziale zostanie omówiona idea bibliotek w języku R, sposób ich instalacji oraz konfiguracji lokalizacji, w której są przechowywane. Dzięki temu możliwe będzie korzystanie z szerokiego zbioru narzędzi dostępnych dla języka R.

1.3. Biblioteki

Możliwość tworzenia zewnętrznych bibliotek (pakietów) stanowi ważne uzupełnienie programu R. Biblioteki są w istocie programami zapisanymi w językach R/C/C++/Fortran, które podczas instalacji są kompilowane i zapisywane na dysku w ustalonej lokalizacji. Domyślną lokalizację bibliotek zewnętrznych można dowolnie zmienić. W systemie Windows może to być następująca ścieżka (np. dla wersji R 4.4.3):

`C:\Users\user\AppData\Local\R\win-library\4.4`.

Tę lokalizację można zmienić za pomocą polecenia `.libPaths()`. Przyjmując, że biblioteki mają być instalowane w katalogu F:\package, należy uruchomić następującą funkcję:

```
.libPaths(c(.libPaths(), "F:\\package")).
```

Ponowne wykonanie polecenia `.libPaths()` pokazuje zmienioną lokalizację bibliotek z uwzględnieniem ścieżki F:\package, którą można wybrać podczas instalacji w programie RStudio. Zastosowania bibliotek są bardzo różnorodne. Wiele z nich zostanie przedstawionych w dalszej części książki. Informacje dotyczące bibliotek są zgromadzone na stronie internetowej:

- https://www.rdocumentation.org/.

Standardową instalację biblioteki wykonuje się za pomocą polecenia wydanego w konsoli programu R lub RStudio:

```
install.packages('nazwa-biblioteki').
```

Instalowanie bibliotek w programie RStudio jest bardzo wygodne. W tym celu w prawym obszarze roboczym należy wybrać polecenie Packages, a następnie Install

i w polu wyboru podać nazwę biblioteki. Na podstawie wprowadzonych znaków pojawiają się propozycje nazw bibliotek. Jeśli w systemie są zadeklarowane różne ścieżki dostępu do bibliotek, to można w następnym polu wybrać żądaną ścieżkę. Po zatwierdzeniu formularza biblioteka jest instalowana z repozytorium CRAN. Jeśli występuje nowsza wersja biblioteki, ale jeszcze nie znalazła się w repozytorium CRAN, można ją zainstalować bezpośrednio z lokalizacji wskazanej przez jej autora. Często taką lokalizacją jest serwis GitHub. W tym celu należy wcześniej zainstalować bibliotekę `devtools` autorstwa H. Wickhama[1] (Wickham i Grolemund, 2016; Wickham i in., 2022):

```
install.packages('devtools').
```

W tej bibliotece występuje funkcja `install_github`. Instalację biblioteki – np. `devtools` – z serwisu GitHub wykonuje się poleceniem:

```
devtools::install_github('r-lib/devtools').
```

Zdarza się, że biblioteka przestała być rozwijana i została usunięta z repozytorium CRAN albo znajduje się poza nim. Wówczas jej instalacja w systemie Windows jest bardziej złożona, gdyż wymaga *kompilacji* kodu źródłowego. Taką bibliotekę można zainstalować za pomocą programu `Rtools`[2].

Program `Rtools` umożliwia kompilację bibliotek R wykorzystujących języki programowania C/C++/Fortran. Kompilacja następuje z postaci źródłowej do postaci wykonywalnej, zrozumiałej dla programu R. Należy pamiętać o zainstalowaniu w określonej lokalizacji (np. `C:\rtools44`) programu `Rtools` zgodnego z numerem wersji programu R (dla wersji `4.4` powinna to być wersja `rtools44`):

- https://cran.r-project.org/bin/windows/Rtools/history.html.

Na etapie instalacji programu `Rtools` można dodać wpis w zmiennej środowiskowej PATH systemu Windows o lokalizacji programu `Rtools` za pomocą opcji `Add rtools to system PATH`. W wersji `Rtools 4.4` ta operacja odbywa się automatycznie. Jeśli ta lokalizacja nie zostanie wprowadzona automatycznie, niezbędne jest późniejsze ustawienie ścieżki dostępu do narzędzi programu `Rtools`. Tę czynność należy przeprowadzić w *Panelu sterowania* systemu Windows:

```
Panel sterowania -> System i zabezpieczenia -> System.
```

W panelu `Zaawansowane ustawienia systemu` należy wybrać opcję `Zmienne środowiskowe` i w oknie `Zmienne systemowe` wybrać pole `Path`, a następnie dodać ścieżkę dostępu do narzędzi programu `Rtools`.

Instalację biblioteki na podstawie jej wersji źródłowej z rozszerzeniem `tar.gz` przeprowadza się w programie R lub RStudio z wykorzystaniem opcji `source`. W poniższym przykładzie podano sposób instalacji biblioteki `jsonlite`, która służy

[1] Por. https://github.com/hadley.
[2] Por. https://cran.r-project.org/bin/windows/Rtools/.

do komunikacji ze stronami internetowymi w popularnym formacie gromadzenia danych JSON (*JavaScript Object Notation*) (Ooms, 2025a).

```
install.packages('jsonlite', type = 'source').
```

Udana instalacja biblioteki kończy się komunikatem:

```
* DONE (jsonlite).
```

Należy pamiętać o wcześniejszym usunięciu starszej wersji biblioteki.

Podobną procedurę można przeprowadzić bezpośrednio w skrypcie programu R/RStudio. Dla przykładu podano sposób instalacji biblioteki MSBVAR, w której zaimplementowano metody analizy bayesowskich modeli wektorowej autoregresji z łańcuchem Markowa (Brandt, 2016). Ta biblioteka została przygotowana z wykorzystaniem języków programowania C++/R/Fortran/C i nie znajduje się już w repozytorium CRAN.

Wszystkie poniższe elementy kodu (chunk) należy umieścić w jednym skrypcie i uruchomić w *nowej* sesji programu RStudio (Session/New Session). Przyjęto, że ścieżki dostępu w systemie Windows znajdują się na dysku F: (należy zwrócić uwagę na symbole \\).

```
path <- "F:\\docs\\R\\b2\\ch1\\"
pack <- "MSBVAR-master"
```

Bibliotekę MSBVAR należy zapisać w katalogu path w formacie zip pod nazwą MSBVAR-master.zip z następującej lokalizacji:

- https://github.com/cran/MSBVAR/archive/master.zip.

Następnie trzeba sprawdzić ścieżkę dostępu do programu Rtools. Przyjęto, że ten program został domyślnie zainstalowany w katalogu C:\rtools44 (niektóre elementy kodu zostaną omówione w dalszej części książki).

```
get.paths <- strsplit(Sys.getenv("PATH"), ";")[[1]]
rtools <- grep(
  "rtools44",
  get.paths,
  value = TRUE,
  ignore.case = TRUE
)
cat("Ścieżki zawierające 'rtools44':\n")
```

```
## Ścieżki zawierające 'rtools44':
```

```
print(rtools)
```

```
## [1] "C:
rtools44/x86_64-w64-mingw32.static.posix/bin"
## [2] "C:
rtools44/usr/bin"
## [3] "C:
rtools44
x86_64-w64-mingw32.static.posix
bin"
## [4] "C:
rtools44
usr
bin"
## [5] "C:
rtools44"
```

Funkcja `Sys.getenv` zwraca ścieżki dostępu w systemie `Windows`. Ścieżka ta powinna zawierać dane dostępowe do programu `rtools44`. Jeśli powyższe ścieżki nie występują w zmiennej `PATH` użytkownika systemu `Windows`, to należy je ustawić za pomocą poniższego kodu.

```
Sys.setenv(
  PATH = paste0(
    paste(rtools, collapse = ";"), ";",
    Sys.getenv("PATH")
  )
)
```

W kolejnym kroku następuje *rozpakowanie* pliku `MSBVAR-master.zip` do katalogu o nazwie `MSBVAR-master` i *zbudowanie* archiwum `*.tar.gz`, które zawiera wszystkie pliki potrzebne w procesie kompilacji. W tym celu należy ustawić bieżący katalog roboczy i uruchomić procedurę kompilacji.

```
setwd(path)
unzip(paste0(
  path,
  pack,
  ".zip"
  )
)
shell(paste0(
  "R CMD build ",
  pack
  )
)
```

Ostatecznie powstaje archiwum o nazwie `MSBVAR_0.9-3.tar.gz` oznaczone numerem wersji biblioteki, zawierające pliki potrzebne do jej instalacji. Wystarczy teraz wydać polecenie do zainstalowania biblioteki z opcją `source` (wcześniej trzeba zainstalować niezbędne biblioteki:

- `coda` (Plummer i in., 2024),
- `bit` (Chirico i Oehlschlägel, 2025),
- `mvtnorm` (Genz i in., 2025),
- `xtable` (Dahl i in., 2019).

```
src.name = "MSBVAR_0.9-3.tar.gz"
install.packages(
  src.name,
  repos = NULL,
  type = "source"
)
```

Udana procedura instalacji biblioteki powinna zakończyć się komunikatem:

```
* DONE (MSBVAR).
```

Na koniec należy sprawdzić, czy biblioteka została poprawnie zainstalowana.

```
library(MSBVAR)
```

```
## ##
## ## MSBVAR Package v.0.9-2
## ## Build date:  Sun Dec 21 16:10:47 2025
## ## Copyright (C) 2005-2025, Patrick T. Brandt
## ## Written by Patrick T. Brandt
## ##
## ## Support provided by the U.S. National Science Foundation
## ## (Grants SES-0351179, SES-0351205, SES-0540816, and SES-0921051)
## ##
```

W kolejnym rozdziale przedstawione zostanie wprowadzenie do programowania w języku R, które stanowi fundament dalszych analiz i przykładów zawartych w niniejszej książce. Omówione zostaną podstawowe zasady tworzenia kodu, struktura programu oraz rola algorytmicznego myślenia w pracy statystyka i ekonometryka.

Rozdział 2

Struktury danych i programowanie

2.1. Wprowadzenie

Programowanie stanowi nieodłączny element warsztatu pracy statystyka i ekonometryka. Opanowanie umiejętności programowania może mieć zasadnicze znaczenie dla rozwiązywania problemów zarówno w obszarze teorii ekonomii, jak i w badaniach empirycznych. Tworzenie oprogramowania nie jest już wyłącznie rzemiosłem – stało się dyscypliną naukową, która ma ścisły związek z naukami inżynieryjno-technicznymi oraz naukami ścisłymi i przyrodniczymi. Algorytmika i programowanie stanowią filary dyscypliny naukowej, jaką jest informatyka. Programowanie stało się dziedziną wiedzy, którą można analizować i przedstawiać w sposób naukowy (Dahl i in., 1972; Wirth, 1989). Warto wspomnieć o niezwykle istotnym czasopiśmie naukowym *Journal of Statistical Software* (J STAT SOFTW), założonym w 1996 roku, w którym publikowane są artykuły dotyczące konstrukcji i zastosowań języka R oraz innych języków programowania w statystyce. To czasopismo jest indeksowane w najważniejszych bazach danych[1]. Wśród innych interdyscyplinarnych wydawnictw warto wymienić indeksowane czasopismo *Statistical Analysis and Data Mining*, wydawane od 2008 roku[2].

[1] Wskaźnik cytowań w 2023 roku wynosił 5,4.
[2] Wskaźnik cytowań w 2023 roku wynosił 3,2.

2.2. Podstawy programowania i obiekty

Metodyka programowania wymaga uwzględnienia algorytmów i struktur danych, które pozwalają rozwiązać dany problem numeryczny. Program zapisany w języku programowania stanowi implementację tych struktur oraz algorytmu, czyli opisu postępowania prowadzącego do rozwiązania. Podejście naukowe do programowania jest szczególnie istotne w przypadku przetwarzania dużych i wielowymiarowych zbiorów danych. Algorytmy, struktury danych i język R umożliwiają tworzenie bibliotek obliczeniowych, które znajdują zastosowanie przede wszystkim w statystyce i ekonometrii. Podstawy programowania omówiono szczegółowo w opracowaniu Wdowiński (2020)[3]. W niniejszej książce, w obszarze programowania, skupiono się przede wszystkim na zastosowaniu obiektów języka R i struktur danych w przetwarzaniu potokowym oraz w graficznym raportowaniu wyników. Przedstawiono także wybrane zagadnienia modelowania i prognozowania ekonometrycznego. W kolejnych podrozdziałach omówiono najważniejsze aspekty struktur danych oraz programowania w języku R.

2.2.1. Przetwarzanie potokowe

Metoda przetwarzania potokowego (*pipeline*) stanowi ważny element programowania w języku R. Polega na przekazywaniu wartości z lewej strony wyrażenia do wyrażenia po prawej stronie, gdzie staje się ona argumentem funkcji zdefiniowanej przez kolejne wyrażenie. Polecenia zapisuje się w kolejności odpowiadającej naturalnemu przebiegowi operacji. W tym podejściu nie tworzy się zmiennych pomocniczych, dzięki czemu zapis całego polecenia jest przejrzysty i łatwy do odczytania (zob. szerzej: Wickham i Grolemund, 2016). W przetwarzaniu potokowym stosuje się specjalny operator %>% wprowadzony w bibliotece `magrittr` (Bache i Wickham, 2022). Struktura zapisu polecenia jest następująca:

- `x %>% f` oznacza zwrócenie wartości `f(x)`,
- `x %>% f(y)` oznacza zwrócenie wartości `f(x, y)`,
- `x %>% f %>% g %>% z` oznacza zwrócenie wartości `z(g(f(x)))`.

Wartość wyrażenia z lewej strony (argument) można przekazać do wyrażenia po prawej stronie za pomocą operatora `.` (kropki):

- `x %>% f(y, .)` oznacza zwrócenie wartości `f(y, x)`,
- `x %>% f(y, h = .)` oznacza zwrócenie wartości `f(y, h = x)`.

[3] Zagadnienia przedstawione w podrozdziale 2.2 odnoszą się do pojęć omówionych w książce Wdowiński (2020).

Operator . można stosować wielokrotnie w złożonych wyrażeniach. Należy przy tym pamiętać, że argument z lewej strony wyrażenia zostanie automatycznie wykorzystany w funkcji po prawej stronie wyrażenia:

- `x %>% f(y = mean(.), z = median(.))` oznacza zwrócenie wartości `f(x, y = mean(x), z = median(x))`.

Można temu zapobiec, stosując nawiasy klamrowe `{}`:

- `x %>% { f(y = mean(.), z = median(.)) }` oznacza zwrócenie wartości `f(y = mean(x), z = median(x))`.

Za pomocą przetwarzania potokowego można również definiować nowe funkcje. Wystarczy dokonać przypisania, które rozpoczyna się od symbolu kropki (`.`):

- `g <- . %>% mean` oznacza zwrócenie funkcji `g <- function(x) mean(x)`.

W kolejnym podrozdziale zostaną przedstawione instrukcje warunkowe, które stanowią podstawowy element logiki programowania. Umożliwiają one wykonywanie różnych fragmentów kodu w zależności od tego, czy określone warunki są spełnione, co pozwala tworzyć bardziej elastyczne programy w języku R.

2.2.2. Instrukcje warunkowe

W języku R, podobnie jak w innych językach programowania wyższego poziomu, stosuje się instrukcje warunkowe. Bez takich instrukcji tworzenie rozbudowanego kodu nie byłoby możliwe. Najprostszą formą instrukcji warunkowej jest funkcja `if`. Jeśli warunek jest spełniony, czyli ma wartość TRUE, wykonywany jest odpowiedni fragment kodu programu. W przeciwnym razie kod programu jest ignorowany.

```
if (warunek) {
    `[kod]`
}
```

Jeśli `warunek` jest spełniony, funkcja `if` przyjmuje wartość TRUE i `kod` pomiędzy znacznikami `{}` jest wykonywany. Jeśli `warunek` nie jest spełniony, funkcja `if` przyjmuje wartość FALSE. Można również uwzględnić przypadek, gdy `warunek` nie jest spełniony, stosując rozszerzoną postać funkcji `if` z klauzulą `else`.

```
if (warunek) {
    `[kod.1]`
} else {
    `[kod.2]`
}
```

Funkcje `if` można zagnieżdżać, aby precyzyjniej sterować wykonywaniem kodu w bardziej złożony sposób.

```
if (warunek.1) {
    `[kod.1]`
} else {
    if (warunek.2) {
        `[kod.2]`
    } else {
        `[kod.3]`
    }
}
```

W powyższym przykładzie `kod.1` zostanie wykonany, jeśli *jest* spełniony `warunek.1`. Następnie zostanie wykonany `kod.2` lub `kod.3`, jeśli `warunek.1` *nie jest* spełniony. Łatwo zauważyć, że `kod.3` zostanie wykonany tylko wówczas, gdy warunki `warunek.1` i `warunek.2` *nie są* spełnione. Można stosować wiele warunków w funkcji `if` i łączyć je za pomocą odpowiednich operatorów logicznych.

```
if (warunek.1 & warunek.2) {
    `[kod]`
}
```

Powyższa sekwencja oznacza, że kod zostanie wykonany, jeśli *jednocześnie* spełnione są oba warunki. Czasami wygodne jest zastosowanie negacji warunku przy użyciu symbolu `!`. Można na przykład zastosować następujące rozwiązanie:

```
if (!warunek) {
    `[kod]`
}
```

Warunki mogą przyjmować różne formy. Jeśli zachodzi potrzeba sprawdzenia, czy ciąg znaków (`string`) występuje w wektorze (zob. podrozdział 2.2.5), można zastosować operator `%in%`.

```
l <- c("jeden", "dwa", "trzy")
x <- "jeden"
if (x %in% l) { print("kod") }
```

W powyższym przykładzie kod pomiędzy znacznikami `{}` zostanie wykonany, ponieważ obiekt (zmienna) `x` przyjmuje wartość, która znajduje się w wektorze `l`.

Warto zauważyć, że instrukcje warunkowe stanowią podstawę sterowania przepływem programu, jednak w wielu przypadkach konieczne jest powtarzanie określonych operacji dla różnych wartości zmiennych lub elementów zbioru danych. W takich sytuacjach niezbędne staje się wykorzystanie mechanizmów umożliwiających automatyczne, iteracyjne wykonywanie kodu.

W kolejnym podrozdziale omówione zostaną *pętle*, które pozwalają na realizację powtarzalnych obliczeń w języku R. Zostaną przedstawione ich główne rodzaje, zasady działania oraz przykłady zastosowań z uwzględnieniem pętli prostych i zagnieżdżonych. Szczególna uwaga zostanie zwrócona na efektywność obliczeń i znaczenie wektoryzacji w programowaniu.

2.2.3. Pętle

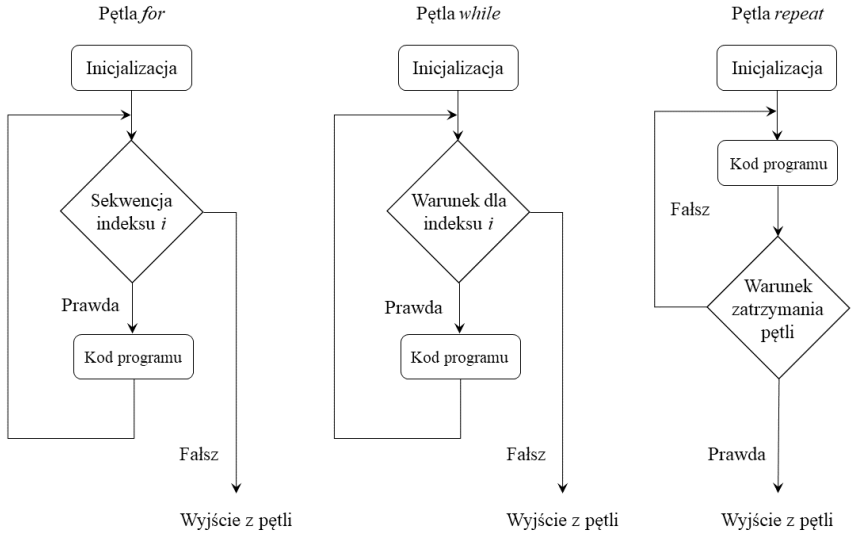
W podrozdziale 2.2.9 zostanie zwrócona uwaga na możliwość wykonywania obliczeń *iteracyjnych* przy wykorzystaniu obiektu typu `lista`. Jest to bardzo przydatny sposób programowania ze względu na łatwość tworzenia i indeksowania obiektów za pomocą funkcji `lapply`. W programie R istnieje również inna możliwość wykonywania obliczeń iteracyjnych – służą do tego pętle (`loop`).

Pętle mają szerokie zastosowanie w wielu językach programowania. Należy zwrócić uwagę, że w pętlach – wszędzie tam, gdzie to możliwe – należy stosować dostępną wektoryzację obliczeń (zob. podrozdział 2.2.5). Inne zastosowania pętli będą z reguły nieefektywne (Spector, 2008).

W języku R występuje kilka rodzajów pętli:

- `for`,
- `while`,
- `repeat`.

Dodatkowo stosuje się *klauzule* sterujące kodem `break` i `next`. Klauzula `break` służy do natychmiastowego przerwania działania pętli. Po jej wykonaniu działanie pętli kończy się całkowicie, a sterowanie programu przechodzi do pierwszego polecenia znajdującego się po pętli. Stosuje się ją np. wtedy, gdy dalsze iteracje nie mają sensu po spełnieniu określonego warunku. Klauzula `next` powoduje przerwanie bieżącej iteracji pętli, ale nie kończy działania pętli jako całości. Program przechodzi od razu do kolejnej iteracji, pomijając pozostałe instrukcje w aktualnym przebiegu pętli. Jest użyteczna, gdy dla pewnych wartości spełniony warunek uniemożliwia wykonanie reszty instrukcji, ale kolejne iteracje są nadal potrzebne. Rysunek 2.1 przedstawia diagramy działania poszczególnych rodzajów pętli.



Rysunek 2.1. Rodzaje pętli w programie R

Przykład pętli `for` przedstawiono poniżej. Indeksem tej pętli jest zmienna `i`, która przyjmuje kolejno wartości od 1 do 10.

```
for (i in 1:10) {
    print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

Można również tworzyć zagnieżdżone pętle `for`, co ilustruje poniższy przykład. Należy zauważyć, że każda pętla ma inny indeks `i` lub `j`. Indeks `j` drugiej pętli może nawet zależeć od indeksu `i` pierwszej pętli.

```
for (i in 1:3) {
    for (j in 8:10) {
        print(i*j)
    }
}
```

```
## [1] 8
## [1] 9
## [1] 10
## [1] 16
## [1] 18
## [1] 20
## [1] 24
## [1] 27
## [1] 30
```

Pętla `for` może być również indeksowana za pomocą wartości innych niż numeryczne. Ilustruje to poniższy przykład, w którym indeks `i` przyjmuje wartości znakowe kolejnych małych liter (wektor `letters`).

```
for (i in letters[1:5]) {
    print(i)
}
```

```
## [1] "a"
## [1] "b"
## [1] "c"
## [1] "d"
## [1] "e"
```

Pętla `for` może być indeksowana wartościami wektora. Trzeba przy tym pamiętać, że wektor powinien zawierać wartości tego samego typu.

```
for (i in c(1, letters[1:3], "text")) {
    print(i)
}
```

```
## [1] "1"
## [1] "a"
## [1] "b"
## [1] "c"
## [1] "text"
```

Zagnieżdżoną pętlę `for` można wykorzystać do utworzenia ramki danych (zob. podrozdział 2.2.7), jak w poniższym przykładzie.

```
df <- data.frame()
for (i in 1:3) {
    for (j in 1:5) {
        df[i, j] <- i*j
    }
}
colnames(df) <- c(letters[1:ncol(df)])
df
```

```
##   a b c  d  e
## 1 1 2 3  4  5
## 2 2 4 6  8 10
## 3 3 6 9 12 15
```

Pętla `while` ma inną konstrukcję niż pętla `for`, gdyż jej wykonywanie jest uzależnione od spełnienia warunku (zwanego dalej warunkiem logicznym) zapisanego w nawiasie. W poniższym przykładzie indeksem pętli jest zmienna `i`, której wartość początkowa musi zostać przypisana przed rozpoczęciem wykonywania pętli. Jeśli warunek logiczny w nawiasie pętli jest spełniony, to pętla jest wykonywana. Dalsze wykonywanie pętli jest zależne od aktualnej wartości indeksu `i`. Przyjmuje się, że pętla `while` jest wykonywana co najmniej jeden raz. Następnie indeks pętli może być zmodyfikowany tak, aby pętla nie została wykonana po raz kolejny. Dzięki temu można sterować liczbą wykonań pętli. Należy w związku z tym pamiętać, że kod wewnątrz pętli `while` *musi* mieć wpływ na wartości indeksu pętli.

```
i <- 1
while (i < 5) {
    print(i)
    i <- i+1
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
```

Pętlę `while` można wykorzystać do obliczeń, w których liczba iteracji nie jest znana z góry, np. w zagadnieniach optymalizacyjnych wykonywanych aż do momentu, gdy algorytm osiągnie zbieżność do pewnej wartości ϵ.

Rodzajem pętli w języku R jest również pętla `repeat`, której konstrukcja jest zbliżona do pętli `while`. Główna różnica polega na tym, że pętla `repeat` wykonuje się co najmniej jeden raz, niezależnie od warunku. Instrukcje w pętli są wykonywane tak długo, aż zostanie spełniony warunek przerwania pętli. Poniżej podano przykład pętli `repeat`.

```
i <- 1
repeat {
    print(i)
    i <- i+1
    if (i == 5){
        break
    }
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
```

W każdej chwili można przerwać wykonywanie pętli za pomocą polecenia `break`, które zostało wprowadzone w pętli `repeat`. Jeśli interpreter programu R napotka na polecenie `break`, to natychmiast zakończy działanie bieżącej pętli i wykona polecenia znajdujące się bezpośrednio po niej.

Kolejnym przykładem kontroli nad przebiegiem pętli jest polecenie `next`. Powoduje ono przejście pętli `for` do następnej iteracji. Ilustruje to poniższy przykład.

```
for (i in 1:5) {
    if (i == 4){
        next
    }
    print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 5
```

W następnym podrozdziale podano zarys zagadnień dotyczących tworzenia *funkcji*.

2.2.4. Funkcje

W programie R można korzystać z wbudowanych funkcji. Można do nich zaliczyć funkcję `date`. Funkcja ta zwraca aktualną wartość daty i czasu uzyskaną z systemu operacyjnego.

```
date()
```

```
## [1] "Sun Dec 21 16:10:53 2025"
```

Można również budować własne funkcje. Służy do tego funkcja o nazwie `function`. Do funkcji przekazuje się parametry, natomiast wartość funkcji zwraca się za pomocą funkcji `return`. Najprostsza funkcja może mieć postać podaną poniżej.

```
func <- function() {
    print("To jest funkcja")
}
func()
```

```
## [1] "To jest funkcja"
```

Przekazanie parametrów do funkcji odbywa się w następujący sposób:

```
func <- function(x) {
    print(x)
}
func(1)
```

```
## [1] 1
```

```
func("To jest funkcja z parametrem x")
```

```
## [1] "To jest funkcja z parametrem x"
```

Jeśli w deklaracji funkcji występuje więcej poleceń, to należy wskazać, który obiekt ma zostać zwrócony jako wartość funkcji. W przeciwnym razie zwracany jest ostatni wygenerowany obiekt.

```
func.1 <- function(x) {
    x^3
    x^2
}
func.1(2)
```

```
## [1] 4
```

```
func.2 <- function(x) {
    y <- x^3
    z <- x^2
    return(y)
}
func.2(2)
```

```
## [1] 8
```

Za pomocą funkcji `return` można zwrócić wartość tylko jednego obiektu. Aby zwrócić większą liczbę obiektów, należy utworzyć listę tych obiektów (zob. podrozdział 2.2.9) i zwrócić jej wartość.

```
func <- function(x) {
    a <- 1
    b <- "dwa"
    c <- x^3
    myList <- list(a = a, b = b, c = c)
    return(myList)
}
func(2)
```

```
## $a
## [1] 1
##
## $b
## [1] "dwa"
##
## $c
## [1] 8
```

Dla pełnej powtarzalności wyników symulacyjnych uzyskanych w tej książce ustawione zostało również *ziarno* generatora liczb pseudolosowych[4]. Ziarno generatora należy ustawić w każdym fragmencie kodu (`chunk code`). Najlepiej w tym celu przygotować funkcję, która będzie wywoływana przy każdym uruchomieniu kodu.

```
set_seed <- function() set.seed(1234)
```

W R występuje ponadto operator `...` (trzy kropki), który pozwala na przekazanie dowolnej liczby dodatkowych argumentów do funkcji. Zwiększa to jej elastyczność

[4] Ziarno generatora liczb pseudolosowych to początkowy stan deterministycznego algorytmu, od którego zależy cały późniejszy przebieg działania generatora i który jednoznacznie określa generowaną sekwencję liczb.

i umożliwia łatwą integrację z innymi funkcjami. Funkcje wykorzystujące ten operator są szczególnie przydatne w przypadku dynamicznie zmieniających się wymagań, gdzie nie wszystkie parametry mogą być z góry określone. Dzięki operatorowi `...` można łatwo przekazać dodatkowe argumenty do wewnętrznych funkcji, takich jak `plot` lub zarządzać specyficznymi ustawieniami w sposób przejrzysty i intuicyjny. Poniżej podano przykład zastosowania funkcji z takim operatorem.

```
f <- function(x, y, ...) {
  if (missing(x) || missing(y)) {
    print("Nie podano wartości x lub y.")
    return(invisible(NULL))
  }
  extra <- list(...)
  result <- mean(x + y)
  print(paste("Średnia wartość x+y:", result))
  plot(x, y, ...)
  invisible(result)
  return(extra)
}
```

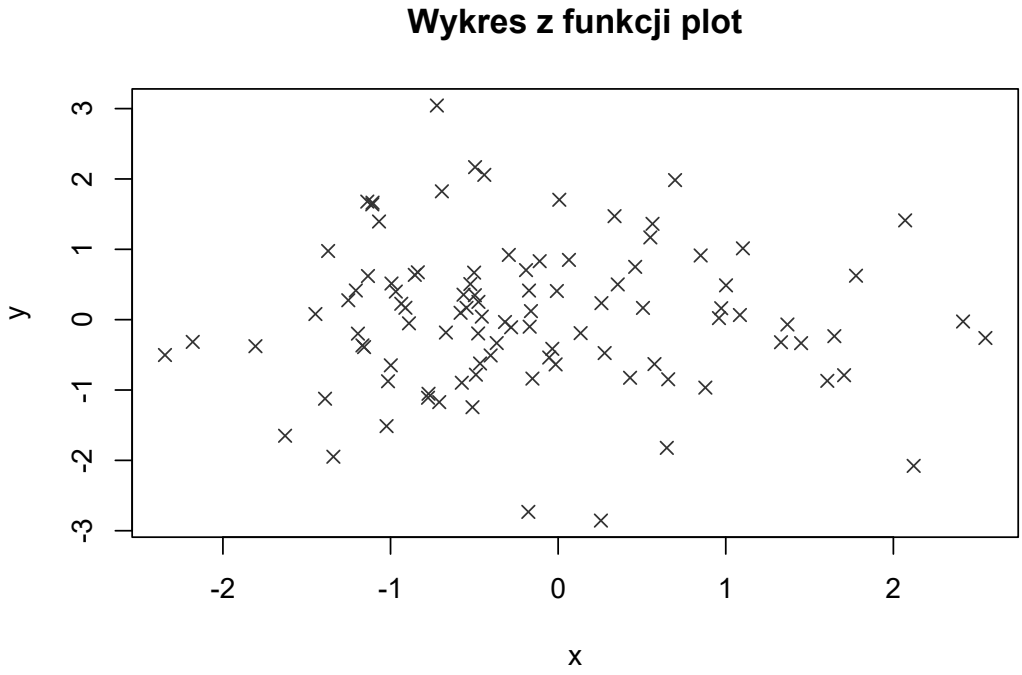
Należy zauważyć, że zastosowanie operatora logicznego `||` (OR, lub) zamiast `|` w warunku `missing(x) || missing(y)` wynika z faktu, że `||` jest operatorem logicznym używanym do *szybkiej* (*short-circuit evaluation*) oceny pojedynczych wartości logicznych. Operator `|` dokonuje oceny element po elemencie (*element-wise*) w wektorach logicznych. Ta różnica ma znaczenie, gdyż operator `||` ocenia tylko pierwszy element każdego warunku i przerywa dalszą ocenę, jeśli pierwszy warunek jest spełniony (`TRUE`). To zwiększa efektywność obliczeń w przypadku pracy na wartościach skalarnych, takich jak `missing(x)` i `missing(y)`. Wywołanie funkcji `f` podano poniżej (por. również rys. 2.2).

```
f(y = rnorm(100), col = "blue", pch = 19, type = "l")
```

```
## [1] "Nie podano wartości x lub y."
```

```
set_seed()
f(
  x = rnorm(100),
  y = rnorm(100),
  col = grey(0.2),
  pch = 4,
  main = "Wykres z funkcji plot"
)
```

```
## [1] "Średnia wartość x+y: -0.115518562495681"
```



Rysunek 2.2. Zmienna o rozkładzie normalnym standaryzowanym – funkcja `plot` z argumentami przydzielanymi dynamicznie

```
## $col
## [1] "#333333"
##
## $pch
## [1] 4
##
## $main
## [1] "Wykres z funkcji plot"
```

Argumenty `x` i `y` są wymagane i służą do wykonania podstawowych obliczeń lub operacji. Operator `...` pozwala na dodanie dodatkowych argumentów, które mogą być używane wewnątrz funkcji (`extra`) lub przekazywane do innych funkcji (np. `plot`). Funkcja `invisible` zwraca wynik (średnia `x+y`) bez automatycznego wyświetlania. W powyższym przykładzie operator `...` pozwala na przekazanie argumentów do funkcji `plot` wewnątrz funkcji `f` oraz umożliwia obsługę dowolnych, elastycznie definiowanych dodatkowych parametrów. Operator `...` pozwala również przekazywać do funkcji argumenty, które nie zostały wcześniej wymienione w jej liście parametrów, dzięki czemu funkcja może obsługiwać dowolną liczbę dodatkowych ustawień, w zależności od potrzeb użytkownika.

Na zakończenie warto podkreślić, że każda analiza danych w języku R opiera się na odpowiednim sposobie ich przechowywania i przetwarzania. Zrozumienie struktury danych jest zatem kluczowe dla efektywnego programowania i wykonywania obliczeń statystycznych.

W kolejnym podrozdziale zostanie przedstawiona jedna z najważniejszych struktur danych w języku R – *wektor*. Omówione zostaną jego podstawowe typy, sposoby tworzenia oraz znaczenie w kontekście operacji arytmetycznych, logicznych i tekstowych. Wektory stanowią fundament dla bardziej złożonych obiektów, takich jak macierze, ramki danych czy listy, dlatego ich znajomość jest niezbędna do dalszej pracy w środowisku R.

2.2.5. Wektory

Wektor stanowi jedną z podstawowych struktur danych w programie R (Wickham i in., 2023a). Wyróżnia się cztery podstawowe typy wektorów:

- logiczne (*logical*),
- całkowitoliczbowe (*integer*),
- zmiennoprzecinkowe (numeryczne) (*double*),
- znakowe (*character*).

Wektor tworzy się za pomocą polecenia `c()` (*combine*).

```
v.1 <- c(1, 2, 3)
v.1
```

```
## [1] 1 2 3
```

Wektor `v.1` jest wektorem wartości liczbowych (rzeczywistych). Jeśli wektor zawiera wyłącznie liczby, można na nim wykonywać operacje arytmetyczne.

```
v.1 * 2
```

```
## [1] 2 4 6
```

```
v.1 - 1
```

```
## [1] 0 1 2
```

Liczbę współrzędnych wektora określa się za pomocą funkcji `length`. Typ wektora oraz innych obiektów można sprawdzić za pomocą funkcji `typeof` lub bardziej ogólnej funkcji `class`.

```
length(v.1)
```

```
## [1] 3
```

```
typeof(v.1)
```

```
## [1] "double"
```

```
class(v.1)
```

```
## [1] "numeric"
```

Wektor `v.2` jest wektorem liczb całkowitych.

```
v.2 <- c(1L, 5L, 10L)
v.2
```

```
## [1]  1  5 10
```

```
typeof(v.2)
```

```
## [1] "integer"
```

```
class(v.2)
```

```
## [1] "integer"
```

Wektory numeryczne tworzy się również w sposób podany poniżej.

```
v.3 <- c(1:5)
v.3
```

```
## [1] 1 2 3 4 5
```

```
v.4 <- c(1, 2, 3, 4, 5)
v.4
```

```
## [1] 1 2 3 4 5
```

Możliwe jest zagnieżdżanie wektorów, przy czym nie powoduje to zmiany wymiaru wynikowego wektora.

```
v.5 <- c(1, c(2, 3), c(4, c(5, 6)))
v.5
```

```
## [1] 1 2 3 4 5 6
```

```
v.6 <- c(1:6)
v.6
```

```
## [1] 1 2 3 4 5 6
```

Wektory znakowe tworzy się w sposób podany poniżej.

```
v.7 <- c("jeden", "dwa", "trzy")
v.7
```

```
## [1] "jeden" "dwa"   "trzy"
```

```
typeof(v.7)
```

```
## [1] "character"
```

```
class(v.7)
```

```
## [1] "character"
```

Wektor w języku R zawiera dane *tego samego* typu.

```
v.8 <- c(1, "dwa", "3")
v.8
```

```
## [1] "1"   "dwa" "3"
```

```
typeof(v.8)
```

```
## [1] "character"
```

```
class(v.8)
```

```
## [1] "character"
```

Wektor `v.8` to również wektor znakowy, mimo iż jego pierwsza współrzędna ma wartość liczbową, gdyż wektor zawiera dane *tego samego* typu.

Wektor wartości logicznych jest przykładem najprostszego wektora, ponieważ jego współrzędne mogą przyjąć jedną z trzech wartości: `TRUE`, `FALSE` i `NA`. Wektor wartości logicznych tworzy się w sposób pokazany poniżej.

```
v.9 <- c(TRUE, FALSE, T, F)
v.9
```

```
## [1]  TRUE FALSE  TRUE FALSE
```

```
typeof(v.9)
```

```
## [1] "logical"
```

```
class(v.9)
```

```
## [1] "logical"
```

Prawie każda analiza w języku R zawiera wektory logiczne, na których wykonywane są operacje. Powszechnym sposobem tworzenia wektorów logicznych jest zastosowanie operatorów porównania <, <=, >, >=, != i ==. Łatwo teraz sprawdzić, że wektory `v.5` i `v.6` są identyczne.

```
v.5 == v.6
```

```
## [1] TRUE TRUE TRUE TRUE TRUE TRUE
```

Można utworzyć wektor wartości logicznych na podstawie porównania wektorów.

```
v.10 <- v.5 == v.6
v.10
```

```
## [1] TRUE TRUE TRUE TRUE TRUE TRUE
```

Szczególną uwagę należy zwrócić na operator porównania ==. Można np. utworzyć następujący wektor:

```
v.11 <- c(1/49*49, sqrt(2)^2)
v.11
```

```
## [1] 1 2
```

Dokładniejsze przyjrzenie się wektorowi `v.11` wskazuje na wartości podane poniżej.

```
print(v.11, digits = 16)
```

```
## [1] 0.9999999999999999 2.0000000000000004
```

Z podanego powodu porównanie `v.11 == c(1, 2)` daje następujący wektor:

```
v.11 == c(1, 2)
```

```
## [1] FALSE FALSE
```

Do wektorów logicznych można stosować strukturę algebraiczną Boole'a (Boole, 1854). W języku R symbol & oznacza koniunkcję, | alternatywę, `xor()` alternatywę rozłączną oraz ! negację.

```
v.12 <- c(-5:5)
v.12 < -3 & v.12 > 3
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 ↪  FALSE
```

```
v.12 < -3 | v.12 > 3
```

```
##  [1]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
 ↪  TRUE
```

```
v.12 > 3
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
 ↪  TRUE
```

```
v.12 != 0
```

```
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
 ↪  TRUE
```

```
!v.12 < 0
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
 ↪  TRUE
```

```
v.12 < -2
```

```
##  [1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 ↪  FALSE
```

```
v.12 < -4
```

```
##  [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 ↪  FALSE
```

```
xor(v.12 < -2, v.12 < -4)
```

```
##  [1] FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 ↪  FALSE
```

Wcześniej wspomniano, że wektor wartości logicznych może również przyjąć wartość nieokreśloną NA. W konsekwencji wyniki porównań z udziałem NA mogą być nieokreślone, jeśli zastosuje się standardowe operatory porównania.

```
NA > 0
```

```
## [1] NA
```

```
NA == 0
```

```
## [1] NA
```

```
NA == NA
```

```
## [1] NA
```

Do przypadków wektorów logicznych zawierających wartość NA należy podchodzić z ostrożnością. Taką sytuację przedstawia poniższy przykład.

```
NA | TRUE
```

```
## [1] TRUE
```

```
NA | FALSE
```

```
## [1] NA
```

```
NA & FALSE
```

```
## [1] FALSE
```

W języku R występują dwie funkcje agregujące wyrażenia logiczne: any i all. Pierwsza odpowiada alternatywie |, natomiast druga koniunkcji &. Funkcje te mogą zwrócić wartość NA, jeśli w wyrażeniu logicznym występują wartości NA. Można temu zapobiec poprzez użycie argumentu na.rm = TRUE.

```
v.13 <- c(1:5)
v.14 <- v.13 + 1
any(v.13 == v.14)
```

```
## [1] FALSE
```

```
all(v.13 < v.14)
```

```
## [1] TRUE
```

Wektory logiczne można również stosować w kontekście numerycznym. W tym przypadku wartość TRUE oznacza 1, natomiast FALSE jest równe 0.

```
v.15 <- c(TRUE, TRUE, FALSE, FALSE)
sum(v.15)
```

```
## [1] 2
```

```
mean(v.15)
```

```
## [1] 0.5
```

```
sum(v.15) / length(v.15)
```

```
## [1] 0.5
```

Do porównywania z wartością NA stosuje się funkcję `is.na`.

```
is.na(c(TRUE, FALSE, NA))
```

```
## [1] FALSE FALSE  TRUE
```

Typ wektora można ustalić za pomocą funkcji:

- `is.integer`,
- `is.atomic`,
- `is.double`,
- `is.numeric`.

Każda z tych funkcji zwraca wartość logiczną TRUE lub FALSE w zależności od typu wektora. Warto zauważyć, że funkcja `is.numeric` zwraca wartość TRUE zarówno dla wektorów typu `integer`, jak i `double`.

Współrzędnym wektora można nadać nazwę.

```
v <- c(x = 1, y = 2, z = 3)
v
```

```
## x y z
## 1 2 3
```

```
names(v)
```

```
## [1] "x" "y" "z"
```

Nazwy współrzędnych można łatwo usunąć.

```
names(v) <- NULL
names(v)
```

```
## NULL
```

Aby odwołać się do elementu wektora, należy podać jego współrzędną w nawiasie kwadratowym, tj. `[i]`.

```
v[1]
```

```
## [1] 1
```

```
v[1:2]
```

```
## [1] 1 2
```

```
v[c(1, 3)]
```

```
## [1] 1 3
```

```
v[c(3, 1)]
```

```
## [1] 3 1
```

Zapisanie współrzędnej w podwójnym nawiasie `[[]]` powoduje podanie wartości wektora bez nazw współrzędnych.

```
v <- c(x = 1, y = 2, z = 3)
names(v)
```

```
## [1] "x" "y" "z"
```

```
v[1]
```

```
## x
## 1
```

```
v[[1]]
```

```
## [1] 1
```

Wektory można również tworzyć za pomocą funkcji `rep` i `seq`.

```
seq(from = 1, to = 10, by = 2)
```

```
## [1] 1 3 5 7 9
```

```
seq(1, 10, 2)
```

```
## [1] 1 3 5 7 9
```

```
rep(1, 10)
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1
```

```
rep("A", 10)
```

```
##  [1] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
```

```
rep(1:2, 5)
```

```
##  [1] 1 2 1 2 1 2 1 2 1 2
```

```
rep(c(1, 4), 5)
```

```
##  [1] 1 4 1 4 1 4 1 4 1 4
```

W języku `R` można również tworzyć wektory czynnikowe (`factors`) dla zmiennych jakościowych, które przyjmują wartości ze skończonego zbioru. Dla ilustracji problemu został utworzony wektor znakowy z nazwami miesięcy.

```
x <- c("styczeń", "kwiecień", "luty", "marzec")
x
```

```
## [1] "styczeń"  "kwiecień" "luty"     "marzec"
```

```
typeof(x)
```

```
## [1] "character"
```

```
class(x)
```

```
## [1] "character"
```

Sortowanie wartości takiej zmiennej zwykle nie przynosi pożądanego efektu.

```
sort(x)
```

```
## [1] "kwiecień" "luty"     "marzec"   "styczeń"
```

Zmienną znakową można przekształcić w zmienną czynnikową. Najpierw trzeba określić czynniki (`levels`).

```
y <- c(
  "styczeń",
  "luty",
  "marzec",
  "kwiecień",
  "maj",
  "czerwiec",
  "lipiec",
  "sierpień",
  "wrzesień",
  "październik",
  "listopad",
  "grudzień"
)
y
```

```
##  [1] "styczeń"     "luty"        "marzec"      "kwiecień"    "maj"
##  [6] "czerwiec"    "lipiec"      "sierpień"    "wrzesień"
 ↪  "październik"
## [11] "listopad"    "grudzień"
```

Następnie można przekształcić zmienną x w zmienną czynnikową z.

```
z <- factor(x, levels = y)
z
```

```
## [1] styczeń  kwiecień luty     marzec
## 12 Levels: styczeń luty marzec kwiecień maj czerwiec lipiec ...
 ↪  grudzień
```

Wartości zmiennej z można już łatwo sortować.

```
sort(z)
```

```
## [1] styczeń  luty     marzec   kwiecień
## 12 Levels: styczeń luty marzec kwiecień maj czerwiec lipiec ...
 ↪  grudzień
```

Czynniki pobiera się za pomocą funkcji `levels`.

```
levels(z)
```

```
##  [1] "styczeń"     "luty"        "marzec"      "kwiecień"    "maj"
##  [6] "czerwiec"    "lipiec"      "sierpień"    "wrzesień"
 ↪  "październik"
## [11] "listopad"    "grudzień"
```

Jeśli nie uwzględni się odrębnych czynników, to zostaną one pobrane z danych i posortowane alfabetycznie.

```
z <- factor(x)
z
```

```
## [1] styczeń  kwiecień luty     marzec
## Levels: kwiecień luty marzec styczeń
```

```
levels(z)
```

```
## [1] "kwiecień" "luty"     "marzec"   "styczeń"
```

W celu określenia zastosowań czynników wykorzystano zbiór danych `iris`. Źródłem danych są następujące artykuły naukowe: Anderson (1935) i Fisher (1936). Zbiór `iris` to wbudowany w R zbiór danych, który zawiera pomiary 4 różnych atrybutów (w centymetrach) dla 50 kwiatów z 3 różnych gatunków (*iris setosa*, *iris versicolor*, *iris virginica*). Informację o zbiorach dostępnych w programie R można wyświetlić za pomocą polecenia `library(help = "datasets")`.

```
l <- library(help = "datasets")
```

Obiekt `l` jest listą. Więcej informacji na temat list można znaleźć w podrozdziale 2.2.9.

```
id <- grep("iris", l[[3]][[2]])
l[[3]][[1]]
```

```
##  [1] "Package:       datasets"
##  [2] "Version:       4.4.3"
##  [3] "Priority:      base"
##  [4] "Title:         The R Datasets Package"
##  [5] "Author:        R Core Team and contributors worldwide"
##  [6] "Maintainer:    R Core Team
 ↪  <do-use-Contact-address@r-project.org>"
##  [7] "Contact:       R-help mailing list <r-help@r-project.org>"
##  [8] "Description:   Base R datasets."
##  [9] "License:       Part of R 4.4.3"
## [10] "Built:         R 4.4.3; ; 2025-02-28 10:25:04 UTC; windows"
```

```
l[[3]][[2]][id]
```

```
## [1] "iris                     Edgar Anderson's Iris Data"
```

Najpierw trzeba zainstalować bibliotekę `forcats` (Wickham, 2023a) z ekosystemu `tidyverse` (Wickham, 2023b) do obsługi zmiennych jakościowych.

```
library(forcats)
```

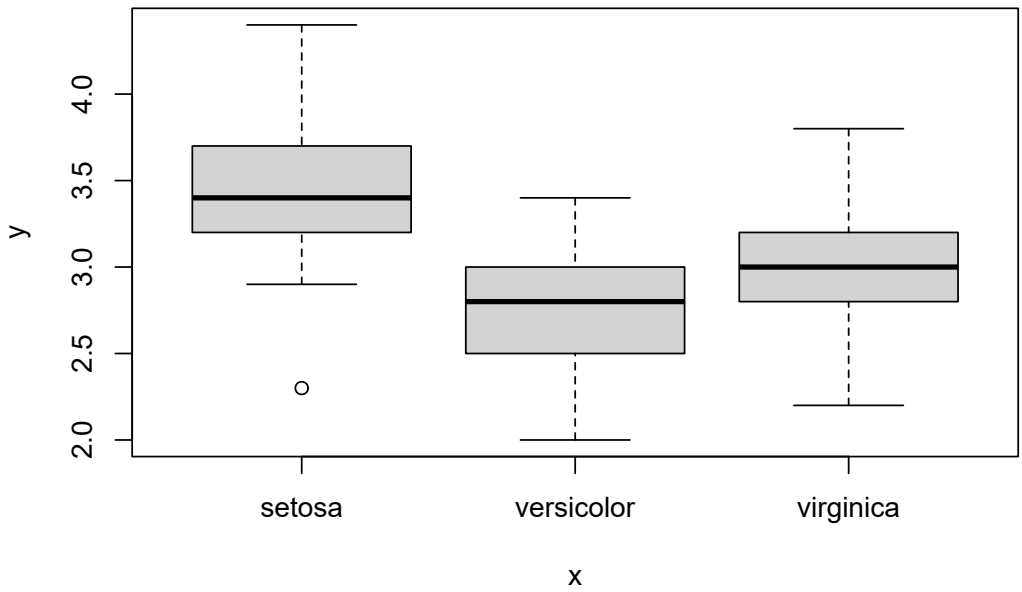
Następnie należy pobrać zmienne ze zbioru `iris` i wykonać wykres pudełkowy (rys. 2.3) szerokości działki kielicha (`sepal width`) względem nazw gatunków (`species`) (szerzej na temat tworzenia wykresów zob. podrozdział 2.2.10).

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
x <- iris$Species
y <- iris$Sepal.Width
```

```
plot(x, y)
```



Rysunek 2.3. Działka kielicha względem nazw gatunków – przypadek 1

Można zauważyć, że zmienna `iris$Species` jest czynnikiem.

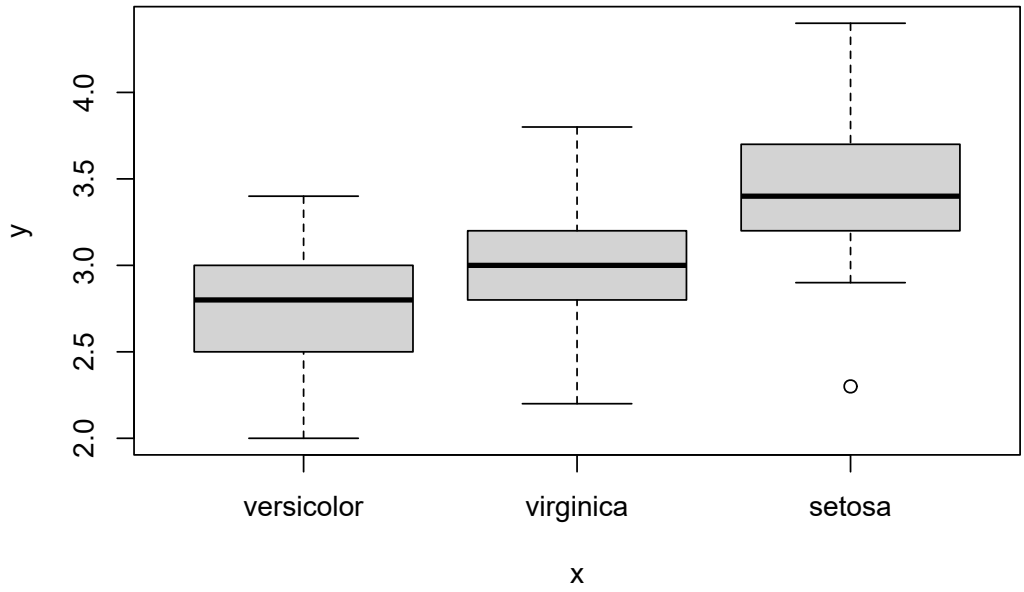
```
str(iris$Species)
```

```
##  Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1
 ↪  ...
```

Wobec powyższego, łatwo można ją posortować według wartości innej zmiennej za pomocą funkcji `forcats::fct_reorder` (rys. 2.4).

```
z <- fct_reorder(x, y)
```

```
plot(z, y)
```



Rysunek 2.4. Działka kielicha względem nazw gatunków – przypadek 2

Łatwo zauważyć, że kategorie osi X zostały posortowane rosnąco według wartości zmiennej `iris$Sepal.Width`. Podany przykład można również przedstawić z wykorzystaniem techniki przetwarzania potokowego (szerzej zob. podrozdział 2.2.1 oraz biblioteka `ggplot2` (szerzej zob. Wickham i in., 2024a i podrozdział 2.2.10).

W kolejnym podrozdziale zostaną przedstawione *macierze*, będące naturalnym rozszerzeniem pojęcia wektora dla dwóch wymiarów. Omówione zostaną sposoby ich tworzenia, własności oraz podstawowe operacje na macierzach, które odgrywają istotną rolę w analizie danych.

2.2.6. Macierze

Macierz jest szczególnym przypadkiem dwuwymiarowej tablicy, która zawiera dane wyłącznie jednego typu (Abedin, 2017). Macierze tworzy się za pomocą funkcji `matrix`.

```
m <- matrix(1:10, nrow = 2, ncol = 5)
m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

Powyższa macierz powstała przez przyporządkowanie liczb od 1 do 10 poszczególnym kolumnom. Można również utworzyć macierz, przypisując wartości wierszami.

```
m <- matrix(1:10, nrow = 2, ncol = 5, byrow = TRUE)
m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
```

Wymiary macierzy można sprawdzić za pomocą funkcji `nrow` i `ncol` oraz `dim`.

```
nrow(m)
```

```
## [1] 2
```

```
ncol(m)
```

```
## [1] 5
```

```
dim(m)
```

```
## [1] 2 5
```

```
dim(m)[1]
```

```
## [1] 2
```

```
dim(m)[2]
```

```
## [1] 5
```

Można również nadać nazwy wierszom i kolumnom macierzy (należy zwrócić uwagę na różnicę pomiędzy wektorami `letters` i `LETTERS`) za pomocą funkcji `rownames` i `colnames`.

```
rownames(m) <- c(letters[1:nrow(m)])
rownames(m)
```

```
## [1] "a" "b"
```

```
colnames(m) <- c(LETTERS[1:ncol(m)])
colnames(m)
```

```
## [1] "A" "B" "C" "D" "E"
```

```
m
```

```
##   A B C D  E
## a 1 2 3 4  5
## b 6 7 8 9 10
```

Można również tworzyć bardziej złożone nazwy wierszy i kolumn.

```
rownames(m) <- paste0("row_", 1:nrow(m))
colnames(m) <- paste0("col_", 1:ncol(m))
m
```

```
##       col_1 col_2 col_3 col_4 col_5
## row_1     1     2     3     4     5
## row_2     6     7     8     9    10
```

Macierz można utworzyć także poprzez łączenie wektorów za pomocą funkcji `rbind` (wierszami) lub `cbind` (kolumnami), które stanowią uogólnienie funkcji `c` (Wickham, 2019).

```
v.1 <- c(1:5)
v.2 <- c(5:1)
m.1 <- rbind(v.1, v.2)
m.2 <- cbind(v.1, v.2)
m.1
```

```
##     [,1] [,2] [,3] [,4] [,5]
## v.1    1    2    3    4    5
## v.2    5    4    3    2    1
```

```
m.2
```

```
##      v.1 v.2
## [1,]   1   5
## [2,]   2   4
## [3,]   3   3
## [4,]   4   2
## [5,]   5   1
```

Do współrzędnych macierzy należy odwoływać się, podając numer wiersza `i` oraz kolumny `j` w nawiasach kwadratowych, tj. `[i,j]`. Można również odwoływać się do poszczególnych wierszy lub kolumn.

```
m[1, 1]
```

```
## [1] 1
```

```
m[1, ]
```

```
## col_1 col_2 col_3 col_4 col_5
##     1     2     3     4     5
```

```
m[, 1]
```

```
## row_1 row_2
##     1     6
```

Do sprawdzenia czy obiekt jest macierzą służy funkcja `is.matrix`. Wektor można przekształcić w macierz za pomocą funkcji `as.matrix`.

```
is.matrix(m)
```

```
## [1] TRUE
```

```
as.matrix(c(1:10))
```

```
##       [,1]
##  [1,]    1
##  [2,]    2
##  [3,]    3
```

```
##  [4,]    4
##  [5,]    5
##  [6,]    6
##  [7,]    7
##  [8,]    8
##  [9,]    9
## [10,]   10
```

Macierze powszechnie stosuje się w statystyce i ekonometrii. W programie R występują operatory i funkcje stosowane w rachunku macierzowym:

- `+` oraz `-` dodawanie i odejmowanie,
- `t` transponowanie,
- %*% mnożenie,
- `solve` odwracanie,
- `diag` pobieranie w postaci wektora elementów głównej przekątnej,
- `det` obliczanie wyznacznika.

Niech będą dane dwie macierze kwadratowe A i B:

$$A = \begin{bmatrix} 1 & 2 & 4 \\ 1 & 5 & 5 \\ 2 & 3 & 2 \end{bmatrix} \quad \text{oraz} \quad B = \begin{bmatrix} 2 & 3 & 6 \\ 1 & 4 & 3 \\ 2 & 1 & 1 \end{bmatrix} \tag{2.1}$$

Ich zapis w programie R jest następujący:

```
A <- matrix(
    c(1, 2, 4, 1, 5, 5, 2, 3, 2),
    nrow = 3,
    ncol = 3,
    byrow = TRUE
)
A
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    4
## [2,]    1    5    5
## [3,]    2    3    2
```

```
B <- matrix(
    c(2, 3, 6, 1, 4, 3, 2, 1, 1),
    nrow = 3,
    ncol = 3,
    byrow = TRUE
)
B
```

```
##      [,1] [,2] [,3]
## [1,]    2    3    6
## [2,]    1    4    3
## [3,]    2    1    1
```

Dla podanych macierzy można wykonać następujące obliczenia:

- dodawanie

```
A+B
```

```
##      [,1] [,2] [,3]
## [1,]    3    5   10
## [2,]    2    9    8
## [3,]    4    4    3
```

- odejmowanie

```
A-B
```

```
##      [,1] [,2] [,3]
## [1,]   -1   -1   -2
## [2,]    0    1    2
## [3,]    0    2    1
```

- transponowanie

```
t(A)
```

```
##      [,1] [,2] [,3]
## [1,]    1    1    2
## [2,]    2    5    3
## [3,]    4    5    2
```

```
t(B)
```

```
##      [,1] [,2] [,3]
## [1,]    2    1    2
## [2,]    3    4    1
## [3,]    6    3    1
```

- mnożenie

```
A %*% B
```

```
##      [,1] [,2] [,3]
## [1,]   12   15   16
## [2,]   17   28   26
## [3,]   11   20   23
```

- odwracanie

```
round(solve(A), digits = 4)
```

```
##         [,1]    [,2]    [,3]
## [1,]  0.2941 -0.4706  0.5882
## [2,] -0.4706  0.3529  0.0588
## [3,]  0.4118 -0.0588 -0.1765
```

```
solve(B)
```

```
##       [,1]  [,2] [,3]
## [1,] -0.04 -0.12  0.6
## [2,] -0.20  0.40  0.0
## [3,]  0.28 -0.16 -0.2
```

```
round(A %*% solve(A), digits = 4)
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

```
B %*% solve(B)
```

```
##              [,1]         [,2] [,3]
## [1,] 1.000000e+00  0.00000e+00    0
## [2,] 0.000000e+00  1.00000e+00    0
## [3,] 3.330669e-16 -1.94289e-16    1
```

- tworzenie wektora z elementów głównej przekątnej

```
diag(A)
```

```
## [1] 1 5 2
```

```
diag(B)
```

```
## [1] 2 4 1
```

- obliczanie wyznacznika macierzy

```
det(A)
```

```
## [1] -17
```

```
det(B)
```

```
## [1] -25
```

Dzięki możliwości *wektoryzacji* argumentów funkcji w programie R wykonanie kodu jest bardzo szybkie.

W odniesieniu do macierzy można podać poniższy przykład obliczania średnich dla wierszy i kolumn. Replikacja przykładu jest możliwa dzięki wcześniejszemu ustawieniu *ziarna* generatora liczb pseudolosowych o rozkładzie normalnym, przy użyciu wcześniej utworzonej funkcji `set_seed`.

```
set_seed()
n.row <- n.col <- 10000
system.time(mat <- matrix(
    rnorm(n.row * n.col),
    n.row,
    n.col
))
```

```
## użytkownik     system   upłynęło
##       8.08       0.56       8.94
```

```
system.time(r.mean <- rowMeans(mat))
```

```
## użytkownik     system   upłynęło
##       0.20       0.00       0.22
```

```
system.time(c.mean <- colMeans(mat))
```

```
## użytkownik     system   upłynęło
##       0.11       0.00       0.11
```

W powyższym przykładzie utworzono dużą macierz o wymiarach $10^4 \times 10^4$, dla której obliczono wektory średnich w wierszach i kolumnach. Dzięki funkcji `system.time` można poznać dokładny czas wykonywania obliczeń (w sekundach).

Na zakończenie warto podkreślić, że macierze stanowią podstawową strukturę umożliwiającą wykonywanie wielu operacji matematycznych w języku R. Ich ograniczeniem jest jednak wymóg jednorodności typu danych, co sprawia, że nie zawsze są wystarczające do reprezentowania rzeczywistych zbiorów danych ekonomicznych.

W kolejnym podrozdziale zostanie omówiona *ramka danych* – uniwersalna struktura danych w języku R, pozwalająca na jednoczesne przechowywanie informacji różnego typu w poszczególnych kolumnach.

2.2.7. Ramki danych

Ramka danych (`data frame`) jest obiektem zorganizowanym na zasadzie dwuwymiarowej macierzy danych alfanumerycznych[5]. W ramce danych występują wiersze i kolumny. Zarówno wiersze, jak i kolumny mają nazwy, które można zmieniać. Ramkę danych tworzy się za pomocą funkcji `data.frame`. Jest to funkcja, której parametry definiują strukturę ramki danych. Niech ramka danych `df` zawiera pięć wierszy i trzy kolumny (5×3).

```
df <- data.frame(
    a = 1:5,
    b = 1,
    c = letters[1:5],
    stringsAsFactors = FALSE
)
df
```

```
##   a b c
## 1 1 1 a
## 2 2 1 b
## 3 3 1 c
## 4 4 1 d
## 5 5 1 e
```

[5] Dane alfanumeryczne to dane, które mogą mieć zarówno charakter liczbowy (numeryczny), jak i tekstowy (literowy). W kontekście ramki danych w R oznacza to, że każda kolumna może przechowywać inny typ danych, co pozwala opisywać rzeczywiste obiekty i zjawiska, w których zmienne mają różną naturę – liczbową, tekstową lub logiczną.

Ramka df zawiera pięć wierszy numerowanych od 1 do 5 i trzy kolumny – a, b, c. Łatwo zauważyć, że w ramce df połączono dane różnego typu – numeryczne i znakowe. Aby ciągi znaków miały postać znakową (string) trzeba do funkcji przekazać parametr stringsAsFactors = FALSE. Strukturę ramki danych można zbadać za pomocą funkcji str.

```
str(df)
```

```
## 'data.frame':    5 obs. of  3 variables:
##  $ a: int  1 2 3 4 5
##  $ b: num  1 1 1 1 1
##  $ c: chr  "a" "b" "c" "d" ...
```

Ramka danych jest wygodną strukturą danych, na której można przeprowadzać wiele operacji. Nazwy wierszy pobiera się za pomocą funkcji rownames.

```
rownames(df)
```

```
## [1] "1" "2" "3" "4" "5"
```

Nazwy wierszy tworzą wektor znakowy, któremu można przypisać inne wartości.

```
rownames(df) <- paste0("row_", 1:nrow(df))
```

Nazwy kolumn pobiera się odpowiednio za pomocą funkcji colnames.

```
colnames(df)
```

```
## [1] "a" "b" "c"
```

Podobnie jak w przypadku nazw wierszy, kolumnom również można przypisać nowe wartości.

```
colnames(df) <- paste0("col_", 1:ncol(df))
df
```

```
##       col_1 col_2 col_3
## row_1     1     1     a
## row_2     2     1     b
## row_3     3     1     c
## row_4     4     1     d
## row_5     5     1     e
```

Jeśli zachodzi potrzeba wyświetlenia części ramki danych lub utworzenia nowej zmiennej na podstawie ramki danych, stosuje się następującą składnię:

```
df[, 1]
```

```
## [1] 1 2 3 4 5
```

```
df[, "col_1"]
```

```
## [1] 1 2 3 4 5
```

```
df$col_1
```

```
## [1] 1 2 3 4 5
```

```
col_1 <- df$col_1
```

Oznacza to, że do elementów ramki danych należy odwoływać się analogicznie jak do współrzędnych macierzy. Powyższe polecenia powodują wyświetlenie wszystkich wierszy i pierwszej kolumny ramki `df`. Wyświetlenie dwóch lub większej liczby kolumn wymaga podania wektora indeksów lub nazw kolumn.

```
df[, 1:2]
```

```
##       col_1 col_2
## row_1     1     1
## row_2     2     1
## row_3     3     1
## row_4     4     1
## row_5     5     1
```

```
df[, c(1, 2)]
```

```
##       col_1 col_2
## row_1     1     1
## row_2     2     1
## row_3     3     1
## row_4     4     1
## row_5     5     1
```

```
df[, c("col_1", "col_2")]
```

```
##       col_1 col_2
## row_1     1     1
## row_2     2     1
## row_3     3     1
## row_4     4     1
## row_5     5     1
```

Wyświetlenie ustalonych wierszy i kolumn jest również możliwe.

```
df[1, ]
```

```
##       col_1 col_2 col_3
## row_1     1     1     a
```

```
df[1:3, ]
```

```
##       col_1 col_2 col_3
## row_1     1     1     a
## row_2     2     1     b
## row_3     3     1     c
```

```
df[c(1, 4), ]
```

```
##       col_1 col_2 col_3
## row_1     1     1     a
## row_4     4     1     d
```

```
df[c(1, 4), c(1, 2)]
```

```
##       col_1 col_2
## row_1     1     1
## row_4     4     1
```

Jako pierwszą współrzędną należy podać indeksy wierszy, a jako drugą – indeksy kolumn. Analogicznie możliwe jest również pokazanie wybranej wartości z ramki danych, podając współrzędne wiersza i kolumny.

```
df[3, 3]
```

```
## [1] "c"
```

Można również usunąć z widoku wybrany wiersz lub kolumnę, podając indeks ze znakiem ujemnym.

```
df[-1, ]
```

```
##       col_1 col_2 col_3
## row_2     2     1     b
## row_3     3     1     c
## row_4     4     1     d
## row_5     5     1     e
```

```
df[, -1]
```

```
##       col_2 col_3
## row_1     1     a
## row_2     1     b
## row_3     1     c
## row_4     1     d
## row_5     1     e
```

```
df[, c(-1, -3)]
```

```
## [1] 1 1 1 1 1
```

```
df[, -c(1, 3)]
```

```
## [1] 1 1 1 1 1
```

```
df[-1, -1]
```

```
##       col_2 col_3
## row_2     1     b
## row_3     1     c
## row_4     1     d
## row_5     1     e
```

Ramki danych można łączyć zarówno wierszami, jak i kolumnami. Ramki łączy się wierszami za pomocą funkcji `rbind`, przy czym nazwy kolumn w obu ramkach muszą być identyczne. Niech dane będą dwie ramki danych `a` i `b`.

```
a <- data.frame(
    a = 1:3,
    b = 1
)
a
```

```
##   a b
## 1 1 1
## 2 2 1
## 3 3 1
```

```
b <- data.frame(
    a = 3,
    b = letters[1:3],
    stringsAsFactors = FALSE
)
b
```

```
##   a b
## 1 3 a
## 2 3 b
## 3 3 c
```

```
c <- rbind(a, b)
c
```

```
##   a b
## 1 1 1
## 2 2 1
## 3 3 1
## 4 3 a
## 5 3 b
## 6 3 c
```

W powyższy sposób ramki `a` i `b` połączono wierszami, a wynik zapisano w zmiennej `c`. Ramki można również łączyć kolumnami za pomocą funkcji `cbind`, przy czym nazwy kolumn w łączonych ramkach nie mogą się powtarzać.

```
a <- data.frame(
    a = 1:3,
    b = 1
)
a
```

```
##   a b
## 1 1 1
## 2 2 1
## 3 3 1
```

```
b <- data.frame(
    c = 3,
    d = letters[1:3],
    stringsAsFactors = FALSE
)
b
```

```
##   c d
## 1 3 a
## 2 3 b
## 3 3 c
```

```
c <- cbind(a, b)
c
```

```
##   a b c d
## 1 1 1 3 a
## 2 2 1 3 b
## 3 3 1 3 c
```

Ważnym elementem pracy z ramką danych jest możliwość wyboru wierszy i kolumn na podstawie warunków logicznych. Weźmy pod uwagę ramkę `c` powstałą z połączenia kolumnowego ramek `a` i `b`. Dla takiej ramki danych można np. pokazać tylko te wiersze, które spełniają warunek logiczny. W tym celu można skorzystać z funkcji `which`, która zwraca indeksy na podstawie warunku logicznego.

```
c[which(c$a == 3), ]
```

```
##   a b c d
## 3 3 1 3 c
```

```
c[which(c$a > 1), ]
```

```
##   a b c d
## 2 2 1 3 b
## 3 3 1 3 c
```

Można również zadawać warunki złożone wynikające np. z *koniunkcji* (&) lub *alternatywy* (|).

```
c[which(c$a == 3 & c$d == "c"), ]
```

```
##   a b c d
## 3 3 1 3 c
```

```
c[which(c$a == 2 | c$d == "c"), ]
```

```
##   a b c d
## 2 2 1 3 b
## 3 3 1 3 c
```

Należy zwrócić uwagę na operator porównania (==). Można również sformułować podobne warunki dla kolumn ramki danych korzystając np. z operatora `%in%`, za pomocą którego sprawdza się, czy element należy do wektora.

```
c[, which(colnames(c) %in% c("a", "d"))]
```

```
##   a d
## 1 1 a
## 2 2 b
## 3 3 c
```

Warunki dla wierszy i kolumn można łączyć.

```
c[
    which(c$a >= 2),
    which(colnames(c) %in% c("a", "c"))
]
```

```
##   a c
## 2 2 3
## 3 3 3
```

Warunki logiczne dla zakresów ramki danych można nakładać również za pomocą funkcji `subset`.

```
subset(c, a > 2)
```

```
##   a b c d
## 3 3 1 3 c
```

Funkcja `subset` zawiera argument `select` wskazujący kolumny, które mają zostać zwrócone w postaci wyniku.

```
subset(
    c,
    d == "a" | d == "b",
    select = c:d
)
```

```
##   c d
## 1 3 a
## 2 3 b
```

Istnieje kilka środowisk dla gromadzenia danych w tablicach w programie R. Funkcja `data.frame` jest elementem biblioteki `base` (R Core Team, 2025). Pozostałe zrealizowano za pomocą odpowiednich bibliotek zewnętrznych:

- `tibble` (Müller i Wickham, 2023), która jest elementem *ekosystemu* `tidyverse` (https://www.tidyverse.org/) (Wickham i Grolemund, 2016),
- `data.table` (Barrett i in., 2024).

Poniżej podano elementarne zagadnienia związane z bibliotekami `tibble` oraz `data.table`. Ramkę `tibble` tworzy się za pomocą funkcji `tibble`. Funkcja ta nie zmienia typów argumentów, nie modyfikuje nazw zmiennych oraz nie tworzy nazw wierszy. Na początku należy wczytać odpowiednią bibliotekę.

```
library(tibble)
```

Następnie można utworzyć ramkę `tb`.

```
tb <- tibble(
  a = 1:3,
  b = 1,
  c = letters[1:3]
)
tb
```

```
## # A tibble: 3 x 3
##       a     b c
##   <int> <dbl> <chr>
## 1     1     1 a
## 2     2     1 b
## 3     3     1 c
```

Dla każdej kolumny ramki `tb` pokazany jest również jej typ. Ramkę `tibble` można utworzyć z ramki `c` o strukturze `data.frame` za pomocą funkcji `as_tibble`. Jedną z kluczowych różnic pomiędzy ramkami `tibble` i klasycznymi ramkami `data.frame` jest sposób selekcji kolumn – w przypadku `tibble` wybór pojedynczej kolumny (np. `tibble$kolumna`) zwraca obiekt typu `tibble` o jednej kolumnie, natomiast w klasycznej ramce `data.frame` domyślnie następuje uproszczenie wyniku do wektora.

```
c.tibble <- as_tibble(c)
c.tibble
```

```
## # A tibble: 3 x 4
##       a     b     c d
##   <int> <dbl> <dbl> <chr>
## 1     1     1     3 a
## 2     2     1     3 b
## 3     3     1     3 c
```

Dla ilustracji użycia biblioteki `data.table` wykorzystano zbiór danych `mtcars`, zawierający dane o cechach 32 samochodów, opublikowany w czasopiśmie *1974 Motor Trend US*[6]. Na początku należy wczytać odpowiednią bibliotekę.

```
library(data.table)
```

Ramka `dt` zawiera wszystkie wiersze i pierwszych 7 kolumn zbioru `mtcars`. Ponadto, w pierwszej kolumnie zamieszczono nazwy modeli samochodów.

```
dt <- mtcars
dt <- cbind(
  name = rownames(dt),
  dt
)
dt <- dt[, 1:7]
rownames(dt) <- c(1:nrow(dt))
dt <- setDT(
  dt,
```

[6] Por. https://www.rdocumentation.org/packages/datasets/versions/3.6.2/topics/mtcars.

```
  keep.rownames = FALSE
)
dt
```

```
##                   name   mpg   cyl  disp    hp  drat    wt
##                 <char> <num> <num> <num> <num> <num> <num>
##  1:           Mazda RX4  21.0     6 160.0   110  3.90 2.620
##  2:       Mazda RX4 Wag  21.0     6 160.0   110  3.90 2.875
##  3:          Datsun 710  22.8     4 108.0    93  3.85 2.320
##  4:      Hornet 4 Drive  21.4     6 258.0   110  3.08 3.215
##  5:   Hornet Sportabout  18.7     8 360.0   175  3.15 3.440
##  6:             Valiant  18.1     6 225.0   105  2.76 3.460
##  7:          Duster 360  14.3     8 360.0   245  3.21 3.570
##  8:           Merc 240D  24.4     4 146.7    62  3.69 3.190
##  9:            Merc 230  22.8     4 140.8    95  3.92 3.150
## 10:            Merc 280  19.2     6 167.6   123  3.92 3.440
## 11:           Merc 280C  17.8     6 167.6   123  3.92 3.440
## 12:          Merc 450SE  16.4     8 275.8   180  3.07 4.070
## 13:          Merc 450SL  17.3     8 275.8   180  3.07 3.730
## 14:         Merc 450SLC  15.2     8 275.8   180  3.07 3.780
## 15:  Cadillac Fleetwood  10.4     8 472.0   205  2.93 5.250
## 16: Lincoln Continental  10.4     8 460.0   215  3.00 5.424
## 17:   Chrysler Imperial  14.7     8 440.0   230  3.23 5.345
## 18:            Fiat 128  32.4     4  78.7    66  4.08 2.200
## 19:         Honda Civic  30.4     4  75.7    52  4.93 1.615
## 20:      Toyota Corolla  33.9     4  71.1    65  4.22 1.835
## 21:       Toyota Corona  21.5     4 120.1    97  3.70 2.465
## 22:    Dodge Challenger  15.5     8 318.0   150  2.76 3.520
## 23:         AMC Javelin  15.2     8 304.0   150  3.15 3.435
## 24:          Camaro Z28  13.3     8 350.0   245  3.73 3.840
## 25:    Pontiac Firebird  19.2     8 400.0   175  3.08 3.845
## 26:           Fiat X1-9  27.3     4  79.0    66  4.08 1.935
## 27:       Porsche 914-2  26.0     4 120.3    91  4.43 2.140
## 28:        Lotus Europa  30.4     4  95.1   113  3.77 1.513
## 29:      Ford Pantera L  15.8     8 351.0   264  4.22 3.170
## 30:        Ferrari Dino  19.7     6 145.0   175  3.62 2.770
## 31:       Maserati Bora  15.0     8 301.0   335  3.54 3.570
## 32:          Volvo 142E  21.4     4 121.0   109  4.11 2.780
##                   name   mpg   cyl  disp    hp  drat    wt
```

Ramka `dt` została przekształcona w obiekt `data.table` za pomocą funkcji `setDT`. Filtrowanie ramki może odbywać się za pomocą następujących poleceń:

```
dt[hp == 110]
```

```
##             name   mpg   cyl  disp    hp  drat    wt
```

```
##            <char> <num> <num> <num> <num> <num> <num>
## 1:      Mazda RX4  21.0     6   160   110  3.90 2.620
## 2:  Mazda RX4 Wag  21.0     6   160   110  3.90 2.875
## 3: Hornet 4 Drive  21.4     6   258   110  3.08 3.215
```

```
dt[hp == 110 & grepl("Mazda", name, fixed = TRUE)]
```

```
##            name   mpg   cyl  disp    hp  drat    wt
##          <char> <num> <num> <num> <num> <num> <num>
## 1:     Mazda RX4    21     6   160   110   3.9 2.620
## 2: Mazda RX4 Wag    21     6   160   110   3.9 2.875
```

W pierwszym przykładzie filtrowania pokazano wszystkie wiersze, które spełniają warunek dla kolumny hp (*horse power*). W drugim przykładzie zastosowano koniunkcję dwóch warunków logicznych dla kolumn hp i name, w którym wykorzystano funkcję grepl do wyszukania podciągu Mazda. Nazwy kolumn przekazuje się tak samo jak nazwy zmiennych, bez konieczności korzystania z operatora $ (dt$hp).

Na koniec należy zwrócić uwagę, że występują różnice pomiędzy macierzą a ramką danych (Burns, 2012).

```
m <- cbind(1:3, 4:2)
d <- data.frame(a = 1:3, b = 4:2)
```

Można następnie zauważyć, że zastosowanie prostych funkcji statystycznych daje następujące wyniki:

```
mean(m)
```

```
## [1] 2.5
```

```
median(m)
```

```
## [1] 2.5
```

```
sum(m)
```

```
## [1] 15
```

Ponieważ funkcji mean i median nie można zastosować bezpośrednio do ramki danych, w celu wykonania kodu zastosowano obsługę błędów za pomocą funkcji tryCatch.

```
tryCatch(
  { mean(d) },
  error = function(x) {
    message(conditionMessage(x))
  },
  warning = function(x) {
    message(conditionMessage(x))
  }
)
```

```
## argument nie jest wartością liczbową ani logiczną: zwracanie
 ↪  wartości NA
```

```
tryCatch(
  { median(d) },
  error = function(x) {
    message(conditionMessage(x))
  },
  warning = function(x) {
    message(conditionMessage(x))
  }
)
```

```
## potrzeba danych liczbowych
```

```
sum(d)
```

```
## [1] 15
```

Poprawne zastosowanie funkcji `mean` i `median` dla całej ramki danych polega na uprzednim przekształceniu jej do macierzy:

```
d <- as.matrix(d)
mean(d)
```

```
## [1] 2.5
```

```
median(d)
```

```
## [1] 2.5
```

W podrozdziale 2.2.1 podano zasady przetwarzania potokowego w R. Metoda przetwarzania potokowego ma wiele zastosowań i może być wykorzystywana do porządkowania oraz optymalizacji kodu programu. Poniżej przedstawiono jej użycie w odniesieniu do ramek danych. Na początku należy wczytać wcześniej omówioną bibliotekę.

```
library(magrittr)
```

W celu replikacji podanych przykładów, uwzględniających liczby pseudolosowe, należy ustawić *ziarno* generatora na podstawie wcześniej zdefiniowanej funkcji.

```
set_seed()
```

Poniższy kod tworzy dwie ramki danych x i y, łączy je w jedną ramkę z, a następnie wybiera z niej poszczególne kolumny lub fragmenty danych.

```
set_seed()
x <- data.frame(a = 1:10, b = rnorm(10))
x
```

```
##     a          b
## 1   1 -1.2070657
## 2   2  0.2774292
## 3   3  1.0844412
## 4   4 -2.3456977
## 5   5  0.4291247
## 6   6  0.5060559
## 7   7 -0.5747400
## 8   8 -0.5466319
## 9   9 -0.5644520
## 10 10 -0.8900378
```

```
y <- data.frame(c = letters[1:10], stringsAsFactors = FALSE)
y
```

```
##    c
## 1  a
## 2  b
## 3  c
## 4  d
## 5  e
## 6  f
## 7  g
## 8  h
## 9  i
## 10 j
```

```
z <- cbind(x, y)
z
```

```
##     a          b c
## 1   1 -1.2070657 a
## 2   2  0.2774292 b
## 3   3  1.0844412 c
## 4   4 -2.3456977 d
## 5   5  0.4291247 e
## 6   6  0.5060559 f
## 7   7 -0.5747400 g
## 8   8 -0.5466319 h
## 9   9 -0.5644520 i
## 10 10 -0.8900378 j
```

```
z[, 1]
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
z[, 2][1:5]
```

```
## [1] -1.2070657  0.2774292  1.0844412 -2.3456977  0.4291247
```

```
z[, 3]
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

Ten sam wynik można otrzymać, stosując metodę potokową lub wykorzystując operator . (kropki).

```
z.1 <- x %>%
  cbind(y)
z.1
```

```
##     a          b c
## 1   1 -1.2070657 a
## 2   2  0.2774292 b
## 3   3  1.0844412 c
## 4   4 -2.3456977 d
## 5   5  0.4291247 e
## 6   6  0.5060559 f
## 7   7 -0.5747400 g
## 8   8 -0.5466319 h
## 9   9 -0.5644520 i
## 10 10 -0.8900378 j
```

```
z.1[, 1]
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
z.1[, 2][1:5]
```

```
## [1] -1.2070657  0.2774292  1.0844412 -2.3456977  0.4291247
```

```
z.1[, 3]
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

```
z.2 <- x %>%
  cbind(., y)
z.2
```

```
##     a          b c
## 1   1 -1.2070657 a
## 2   2  0.2774292 b
## 3   3  1.0844412 c
## 4   4 -2.3456977 d
## 5   5  0.4291247 e
## 6   6  0.5060559 f
## 7   7 -0.5747400 g
## 8   8 -0.5466319 h
## 9   9 -0.5644520 i
## 10 10 -0.8900378 j
```

```
z.2[, 1]
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
z.2[, 2][1:5]
```

```
## [1] -1.2070657  0.2774292  1.0844412 -2.3456977  0.4291247
```

```
z.2[, 3]
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

Dalsze zagnieżdżanie poleceń jest również możliwe. Ilustruje to poniższy przykład.

```
x[9:10, 1] <- NA
x
```

```
##     a          b
## 1   1 -1.2070657
## 2   2  0.2774292
## 3   3  1.0844412
## 4   4 -2.3456977
## 5   5  0.4291247
## 6   6  0.5060559
## 7   7 -0.5747400
## 8   8 -0.5466319
## 9  NA -0.5644520
## 10 NA -0.8900378
```

```
y <- data.frame(a = rep(NA, 5), b = rep(NA, 5))
y
```

```
##    a  b
## 1 NA NA
## 2 NA NA
## 3 NA NA
## 4 NA NA
## 5 NA NA
```

```
z <- x %>% rbind(., y)
z
```

```
##     a          b
## 1   1 -1.2070657
## 2   2  0.2774292
## 3   3  1.0844412
## 4   4 -2.3456977
## 5   5  0.4291247
## 6   6  0.5060559
## 7   7 -0.5747400
## 8   8 -0.5466319
## 9  NA -0.5644520
## 10 NA -0.8900378
## 11 NA         NA
## 12 NA         NA
## 13 NA         NA
## 14 NA         NA
## 15 NA         NA
```

```
h <- x %>%
  rbind(., y) %>%
  na.omit %>%
  round(digits = 4)
h
```

```
##   a       b
## 1 1 -1.2071
## 2 2  0.2774
## 3 3  1.0844
## 4 4 -2.3457
## 5 5  0.4291
## 6 6  0.5061
## 7 7 -0.5747
## 8 8 -0.5466
```

Jak widać, ramka danych z zawiera wartości NA, ponieważ powstała z połączenia ramek x i y. Ramka h nie zawiera wartości NA, gdyż zastosowano funkcję na.omit, która usuwa wszystkie wiersze zawierające brakujące wartości. Ponadto wartości w ramce h zostały zaokrąglone do czterech miejsc po przecinku.

Jak wspomniano wcześniej, możliwe jest także utworzenie funkcji z wykorzystaniem techniki potokowej.

```
set_seed()
x <- rnorm(10)
g <- . %>% mean
g
```

```
## Functional sequence with the following components:
##
##  1. mean(.)
##
## Use 'functions' to extract the individual functions.
```

```
g(1:10)
```

```
## [1] 5.5
```

```
g(x)
```

```
## [1] -0.3831574
```

```
mean(x)
```

```
## [1] -0.3831574
```

Poniżej utworzono funkcję myround.

```
set_seed()
myround <- . %>% round(digits = 4)
myround
```

```
## Functional sequence with the following components:
##
##  1. round(., digits = 4)
##
## Use 'functions' to extract the individual functions.
```

```
myround(rnorm(1))
```

```
## [1] -1.2071
```

Funkcję myround można teraz wykorzystać do zaokrąglenia wartości ramki h.

```
h <- h %>% myround()
h
```

```
##   a       b
## 1 1 -1.2071
## 2 2  0.2774
## 3 3  1.0844
## 4 4 -2.3457
## 5 5  0.4291
## 6 6  0.5061
## 7 7 -0.5747
## 8 8 -0.5466
```

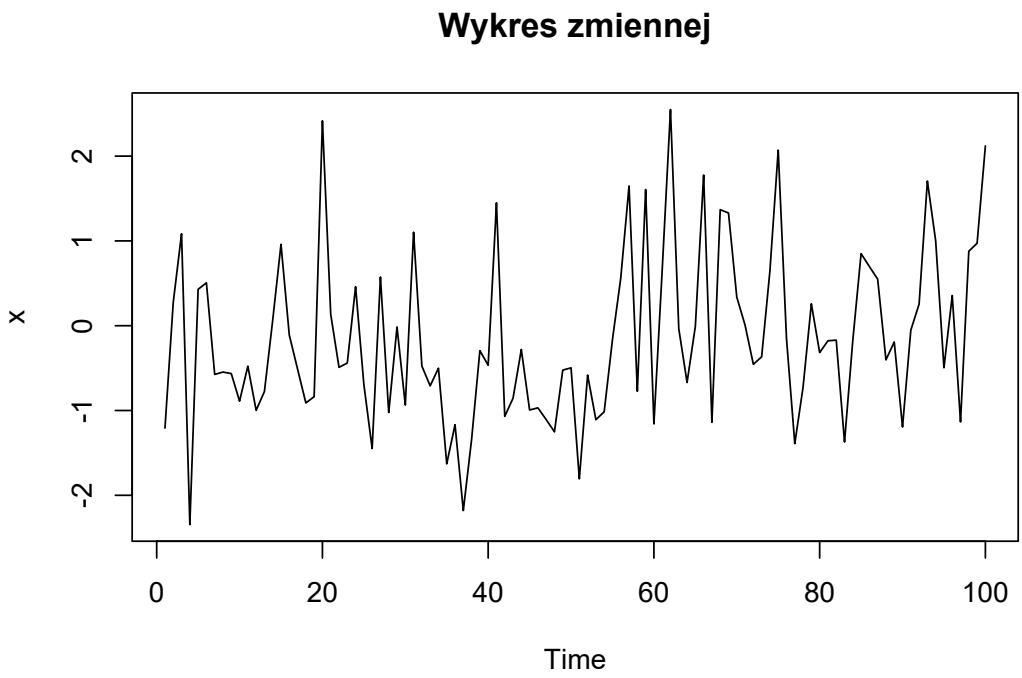
Tę samą czynność można wykonać za pomocą funkcji myround zapisanej w sposób klasyczny.

```
myround <- function(x) {
  round(x, digits = 4)
}
h <- h %>% myround()
h
```

```
##   a       b
## 1 1 -1.2071
## 2 2  0.2774
## 3 3  1.0844
## 4 4 -2.3457
## 5 5  0.4291
## 6 6  0.5061
## 7 7 -0.5747
## 8 8 -0.5466
```

Występuje również operator potokowy, który daje możliwość odwołania się do *argumentu* wyrażenia z lewej strony. Stosuje się wówczas operator %$% (rys. 2.5).

```
set_seed()
data.frame(x = rnorm(100)) %$% ts.plot(
  x,
  main = "Wykres zmiennej"
)
```



Rysunek 2.5. Zmienna o rozkładzie normalnym standaryzowanym – `ts.plot`

Jednym z praktycznych zastosowań ramek danych i przetwarzania potokowego jest transformacja danych statystycznych. Jeśli dane są wyrażone w postaci poziomów

zmiennych, można przeliczyć je do postaci logarytmów, przyrostów lub temp wzrostu. Ilustruje to poniższy przykład.

W ramce danych zostaną zapisane dwa szeregi czasowe dziennych kursów walutowych EUR/PLN i USD/PLN w okresie 1-29 października 2021 r. Kursy pochodzą z serwisu https://stooq.pl/. Zostały zapisane w zbiorze `Excel` `kursy_10_2021.xlsx`. Na początku należy wczytać odpowiednie biblioteki.

```
library(dplyr)
library(magrittr)
library(readxl)
```

Do wczytania danych z pliku `Excel` zostanie wykorzystana biblioteka `readxl` (Wickham i Bryan, 2023). Założenia biblioteki `magrittr` zostały przedstawione wcześniej. Bibliotekę `dplyr` (Wickham i in., 2023b) stosuje się w przetwarzaniu potokowym do przekształcania i filtrowania ramek danych. Wczytanie zbioru danych przeprowadza się na podstawie poniższego kodu.

```
path <- "F:\\docs\\R\\b2\\ch2\\kursy_10_2021.xlsx"
sheet <- "kursy"
df <- read_excel(path, sheet = sheet) %>%
  set_colnames(c("data", "eurpln", "usdpln")) %>%
  mutate_if(is.numeric, myround)
df
```

```
## # A tibble: 21 x 3
##    data                eurpln usdpln
##    <dttm>               <dbl>  <dbl>
##  1 2021-10-01 00:00:00   4.58   3.95
##  2 2021-10-04 00:00:00   4.60   3.96
##  3 2021-10-05 00:00:00   4.60   3.97
##  4 2021-10-06 00:00:00   4.55   3.94
##  5 2021-10-07 00:00:00   4.59   3.97
##  6 2021-10-08 00:00:00   4.61   3.98
##  7 2021-10-11 00:00:00   4.59   3.97
##  8 2021-10-12 00:00:00   4.58   3.98
##  9 2021-10-13 00:00:00   4.58   3.95
## 10 2021-10-14 00:00:00   4.57   3.94
## # i 11 more rows
```

Wartości zmiennych EUR/PLN i USD/PLN zapisano w ramce danych `df`. Warto zauważyć, że wartości numeryczne zostały zaokrąglone do czterech miejsc po przecinku za pomocą wcześniej utworzonej funkcji `myround` i funkcji `mutate_if` z biblioteki `dplyr`. Do przeliczenia wartości kursów walutowych na logarytmy naturalne można wykorzystać funkcję `mutate_at` również z biblioteki `dplyr`.

```
df.1 <- df %>%
  mutate_at(
    .vars = vars(eurpln, usdpln),
    .funs = funs(log)
  ) %>%
  mutate_if(is.numeric, myround)
df.1
```

```
## # A tibble: 21 x 3
##    data                eurpln usdpln
##    <dttm>               <dbl>  <dbl>
##  1 2021-10-01 00:00:00   1.52   1.37
##  2 2021-10-04 00:00:00   1.53   1.38
##  3 2021-10-05 00:00:00   1.53   1.38
##  4 2021-10-06 00:00:00   1.51   1.37
##  5 2021-10-07 00:00:00   1.52   1.38
##  6 2021-10-08 00:00:00   1.53   1.38
##  7 2021-10-11 00:00:00   1.52   1.38
##  8 2021-10-12 00:00:00   1.52   1.38
##  9 2021-10-13 00:00:00   1.52   1.37
## 10 2021-10-14 00:00:00   1.52   1.37
## # i 11 more rows
```

Argument `vars` zawiera listę nazw kolumn z ramki danych `df`, dla których jest wykonywana transformacja. Argument `funs` zawiera nazwę funkcji stanowiącej transformację. W tym przypadku dla wskazanych kolumn policzono logarytm naturalny. Można stosować funkcje wbudowane i własne. Łatwo sprawdzić poprawność rachunków porównując wyniki z ramek danych `df` i `df.1`.

```
log(df[1:5, 2]) %>% myround
```

```
## # A tibble: 5 x 1
##   eurpln
##    <dbl>
## 1   1.52
## 2   1.53
## 3   1.53
## 4   1.51
## 5   1.52
```

```
log(df[1:5, 3]) %>% myround
```

```
## # A tibble: 5 x 1
##   usdpln
##    <dbl>
```

```
## 1   1.37
## 2   1.38
## 3   1.38
## 4   1.37
## 5   1.38
```

Przeliczenie wartości zmiennych do przyrostów i temp wzrostu można wykonać w sposób podany poniżej.

```
df.2 <- df %>%
mutate_at(
    .vars = vars(eurpln,  usdpln),
    .funs = funs(c(first(NA), diff(.)))
) %>%
  mutate_if(is.numeric, myround)
df.2
```

```
## # A tibble: 21 x 3
##    data                eurpln  usdpln
##    <dttm>               <dbl>   <dbl>
##  1 2021-10-01 00:00:00 NA      NA
##  2 2021-10-04 00:00:00  0.0184  0.0086
##  3 2021-10-05 00:00:00  0.0083  0.0149
##  4 2021-10-06 00:00:00 -0.0559 -0.034
##  5 2021-10-07 00:00:00  0.0386  0.0329
##  6 2021-10-08 00:00:00  0.0184  0.0109
##  7 2021-10-11 00:00:00 -0.0191 -0.0108
##  8 2021-10-12 00:00:00 -0.0025  0.006
##  9 2021-10-13 00:00:00 -0.0076 -0.0297
## 10 2021-10-14 00:00:00 -0.0067 -0.0059
## # i 11 more rows
```

```
df.3 <- df %>%
mutate_at(
    .vars = vars(eurpln, usdpln),
    .funs = funs(100*(log(.)-log(lag(., 1))))
) %>%
  mutate_if(is.numeric, myround)
df.3
```

```
## # A tibble: 21 x 3
##    data                eurpln usdpln
##    <dttm>               <dbl>  <dbl>
##  1 2021-10-01 00:00:00 NA      NA
##  2 2021-10-04 00:00:00  0.401   0.218
##  3 2021-10-05 00:00:00  0.180   0.376
```

```
##  4 2021-10-06 00:00:00 -1.22   -0.860
##  5 2021-10-07 00:00:00  0.845   0.832
##  6 2021-10-08 00:00:00  0.400   0.274
##  7 2021-10-11 00:00:00 -0.416  -0.272
##  8 2021-10-12 00:00:00 -0.0545  0.151
##  9 2021-10-13 00:00:00 -0.166  -0.750
## 10 2021-10-14 00:00:00 -0.146  -0.150
## # i 11 more rows
```

Poprawność rachunków można sprawdzić, porównując wyniki odpowiednio ramek `df` i `df.2`, oraz `df` i `df.3`.

```
diff(as.matrix(df[1:5, 2])) %>% myround
```

```
##       eurpln
## [1,]  0.0184
## [2,]  0.0083
## [3,] -0.0559
## [4,]  0.0386
```

```
diff(as.matrix(df[1:5, 3])) %>% myround
```

```
##       usdpln
## [1,]  0.0086
## [2,]  0.0149
## [3,] -0.0340
## [4,]  0.0329
```

```
100*diff(as.matrix(log(df[1:5, 2]))) %>% myround
```

```
##      eurpln
## [1,]   0.40
## [2,]   0.18
## [3,]  -1.22
## [4,]   0.85
```

```
100*diff(as.matrix(log(df[1:5, 3]))) %>% myround
```

```
##      usdpln
## [1,]   0.22
## [2,]   0.38
## [3,]  -0.86
## [4,]   0.83
```

W przypadku ramek danych niekiedy zachodzi potrzeba odwołania się do argumentu wyrażenia z lewej strony. Wówczas stosuje się operator %$%:

- `data.frame(x = rnorm(10)) %$% hist(x)` oznacza utworzenie wykresu histogramu ciągu 10 wartości `x` zmiennej o rozkładzie normalnym standaryzowanym.

Przetwarzanie potokowe można stosować do wszelkich obiektów, w tym do list, omówionych w podrozdziale 2.2.9. Zdarza się, że w ramce danych występują wartości nienumeryczne, np. ciągi znaków, które należy w określony sposób skwantyfikować w celu wykonania dalszych obliczeń. Niech będzie dana ramka danych `df`, zawierająca informację o sprzedaży samochodów.

```
df <- data.frame(
    marka = "Volkswagen",
    model = c(
        "Tiguan",
        "Arteon",
        "Touareg",
        "Golf",
        "Passat"
    ),
    sprzedawca = c(
        "firma",
        "firma",
        "firma",
        "prywatnie",
        "prywatnie"
    ),
    finansowanie = c(
        "kredyt",
        "leasing",
        "leasing",
        "gotówka",
        "gotówka"
    )
)
df
```

```
##        marka   model sprzedawca finansowanie
## 1 Volkswagen  Tiguan      firma       kredyt
## 2 Volkswagen  Arteon      firma      leasing
## 3 Volkswagen Touareg      firma      leasing
## 4 Volkswagen    Golf  prywatnie      gotówka
## 5 Volkswagen  Passat  prywatnie      gotówka
```

Do przekształcenia tej ramki można zastosować podejście potokowe.

```
df.1 <- df %>%
  mutate_at(
    .vars = vars(sprzedawca),
    .funs = funs(
        case_when(
            . == "firma"     ~ 1,
            . == "prywatnie" ~ 0
        )
    )
  ) %>%
  mutate_at(
    .vars = vars(finansowanie),
    .funs = funs(
        case_when(
            . == "kredyt"  ~ 0,
            . == "leasing" ~ 1,
            . == "gotówka" ~ 2
        )
    )
  )
df.1
```

```
##        marka   model sprzedawca finansowanie
## 1 Volkswagen  Tiguan          1            0
## 2 Volkswagen  Arteon          1            1
## 3 Volkswagen Touareg          1            1
## 4 Volkswagen    Golf          0            2
## 5 Volkswagen  Passat          0            2
```

Jak widać, ciągi znaków w kolumnach `sprzedawca` i `finansowanie` w ramce `df` zostały odpowiednio przekształcone i przypisane do ramki `df.1`. Takie transformacje można również wprowadzać od razu (*w locie*) do ramki `df`, bez konieczności tworzenia dodatkowego obiektu.

W bibliotece `dplyr`, funkcja `if_else` jest często używana do przypisania wartości do kolumny w zależności od spełnienia określonych warunków. Poniżej przedstawiono przykład zastosowania funkcji `if_else` w połączeniu z innymi funkcjami biblioteki `dplyr`.

Niech będzie dany zbiór `df` zawierający informację o studentach, w tym ich oceny z matematyki, fizyki i chemii. Na podstawie tych danych można:

1. Przydzielić kategorię wyniku (np. `wysoki`, `średni`, `niski`) na podstawie średniej ocen.
2. Oznaczyć studentów, którzy osiągnęli niski wynik w którymkolwiek przedmiocie (poniżej 40%).

Przykład zamieszczony poniżej zrealizowano za pomocą biblioteki `dplyr`.

```
library(dplyr)
```

Utworzono ramkę `df`.

```
df <- tibble(
  id = 1:6,
  m  = c(85, 40, 78, 92, 30, 56),
  f  = c(75, 50, 89, 65, 39, 85),
  c  = c(90, 45, 67, 82, 28, 74)
)
```

Następnie utworzono ramkę `df.processed`.

```
df.processed <- df %>%
  rowwise() %>%
  mutate(
    score = mean(c(m, f, c)),
    performance = if_else(
      score >= 80, "wysoki",
      if_else(score >= 60, "średni", "niski")
    ),
    warning = if_else(
      m < 40 | f < 40 | c < 40, TRUE, FALSE
    )
  ) %>%
  ungroup()
print(df.processed)
```

```
## # A tibble: 6 x 7
##      id     m     f     c score performance warning
##   <int> <dbl> <dbl> <dbl> <dbl> <chr>       <lgl>
## 1     1    85    75    90  83.3 wysoki      FALSE
## 2     2    40    50    45  45   niski       FALSE
## 3     3    78    89    67  78   średni      FALSE
## 4     4    92    65    82  79.7 średni      FALSE
## 5     5    30    39    28  32.3 niski       TRUE
## 6     6    56    85    74  71.7 średni      FALSE
```

Przedstawiony kod realizuje następujące zadania:

- `mutate` – tworzy nowe kolumny na podstawie istniejących,
- `rowwise` – zapewnia, że operacje są wykonywane dla każdego wiersza z osobna, co jest istotne przy obliczaniu średniej z kilku kolumn,

- `if_else` – przypisuje kategorie wyników; najpierw sprawdza, czy średnia ocen wynosi co najmniej 80, jeśli tak, przypisuje kategorię `wysoki`; w przeciwnym razie, jeśli średnia wynosi co najmniej 60, przypisuje `średni`, a w pozostałych przypadkach `niski`,
- `warning` – używa `if_else` do oznaczenia studentów, którzy uzyskali wynik poniżej 40% z któregokolwiek przedmiotu.

Jako wynik otrzymano nową tablicę z kolumnami `score`, `performance` i `warning`.

Warto zauważyć, że ramki danych są niezwykle elastycznym narzędziem do przechowywania danych o zróżnicowanej strukturze. W wielu zastosowaniach analitycznych pojawia się jednak potrzeba pracy z danymi o większej liczbie wymiarów niż dwa, co wykracza poza możliwości zarówno klasycznych macierzy, jak i ramek danych.

W kolejnym podrozdziale zostanie omówiona *tablica* – struktura danych stanowiąca uogólnienie wektora i macierzy. Dzięki niej możliwe jest przechowywanie oraz przetwarzanie danych wielowymiarowych, co ma istotne znaczenie w analizie bardziej złożonych zbiorów danych.

2.2.8. Tablice

Tablica jest uogólnieniem struktur danych, takich jak wektory i macierze. Tworzy się ją za pomocą funkcji `array`. W tablicach można łatwo gromadzić wielowymiarowe zbiory danych. Odwołanie się do elementów tablicy odbywa się poprzez podanie współrzędnych w nawiasach kwadratowych `[i, j, k]`, gdzie `i` oznacza numer wiersza, `j` – numer kolumny, zaś `k` – numer warstwy (trzeciego wymiaru).

```
a <- array(
    1:20,
    dim = c(2, 5, 2),
    dimnames = list(
        letters[1:2],
        LETTERS[1:5]
    )
)
a
```

```
## , , 1
##
##   A B C D  E
## a 1 3 5 7  9
## b 2 4 6 8 10
##
## , , 2
##
```

```
##    A  B  C  D  E
## a 11 13 15 17 19
## b 12 14 16 18 20
```

```
a[, , 1]
```

```
##   A B C D  E
## a 1 3 5 7  9
## b 2 4 6 8 10
```

```
a[, , 2]
```

```
##    A  B  C  D  E
## a 11 13 15 17 19
## b 12 14 16 18 20
```

Kolejne wartości wektora podanego w argumencie `dim` oznaczają, że każda macierz w tablicy będzie miała 2 wiersze i 5 kolumn, a takich macierzy (warstw) będą dwie. Za pomocą argumentu `dimnames` można zmienić nazwy wierszy i kolumn w macierzach. Wektory nazw uporządkowano w postaci listy (szerzej zob. rozdział 2.2.9). Do sprawdzenia, czy obiekt jest tablicą, służy funkcja `is.array`.

```
is.array(a)
```

```
## [1] TRUE
```

Warto zauważyć, że wszystkie elementy tablicy muszą mieć ten sam typ danych. W praktyce analitycznej często zachodzi jednak potrzeba pracy z danymi o różnej strukturze – na przykład łączenia liczb, tekstów, ramek danych czy funkcji w jednym obiekcie. Dlatego w kolejnym podrozdziale zostanie przedstawiona *lista* – niezwykle uniwersalna struktura danych w języku R, umożliwiająca przechowywanie elementów o różnym typie i rozmiarze. Pozwala ona budować złożone obiekty, które mogą stanowić podstawę bardziej zaawansowanych operacji programistycznych i analitycznych.

2.2.9. Listy

Lista jest obiektem, który może zawierać inne obiekty różnych typów, np. liczby, ciągi znaków, wektory, macierze, a nawet ramki danych i funkcje. Jest to bardzo często stosowany obiekt w skryptach programu R. Listę deklaruje się za pomocą funkcji `list`. Listy mogą być jedno- lub wielowymiarowe[7]. Poniżej zadeklarowano *pustą* listę jednowymiarową.

[7] Często mówi się również o listach *wielopoziomowych* lub *zagnieżdżonych*.

```
x <- list()
x
```

```
## list()
```

Jak wspomniano, elementami listy mogą być różne obiekty.

```
x <- list(
    a = 1,
    b = "text",
    c = letters[1:5]
)
x
```

```
## $a
## [1] 1
##
## $b
## [1] "text"
##
## $c
## [1] "a" "b" "c" "d" "e"
```

Do elementów listy można odwołać się na dwa sposoby. W pierwszym stosuje się podwójny nawias kwadratowy `[[k]]`, gdzie `k` oznacza numer współrzędnej listy. W drugim korzysta się z operatora `$`, podobnie jak w przypadku ramek danych, przy czym wcześniej muszą zostać zadeklarowane nazwy elementów listy.

```
x[[1]]
```

```
## [1] 1
```

```
x$a
```

```
## [1] 1
```

```
x[[3]]
```

```
## [1] "a" "b" "c" "d" "e"
```

```
x$c
```

```
## [1] "a" "b" "c" "d" "e"
```

Listę wielowymiarową (wielopoziomową) tworzy się poprzez zagnieżdżanie list jednowymiarowych. W ten sposób powstaje lista złożona z innych list.

```
x <- list(
    a = list(
        a = c(1, 2),
        b = c(1:5),
        c = c(5:10)
    ),
    b = list(
        a = c(letters[1:5]),
        b = c("text1", "text2")
    ),
    c = list(
        a = c("a", "b", "c")
    )
)
x
```

```
## $a
## $a$a
## [1] 1 2
##
## $a$b
## [1] 1 2 3 4 5
##
## $a$c
## [1]  5  6  7  8  9 10
##
##
## $b
## $b$a
## [1] "a" "b" "c" "d" "e"
##
## $b$b
## [1] "text1" "text2"
##
##
## $c
## $c$a
## [1] "a" "b" "c"
```

Do elementów listy wielowymiarowej należy odwoływać się w podobny sposób jak do elementów listy jednowymiarowej, tj. poprzez podanie jej współrzędnych. Jeśli elementy list mają nazwy, co nie jest konieczne, to można skorzystać z operatora $.

```
x[[1]][[1]]
```

```
## [1] 1 2
```

```
x[[1]][[2]]
```

```
## [1] 1 2 3 4 5
```

```
x$a$c
```

```
## [1]  5  6  7  8  9 10
```

```
x[[2]][[1]]
```

```
## [1] "a" "b" "c" "d" "e"
```

```
x[[2]]$b
```

```
## [1] "text1" "text2"
```

```
x$c$a
```

```
## [1] "a" "b" "c"
```

Wymiar listy można uzyskać za pomocą funkcji `length`.

```
length(x)
```

```
## [1] 3
```

```
length(x[[1]])
```

```
## [1] 3
```

```
length(x[[2]])
```

```
## [1] 2
```

```
length(x$c$a)
```

```
## [1] 3
```

Listy są bardzo ważnym elementem organizacji i optymalizacji kodu programu. Wykorzystuje się je powszechnie do gromadzenia wyników obliczeń. W języku R występuje niezwykle ważna funkcja `lapply` przeznaczona do tworzenia i przetwarzania list.

```
y <- lapply(
    c(1:5),
    function(x) print(x)
)
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

```
str(y)
```

```
## List of 5
##  $ : int 1
##  $ : int 2
##  $ : int 3
##  $ : int 4
##  $ : int 5
```

```
y
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
```

```
## [1] 3
##
## [[4]]
## [1] 4
##
## [[5]]
## [1] 5
```

Za pomocą funkcji `str` sprawdzono, że obiekt y jest listą indeksowaną w zbiorze pięcioelementowym. Za indeksację listy jest odpowiedzialny wektor `c(1:5)`, tj. pierwszy argument funkcji `lapply`. Elementami listy są wartości tego wektora. Za przypisanie tych wartości do współrzędnych listy jest odpowiedzialna funkcja z argumentem `x`. Do tej funkcji są kolejno przekazywane wartości wektora `c(1:5)`, tj. liczby od 1 do 5. Funkcja `function` jest dowolną funkcją zwracającą wyniki obliczeń.

Zastosowanie funkcji `lapply` może być bardzo szerokie. W poniższym kodzie najpierw utworzono ramkę danych `df`.

```
df <- data.frame(
    x = 1:4,
    y = c(2, 2, 3, 3),
    z = rep(1, 4)
)
df
```

```
##   x y z
## 1 1 2 1
## 2 2 2 1
## 3 3 3 1
## 4 4 3 1
```

Następnie wykonano na jej zbiorze obliczenia średnich arytmetycznych w kolumnach.

```
df.1 <- data.frame(
    lapply(
        df,
        function(x) mean(x)
    )
)
df.1
```

```
##     x   y z
## 1 2.5 2.5 1
```

W powyższej funkcji `lapply` zbiorem indeksującym listę przekształconą w ramkę danych `df.1` są kolumny ramki danych `df`. Poniższy kod realizuje obliczenia w ramce danych `df`, dla której wykonano unitaryzację zerowaną.

```
df.2 <- data.frame(
  lapply(
    df,
    function(x) {
        if (min(x) != max(x))
            {(x - min(x)) / (max(x) - min(x))}
        else {0}
    }
  )
)
round(df.2, digits = 2)
```

```
##      x y z
## 1 0.00 0 0
## 2 0.33 0 0
## 3 0.67 1 0
## 4 1.00 1 0
```

Ramka danych `df.2` stanowi przekształconą postać ramki `df`, dla której wykonano unitaryzację (standaryzację) wartości w kolumnach. Można zauważyć, że ponownie wykorzystano funkcję `lapply` indeksowaną kolumnami ramki `df`. Następnie w funkcji `function(x)`, gdzie `x` oznacza wartości z poszczególnych kolumn ramki `df`, zostały wykonane obliczenia. Ostatecznie wartości liczbowe zostały zaokrąglone do dwóch miejsc po przecinku.

Lista może służyć do deklarowania wielopoziomowych, uporządkowanych struktur danych, służących do opisu zmiennych. Poniżej podano przykład takiej listy.

```
key <- list(
    list(
        cat = "telefony",
        subcat = "telefony komórkowe",
        id = 1,
        prod = c(
            "APPLE",
            "HUAWEI",
            "LENOVO",
            "LG",
            "SAMSUNG"
        )
    ),
    list(
        cat = "telewizory",
```

```
        subcat = "telewizory LED",
        id = 2,
        prod = c(
            "JVC",
            "LG",
            "SAMSUNG",
            "SONY",
            "TOSHIBA"
        )
    ),
    list(
        cat = "aparaty",
        subcat = "aparaty cyfrowe",
        id = 3,
        prod = c(
            "CANON",
            "KODAK",
            "NIKON",
            "OLYMPUS",
            "SONY"
        )
    ),
    list(
        cat = "komputery",
        subcat = "laptopy",
        id = 4,
        prod = c(
            "APPLE",
            "ASUS",
            "DELL",
            "HP",
            "LENOVO"
        )
    ),
    list(
        cat = "monitory",
        subcat = "monitory LED",
        id = 5,
        prod = c(
            "HP",
            "LENOVO",
            "LG",
            "PHILIPS",
            "SAMSUNG"
        )
    ),
    list(
        cat = "drukarki",
        subcat = "drukarki laserowe",
```

```
        id = 6,
        prod = c(
            "CANON",
            "HP",
            "LEXMARK",
            "RICOH",
            "SAMSUNG"
        )
    )
)
```

Po utworzeniu listy `key` można utworzyć kolejną listę zawierającą np. kategorie `cat`.

```
cat <- lapply(
    1:length(key),
    function(x)
        paste(key[[x]][[1]], collapse = ",")
)
cat <- sort(
    paste(cat),
    decreasing = FALSE
)
data.frame(cat)
```

```
##          cat
## 1    aparaty
## 2   drukarki
## 3  komputery
## 4   monitory
## 5   telefony
## 6 telewizory
```

W podobny sposób można utworzyć listę `prod` zawierającą nazwy producentów urządzeń elektroniki użytkowej.

```
prod <- lapply(
    1:length(key),
    function(x) key[[x]]$prod
)
prod <- sort(
    unique(unlist(prod)),
    decreasing = FALSE
)
data.frame(prod)
```

```
##       prod
```

```
## 1    APPLE
## 2     ASUS
## 3    CANON
## 4     DELL
## 5       HP
## 6   HUAWEI
## 7      JVC
## 8    KODAK
## 9   LENOVO
## 10 LEXMARK
## 11      LG
## 12   NIKON
## 13 OLYMPUS
## 14 PHILIPS
## 15   RICOH
## 16 SAMSUNG
## 17    SONY
## 18 TOSHIBA
```

W powyższym kodzie zastosowano funkcję `unique`, która zwraca wektor, ramkę danych lub tablicę, z których usunięte zostały elementy powtarzające się. Wcześniej struktura listy `prod` została uproszczona do postaci wektora za pomocą funkcji `unlist`. Jak widać, lista może być podstawą dalszych przekształceń, na podstawie których można otrzymać potrzebne obiekty.

Konstrukcję listy można również wykorzystać do filtrowania ramki danych. Ilustruje to poniższy przykład. Na początku utworzona została lista db warunków.

```
db <- list(
  a = letters[1:3],
  b = LETTERS[1:3]
)
db
```

```
## $a
## [1] "a" "b" "c"
##
## $b
## [1] "A" "B" "C"
```

Następnie utworzono ramkę danych `df.1`, która będzie poddana filtrowaniu.

```
df.1 <- data.frame(
  a = letters[1:6],
  b = LETTERS[1:6]
)
df.1
```

```
##   a b
## 1 a A
## 2 b B
## 3 c C
## 4 d D
## 5 e E
## 6 f F
```

W kolejnym kroku utworzono listę cond warunków logicznych. Elementy listy zostały następnie zagregowane do ciągu znaków z separatorem & i ponownie zapisane w zmiennej cond[8].

```
cond <- lapply(
  names(db),
  function(x) paste0(x, " %in% db$", x)
)
cond <- paste(cond, collapse = " & ")
cond
```

```
## [1] "a %in% db$a & b %in% db$b"
```

W kolejnym kroku filtrowaniu poddano ramkę df.1 w oparciu o złożony warunek cond. Wynik zapisano w ramce danych df.2.

```
df.2 <- subset(df.1, eval(parse(text = cond)))
df.2
```

```
##   a b
## 1 a A
## 2 b B
## 3 c C
```

Zastosowanie konstrukcji eval(parse(...)) umożliwia dynamiczne zbudowanie wyrażenia logicznego na podstawie zawartości listy db, jednak w praktyce zaleca się ostrożność przy tego typu rozwiązaniach (zwłaszcza dla danych pochodzących z zewnętrznych źródeł). Zmiana warunków podanych w liście db spowoduje odmienne filtrowanie ramki df.1. Proces tworzenia listy warunków db, ich koniunkcji (lub np. alternatywy po zmianie separatora & na |) i wyniku filtrowania df.2 można uwzględnić w funkcji, której parametrem jest filtrowana ramka df.1.

W kolejnym podrozdziale zostaną przedstawione instrukcje graficzne, które pozwalają na wizualizację danych w języku R. Graficzne przedstawienie wyników analiz

[8] Na podstawie podrozdziału 2.2.1, Czytelnik może zapisać tę sekwencję samodzielnie za pomocą operatora potokowego %>%.

jest ważnym elementem pracy badawczej – umożliwia szybkie rozpoznanie zależności, trendów i nietypowości w danych. Zostaną omówione zarówno podstawowe funkcje graficzne wbudowane w R, jak i wprowadzenie do bardziej zaawansowanych metod wizualizacji.

2.2.10. Instrukcje graficzne

Przedstawienie danych statystycznych w postaci graficznej stanowi istotny element warsztatu pracy statystyka i ekonometryka. W języku R dane można przedstawić w postaci graficznej na wiele sposobów. Istnieją zarówno wbudowane funkcje graficzne, jak i specjalistyczne biblioteki do wizualizacji danych. Najpierw zostanie pokazane zastosowanie funkcji wbudowanych w R (funkcji systemowych).

Niech y_t oznacza wektor pięciu zmiennych, z których każda ma rozkład normalny standaryzowany:

$$y_t = [y_{1t}, \dots, y_{kt}], \quad k = 5 \tag{2.2}$$

Zmienne wektora y_t można wygenerować w pętli lub zastosować operacje na listach. Zastosowano drugie rozwiązanie.

```
set_seed()
y <- lapply(
    1:5,
    function(x) rnorm(100, mean = 0, sd = 1)
)
```

Zmienna y jest listą pięciu 100-elementowych zbiorów wartości zmiennych o rozkładzie normalnym standaryzowanym. Listę tę można przekształcić w ramkę danych, stosując na przykład przetwarzanie potokowe.

```
library(magrittr)
digits <- 2
myfun <- function(x, d) round(x, digits = d)
df <- do.call(cbind, y) %>%
    data.frame() %>%
    set_colnames(c(paste0("y", 1:5))) %>%
    myfun(., digits)
head(df)
```

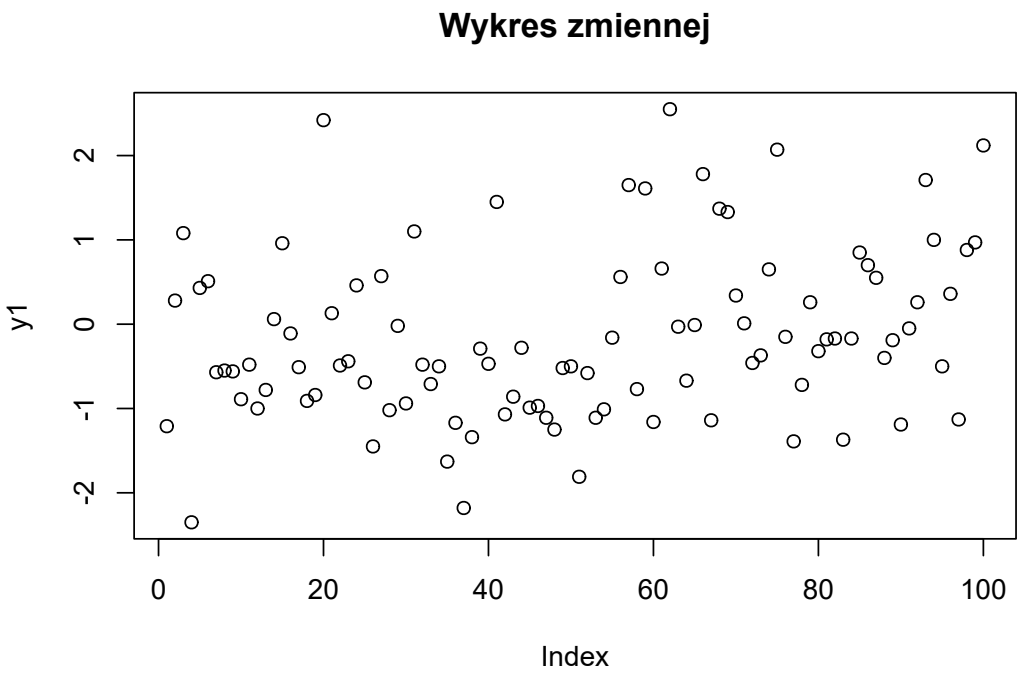
```
##      y1    y2   y3    y4    y5
## 1 -1.21  0.41 0.49 -0.58 -1.23
## 2  0.28 -0.47 0.70 -0.95  0.04
```

```
## 3  1.08  0.07 0.19 -0.18 -0.42
## 4 -2.35 -0.50 0.70  1.01 -0.90
## 5  0.43 -0.83 0.31  0.02  0.42
## 6  0.51  0.17 0.76 -0.65  0.15
```

Ramka danych `df` powstała przez zastosowanie funkcji `do.call` i `cbind` do elementów listy `y`. Następnie przypisano nazwy kolumn `y1` do `y5`. Wartości zmiennych zostały zaokrąglone do 2 miejsc po przecinku.

Ramka `df` będzie podstawą konstrukcji wykresów zmiennych. Na początku można wykonać wykres jednej zmiennej na podstawie funkcji wbudowanej w języku R. Do tworzenia wykresów służy funkcja `plot`, do której przekazuje się parametry. Najprostszy wykres można utworzyć, przekazując do funkcji wyłącznie jeden zbiór danych (rys. 2.6).

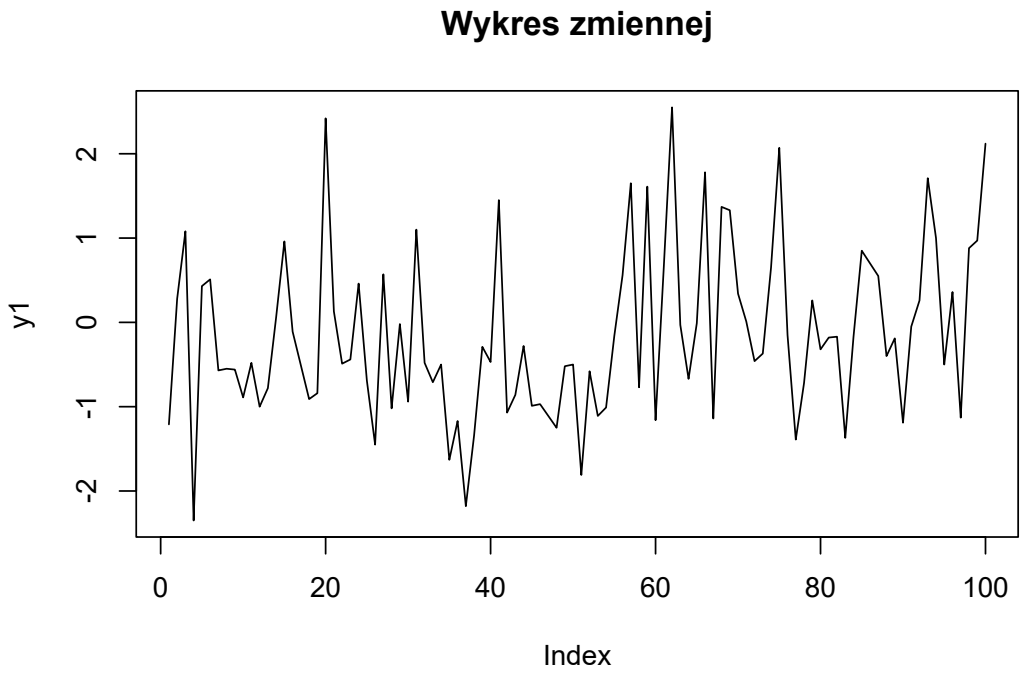
```
y1 <- df[, 1]
plot(y1, main = "Wykres zmiennej")
```



Rysunek 2.6. Zmienna o rozkładzie normalnym standaryzowanym – przypadek 1

Stosując argument `type` w funkcji `plot` można zmienić sposób przedstawienia zmiennej (rys. 2.7 i 2.8).

```
plot(y1, type = "l", main = "Wykres zmiennej")
```



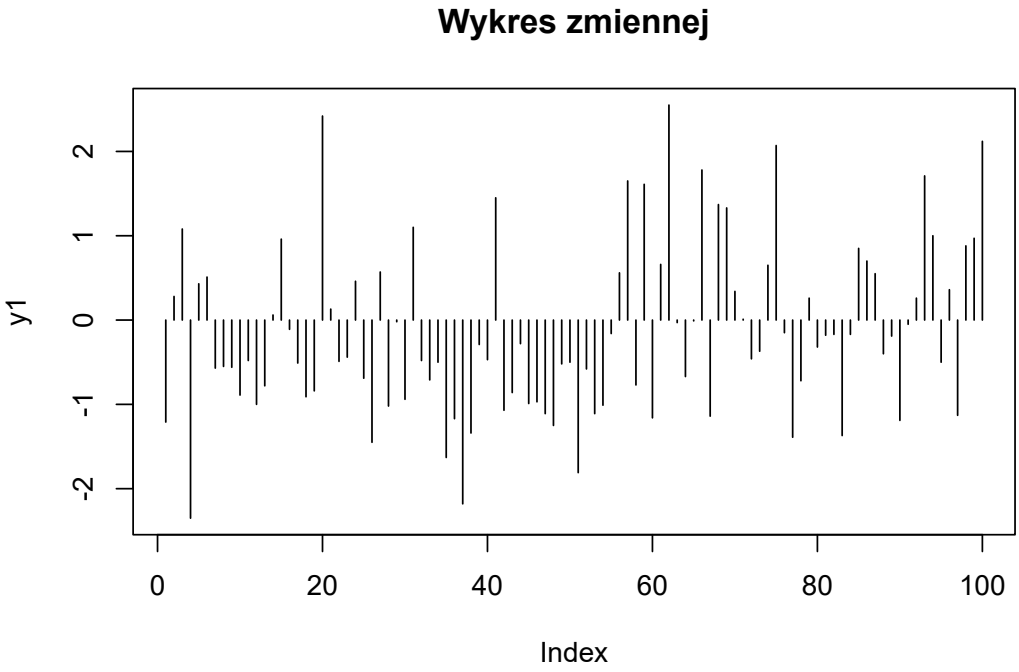
Rysunek 2.7. Zmienna o rozkładzie normalnym standaryzowanym – przypadek 2

```
plot(y1, type = "h", main = "Wykres zmiennej")
```

Przekazanie kolejnych parametrów do funkcji `plot` pozwala dodać na wykresie następne informacje (rys. 2.9).

```
plot(
    y1,
    type = "l",
    main = "Wykres zmiennej",
    xlab = "t",
    ylab = "y1"
)
```

Jak widać, utworzenie wykresu jednej zmiennej jest stosunkowo proste. Jeśli przekaże się do funkcji `plot` dwie lub więcej zmiennych, to otrzyma się wykres punktowy (rozrzutu) w układzie współrzędnych (rys. 2.10).



Rysunek 2.8. Zmienna o rozkładzie normalnym standaryzowanym – przypadek 3

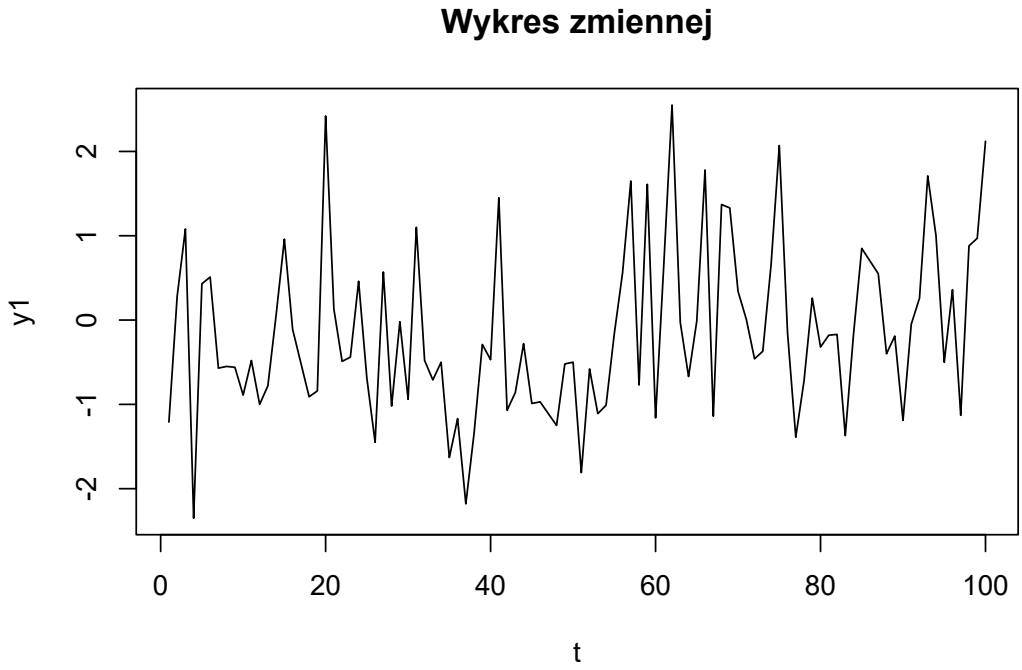
```
y2 <- df[, 2]
plot(y1, y2, main = "Wykres zmiennych")
```

Przekazanie do funkcji `plot` ramki danych `df` daje obraz pokazany na rys. 2.11.

```
plot(df, main = "Wykres zmiennych")
```

Na rys. 2.11 widoczna jest macierz wykresów punktowych. Każdy wykres został utworzony w taki sposób, że w pierwszej kolumnie zmienna `y1` występuje na osi `X`, natomiast na osi `Y` zmienne `y2` do `y5`. Wykresy w kolejnych kolumnach powstały analogicznie. Utworzenie odrębnego rysunku dla przebiegu każdej zmiennej jest również możliwe. Należy w tym celu przekształcić ramkę danych `df` do postaci szeregów czasowych za pomocą funkcji `ts` i ponownie zastosować funkcję `plot` (rys. 2.12).

```
ts <- df %>% ts()
plot(
    ts,
    main = "Wykres zmiennych"
)
```



Rysunek 2.9. Zmienna o rozkładzie normalnym standaryzowanym – przypadek 4

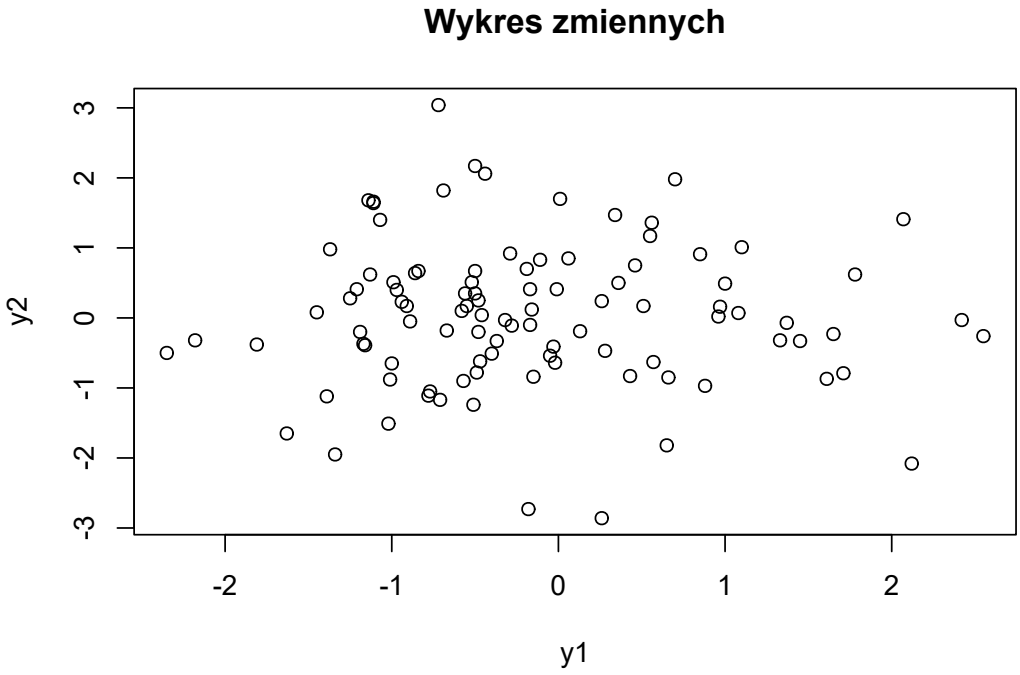
Możliwości prezentacji danych za pomocą funkcji `plot` są często ograniczone, a tworzenie bardziej zaawansowanych wykresów bywa pracochłonne. Aby uzyskać bardziej rozbudowane struktury graficzne, należy skorzystać ze specjalistycznych bibliotek. Powszechnie stosowanym narzędziem jest biblioteka `ggplot2` (Wickham i in., 2024a). Tworzenie kodu należy rozpocząć od wczytania tej biblioteki.

```
library(ggplot2)
```

Na początku rozważań zostanie utworzony wykres zmiennej `y1`. Najpierw trzeba uzupełnić ramkę danych `df` o jedną kolumnę, która będzie zawierać indeks (czasu) t, gdzie $t = 1, \ldots, T$, $T = 100$, i utworzyć nową ramkę danych `df.t`. Aby utworzyć wykres, stosuje się funkcję `ggplot` z parametrami. Wykres tworzy się w warstwach `geom`, które zawierają poszczególne elementy rysunku.

```
df.t <- df %>% cbind(t = 1:nrow(.),.)
ggplot(df.t, aes(x = t)) +
    geom_line(aes(y = y1))
```

Na rys. 2.13 pokazano wykres zmiennej `y1` oznaczonej linią ciągłą. Rysunek utworzono za pomocą dwóch warstw związanych z osiami `X` i `Y`. Struktura polecenia `ggplot` jest następująca:



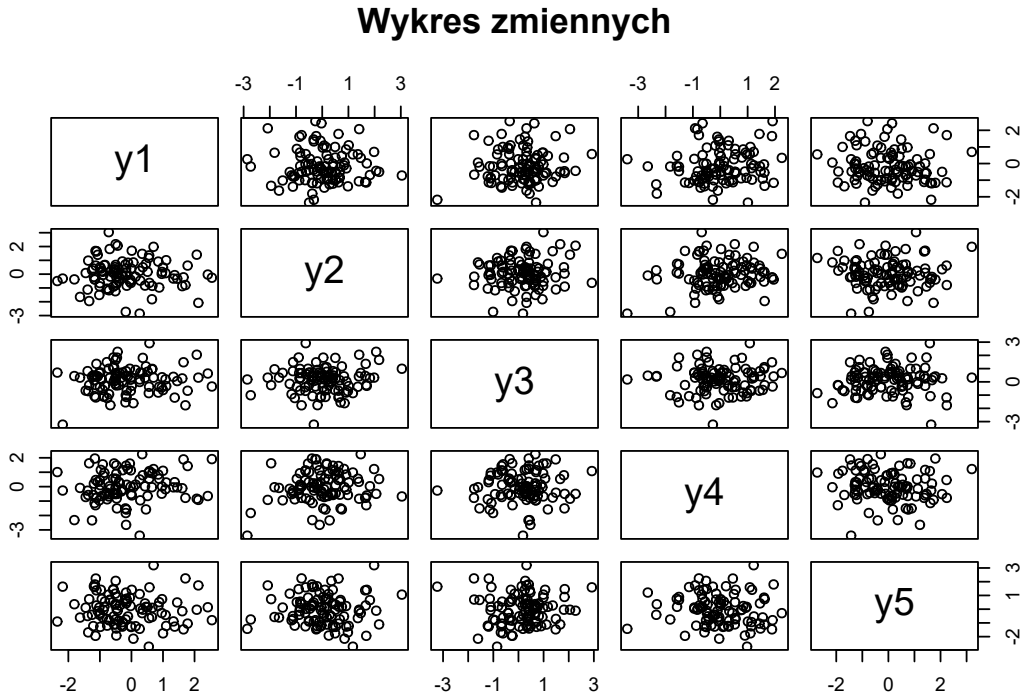
Rysunek 2.10. Rozrzut zmiennych o rozkładzie normalnym standaryzowanym – przypadek 1

- w warstwie pierwszej (ogólnej) podaje się zbiór danych (`df.t`) i oznaczenie osi za pomocą argumentu `aes` (*aesthetics*),
- w warstwach następnych wskazuje się sposób przedstawienia poszczególnych zmiennych,
- warstwy łączy się operatorem + (plus).

Argument `aes(x = t)` oznacza, że na osi X pojawią się wartości zmiennej `t` ze zbioru `df.t`. Argument `geom_line(aes(y = y1))` oznacza, że wartości zmiennej `y1` ze zbioru `df.t` zostaną oznaczone na osi Y, a punkty w czasie połączone linią. Te same efekty można uzyskać w prostszy sposób (rys. 2.14).

```
ggplot(df.t, aes(x = t, y = y1)) +
    geom_line()
```

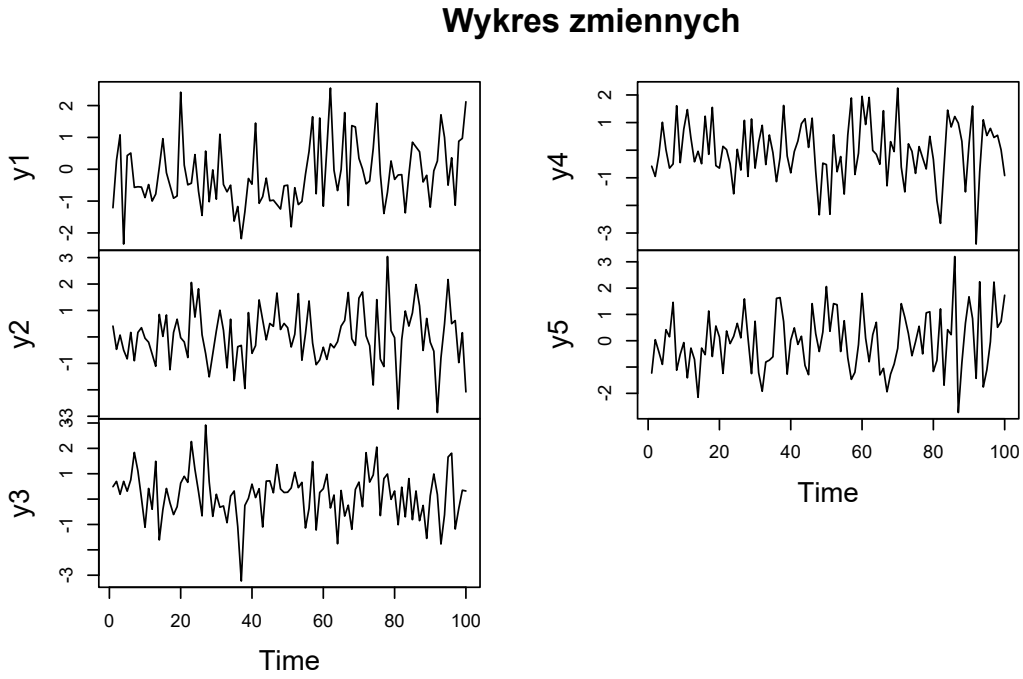
W tym przypadku wartości osi X i Y zostały podane w jednej opcji `aes`. Druga warstwa określa sposób przedstawienia wykresu w postaci linii (`geom_line`). Uzyskanie wykresu dla dwóch zmiennych oznacza, że tworzy się kolejne warstwy (rys. 2.15).



Rysunek 2.11. Rozrzut zmiennych o rozkładzie normalnym standaryzowanym – przypadek 2

```
ggplot(df.t, aes(x = t)) +
    geom_line(
        aes(y = y1),
        color = "#626567",
        linetype = "solid"
    ) +
    geom_line(
        aes(y = y2),
        color = "#BDC3C7",
        linetype = "dashed"
    )
```

Utworzenie jednego wykresu dla większej liczby zmiennych może być jednak uciążliwe, ponieważ dla każdej zmiennej trzeba utworzyć osobną warstwę. Można jednak zmienić konstrukcję obiektu zawierającego dane statystyczne – ramkę danych `df.t` – w taki sposób, aby utworzenie wykresu wielu zmiennych nie sprawiało kłopotów. Wówczas dodatkowo można uzyskać legendę na wykresie. Aby przekształcić ramkę `df.t` trzeba zastosować odpowiednią bibliotekę – `reshape2` (Wickham, 2020). Podobnie jak wcześniej, trzeba wczytać tę bibliotekę.



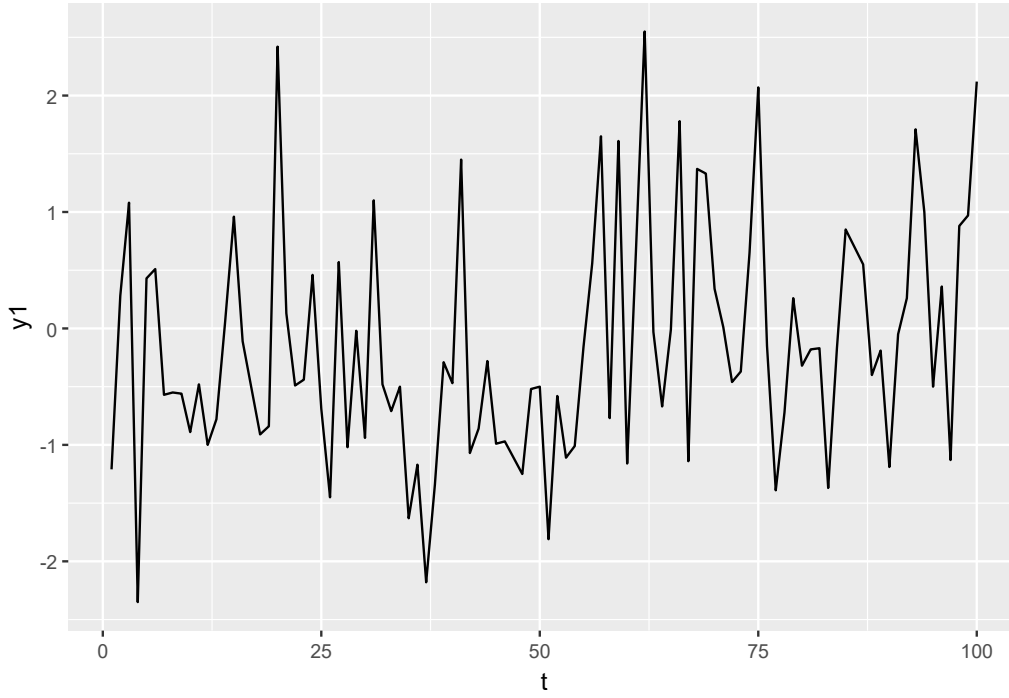
Rysunek 2.12. Wiele zmiennych o rozkładzie normalnym standaryzowanym – `plot`

```
library(reshape2)
```

Następnie należy zmodyfikować ramkę danych za pomocą funkcji `melt` do postaci *stosu*. Warto wspomnieć o różnicy między *szeroką* a *długą* tablicą (ramką danych). W formacie szerokim różne zmienne znajdują się w osobnych kolumnach, natomiast tablica długa przechowuje dane w układzie *zmienna–wartość*, przez co ma więcej wierszy, ale mniej kolumn. Funkcja `melt` służy właśnie do przekształcania danych z formatu szerokiego w długi.

```
df.melt <- melt(df.t, id.var = 't')
head(df.melt)
```

```
##   t variable value
## 1 1       y1 -1.21
## 2 2       y1  0.28
## 3 3       y1  1.08
## 4 4       y1 -2.35
## 5 5       y1  0.43
## 6 6       y1  0.51
```

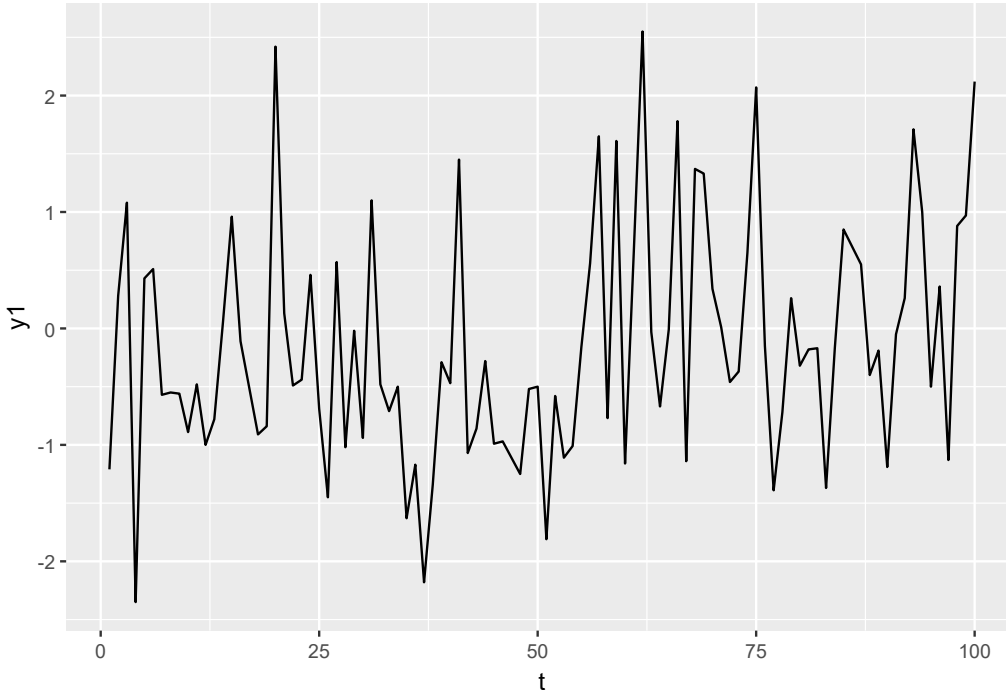


Rysunek 2.13. Zmienna o rozkładzie normalnym standaryzowanym – `ggplot` – przypadek 1

Jak łatwo zauważyć, obiekt `df.melt` zawiera dane statystyczne z ramki `df.t` indeksowane przez zmienną `t`. Teraz można wykorzystać ramkę `df.melt` do utworzenia wykresu wszystkich zmiennych (rys. 2.16).

```
ggplot(
    df.melt,
    aes(x = t, y = value, colour = variable)
) +
geom_line() +
scale_colour_grey() +
theme_bw()
```

Argument `colour = variable` oznacza, że linie dla zmiennych indeksowanych przez kolumnę `variable` będą rysowane innym kolorem. Argument `scale_colour_grey` wprowadza skalę szarości dla linii, natomiast `theme_bw` dodatkowe rozwiązania kolorystyczne (schemat oparty na kolorach czarnym i białym). Rysunek 2.16 można dalej modyfikować. Można np. nadać mu odpowiednią kolorystykę, zamieścić tytuł, etykiety osi `X` i `Y` itd. Zmodyfikowany wykres pokazano na rys. 2.17.

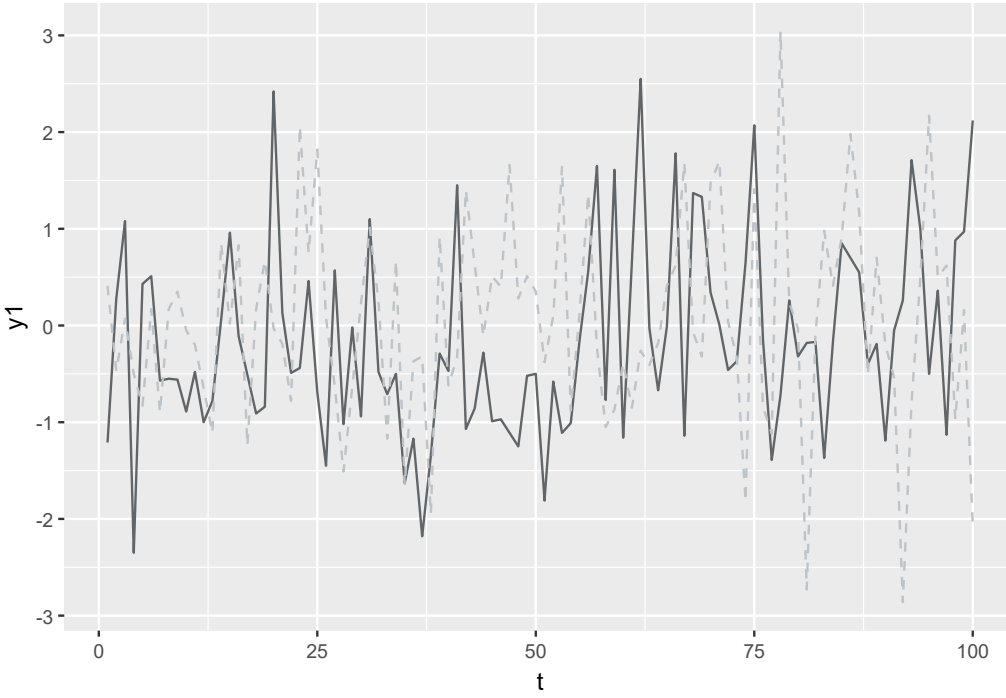


Rysunek 2.14. Zmienna o rozkładzie normalnym standaryzowanym – `ggplot` – przypadek 2

```
ggplot(
    df.melt,
    aes(x = t, y = value, colour = variable)
) +
geom_line() +
theme_bw() +
labs(title = "Wykres zmiennych") +
theme(
    plot.title = element_text(hjust = 0.5, size = 12),
    axis.title.x = element_blank(),
    axis.title.y = element_blank(),
    axis.text.x = element_text(size = 12),
    axis.text.y = element_text(size = 12)
) +
scale_colour_grey()
```

Na rys. 2.17 uwzględniono następujące elementy dodatkowe (warstwy):

- tytuł wykresu (`labs(title = "Wykres wielu zmiennych")`,
- etykiety osi (`theme`), gdzie:



Rysunek 2.15. Wiele zmiennych o rozkładzie normalnym standaryzowanym – `ggplot` – przypadek 1

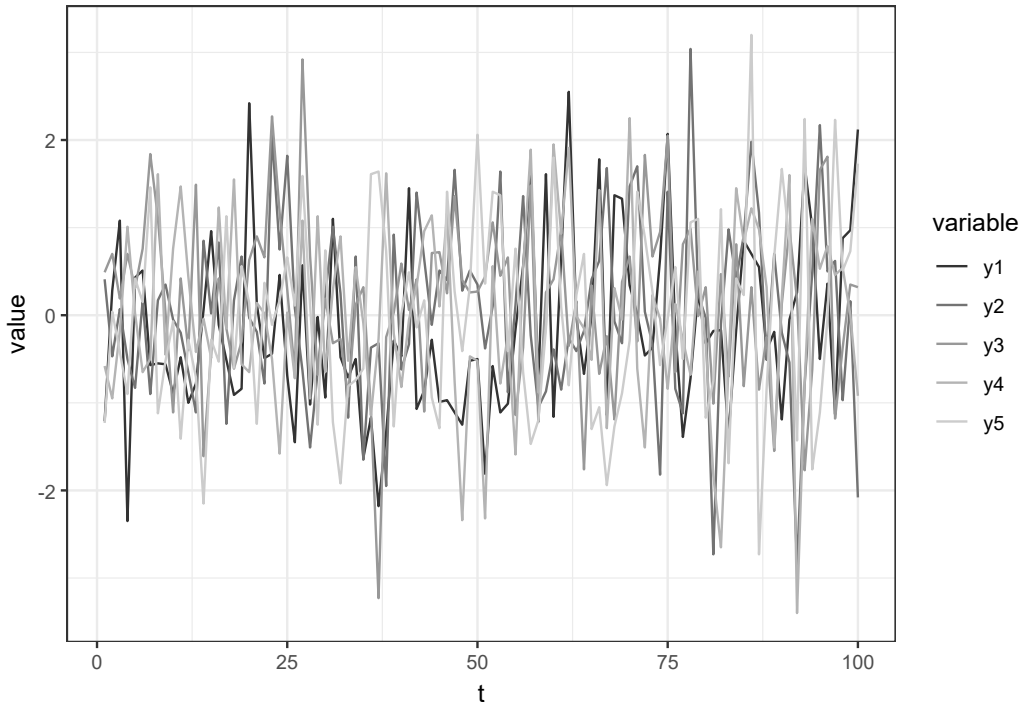
- `hjust` – położenie tytułu,
- `size` – wielkość czcionki,
- `element_blank` – usunięcie informacji z wykresu.

Jak łatwo zauważyć, na rys. 2.17 występuje większa czcionka (`12pt`) względem poprzedniego rysunku i nie występują etykiety osi.

Rysunki można również umieścić w strukturze opisanej za pomocą macierzy. Na przykład przyjęto, że na jednym rysunku mają być umieszczone indywidualne wykresy:

- szeregu czasowego,
- histogramu,
- wykresu pudełkowego.

Niech y_t oznacza szereg czasowy o rozkładzie normalnym standaryzowanym, czyli $y_t \sim N(0, 1)$.



Rysunek 2.16. Wiele zmiennych o rozkładzie normalnym standaryzowanym – `ggplot` – przypadek 2

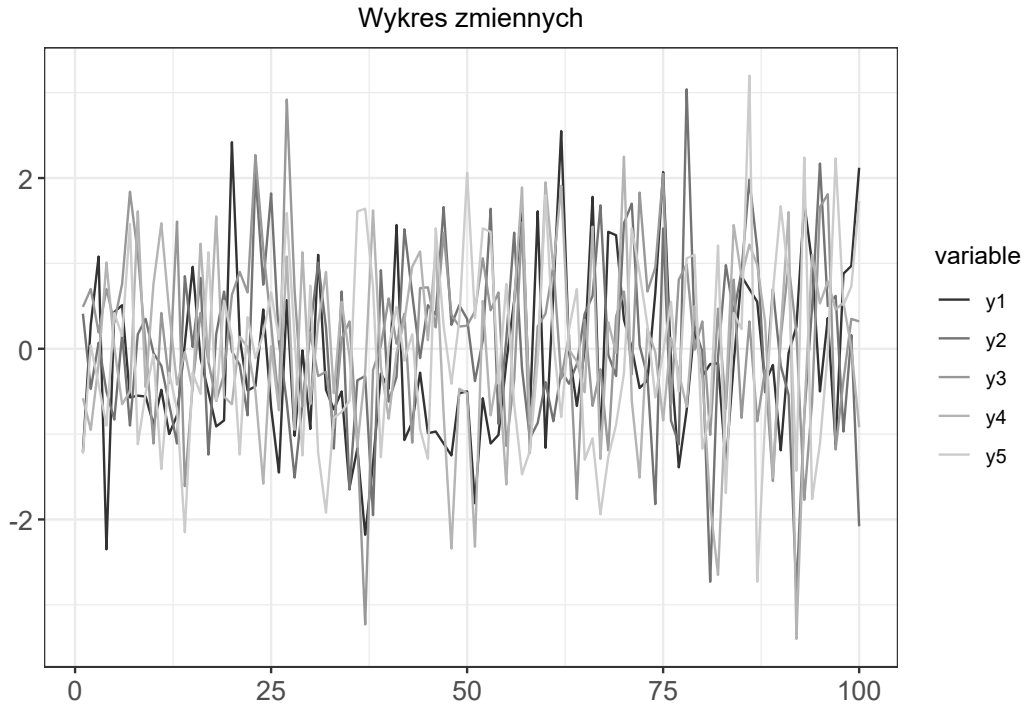
```
set_seed()
n <- 500
y <- data.frame(t = 1:n, y = rnorm(n))
```

Wiadomo już, jak należy utworzyć wykres jednej zmiennej (rys. 2.18).

```
plot.1 <- ggplot(y, aes(x = t, y = y)) +
geom_line() +
theme_bw()
plot.1
```

Wykres rozkładu częstości otrzymuje się za pomocą funkcji `geom_histogram` (rys. 2.19).

```
ggplot(y, aes(x = y)) +
geom_histogram(color = "black", fill = "white") +
theme_bw()
```

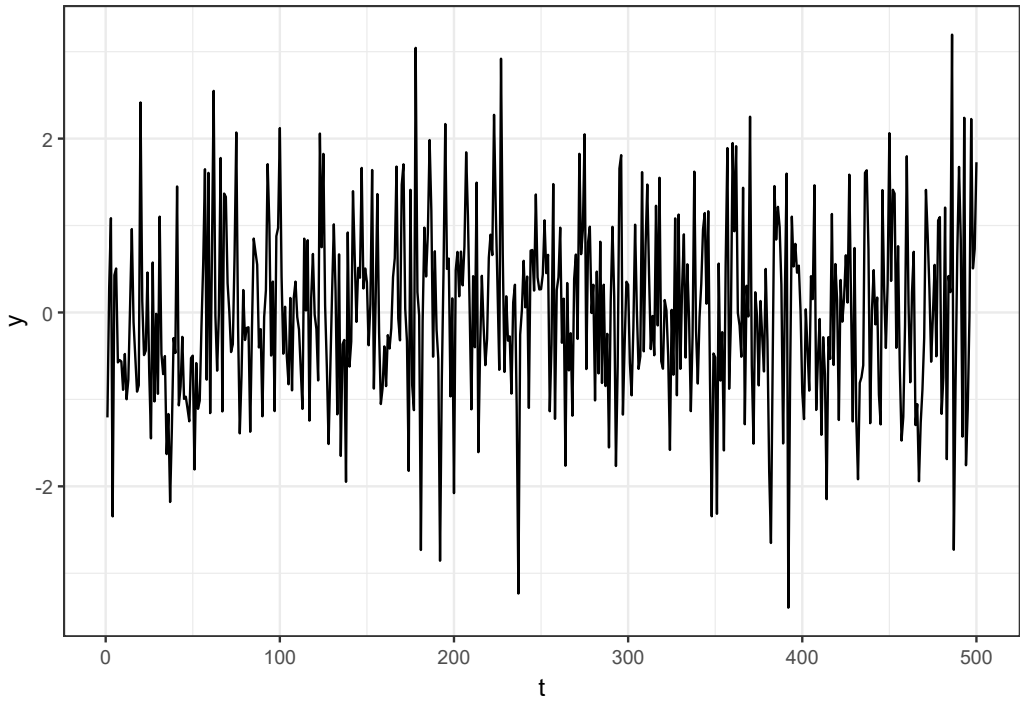


Rysunek 2.17. Wiele zmiennych o rozkładzie normalnym standaryzowanym – ggplot – przypadek 3

Na rys. 2.20 można umieścić średnią (linia ciągła) i medianę (linia przerywana) zmiennej y_t.

```
plot.2 <- ggplot(y, aes(x = y)) +
geom_histogram(color = "black", fill = "white") +
geom_vline(
    aes(xintercept = mean(y)),
    color = "#808080",
    linetype = "solid",
    size = 2
) +
geom_vline(
    aes(xintercept = median(y)),
    color = "#D3D3D3",
    linetype = "dashed",
    size = 2
) +
theme_bw()
plot.2
```

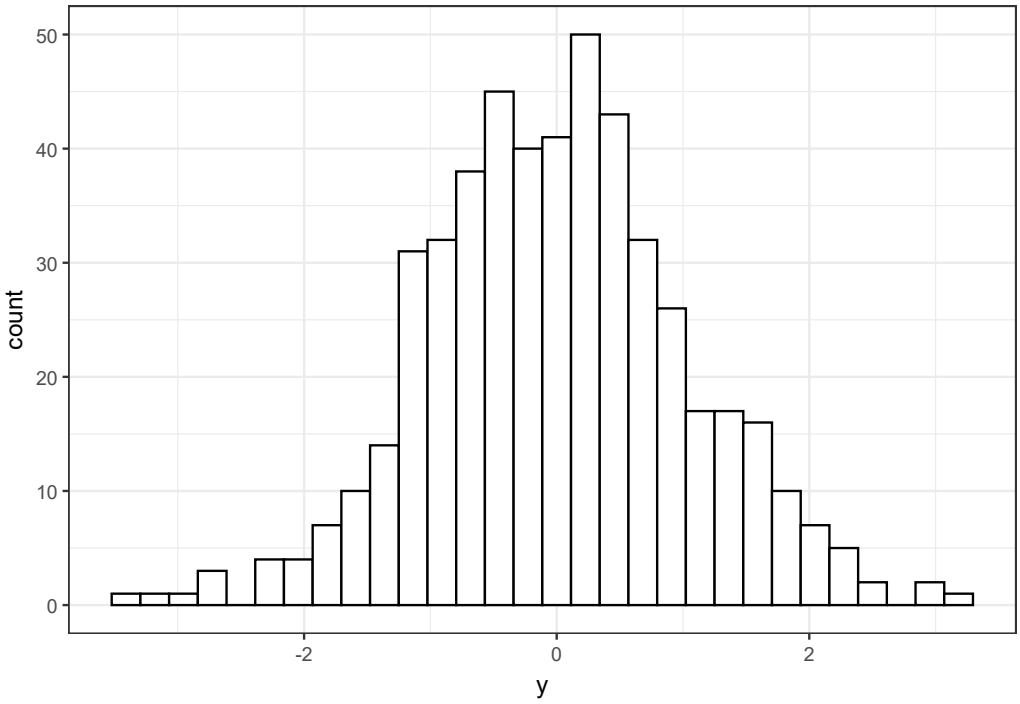
Wykres pudełkowy tworzy się za pomocą funkcji geom_boxplot (rys. 2.21).



Rysunek 2.18. Zmienna o rozkładzie normalnym standaryzowanym

```
plot.3 <- ggplot(y, aes(x = "y", y = y)) +
geom_boxplot(
    outlier.colour = "#A9A9A9",
    outlier.shape = 1,
    outlier.size = 3
) +
theme(
    axis.title.x = element_blank(),
    axis.title.y = element_blank(),
    axis.text.x = element_text(size = 12),
    axis.text.y = element_text(size = 12)
) +
theme_bw()
plot.3
```

Umieszczenie wykresów `plot.1`, `plot.2` oraz `plot.3` na jednym rysunku będzie wymagało utworzenia listy wykresów.



Rysunek 2.19. Histogram zmiennej o rozkładzie normalnym standaryzowanym – przypadek 1

```
plot <- list(plot.1, plot.2, plot.3)
```

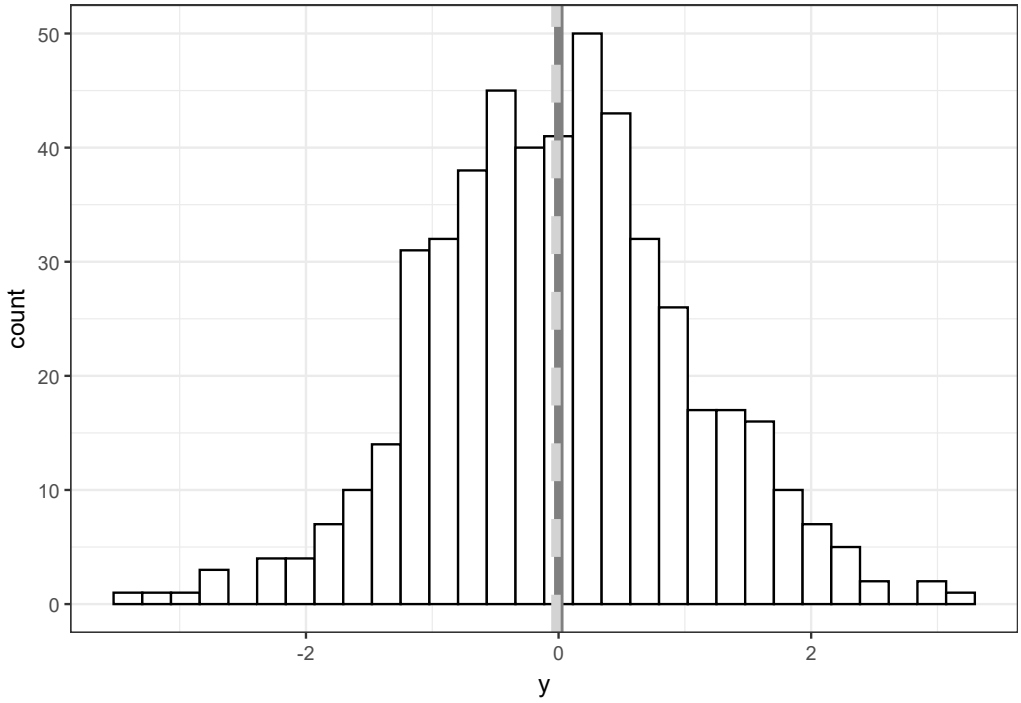
Teraz należy utworzyć macierz, która będzie stanowiła o położeniu wykresów. Niech postać tej macierzy oznacza, że wykresy są umieszczone obok siebie.

```
layout <- matrix(c(1, 2, 3), nrow = 1)
layout
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
```

Wykresy umieszcza się na jednym rysunku za pomocą funkcji `marrangeGrob`, która jest zawarta w bibliotece `gridExtra` (Auguie, 2017).

```
library(gridExtra)
marrangeGrob(
    plot,
    top = NULL,
```



Rysunek 2.20. Histogram zmiennej o rozkładzie normalnym standaryzowanym – przypadek 2

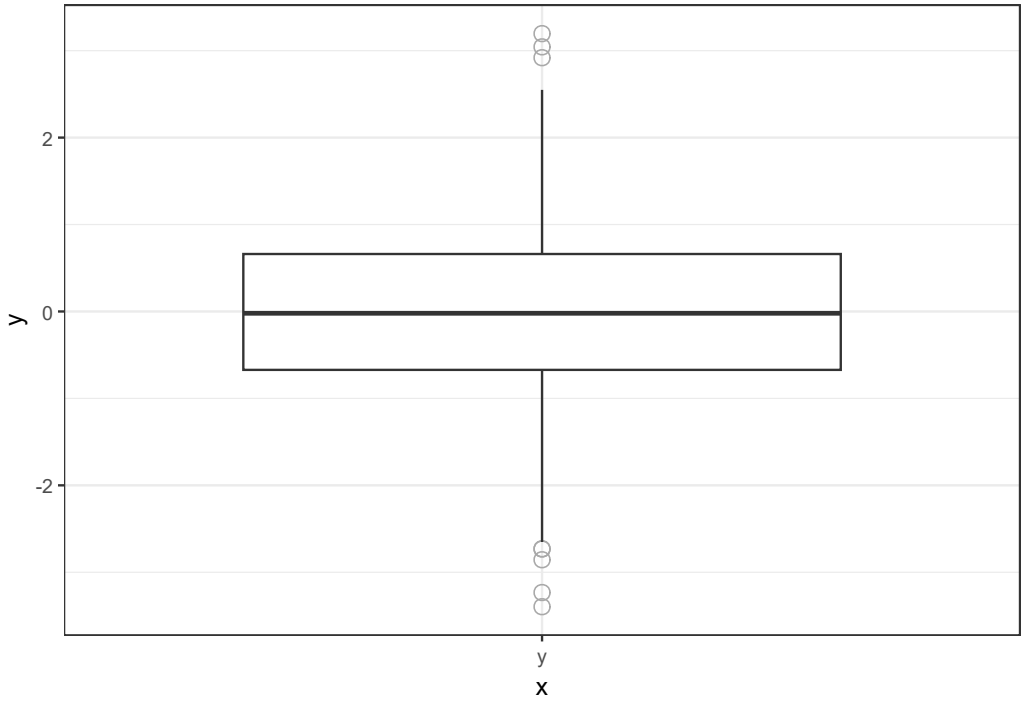
```
    layout_matrix = layout
)
```

Wykresy na rys. 2.22 zostały umieszczone w jednym wierszu zgodnie z konstrukcją macierzy `layout`. Inny układ współrzędnych tej macierzy będzie prowadził do innego układu wykresów (rys. 2.23).

```
layout <- matrix(c(1, 2, 3))
layout
```

```
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
```

```
marrangeGrob(
    plot,
    top = NULL,
```



Rysunek 2.21. Wykres pudełkowy zmiennej o rozkładzie normalnym standaryzowanym

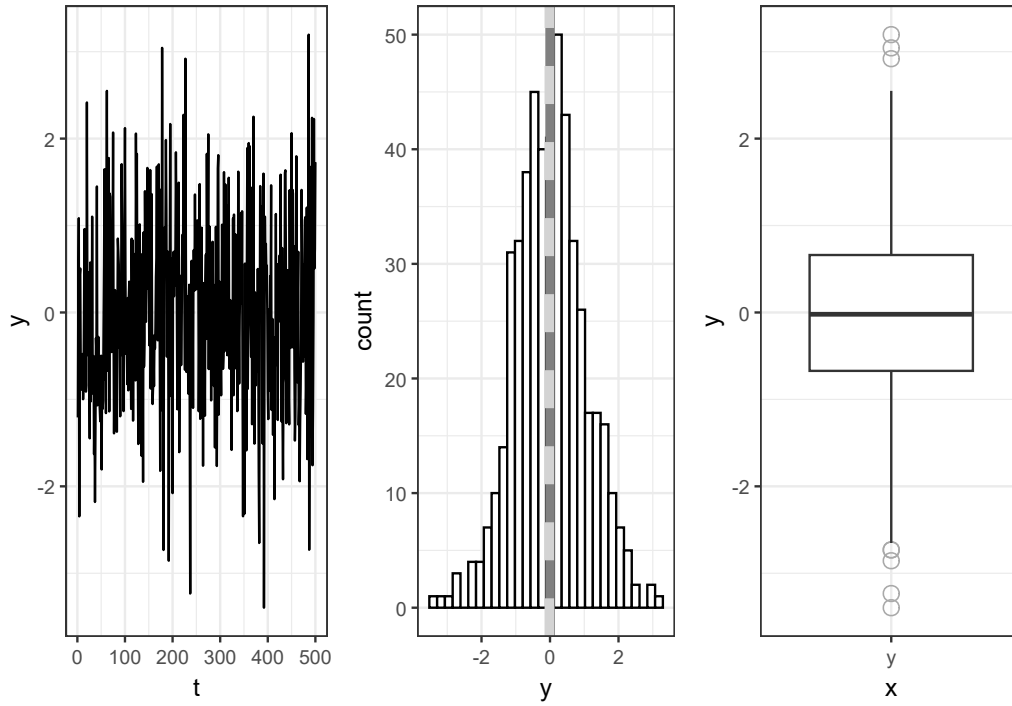
```
    layout_matrix = layout
)
```

Inne przykłady organizacji wykresów na rysunku są następujące (rys. 2.24 i 2.25):

```
layout <- rbind(c(1, 1, 1), c(1, 2, 3))
layout
```

```
##      [,1] [,2] [,3]
## [1,]    1    1    1
## [2,]    1    2    3
```

```
marrangeGrob(
    plot,
    top = NULL,
    layout_matrix = layout
)
```



Rysunek 2.22. Rysunek wykresów – przypadek 1

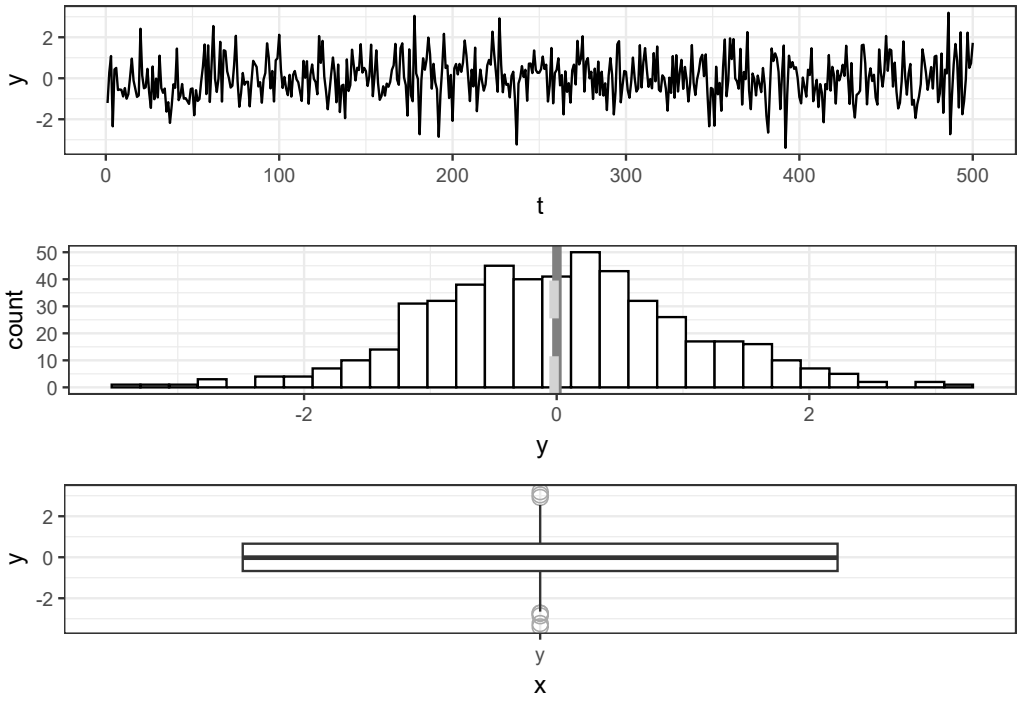
```
layout <- rbind(c(1, 1, 2), c(1, 1, 3))
layout
```

```
##      [,1] [,2] [,3]
## [1,]    1    1    2
## [2,]    1    1    3
```

```
marrangeGrob(
    plot,
    top = NULL,
    layout_matrix = layout
)
```

Jak widać, argument `layout_matrix` w funkcji `marrangeGrob` daje duże możliwości organizacji wykresów na jednym rysunku.

Teraz zostaną pokazane dalsze możliwości biblioteki `ggplot2`. Zostanie również przedstawiona graficzna biblioteka `plotly` (Sievert, 2020; Sievert i in., 2024). Jak zwykle, na początku skryptu trzeba umieścić polecenia wczytania odpowiednich bibliotek.



Rysunek 2.23. Rysunek wykresów – przypadek 2

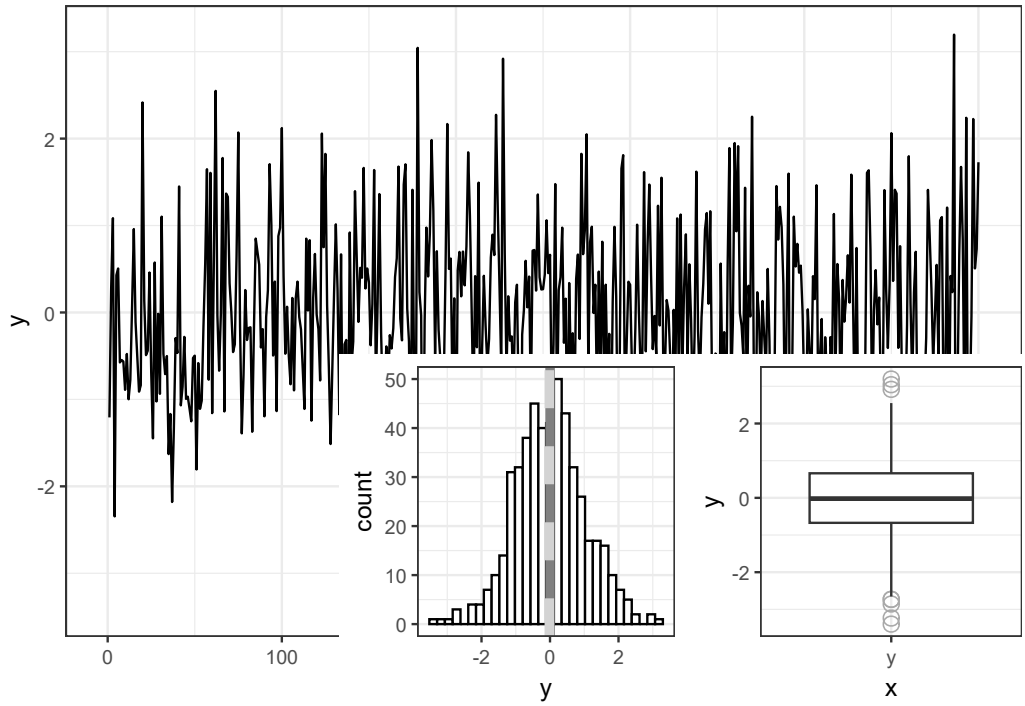
```
library(dplyr)
library(ggplot2)
library(magrittr)
library(plotly)
```

Następnie w celu replikacji wyników ustawiono ziarno algorytmu generowania liczb pseudolosowych.

```
set_seed()
```

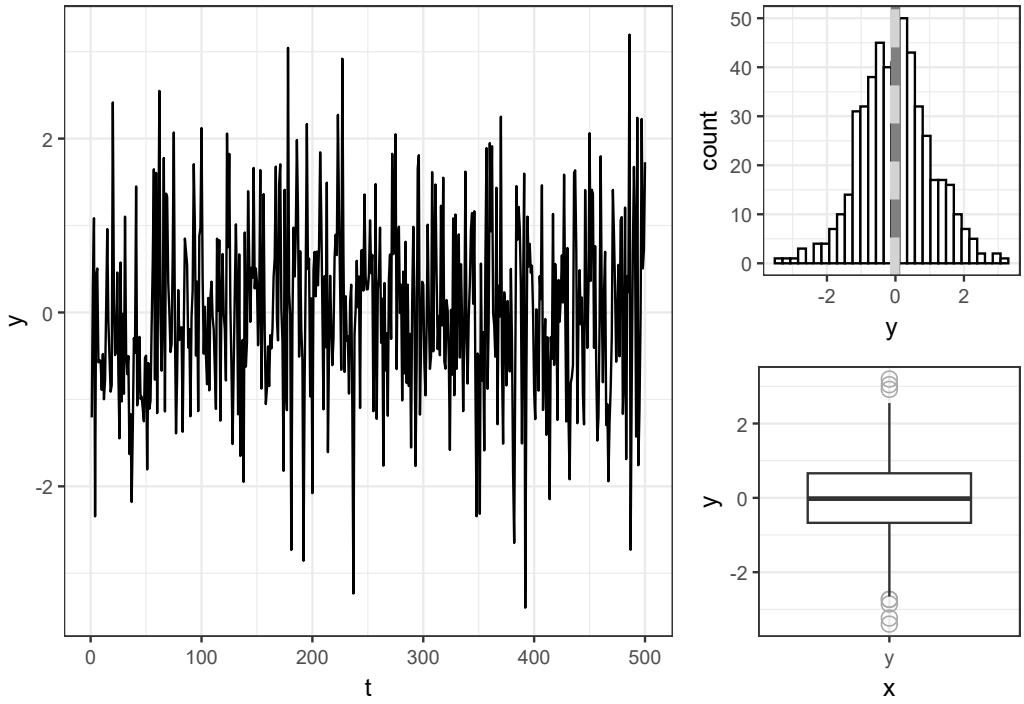
Dla celów ilustracji poleceń graficznych utworzono dwie ramki danych `df.1` i `df.2`.

```
df.1 <- lapply(
    1:10,
    function(x) {
        p <- runif(1, min = 0, max = 1)
        rbinom(100, 100, p)
    }
) %>%
```



Rysunek 2.24. Rysunek wykresów – przypadek 3

```
    do.call(cbind, .) %>%
    data.frame() %>%
    .[-c(1, 5)] %>%
    set_colnames(
        c(rep(paste0(toupper(letters[1:ncol(.)])))))
    )
df.2 <- lapply(
    1:10,
    function(x) {
        p <- runif(1, min = 0, max = 1)
        rbinom(100, 100, p)
    }
) %>%
    do.call(cbind, .) %>%
    data.frame() %>%
    .[-c(2, 7)] %>%
    set_colnames(
        c(rep(paste0(toupper(letters[1:ncol(.)])))))
    )
```



Rysunek 2.25. Rysunek wykresów – przypadek 4

Przy konstrukcji powyższych ramek zawierających dane statystyczne przyjęto następujące założenia:

- dane pochodzą z rozkładu dwumianowego o prawdopodobieństwie p w liczbie losowań $n = 100$ i przedstawiają liczbę k *sukcesów*,
- p jest zmienną losową, która pochodzi z rozkładu jednostajnego na przedziale $\langle 0, 1 \rangle$,
- liczba obserwacji wynosi 100.

Wygenerowano 10 szeregów i wyniki wpisano do listy za pomocą funkcji `lapply`. Następnie w przetwarzaniu potokowym na wszystkich elementach listy wykonano funkcję `cbind` stosując funkcję do `.call`. W ten sposób utworzona została ramka danych. Dla ilustracji operacji na ramce danych usunięto dwie kolumny z każdej ramki (kolumny 1 i 5 z ramki `df.1` oraz kolumny 2 i 7 z ramki `df.2`). Na koniec nadano nazwy poszczególnym szeregom (kolumnom), nazywanym następnie *klasami*, w postaci wielkich liter od A do H. Ramki mają następującą postać:

```
head(df.1)
```

```
##    A  B  C   D E  F  G  H
## 1 53 73 49 100 1 19 62 77
## 2 57 70 48  99 0 15 62 75
## 3 55 68 45 100 0 23 61 76
## 4 65 71 57  99 0 16 60 83
## 5 54 74 56 100 0 15 57 76
## 6 60 63 54  99 1 24 54 78
```

```
head(df.2)
```

```
##    A  B  C D  E  F  G H
## 1 29  4 60 5  4 64 71 6
## 2 25  7 62 6  7 73 71 2
## 3 37 16 70 2  5 78 73 2
## 4 29  9 57 2  8 69 79 3
## 5 37 14 50 3  5 67 76 4
## 6 28 12 51 5 12 73 74 7
```

Następnie przyjęto, że w ramkach `data.1` i `data.2` zostaną odpowiednio utworzone następujące kolumny:

1. Oznaczenie grupy zmiennych (`Grupa 1` i `Grupa 2`).
2. Nazwy wierszy (`rownames`). Łatwo sprawdzić, że nazwy te odpowiadają nazwom klas `A` do `H` (wcześniej powstaje ramka z wartościami przeciętnymi kolumn `data.frame(colMeans(df.1))`).
3. Średnie wartości z obserwacji w kolumnach otrzymane za pomocą funkcji `colMeans` (należy zwrócić uwagę na *wektoryzację* tego polecenia). Ostatecznie kolumnom ramek danych `data.1` i `data.2` zostały nadane nazwy i przyjęto porządkową numerację wierszy. Wszystkie polecenia zostały wykonane w przetwarzaniu potokowym.

```
data.1 <- data.frame(colMeans(df.1)) %>%
    cbind(
        "Grupa 1",
        rownames(.),
        .
    ) %>%
    set_colnames(c("Grupa", "Klasa",  "Wartość")) %>%
    set_rownames(1:nrow(.))
data.2 <- data.frame(colMeans(df.2)) %>%
    cbind(
        "Grupa 2",
        rownames(.),
        .
    ) %>%
    set_colnames(c("Grupa", "Klasa", "Wartość")) %>%
    set_rownames(1:nrow(.))
```

Zbiory danych `data.1` i `data.2` powstały w celu utworzenia wspólnego wykresu słupkowego wartości średnich. Do tego potrzebna jest jedna tablica danych, powstała poprzez połączenie dwóch ramek danych. Wykorzystano w tym celu funkcję `rbind`, gdyż ramki `data.1` i `data.2` mają tę samą konstrukcję, w szczególności takie same nazwy kolumn.

```
barplot <- rbind(data.1, data.2)
barplot[8:12, ]
```

```
##      Grupa Klasa Wartość
## 8  Grupa 1     H   76.10
## 9  Grupa 2     A   28.86
## 10 Grupa 2     B    9.09
## 11 Grupa 2     C   60.05
## 12 Grupa 2     D    4.69
```

Dla ilustracji pokazano wiersze od 8 do 12 ramki `barplot`. Teraz można już przystąpić do utworzenia wykresu słupkowego (rys. 2.26). Wykorzystana zostanie do tego celu ponownie biblioteka `ggplot2`. Poniższy kod ma następującą konstrukcję:

1. Jako zbiór danych wskazano ramkę `barplot`.
2. Do oznaczenia wartości na osiach (`aes`) przyjęto zmienne `Klasa` i `Wartość`, natomiast kolory słupków różnicuje zmienna `Grupa`.
3. Do wykresu dodano warstwy:

- `geom_bar` z argumentami `position = 'dodge'` (zachowanie szerokości słupków) i `stat = 'identity'` (przyjęcie wartości na osi y z kolumn zbioru danych),
- `scale_fill_manual` – kolorystyka wykresu, w której kolory zadano jako wartości w szesnastkowym systemie liczbowym (heksadecymalnym),
- `geom_hline` – dodanie do wykresu linii oznaczającej średnią arytmetyczną z wartości odpowiednio z `Grupa 1` i `Grupa 2`.

Ostatnie polecenia porządkują wykres poprzez nadanie tytułu (`ggtitle`) i nazw osi (`xlab`, `ylab`).

```
ggplot(
    barplot,
    aes(
        x = Klasa,
        y = Wartość,
        fill = Grupa
    )
) +
```

```
geom_bar(
    position = "dodge",
    stat = "identity"
) +
scale_fill_manual(
    values = c("#808080", "#D3D3D3")
) +
geom_hline(
    yintercept = mean(data.1[, "Wartość"]),
    linetype = "dashed",
    color = "#808080",
    size = 2
) +
geom_hline(
    yintercept = mean(data.2[, "Wartość"]),
    linetype = "dashed",
    color = "#D3D3D3",
    size = 2
) +
ggtitle(
    "Wykres słupkowy dla dwóch zbiorów danych"
) +
    xlab("Zmienna") +
    ylab("Wartość") +
    theme(
        plot.title = element_text(hjust = 0.5)
    ) +
  theme_bw() +
  theme(
    plot.title = element_text(hjust = 0.5)
  )
```

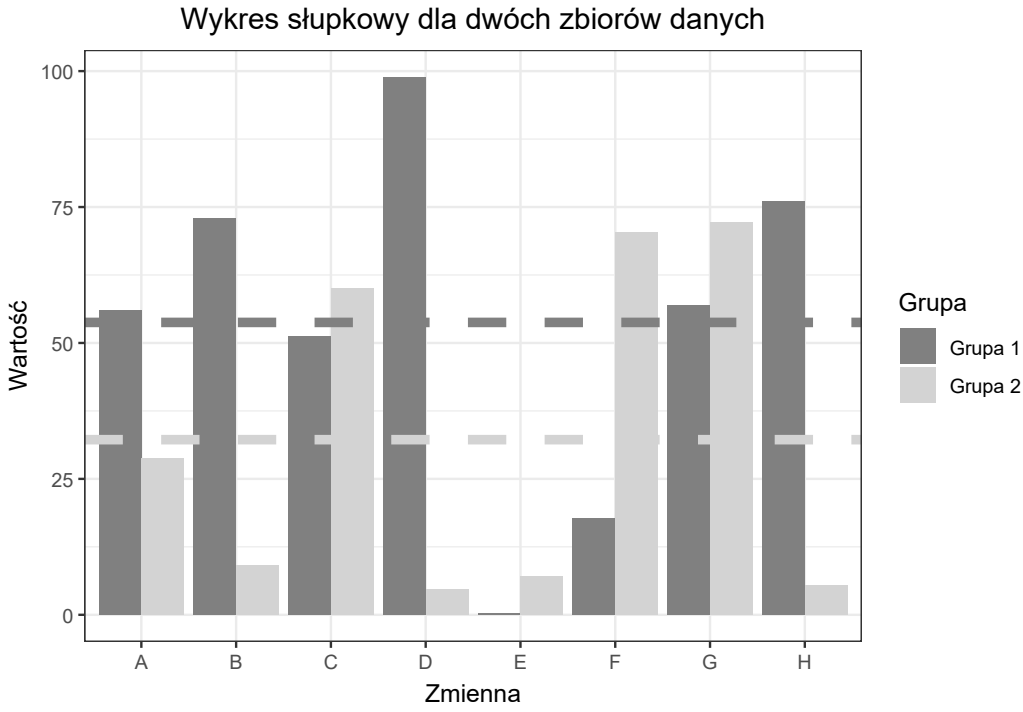
Na koniec utworzono rysunek wykresów pudełkowych na podstawie wartości w ramce `df.1`. Analogicznie można wykonać rysunek dla ramki `df.2`. Najpierw przekształcono ramkę danych za pomocą biblioteki `reshape2`. Podobnie jak wcześniej, należy wczytać tę bibliotekę.

```
library(reshape2)
```

Następnie zmodyfikowano ramkę danych `df.1` za pomocą funkcji `melt`.

```
df.1.melt <- melt(df.1)
head(df.1.melt)
```

```
##   variable value
## 1        A    53
```



Rysunek 2.26. Wykres słupkowy dla dwóch zbiorów danych – ggplot

```
## 2        A    57
## 3        A    55
## 4        A    65
## 5        A    54
## 6        A    60
```

Teraz można wykorzystać ramkę df.1.melt do utworzenia wykresu pudełkowego wszystkich zmiennych. W tym celu wykorzystano bibliotekę plotly. Ciąg poleceń zawiera poniższy kod dla funkcji plot_ly z następującymi parametrami:

- data – nazwa zbioru danych,
- y – kolumna wartości zmiennych ze zbioru danych df.1.melt (zagregowane wielkości z ramki df.1),
- type – box (wykres pudełkowy),
- color – dyskryminacja kolorów ze względu na nazwę zmiennej,
- colors – paleta kolorów z biblioteki RColorBrewer (Neuwirth, 2022),
- name – oznaczenie nazwy wykresu na osi X w postaci nazwy zmiennej,
- line – lista list zawierająca dodatkowe opcje, w tym przypadku kolor obramowania.

```
box.plot <- plot_ly(
    data = df.1.melt,
    y = ~value,
    type = "box",
    color = ~variable,
    colors = "Greys",
    name = ~variable,
    line = list(
        color = "black",
        width = 1
    )
)
```

Na koniec do rysunku `box.plot` dodano tytuł, stosując przetwarzanie potokowe[9].

```
box.plot <- box.plot %>%
    layout(
        title = "Wykres pudełkowy dla zbioru danych"
    )
```

```
orca(
  box.plot,
  file = "output/plots/boxplot_yjvvvu.png",
  width = 800,
  height = 600,
  scale = 3
)
```

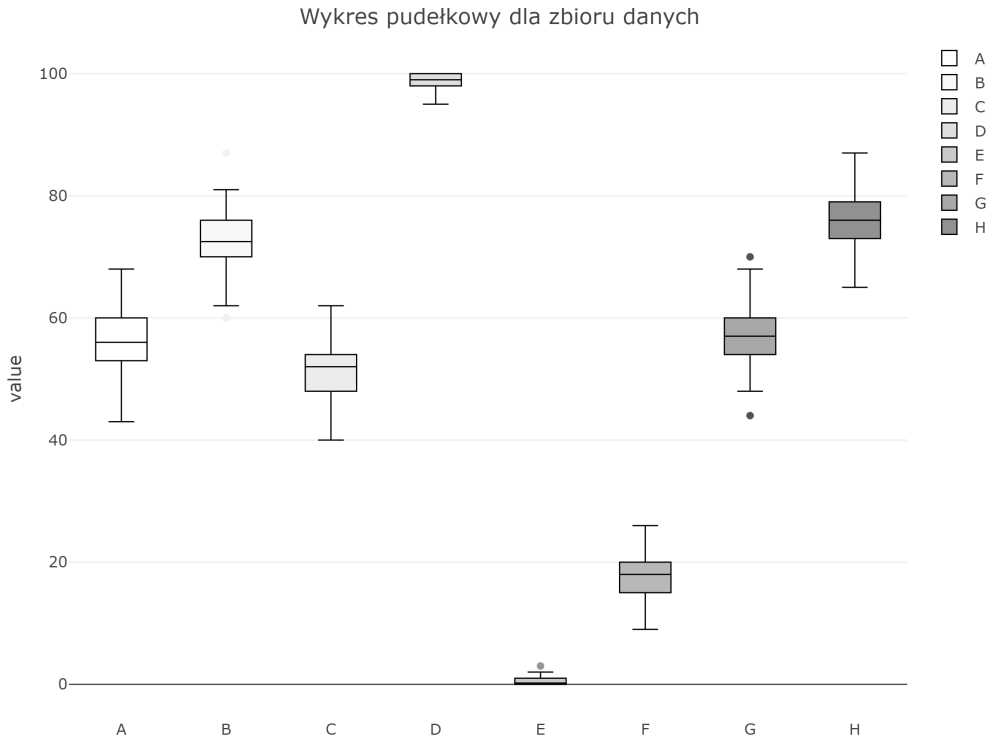
Na rys. 2.27 znajduje się wykres pudełkowy dla wartości kolumn z ramki danych `df.1.melt`.

```
knitr::include_graphics("output/plots/boxplot_yjvvvu.png")
```

Kolejny przykład graficzny (zrealizowany w formule przetwarzania potokowego) dotyczy utworzenia *chmury* słów zawartych w ramce danych. Taka technika może być stosowana do wizualizacji częstotliwości występowania fraz w różnego rodzaju sprawozdaniach. Można ją wykorzystać np. w raportach czy komunikatach, w celu wstępnej oceny zawartego w nich *sentymentu*. Wykorzystane zostaną dwie biblioteki:

- `tidytext` (Robinson i Silge, 2024),
- `wordcloud2` (Fellows, 2018).

[9] Wysokiej rozdzielczości wykres został zapisany do pliku za pomocą funkcji `orca`.



Rysunek 2.27. Wykres pudełkowy dla zbioru danych – `plotly`

Niech obiekt `df` zawiera tekst, którego słowa zostaną przedstawione na rysunku. Słowa pochodzą z tekstu komunikatu FOMC (*Federal Open Market Committee*), Komitetu Fed do spraw Operacji Otwartego Rynku, z dn. 3 listopada 2021 r. – *Press Release - Federal Reserve issues FOMC statement* w obszarze polityki pieniężnej w Stanach Zjednoczonych[10]. Ten fragment tekstu został zapisany w zbiorze `FOMC.txt`. Do wczytania tekstu zostanie wykorzystana biblioteka `readr` (Wickham i in., 2024b). Należy również wczytać biblioteki `magrittr` i `dplyr`.

```
library(dplyr)
library(magrittr)
library(readr)
```

Wczytanie tekstu odbywa się za pomocą funkcji `read_file`.

[10] Źródło: https://www.federalreserve.gov/newsevents/pressreleases/monetary20211103a.htm.

```
path <- "F:\\docs\\R\\b2\\ch2\\FOMC.txt"
df <- read_file(path)
```

Obiekt df (*string*) zawiera tekst komunikatu, którego pierwszych 50 znaków ma następującą postać:

```
stringr::str_trunc(df, 50, "right")
```

```
## [1] "The Federal Reserve is committed to using its f..."
```

Ciąg znaków df został oczyszczony ze znaków specjalnych, np. znaków przejścia do nowej linii (\r\n). Ponieważ ważne są wyłącznie słowa, z tekstu usunięto również znaki interpunkcyjne.

```
df <- df %>%
    gsub("\r\n", " ", .) %>%
    gsub("\\.", "", .) %>%
    gsub(",", "", .)
```

Następnie na podstawie ciągu df utworzono ramkę danych tibble z kolumną o nazwie text.

```
df <- df %>%
    tibble(text = .)
```

Ramkę df przekształcono do postaci, w której w kolumnie zamiast tekstu komunikatu znajdują się poszczególne jego słowa. Tę czynność można wykonać za pomocą funkcji z biblioteki tidytext. Najpierw trzeba wczytać tę bibliotekę.

```
library(tidytext)
```

Następnie stosuje się funkcję unnest_tokens realizującą podział tekstu na słowa.

```
df <- df %>%
    unnest_tokens(., words, text, token = "words")
```

Struktura tego polecenia oznacza, że argumentem funkcji unnest_tokens jest ramka df. W nowej ramce (również df) ma powstać kolumna words na podstawie kolumny text. Jednocześnie podział ma odbyć się dla słów (token = "words"). Ramka df po przekształceniu jest następująca:

```
df[1:20, ]
```

```
## # A tibble: 20 x 1
##    words
##    <chr>
##  1 the
##  2 federal
##  3 reserve
##  4 is
##  5 committed
##  6 to
##  7 using
##  8 its
##  9 full
## 10 range
## 11 of
## 12 tools
## 13 to
## 14 support
## 15 the
## 16 us
## 17 economy
## 18 in
## 19 this
## 20 challenging
```

Teraz można przystąpić do następnego etapu, tj. utworzenia rysunku *chmury* słów. Potrzebna będzie kolejna biblioteka – `wordcloud2`.

```
library(wordcloud2)
```

Po wczytaniu tej biblioteki należy utworzyć odpowiednią ramkę danych (np. `words`) na podstawie ramki `df`, która będzie podstawą utworzenia rysunku.

```
words <- df %>%
    count(words, sort = TRUE) %>%
    set_colnames(c("name", "value"))
words[1:10, ]
```

```
## # A tibble: 10 x 2
##    name      value
##    <chr>     <int>
##  1 the          47
##  2 of           25
##  3 to           25
```

```
##  4 and          24
##  5 in           12
##  6 committee    10
##  7 inflation    10
##  8 will          8
##  9 at            7
## 10 by            7
```

Utworzono nową ramkę danych `words`, zawierającą dwie kolumny: `name` (słowo) oraz `value` (liczba wystąpień słowa w tekście). Można zauważyć, że najliczniej występujące słowa nie mają wartości merytorycznej, zatem można je usunąć ze zbioru. Te słowa zostały umieszczone w wektorze znakowym `delete`. Następnie w ramce `words` wykonano operację usunięcia słów z tego wektora. W tym celu zastosowano funkcję `subset` do filtrowania ramki danych w taki sposób, aby uwzględnić tylko słowa z kolumny `name`, które nie znajdują się (negacja `!`) w zbiorze `delete`. Należy zwrócić uwagę na zastosowanie operatora `%in%`, który służy do sprawdzania, czy dany element (w tym wypadku w kolumnie `name`) należy do wektora.

```
delete <- c(
    "the",
    "in",
    "had",
    "of",
    "to",
    "a",
    "and"
)
words.1 <- words %>%
    subset(., !(name %in% delete)) %>%
  set_colnames(c("word", "freq")) %>%
  slice_head(n = 50)
words.1[1:10, ]
```

```
## # A tibble: 10 x 2
##    word        freq
##    <chr>      <int>
##  1 committee     10
##  2 inflation     10
##  3 will           8
##  4 at             7
##  5 by             7
##  6 for            7
##  7 percent        7
##  8 securities     7
##  9 2              6
## 10 billion        6
```

Ramka danych `words . 1` jest już przygotowana do wizualizacji słów. Aby móc wybrać kolory wykresu z dobranej palety kolorów można zastosować bibliotekę `RColor-Brewer`.

```
library(RColorBrewer)
```

Teraz można wykonać rysunek popularności słów.

```
wordcloud2(
    words.1,
    size = 1,
    color = brewer.pal(8, "Greys"),
    backgroundColor = "grey"
)
```

Rysunek 2.28. Wykres popularności słów – `wordcloud2`

Na rys. 2.28 zawarto najpopularniejsze słowa ze zbioru `words` . 1. W podany sposób można przeprowadzić bardziej złożoną analizę, z uwzględnieniem funkcji zawartych w bibliotece `tidytext`. Podobną analizę można również wykonać za pomocą biblioteki ggwordcloud (Le Pennec i Slowikowski, 2024), stanowiącej wsparcie dla biblioteki `ggplot2`.

Poniżej podano kolejne zastosowania biblioteki `ggplot2`. Funkcja `facet_wrap` służy do tworzenia wielu wykresów (*facets*) na podstawie jednej zmiennej. Każdy wykres pokazuje podzbiór danych. Przykłady podano na podstawie zbioru `mpg`, który pochodzi z testów przeprowadzonych przez Amerykańską Agencję Ochrony Środowiska (*Environmental Protection Agency*, EPA). Agencja ta co roku publikuje dane dotyczące:

- zużycia paliwa,
- emisji spalin,
- specyfikacji technicznej samochodów sprzedawanych na rynku USA.

W szczególności dane `mpg` pochodzą z testów laboratoryjnych, które mierzą spalanie w mieście i poza miastem, wyrażone w milach na galon (rys. 2.29).

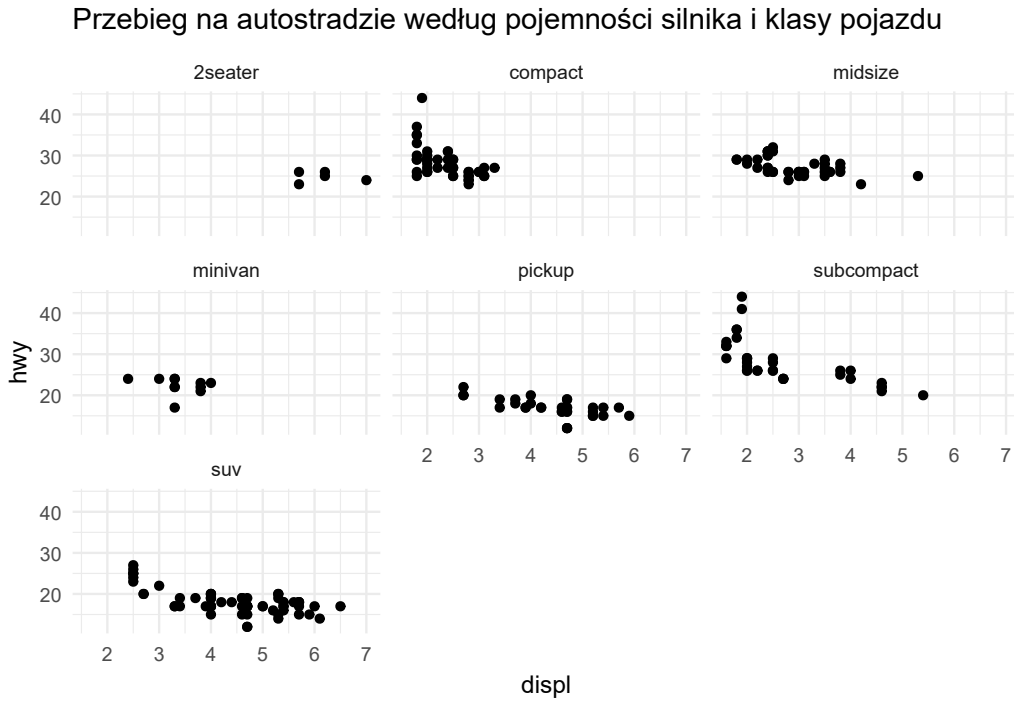
```
data(mpg)
```

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_wrap(~ class) +
  theme_minimal() +
  labs(
    title = "Przebieg na autostradzie według pojemności silnika
     ↪  i klasy pojazdu"
  )
```

Poniżej podano zastosowanie funkcji `facet_grid`, która służy do tworzenia siatki wykresów na podstawie dwóch zmiennych – jednej dla wierszy, drugiej dla kolumn. Oznaczenia osi X i Y są następujące:

- X (`x = displ`) – pojemność silnika w litrach,
- Y (`y = hwy`) – liczba mil przejechanych na galonie paliwa w warunkach drogowych (rys. 2.30).

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_grid(rows = vars(class), cols = vars(cyl)) +
  theme_minimal()
```



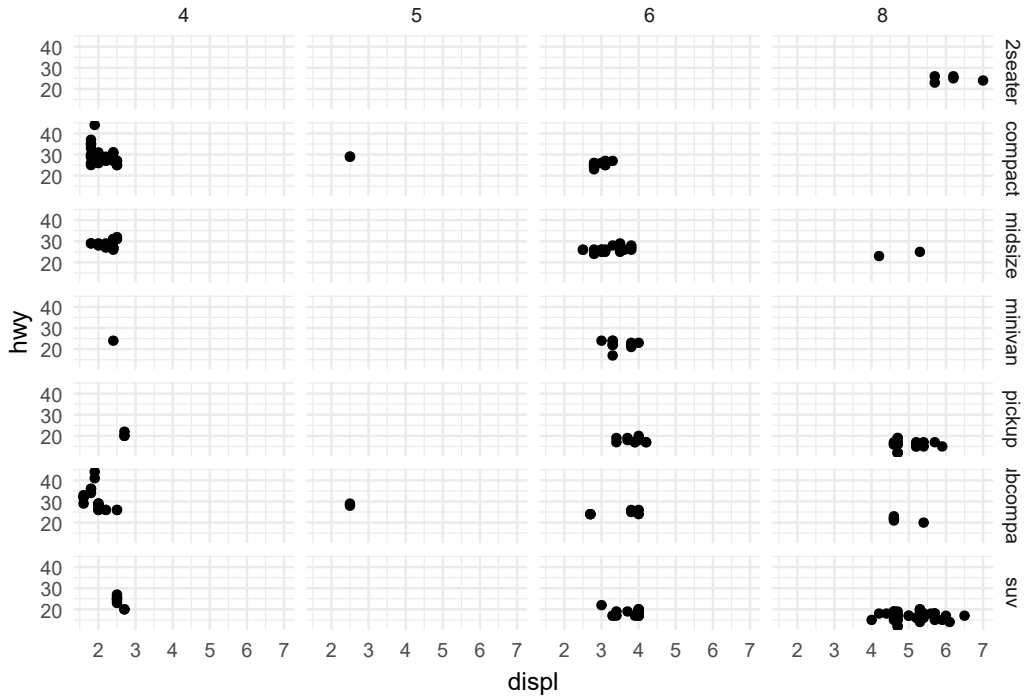
Rysunek 2.29. Przebieg na autostradzie według pojemności silnika i klasy pojazdu

Na rys. 2.31 dodano linię regresji (prostą dopasowaną za pomocą metody najmniejszych kwadratów, zob. rozdział 3) na podstawie `geom_smooth(method = "lm")`, bez wyświetlania przedziału ufności (`se = FALSE`). Wiersze odpowiadają klasie pojazdu (`class`), natomiast kolumny liczbie cylindrów (`cyl`). W rezultacie otrzymano siatkę, w której każdy wykres reprezentuje kombinację klasy i liczby cylindrów.

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE, color = "grey") +
  facet_grid(rows = vars(class), cols = vars(cyl)) +
  theme_minimal()
```

Na zakończenie warto podkreślić, że wizualizacja danych za pomocą wykresów stanowi skuteczne narzędzie wspomagające analizę statystyczną i ekonometryczną. Niemniej jednak, w wielu przypadkach niezbędne jest również uporządkowane i czytelne przedstawienie wyników w formie tabelarycznej, które pozwala na dokładną interpretację wartości liczbowych oraz porównanie różnych wyników.

W kolejnym podrozdziale zostanie omówione tablicowanie wyników. Przedstawione zostaną funkcje i biblioteki języka R – w szczególności `knitr` i `kableExtra` – które



Rysunek 2.30. Liczba mil przejechanych na galonie paliwa w warunkach drogowych

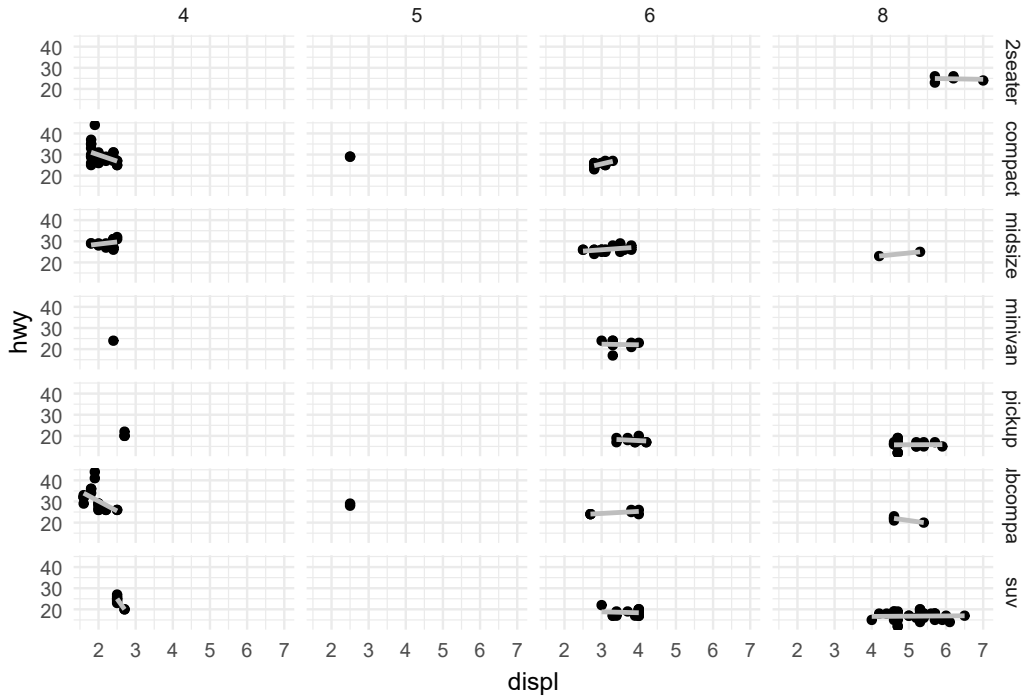
umożliwiają tworzenie estetycznych tablic w formatach HTML i LaTeX, przydatnych w raportach i publikacjach naukowych.

2.2.11. Tablicowanie wyników

Tablicowanie to ważny etap analizy danych, który pozwala na przejrzyste przedstawienie wyników obliczeń. Poprawnie sformatowane tablice ułatwiają interpretację wyników oraz formułowanie wniosków z analizy.

Biblioteka `kableExtra` rozszerza możliwości funkcji `kable` z biblioteki `knitr`, umożliwiając tworzenie rozbudowanych tablic w formatach HTML i LaTeX. Pozwala dodawać nagłówki, formatowanie, kolory, grupowanie wierszy i kolumn, a także stylizować wygląd tablicy. Jest to szczególnie przydatne w raportach R/Markdown (zob. podrozdział 2.5), gdzie czytelność i prezentacja danych mają kluczowe znaczenie. W celu zilustrowania poniższych przykładów wczytano odpowiednie biblioteki.

```
library(dplyr)
library(kableExtra)
library(magrittr)
```



Rysunek 2.31. Liczba mil przejechanych na galonie paliwa w warunkach drogowych z linią regresji

```
library(tibble)
```

Poniższy kod realizuje następujące zadania:

1. Najpierw wywoływana jest funkcja `set_seed`, której celem jest zapewnienie powtarzalności wyników generowanych losowo.
2. Zmienna k zostaje ustawiona na wartość 10, co określa liczbę kolumn w generowanej ramce danych.
3. Następnie generowana jest macierz zawierająca 10000 losowych elementów pochodzących z rozkładu normalnego standaryzowanego, mająca 10 kolumn. Macierz ta zostaje przekształcona w ramkę danych.
4. Kolumnom ramki danych przypisuje się nazwy `Var1`, `Var2` itd.
5. Tworzony jest nowy obiekt `stats`, który zawiera podsumowanie statystyczne każdej zmiennej: nazwę kolumny, średnią oraz odchylenie standardowe.
6. W końcowym kroku wyświetla się ramkę `stats` w formie estetycznej tablicy `HTML`, używając funkcji `kable` z biblioteki `knitr` i funkcji `kable_styling` z biblioteki `kableExtra` (Zhu, 2025) (tabl. 2.1).

```
set_seed()
k <- 10
data <- matrix(rnorm(10000), ncol = k) %>%
  as.data.frame() %>%
  set_colnames(paste("Var", 1:k, sep = ""))
stats <- data.frame(
  variable = colnames(data),
  mean = apply(data, 2, mean),
  sd = apply(data, 2, sd)
)
```

```
stats %>%
  kable(
    digits = 6,
    format = "latex",
    caption = "Stylizowana tablica -- funkcja \\texttt{kable}",
    label = "cuwbha"
  ) %>%
  kable_styling(latex_options = c("striped", "hold_position"))
```

Tablica 2.1. Stylizowana tablica – funkcja `kable`

	variable	mean	sd
Var1	Var1	-0.026597	0.997338
Var2	Var2	0.014508	0.981193
Var3	Var3	0.028951	1.012339
Var4	Var4	-0.009611	0.993759
Var5	Var5	-0.039025	0.971998
Var6	Var6	0.045499	0.972338
Var7	Var7	-0.006176	0.999194
Var8	Var8	0.046091	0.999562
Var9	Var9	0.029308	0.986164
Var10	Var10	-0.021789	0.960361

Kolejny przykład realizuje następujące zadania:

1. Na podstawie ramki `data` tworzona jest nowa ramka `stats`, zawierająca dla każdej kolumny:

- nazwę (`variable`),
- średnią (`mean`),
- odchylenie standardowe (`sd`),
- minimum (`min`),
- maksimum (`max`),

- medianę (`median`),
- rozstęp międzykwartylowy (`IQR`).

2. Tablica `stats` jest formatowana do postaci `LaTeX` przy użyciu funkcji `kable` z argumentem `booktabs = TRUE` oraz podpisem.
3. Dalsze formatowanie tablicy realizowane jest za pomocą funkcji z biblioteki `kableExtra`:

- dodanie stylu *paskowego* (`striped`) oraz dopasowanie szerokości tablicy do strony (`scale_down`),
- ustawienie rozmiaru czcionki na 10 i wyśrodkowanie tablicy (`position = "center"`),
- pogrubienie pierwszej kolumny (nazwy zmiennych),
- ustawienie szerokości kolumn 2–7 i ich tła (kolor #FFFFFF),
- wyróżnienie nagłówka tablicy (kolor #ebedef) i pogrubienie,
- ustawienie tła wierszy z danymi (#FFFFFF),
- obrót tablicy (funkcja `landscape`).

4. Nad kolumnami dodaje się nagłówek grupujący.
5. Na końcu do tablicy dodaje się przykładową stopkę (tabl. 2.2).

```
stats <- data %>%
  {data.frame(
    mean = apply(., 2, mean),
    sd = apply(., 2, sd),
    min = apply(., 2, min),
    max = apply(., 2, max),
    median = apply(., 2, median),
    IQR = apply(., 2, function(x) IQR(x))
  )} %>%
  round(4)
```

```
stats %>%
  kable(
    format = "latex",
    booktabs = TRUE,
    longtable = FALSE,
    linesep = "",
    caption = "Statystyki opisowe",
    label = "vallsb"
  ) %>%
  kable_styling(
    latex_options = c("striped", "scale_down"),
    font_size = 9,
    position = "center"
```

```
) %>%
column_spec(1, bold = TRUE) %>%
column_spec(
  2:7,
  width = "2.5cm",
  background = "#FFFFFF"
) %>%
row_spec(
  0,
  bold = TRUE,
  color = "black",
  background = "#ebedef"
) %>%
row_spec(
  1:10,
  color = "black",
  background = "#FFFFFF"
) %>%
add_header_above(c(" " = 1, "Statystyki opisowe" = 6)) %>%
footnote(
  general = "Przykładowe dane",
  general_title = "Uwagi:",
  footnote_as_chunk = TRUE
) %>%
landscape()
```

Tablica 2.2. Statystyki opisowe

	Statystyki opisowe					
	mean	**sd**	**min**	**max**	**median**	**IQR**
Var1	-0.0266	0.9973	-3.3961	3.1959	-0.0398	1.2891
Var2	0.0145	0.9812	-3.1216	3.1679	0.0136	1.2907
Var3	0.0290	1.0123	-3.0947	3.0225	0.0566	1.4134
Var4	-0.0096	0.9938	-3.0430	3.1199	0.0042	1.3522
Var5	-0.0390	0.9720	-2.8452	2.6986	-0.0590	1.3738
Var6	0.0455	0.9723	-2.8365	3.5608	0.0266	1.3337
Var7	-0.0062	0.9992	-3.2007	3.2950	-0.0106	1.3833
Var8	0.0461	0.9996	-3.1718	3.6181	0.0373	1.3255
Var9	0.0293	0.9862	-3.1857	3.3570	0.0069	1.3614
Var10	-0.0218	0.9604	-3.3351	2.7389	-0.0131	1.1422

Uwagi: Przykładowe dane

Struktura kodu i opracowanie wyników analiz wymagają również komunikacji z plikami zewnętrznymi. W następnym podrozdziale podano zarys operacji I/O na plikach. Umiejętność sprawnego operowania na plikach jest nieodzowna w codziennej pracy statystyka, ekonometryka czy analityka danych. Omówione zostaną podstawowe funkcje służące do odczytu i zapisu danych w popularnych formatach (CSV, TXT, XLSX) oraz zastosowanie nowoczesnych bibliotek, takich jak readr, readxl i writexl, które znacząco ułatwiają pracę z danymi.

2.2.12. Operacje I/O na plikach

Operacje wejścia/wyjścia (I/O) pozwalają na odczytywanie danych z plików i zapisywanie wyników do plików, w formatach takich jak CSV, TXT czy XLSX. Najczęściej wykorzystywane funkcje do odczytu danych to read.csv, read.table oraz funkcje z biblioteki readr, takie jak read_csv. Do zapisu danych służą m.in. funkcje write.csv czy write.table, a nowoczesne biblioteki, jak readr czy openxlsx, oferują wygodne narzędzia do pracy z różnymi formatami plików.

Operacje na plikach stanowią istotne zastosowanie języka R w archiwizowaniu baz danych statystycznych. Za pomocą różnorodnych funkcji można zarówno odczytywać zawartość plików i katalogów, jak i zapisywać pliki na dysku. W tym celu pomocne będą trzy biblioteki: readxl, writexl (Ooms, 2025b) i readr.

```
library(magrittr)
library(readr)
library(readxl)
library(writexl)
```

Na początku analizy ustawiony został katalog roboczy.

```
path <- "F:\\docs\\R\\b2\\ch2"
```

Dzięki funkcji list.files z biblioteki base można odczytać zawartość katalogu i jego podkatalogów.

```
file <- list.files(
    paste0(path, "/data1/"),
    pattern = "\\.txt$",
    recursive = TRUE
) %>%
  data.frame() %>%
  set_colnames(c("files"))
file
```

```
##             files
## 1       dat_1.txt
## 2       dat_2.txt
## 3       dat_3.txt
## 4 data2/dat_4.txt
```

Za pomocą innej funkcji biblioteki `base`, tj. `file.exists`, należącej do zbioru funkcji `files`, można sprawdzić obecność pliku w katalogu.

```
data.file <- file.exists(
  paste0(path, "/data1/data2/dat_4.txt")
)
data.file
```

```
## [1] TRUE
```

Ponieważ funkcja `file.exists` zwraca wartość logiczną, to można ją wykorzystać w warunku logicznym `if`. Jeśli istnieje plik `dat_4.txt`, to zostanie wyświetlona jego zawartość. W tym celu użyto funkcji `read_file` z biblioteki `readr`.

```
if (!data.file) {
  source(paste0(path, "/data1/dat.R"))
} else {
  read_file(paste0(path, "/data1/data2/dat_4.txt"))
}
```

```
## [1] "Plik z danymi dat_4.txt"
```

W przeciwnym razie zostanie wczytany zbiór `dat.R`.

```
source(paste0(path, "/data1/dat.R"))
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

W poniższych przykładach wykorzystano funkcje bibliotek zewnętrznych. Dla ilustracji utworzono prostą ramkę danych `df.1`.

```
df.1 <- data.frame(x = 1:5, y = 1)
df.1
```

```
##   x y
## 1 1 1
## 2 2 1
## 3 3 1
## 4 4 1
## 5 5 1
```

Jej zawartość została następnie zapisana do pliku `dat.xlsx` za pomocą funkcji `write_xlsx` z biblioteki `writexl`.

```
write_xlsx(
    df.1,
    path = paste0(path, "/data3/", "dat.xlsx"),
    col_names = TRUE,
    format_headers = TRUE
)
```

W kolejnym kroku zawartość pliku `dat.xlsx` została odczytana za pomocą funkcji `read_excel` z biblioteki `readxl`.

```
df.2 <- read_excel(paste0(path, "/data3/", "dat.xlsx"))
df.2
```

```
## # A tibble: 5 x 2
##       x     y
##   <dbl> <dbl>
## 1     1     1
## 2     2     1
## 3     3     1
## 4     4     1
## 5     5     1
```

Zapis obiektów R – głównie w postaci ramek danych – do zbiorów `xls` lub `xlsx` nie jest jedynym sposobem zapisywania baz danych. Można w tym celu zastosować również wbudowany standard języka R w postaci zbiorów `RData`. Jest to wygodny sposób zapisu, jeśli bazy danych nie są współdzielone przez inne aplikacje. Dla celów ilustracyjnych utworzono elementarną listę.

```
x.1 <- lapply(
  1:5,
  function(x) x
)
x.1
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 3
```

```
##
## [[4]]
## [1] 4
##
## [[5]]
## [1] 5
```

Następnie obiekt `x.1` został zapisany w zbiorze `dat.RData` za pomocą funkcji `save`.

```
save(x.1, file = paste0(path, "/data3/dat.RData"))
```

W kolejnym kroku zbiór `dat.RData` został wczytany za pomocą funkcji `load`.

```
data.file <- file.exists(paste0(path, "/data3/dat.RData"))
if(data.file) {
  load(file = paste0(path, "/data3/dat.RData"))
}
x.2 <- x.1
```

Ostatecznie sprawdzono, czy obiekty `x.1` i `x.2`, wynikające z podanej manipulacji na plikach, są identyczne.

```
identical(x.1, x.2)
```

```
## [1] TRUE
```

W następnym przykładzie podano sposób zapisu ramki danych do pliku tekstowego za pomocą funkcji `write.table` z biblioteki systemowej `utils` (R Core Team, 2025). Dla ilustracji utworzono ramkę danych `df.1`.

```
df.1 <- data.frame(x = letters[1:3], y = LETTERS[1:3])
df.1
```

```
##   x y
## 1 a A
## 2 b B
## 3 c C
```

Ramka `df.1` została zapisana do pliku tekstowego `letters.txt` w postaci tablicy.

```
write.table(
  df.1,
  paste0(path, "/data3/letters.txt"),
  append = FALSE,
  sep = " ",
  dec = ".",
  row.names = FALSE,
  col.names = FALSE
)
```

Odczytanie tego zbioru jest możliwe dzięki podanej wcześniej funkcji `read_file`.

```
df.2 <- read_file(paste0(path, "/data3/letters.txt"))
```

Zmienna `df.2` zawiera znaki sterujące, gdyż ramka danych `df.1` ma postać tablicy. Taki sposób zapisu nie jest szczególnie wygodny i wymaga dalszego uporządkowania. Tę czynność pozostawiono Czytelnikowi do samodzielnego wykonania[11].

```
read.table(
  paste0(path, "/data3/letters.txt"),
  header = FALSE,
  sep = "",
  dec = "."
) %>%
  set_colnames(c("x", "y"))
```

W następnym podrozdziale zostaną przedstawione sposoby automatycznego gromadzenia danych statystycznych za pomocą języka R.

2.3. Organizacja zbiorów danych statystycznych

W tym rozdziale zostaną omówione mechanizmy wykorzystania interfejsów API (*application programming interface*) oraz metody bezpośredniego pobierania danych z witryn internetowych.

[11] Należy również zwrócić uwagę, że przekazanie obiektu `df.2` do środowiska `LaTeX verbatim` może powodować błędy przy kompilacji dokumentu do formatu PDF. Bowiem `LaTeX` nie radzi sobie dobrze z niektórymi znakami specjalnymi. Sposobem na rozwiązanie problemu jest odczytanie zawartości zbioru `letters.txt` za pomocą funkcji `read.table`.

2.3.1. Interfejs API

API jest interfejsem programowania aplikacji, czyli zbiorem reguł i metod pozwalających na komunikację między aplikacjami komputerowymi przez Internet. Komunikują się one za pomocą wspólnego języka (Jacobson i in., 2012). Interfejs API ma postać oprogramowania zainstalowanego na serwerze sieciowym. Dzięki temu oprogramowaniu dostawca danych udostępnia użytkownikom swoje zasoby. Użytkownik uzyskuje dostęp do wspomnianych zasobów poprzez wykorzystanie własnego oprogramowania, które komunikuje się z oprogramowaniem dostawcy. W komunikacji tej pośredniczy interfejs API. Zasady interfejsu API sprowadzają się do określenia funkcji zwracających wartości w odpowiedzi na zapytanie użytkownika oraz sposobu dostępu i uwierzytelniania użytkownika. Warto zwrócić uwagę na następujące aspekty interfejsów API (Jacobson i in., 2012):

- dostawca określa funkcjonalność interfejsu,
- dostawca może wprowadzać restrykcje w zakresie korzystania z interfejsu, takie jak limity dostępu do aplikacji i uwierzytelniania użytkowników na podstawie klucza dostępu,
- użytkownicy mogą korzystać z interfejsu tylko według zasad określonych przez dostawcę.

Występuje wiele interfejsów API, które są przydatne w pracy analityków danych. Warto zwrócić uwagę na interfejsy dostarczane przez następujące organizacje:

- ECB (Europejski Bank Centralny)[12],
- Eurostat[13],
- Facebook[14],
- FRED (*Federal Reserve Economic Data*, *Federal Reserve Bank of St. Louis*)[15],
- Google[16].

W poniższej analizie wykorzystane zostaną interfejsy ECB, Eurostat oraz FRED.

ECB

Wszystkie oficjalne dane statystyczne Europejskiego Banku Centralnego są dostępne w serwisie internetowym ECB SDW (*Statistical Data Warehouse*). W celu automatycznego pobrania danych, ECB udostępnił interfejs API pod adresem:

- https://data.ecb.europa.eu/help/api/overview/.

[12] Por. https://sdw-wsrest.ecb.europa.eu/help/.
[13] Por. https://ec.europa.eu/eurostat/web/user-guides/data-browser/api-data-access/api-getting-started.
[14] Por. https://developers.facebook.com/.
[15] Por. https://fred.stlouisfed.org/docs/api/fred/.
[16] Por. https://developers.google.com/.

Występuje również biblioteka pod nazwą `ecb` (Persson, 2025). W poniższym przykładzie pokazano dwa podejścia:

1. Pobranie danych poprzez `API`.
2. Zastosowanie biblioteki `ecb`.

Dodatkowo wykorzystano bibliotekę `stringr` do przetwarzania ciągów znaków (Wickham, 2023c). Warto również wspomnieć o innej bibliotece, przydatnej przy pracy na danych tekstowych, jaką jest `stringi` (Gagolewski, 2022). Kod rozpoczęto od wczytania potrzebnych bibliotek.

```
library(dplyr)
library(ecb)
library(ggplot2)
library(gridExtra)
library(httr)
library(magrittr)
library(readxl)
library(stringr)
```

Następnie trzeba zdefiniować niezbędne zmienne znakowe.

```
ecb.address <- "https://data-api.ecb.europa.eu/service/data/"
ecb.flow <- "EXR/"
ecb.var <- "D.CHF+GBP+PLN+USD.EUR.SP00.A"
ecb.st <- "2017-01-01"
ecb.en <- "2021-09-06"
format <- "text/csv"
```

Znaczenie zmiennych jest następujące:

- `ecb.address` – adres interfejsu `API` ECB,
- `ecb.flow` – określenie zasobu danych: *EXR: Exchange Rates* (kursy walutowe)[17],
- `ecb.var` – identyfikator zmiennej z bazy danych ECB SDW; w tym przypadku jest to wskazanie (za pomocą symbolu +) kursów dziennych dla czterech walut (CHF, GBP, PLN, USD) względem EUR,
- `ecb.st` i `ecb.en` – data początku i końca próby statystycznej,
- `format` – format danych (`CSV`).

Na podstawie podanych zmiennych utworzony został adres zapytania do bazy danych ECB.

[17] Por. https://data.ecb.europa.eu/data/datasets/EXR.

```
ecb.request <- paste0(
  ecb.address,
  ecb.flow,
  ecb.var,
  "?startPeriod=", ecb.st,
  "&endPeriod=", ecb.en
)
```

```
stringr::str_trunc(ecb.request, 50, "right")
```

```
## [1] "https://data-api.ecb.europa.eu/service/data/EXR..."
```

W kolejnym kroku sprawdzono, czy odpowiedź serwera na wykonane zapytanie jest poprawna, tj. czy kod statusu odpowiedzi to 200[18]. W celu weryfikacji tego statusu wykorzystano funkcję `GET` z biblioteki `httr`. Jeśli warunek funkcji `if` jest spełniony, to wykonywane jest zapytanie i porządkowana jest wartość funkcji.

```
if (GET(ecb.request)$status_code == 200) {
  # emoygd
}
```

Zadaniem powyższego kodu `emoygd` jest:

- wykonanie zapytania do bazy danych ECB,
- odczytanie zawartości CSV i jej przetworzenie,
- utworzenie rysunku wykresów dla czterech walut.

W kodzie `emoygd` zastosowano przetwarzanie potokowe. Najpierw za pomocą biblioteki `httr` (Wickham, 2023d) zostało przesłane żądanie `GET` do serwera bazy danych ze wskazaniem nagłówka `HTTP` (*Accept*), który określa format danych `CSV` jako zwróconą wartość funkcji.

```
ecb.response <- ecb.request %>%
  GET(
    .,
    add_headers(
      Accept = format
    )
  )
```

[18] Więcej informacji na temat kodów `HTTP` można znaleźć w dokumencie https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html.

W zapytaniu można przekazać również inny format danych, np. JSON (*JavaScript Object Notation*)[19].

```
format <- "application/vnd.sdmx.data+json;version=1.0.0-wd"
```

Następnie z obiektu ecb.request pobrano zawartość tekstową w formacie CSV. Na podstawie tej informacji (zawartej w wektorze znakowym) została utworzona tablica (ramka danych) csv.

```
csv <- ecb.response %>%
  httr::content(
    .,
    "text",
    encoding = "UTF-8"
  ) %>%
  read.csv(
    text = .,
    sep =",",
    header = TRUE,
    stringsAsFactors = FALSE
  )
```

W kolejnym kroku został zdefiniowany wektor znakowy zawierający oznaczenia walut.

```
er <- c("CHF", "GBP", "PLN", "USD")
```

Za pomocą poniższego kodu utworzono listę df ramek danych na podstawie ramki csv względem poszczególnych walut er. W ramce df uwzględnione zostały wybrane kolumny.

```
df <- lapply(
  er,
  function(x) {
    subset(csv, CURRENCY == x) %>%
    { data.frame(
      key.1 = .$TITLE_COMPL,
      key.2 = .$KEY,
      c.1   = .$CURRENCY,
      c.2   = .$CURRENCY_DENOM,
      date  = as.Date(.$TIME_PERIOD),
      obs   = as.numeric(.$OBS_VALUE)
    ) }
```

[19] Por. https://www.json.org/json-pl.html.

```
  }
)
names(df) <- er
```

Następnie przypisano identyfikatorom walut odpowiednie obiekty.

```
for (i in er) {
  assign(i, df[[i]])
}
```

Liczba obserwacji dla poszczególnych kursów walutowych wynosi 1196. Wybrana część tablicy z danymi statystycznymi dla kursu CHF/EUR jest następująca:

```
head(CHF[, -c(1, 2)])
```

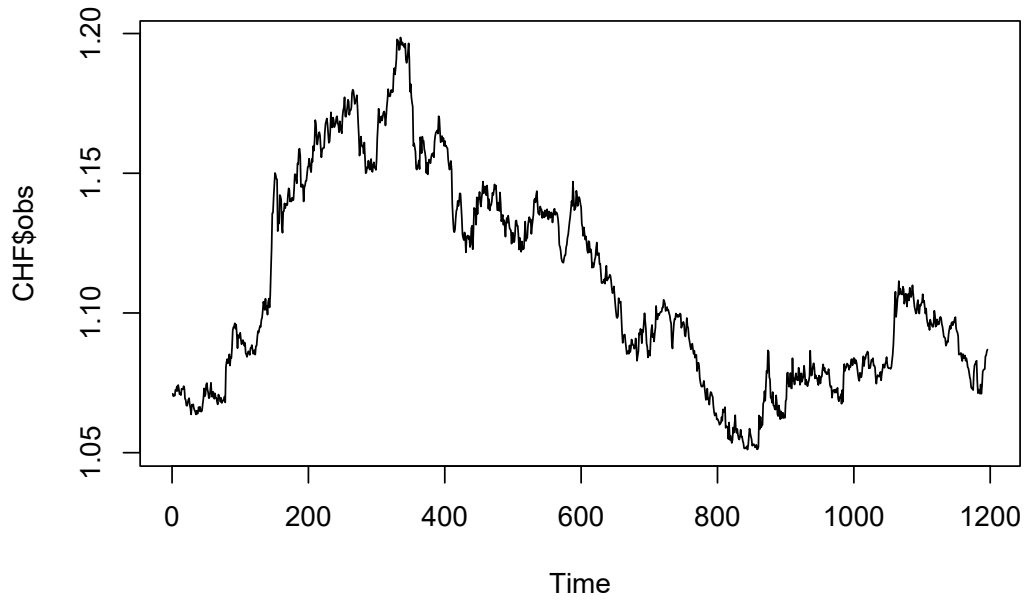
```
##   c.1 c.2       date    obs
## 1 CHF EUR 2017-01-02 1.0711
## 2 CHF EUR 2017-01-03 1.0702
## 3 CHF EUR 2017-01-04 1.0707
## 4 CHF EUR 2017-01-05 1.0704
## 5 CHF EUR 2017-01-06 1.0725
## 6 CHF EUR 2017-01-09 1.0721
```

Następnie utworzono wykres kursu CHF/EUR (rys. 2.32).

```
ts.plot(CHF$obs)
```

W kolejnym kroku zostały utworzone indywidualne wykresy `ggplot` wszystkich zmiennych, uwzględniające tytuły zawarte w funkcji `titles`, w której wykorzystano pola listy `df`.

```
titles <- function(x) {
  title <- x$key.1[1] %>% str_replace_all(., ", ", "\n")
  subtitle <- x$key.2[1]
  caption <- ""
  return(list(
    title = title,
    subtitle = subtitle,
    caption = caption)
  )
}
p <- function(x) {
  ggplot(x, aes(date, obs)) +
```



Rysunek 2.32. Kurs CHF/EUR – ECB SDW

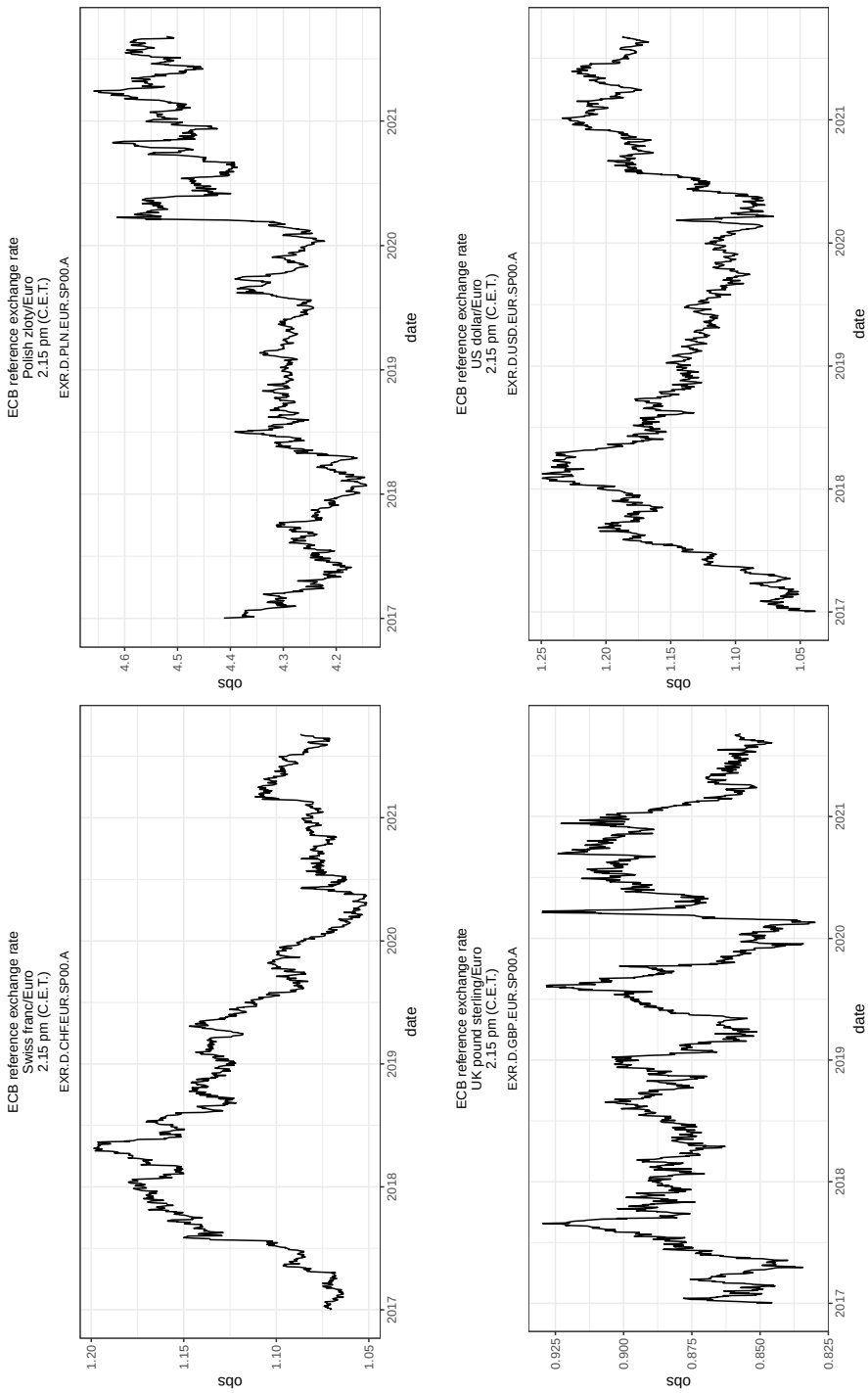
```
    geom_line() +
    labs(
      title = titles(x)$title,
      subtitle = titles(x)$subtitle,
      caption = titles(x)$caption
    ) +
    theme_bw() +
    theme(
      plot.title = element_text(
        hjust = 0.5,
        size = 10
      ),
      plot.subtitle = element_text(
        hjust = 0.5,
        size = 9
      ),
      plot.caption = element_text(
        hjust = 0.5,
        size = 9
      )
    )
}
```

Poszczególne wykresy zostały umieszczone na liście `plot`.

```
plot <- lapply(
  list(CHF, GBP, PLN, USD),
  function(x) {
    p(x)
  }
)
```

Na podstawie listy `plot` został sporządzony wspólny rysunek 2.33 (*landscape*) dla wszystkich wykresów za pomocą funkcji `marrangeGrob` z biblioteki `gridExtra`. Do funkcji należy przekazać listę wykresów jako parametr.

```
marrangeGrob(
  plot,
  top = NULL,
  nrow = 2,
  ncol = 2
)
```



Rysunek 2.33. Rysunek wykresów CHF, GBP, PLN, USD względem EUR – ECB SDW

Na tym zakończono kod emoygd. W kolejnym kroku wykorzystano bibliotekę ecb. Najpierw należało ustalić identyfikator wybranej zmiennej.

```
er.id <- CHF[1, 2]
er.id
```

```
## [1] "EXR.D.CHF.EUR.SP00.A"
```

Zastosowanie biblioteki ecb jest bardzo proste. Do funkcji get_data trzeba przekazać identyfikator zmiennej. Dodatkowe parametry można przekazać do funkcji w postaci listy filter. Parametry zostały zdefiniowane wcześniej. Dla uproszczenia pobrano wyłącznie dane dla kursu CHF/EUR.

```
chfeur <- get_data(
  er.id,
  filter = list(
    startPeriod = ecb.st,
    endPeriod = ecb.en,
    detail = "dataonly"
  )
)
head(chfeur[, c(2, 3, 6, 7)])
```

```
## # A tibble: 6 x 4
##   currency currency_denom obstime    obsvalue
##   <chr>    <chr>          <chr>         <dbl>
## 1 CHF      EUR            2017-01-02     1.07
## 2 CHF      EUR            2017-01-03     1.07
## 3 CHF      EUR            2017-01-04     1.07
## 4 CHF      EUR            2017-01-05     1.07
## 5 CHF      EUR            2017-01-06     1.07
## 6 CHF      EUR            2017-01-09     1.07
```

Funkcja get_data zwraca ramkę danych, dlatego utworzenie wykresu powinno odbyć się przy wykorzystaniu wcześniej zdefiniowanej funkcji p – co pozostawia się Czytelnikowi do samodzielnego wykonania.

Eurostat

W tej części omówione zostaną podstawowe funkcje biblioteki eurostat (Lahti i in., 2023), która pozwala na pobranie danych z bazy danych Eurostat[20]. Biblioteka eurostat korzysta z interfejsu API[21] oraz z funkcji pobierania zbiorczego (*bulk*

[20] Por. https://ec.europa.eu/eurostat.
[21] Por. https://ec.europa.eu/eurostat/api/.

download facility)[22]. Wybór źródła jest zależny od przyjętych parametrów funkcji biblioteki[23]. Na początku rozważań wczytano potrzebne zasoby.

```
library(dplyr)
library(eurostat)
library(ggplot2)
library(gridExtra)
library(httr)
library(jsonlite)
library(lubridate)
library(magrittr)
library(readxl)
library(rlist)
library(stringi)
library(stringr)
library(tidyr)
library(writexl)
library(xts)
```

Poniższy kod realizuje następujące zadania:

1. Ustawia zapisywanie danych lokalnie (`cache`).
2. Funkcja `get_eurostat_toc` pobiera spis dostępnych zbiorów danych z bazy Eurostat.

```
options(eurostat_cache = TRUE)
datasets <- get_eurostat_toc()
```

Następnie kod sprawdza, czy w tablicy `datasets` istnieje zbiór danych o kodzie `prc_hicp_midx` (inflacja HICP, *harmonized index of consumer prices*).

```
code.check <- "prc_hicp_midx" %>%
  grepl(datasets$code) %>%
  any
code.check
```

```
## [1] TRUE
```

Niektóre zbiory zostaną wczytane z odpowiedniego katalogu i dlatego należy ustawić do niego ścieżkę dostępu.

[22] Por. https://ec.europa.eu/eurostat/databrowser/bulk?lang=en.
[23] Por. https://ropengov.github.io/eurostat/.

```
path <- "F:\\docs\\R\\b2\\ch2\\estat\\"
```

Część kodu została przesunięta do zewnętrznych plików R, gdyż definicja pojedynczej zmiennej może być obszerna, a takich zmiennych może być wiele. Poniższy kod realizuje zadanie pobrania listy nazw plików z określonej lokalizacji (path), o określonym wzorcu nazwy (pattern) i z uwzględnieniem podkatalogów (recursive = TRUE)[24].

```
file.0 <- list.files(
  path,
  pattern = "\\.R$",
  recursive = TRUE
)
file.0
```

```
## [1] "prc_hicp_midx_db.R"
```

W pliku *prc_hicp_midx_db.R* znajduje się definicja listy db, która zawiera nazwy pól z bazy Eurostat i inne potrzebne informacje. Definicja listy db jest następująca:

```
db <- list(
    code = "prc_hicp_midx",
    label = "HICP - monthly data (index)",
    coicop = c(
        "All-items HICP",
        "Food",
        "Bread",
        "Meat",
        "Pork",
        "Eggs",
        "Butter",
        "Diesel",
        "Petrol",
        "Energy"
    ),
    unit = c(
        "Index, 2015=100"
    ),
    market = "(Prices)"
)
```

Jako zmienną code przyjęto miesięczny zharmonizowany indeks cen towarów i usług konsumpcyjnych *prc_hicp_midx*. Informacje dotyczące tej zmiennej znajdują się na stronie internetowej Eurostat:

[24] Czytelnik powinien najpierw utworzyć odpowiednie zbiory tekstowe na podstawie podanego kodu i umieścić je w katalogu dyskowym.

- https://ec.europa.eu/eurostat/databrowser/view/prc_hicp_midx/default/table?lang=en.

Łatwo sprawdzić, że etykieta `label` dla tej zmiennej to: *HICP - monthly data (index)*. W bazie danych dla podanej zmiennej występują pola:

- `coicop` – nazwa kategorii cen,
- `geo` – obszar geograficzny (Polska),
- `time` – przedział czasu (cała dostępna próba),
- `unit` – podstawa indeksu cen (Index, 2015=100).

Wektor `coicop` może zawierać nazwy wszystkich kategorii cen dla kodu *prc_hicp_midx*. W przykładzie podano wybrane kategorie. Element `market` listy db jest dodatkową informacją i nie należy do oznaczeń Eurostatu.

Definicję listy db należało wczytać z zewnętrznego zbioru za pomocą funkcji `source`.

```
source(paste0(path, file.0))
```

Utworzono pomocniczo trzy zmienne znakowe `s.1`, `s.2` i `s.3`.

```
s.1 <- c("TIME_PERIOD", "values")
s.2 <- c("coicop", "unit", "TIME_PERIOD", "values")
s.3 <- s.2[!s.2 %in% s.1]
s.1
```

```
## [1] "TIME_PERIOD" "values"
```

```
s.2
```

```
## [1] "coicop"      "unit"        "TIME_PERIOD" "values"
```

```
s.3
```

```
## [1] "coicop" "unit"
```

Powyższe zmienne posłużyły do utworzenia warunku `cond` w postaci ciągu znaków, na podstawie którego realizowane jest filtrowanie ramki danych. Listę oczyszczono (warunek `is.null`) za pomocą funkcji `list.clean` z biblioteki `rlist` (Ren, 2021).

```
cond <- lapply(
  s.2,
  function(x)
    if (!x %in% s.1) paste0(x, " %in% db$", x)
    else NULL
) %>%
  list.clean(., function(x) is.null(x), TRUE) %>%
  paste(., collapse = " & ")
cond
```

```
## [1] "coicop %in% db$coicop & unit %in% db$unit"
```

W poniższym kodzie pobrano do ramki danych `dat.0` wartości i etykiety dla zmiennej *prc_hicp_midx* z bazy Eurostat za pomocą funkcji `get_eurostat` z biblioteki `eurostat`.

```
dat.0 <- get_eurostat(
  db$code,
  time_format = "raw"
) %>%
  label_eurostat(., fix_duplicated = TRUE)
```

Zbiór `dat.0` jest następujący:

```
head(dat.0[1:3, -1])
```

```
## # A tibble: 3 x 5
##   unit            coicop         geo     TIME_PERIOD values
##   <chr>           <chr>          <chr>   <chr>        <dbl>
## 1 Index, 2005=100 All-items HICP Austria 1996-01       86.7
## 2 Index, 2005=100 All-items HICP Austria 1996-02       86.9
## 3 Index, 2005=100 All-items HICP Austria 1996-03       87.2
```

Czytelnik może sprawdzić strukturę ramki `dat.0` za pomocą polecenia `str(dat.0)`. Jak widać, ramka `dat.0` jest dużą tablicą. Do jej kolumn zastosowano filtr `cond`, który został przekazany do funkcji `subset` za pomocą kombinacji funkcji `parse` i `eval`. Funkcja `parse` przekształciła ciąg znaków w wyrażenie, które następnie zostało wykonane za pomocą funkcji `eval`. Dodatkowo ramka `dat.0` zawiera wyłącznie dane dla Polski i obejmuje kolumny zdefiniowane przez wektor `s.2`.

```
dat.1 <- dat.0 %>%
  subset(.,
    eval(parse(text = cond))
```

```
  ) %>%
    {
      if (any(colnames(.) == "geo")) {
          . <- subset(., geo == "Poland")
      } else {
          .
      }
    } %>%
      .[, s.2] %>%
      data.frame(., stringsAsFactors = FALSE)
head(dat.1)
```

```
##           coicop            unit TIME_PERIOD values
## 1 All-items HICP Index, 2015=100     1996-01   43.0
## 2 All-items HICP Index, 2015=100     1996-02   43.6
## 3 All-items HICP Index, 2015=100     1996-03   44.3
## 4 All-items HICP Index, 2015=100     1996-04   45.2
## 5 All-items HICP Index, 2015=100     1996-05   45.8
## 6 All-items HICP Index, 2015=100     1996-06   46.2
```

```
str(dat.1)
```

```
## 'data.frame':    2228 obs. of  4 variables:
##  $ coicop     : chr  "All-items HICP" "All-items HICP" "All-items
 ↪  HICP" "All-items HICP" ...
##  $ unit       : chr  "Index, 2015=100" "Index, 2015=100" "Index,
 ↪  2015=100" "Index, 2015=100" ...
##  $ TIME_PERIOD: chr  "1996-01" "1996-02" "1996-03" "1996-04" ...
##  $ values     : num  43 43.6 44.3 45.2 45.8 46.2 46.3 46.4 47.4
 ↪  48.2 ...
```

Wymiar tablicy `dat.1` został znacznie zredukowany, ponieważ obejmuje ona wyłącznie dane dla Polski. Ze względu na zastosowanie wielu indeksów cen w definicji listy db i potencjalnie wielu podstaw indeksów `unit` oraz innych pól, ramka `dat.1` zawiera pełną informację, którą należy uporządkować. W tym celu utworzono pomocniczą listę `templist`, która zawiera unikalne kombinacje ciągów znaków z kolumn `coicop` oraz `unit`. Zastosowano tu funkcję `crossing` z biblioteki `tidyr` (Wickham i in., 2024c).

```
templist <- dat.1[, which(!s.2 %in% s.1)] %>%
  crossing() %>%
  t() %>%
  data.frame(., stringsAsFactors = FALSE) %>%
  lapply(., function(x) x)
templist
```

```
## $X1
## [1] "All-items HICP"  "Index, 2015=100"
##
## $X2
## [1] "Bread"           "Index, 2015=100"
##
## $X3
## [1] "Butter"          "Index, 2015=100"
##
## $X4
## [1] "Diesel"          "Index, 2015=100"
##
## $X5
## [1] "Eggs"            "Index, 2015=100"
##
## $X6
## [1] "Energy"          "Index, 2015=100"
##
## $X7
## [1] "Food"            "Index, 2015=100"
##
## $X8
## [1] "Meat"            "Index, 2015=100"
##
## $X9
## [1] "Petrol"          "Index, 2015=100"
##
## $X10
## [1] "Pork"            "Index, 2015=100"
```

Następnie na podstawie listy `templist` utworzono klucze `key`, które zostały przekazane do funkcji `subset`. W rezultacie powstała lista `dat.2` zawierająca ramki danych z obserwacjami dla poszczególnych kategorii cen.

```
dat.2 <- lapply(
  1:length(templist),
  function(x) {
    key <- ""
    for (j in 1:length(s.3)) {
      if (j != length(s.3))
        key <- paste0(
          key,
          s.3[j],
          " == ",
          "'",
          templist[[x]][j],
          "'",
          " & "
```

```
        )
      else
        key <- paste0(
          key,
          s.3[j],
          " == ",
          "'",
          templist[[x]][j],
          "'"
        )
    }
    key <- key %>% str_trim()
    subset(dat.1, eval(parse(text = key)))
  }
)
head(dat.2[[1]])
```

```
##           coicop            unit TIME_PERIOD values
## 1 All-items HICP Index, 2015=100     1996-01   43.0
## 2 All-items HICP Index, 2015=100     1996-02   43.6
## 3 All-items HICP Index, 2015=100     1996-03   44.3
## 4 All-items HICP Index, 2015=100     1996-04   45.2
## 5 All-items HICP Index, 2015=100     1996-05   45.8
## 6 All-items HICP Index, 2015=100     1996-06   46.2
```

Czytelnik może zauważyć, że podana procedura jest szczególnie przydatna, jeśli chce się utworzyć wiele zbiorów `[eurostat_code]_db.R` ze zdefiniowanymi zmiennymi, podobnych do pliku *prc_hicp_midx_db.R*. W następnym kroku utworzona została lista `dat.3`, która zawiera daty i wartości zmiennych cenowych.

```
dat.3 <- lapply(
  1:length(dat.2),
  function(x) {
    df <- dat.2[[x]] %>%
    data.frame(., stringsAsFactors = FALSE) %>%
    unite(
      .,
      name,
      which(colnames(.) %in% s.3),
      sep = " ",
      remove = TRUE
    )
    colnames(df)[which(
      colnames(df) == "values"
    )] <- df[1, 1]
    df <- df[, -1]
    return(df)
```

```
  }
) %>%
  list.clean(., function(x) is.null(x), TRUE)
```

Wybrane elementy listy `dat.3` podano poniżej. Należy pamiętać, że obserwacje nie są jeszcze posortowane w czasie.

```
dat.3[[1]][[1]] %>% .[1:5]
```

```
## [1] "1996-01" "1996-02" "1996-03" "1996-04" "1996-05"
```

```
dat.3[[1]][[2]] %>% .[1:5]
```

```
## [1] 43.0 43.6 44.3 45.2 45.8
```

Następnie utworzono listę `dat.4`, która zawiera tylko zmienne o liczebności obserwacji większej od przyjętego progu (96 dla danych miesięcznych i 32 dla kwartalnych). W tym celu wykorzystano bibliotekę `xts` (strukturalizacja danych statystycznych do postaci szeregów czasowych) (Ryan i Ulrich, 2024).

```
dat.4 <- lapply(
  dat.3,
  function(x) {
    x$TIME_PERIOD <- as.Date(paste0(x$TIME_PERIOD, "-01"))
    x %>%
      data.frame(stringsAsFactors = FALSE) %>%
      na.omit() %>%
      {
        if (nrow(.) == 0 | nrow(.) == 1) NULL
        else {
          period <- data.frame(.) %>%
              xts(.[, -1], order.by = .[, 1])
          period <- periodicity(period)$scale
          if (period == "monthly" & nrow(.) < 96) {
            NULL
          } else if(period == "quarterly" & nrow(.) < 32) {
            NULL
          } else .
        }
      }
  }
) %>%
  list.clean(., function(x) is.null(x), TRUE)
```

Na podstawie poniższego kodu utworzono listę `dat.5` zawierającą tylko te zmienne, dla których ostatnia obserwacja nie jest starsza niż 4 kwartały.

```
dat.5 <- lapply(
  1:length(dat.4),
  function(x) {
    last <- dat.4[[x]][[1]] %>%
      unlist %>%
      dplyr::last() %>%
      as.Date()
    last.4q <- Sys.Date() %m-% months(12) %>%
      floor_date("month")
    if (last < last.4q) NULL
    else dat.4[[x]]
  }
) %>%
  list.clean(., function(x) is.null(x), TRUE)
```

Na koniec porządkowania danych statystycznych utworzono indywidualne obiekty o nazwach odpowiadających poszczególnym zmiennym za pomocą funkcji `assign`.

```
if (length(dat.5) != 0) {

  for (i in 1:length(dat.5)) {
    if (i == 1) {
      df.1 <- as_tibble(
        dat.5[[1]],
        stringsAsFactors = FALSE
      )
    } else {
      df.2 <- as_tibble(
        dat.5[[i]],
        stringsAsFactors = FALSE
      )
      df.1 <- inner_join(df.1, df.2, by = "TIME_PERIOD")
    }
  }

  temp <- seq(
    as.Date("1995-01-01"),
    as.Date("2025-10-01"),
    by = "quarter"
  ) %>%
    data.frame(
      time = .,
      temp = 1,
      stringsAsFactors = FALSE
    )
```

```
  colnames(df.1) <- colnames(df.1) %>%
    gsub("\\(", "", .) %>%
    gsub("\\)", "", .) %>%
    gsub("TIME_PERIOD", "time", .) %>%
    str_squish() %>% paste(., db$market, sep = " ")
  colnames(df.1)[1] = "time"

  assign(db$code, df.1 %>%
    .[with(., order(time)), ] %>%
    .[, !colSums(is.na(.)) == nrow(.)] %>%
    left_join(temp, ., by = "time") %>%
    .[, -which(colnames(.) == "temp")]
  )

} else {

  df.1 <- seq(
    as.Date("1995-01-01"),
    as.Date("2025-10-01"),
    by = "quarter"
  ) %>%
    data.frame(
      time = .,
      N_A = NA,
      stringsAsFactors = FALSE
    )

  assign(db$code, df.1)

}
```

Powyższy kod zawiera kilka istotnych etapów. Najpierw w pętli `for` utworzono ramkę danych `df.1`, łącząc informacje z listy `dat.5` według dat obserwacji (`TIME_PERIOD`). Następnie utworzono pomocniczą ramkę `temp` zawierającą żądany przedział czasu dla obserwacji.

W kolejnym kroku oczyszczono nazwy kolumn w ramce `df.1`, eliminując nawiasy, nadmiarowe spacje oraz zmieniając nazwę `TIME_PERIOD` na `time`. Dodatkowo do nazw kolumn dołączono informację o rynku *(Prices)* zdefiniowaną w `db$market`.

Ostatecznie utworzono obiekt o nazwie odpowiadającej zmiennej *prc_hicp_midx*, w którym obserwacje są posortowane od najstarszej do najmłodszej i wyrażone kwartalnie. Jeśli lista `dat.5` była pusta, utworzono ramkę danych `df.1` z pełnym przedziałem czasowym oraz brakami danych (`NA`).

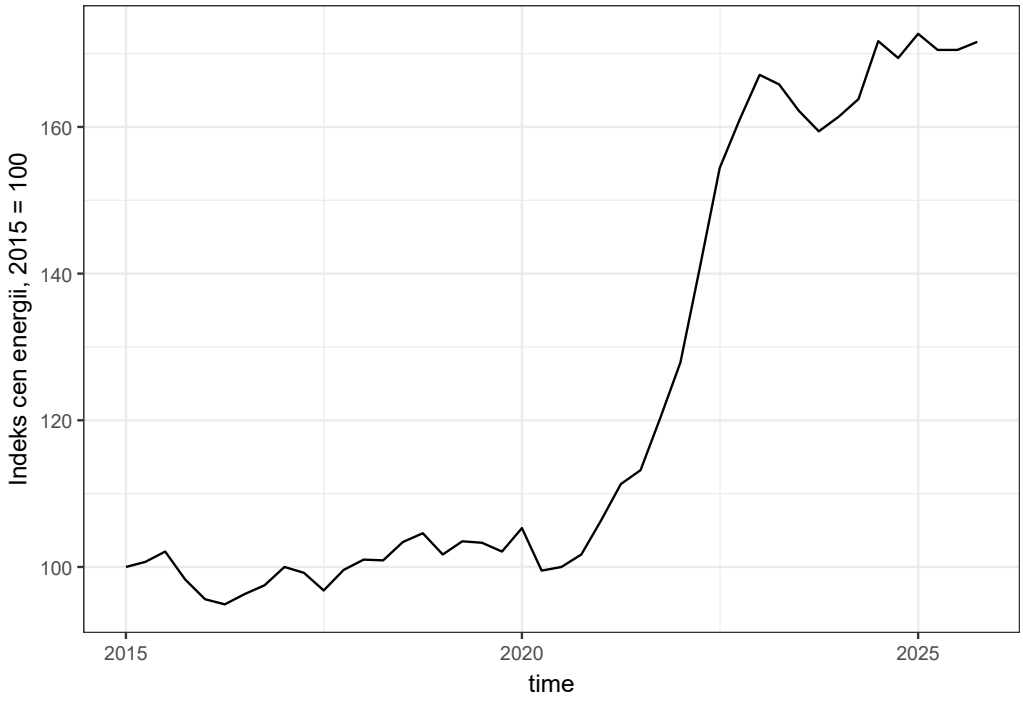
Ramkę danych można wyświetlić za pomocą poniższego kodu.

```
head(eval(parse(text = db$code)))
```

Na tej podstawie można już utworzyć rys. 2.34 dla wybranej zmiennej.

```
df <- eval(parse(text = db$code)) %>%
  na.omit() %>%
  .[, c("time", "Energy.Index..2015.100 (Prices)")] %>%
  set_colnames(
    c("time", "Indeks cen energii, 2015 = 100")
  )
```

```
ggplot(
  df,
  aes(x = time, y =`Indeks cen energii, 2015 = 100`)
) +
  geom_line() +
  theme_bw()
```



Rysunek 2.34. Indeks cen energii, 2015 = 100 – Eurostat

Dla celów archiwizacji ramka danych `df.1` została również zapisana na dysku w formacie pliku `XLSX`.

```
write_xlsx(
  eval(parse(text = db$code)),
  path = paste0(path, db$code, ".xlsx"),
  col_names = TRUE,
  format_headers = TRUE
)
```

Czytelnik może rozszerzyć podany kod o kolejne zmienne z bazy Eurostat, zarówno o częstotliwości miesięcznej, jak i kwartalnej. Na koniec podano sposób organizacji bazy danych FRED (*Federal Reserve Economic Data*) .

FRED

FRED oznacza bazę danych ekonomicznych Systemu Rezerwy Federalnej (Fed), banku centralnego Stanów Zjednoczonych. FRED zawiera szeregi czasowe dla zmiennych gospodarki USA i innych gospodarek z częstotliwością roczną, kwartalną, miesięczną, tygodniową i dzienną. Baza ta agreguje dane ekonomiczne z wielu źródeł, przy czym największy udział w tworzeniu danych mają amerykańskie agencje rządowe.

Aby przedstawić kod, należy wczytać odpowiednie biblioteki. Niektóre z poniższych bibliotek pojawiają się w książce po raz pierwszy:

- `fredr` (Boysel i Vaughan, 2021),
- `jsonlite`.

```
library(dplyr)
library(fredr)
library(ggplot2)
library(httr)
library(jsonlite)
library(magrittr)
library(stringr)
```

W poniższej dyskusji podano sposób komunikacji z interfejsem `API` za pomocą biblioteki `httr`. Aby skorzystać z FRED API, należy utworzyć konto w serwisie (https://fred.stlouisfed.org/) i złożyć elektroniczny wniosek o przydział klucza identyfikacyjnego (`API key`). Klucz został przypisany do zmiennej znakowej `fred.apikey`, zapisany w pliku `fredAPIKey.R` i następnie wczytany do kodu za pomocą funkcji `source`.

```
source("fredAPIKey.R")
```

Jeśli pomiędzy komputerem użytkownika a serwerem FRED API znajduje się serwer pośredniczący (proxy), należy wskazać jego adres IP (np. “192.168.1.1”, *string*) i port (np. 8080, *numeric*) tego serwera.

```
set_config(use_proxy(url = "Adres IP", port = port))
```

Następnie zdefiniowano niezbędne zmienne znakowe, tj. adres serwera FRED API (`fred.address`) i typ odpowiedzi serwera na zapytanie użytkownika (`fred.filetype`). Określono, że wymiana danych ma odbywać się w formacie `JSON`.

```
fred.address <- "https://api.stlouisfed.org/fred/"
fred.filetype <- "json"
```

W następnym kroku należało zdefiniować postać zapytania do bazy danych interfejsu FRED API. W tym celu wskazano właściwą *końcówkę* (*endpoint*) tego interfejsu. Jest to adres lokalizacji funkcji o nazwie `search` w katalogu `fred/series/`. Do funkcji `search` przekazuje się odpowiednie parametry: `search_text`, `search_type` itd., za pomocą symbolu ? w adresie internetowym. Parametry łączy się za pomocą symbolu &. Specyfikację *końcówek* można znaleźć w dokumentacji dostępnej pod adresem:

- https://fred.stlouisfed.org/docs/api/fred/.

Dla funkcji `fred/series/search` jest to adres:

- https://fred.stlouisfed.org/docs/api/fred/series_search.html.

W dokumentacji znajduje się również opis parametrów funkcji.

```
fred.query <- paste0(
  "series/search",
  "?search_text=poland+gdp",
  "&search_type=full_text"
)
```

Można zauważyć, że do powyższej funkcji `search` przekazano parametry:

`search_text=poland+gdp` oraz `search_type=full_text`.

Następnie należało zdefiniować zapytanie do bazy danych w postaci adresu internetowego[25].

[25] Warto zwrócić uwagę, że w druku zmienna `fred.request` jest skracana, aby nie ujawniać postaci pełnego klucza `API`.

```
fred.request <- paste0(
  fred.address,
  fred.query,
  "&api_key=", fred.apikey,
  "&file_type=", fred.filetype
)
fred.request %>% str_trunc(., 40, side = "center")
```

```
## [1] "https://api.stlouis...8b1&file_type=json"
```

W kolejnym kroku sprawdzono, czy odpowiedź serwera na zapytanie jest poprawna, tj. czy status odpowiedzi otrzymuje kod 200. W celu weryfikacji statusu zapytania zastosowano funkcję `GET` z biblioteki `httr`. Jeśli warunek funkcji `if` jest spełniony, wykonywane jest zapytanie i następuje porządkowanie wartości funkcji.

```
if (GET(fred.request)$status_code == 200) {
  # rltumd
}
```

Zadaniem kodu `rltumd` jest:

- przeszukanie bazy danych na obecność ciągu `poland+gdp`,
- pobranie wyników poszukiwania w postaci ramki danych,
- filtrowanie ramki danych,
- pobranie kodu (`id`) wybranej zmiennej z bazy danych FRED.

W poniższym kodzie `rltumd` zastosowano przetwarzanie potokowe.

```
if (GET(fred.request)$status_code == 200) {
  fred.response <- fred.request %>%
    GET() %>%
    content(., "text") %>%
    fromJSON()
  df.1 <- fred.response$seriess
  df.2 <- df.1 %>%
    subset(
      frequency_short == "Q" &
        units == "Millions of Polish Zloty" &
        seasonal_adjustment_short == "SA"
    ) %>%
    t()
  id <- df.2[rownames(df.2) == "id", ]
}
```

Po pierwsze, wykonano zapytanie za pomocą metody GET protokołu HTTP dla adresu fred.request, wartość funkcji przekształcono na tekst (content) a następnie w obiekt języka R (fromJSON) za pomocą funkcji biblioteki jsonlite. Ten obiekt to lista.

```
class(fred.response)
```

```
## [1] "list"
```

Po drugie, z obiektu (zmiennej) fred.response pobrana została ramka danych seriess i zapisana w obiekcie df.1. Po trzecie, wartości obiektu df.1 zostały poddane filtrowaniu za pomocą funkcji subset na obecność zmiennych o częstotliwości kwartalnej (Q), wyrażonych w milionach jednostek waluty krajowej (Millions of Polish Zloty) i wyrównanych sezonowo (SA). Ostatecznie obiekt df.2 zawiera transpozycję wierszy i kolumn. Po czwarte, z ramki danych df.2 pobrano identyfikator z wiersza id i zapisano w zmiennej id. Ramka df.2 ma następującą postać:

```
df.2 %>% str_trunc(., 25, side = "right")
```

```
##                            6
## id                         "CPMNACSCAB1GQPL"
## realtime_start             "2025-12-21"
## realtime_end               "2025-12-21"
## title                      "Gross Domestic Product..."
## observation_start          "1995-01-01"
## observation_end            "2025-07-01"
## frequency                  "Quarterly"
## frequency_short            "Q"
## units                      "Millions of Polish Zloty"
## units_short                "Mil. of Polish Zloty"
## seasonal_adjustment        "Seasonally Adjusted"
## seasonal_adjustment_short "SA"
## last_updated               "2025-12-05 04:03:44-06"
## popularity                 "25"
## group_popularity           "30"
## notes                      "Eurostat unit ID: CP_M..."
```

Identyfikator id uzyskanej zmiennej jest równy: CPMNACSCAB1GQPL. Należy zauważyć, że powyższe przekształcenia doprowadziły do wyspecyfikowania tylko jednej zmiennej. Następnie dla wybranej zmiennej zostały pobrane wartości funkcji (*końcówek*) zawarte w liście fred.task.

```
fred.task <- list(
  category = "series/categories",
```

```
  release  = "series/release",
  tags     = "series/tags",
  series   = "series",
  obs      = "series/observations"
)
```

Następnie zastosowano funkcję `lapply` i zapisano wartości funkcji do listy `fred.output`. Warto zwrócić uwagę, że w zmiennej `fred.request` wstawiono parametr `series_id`=CPMNACSCAB1GQPL. Jeśli w wyniku zapytania do serwera status operacji HTTP jest inny niż 200, funkcja przypisuje wartość NA.

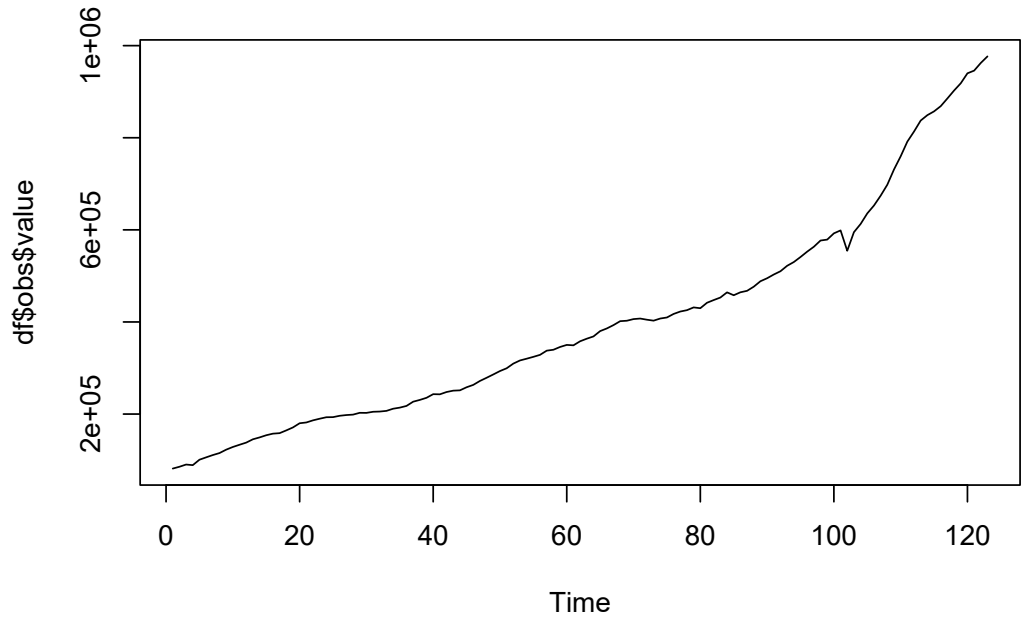
```
fred.output <- lapply(
  fred.task,
  function(x) {
    fred.request <- paste0(
      fred.address,
      x,
      "?series_id=",
      id,
      "&api_key=", fred.apikey,
      "&file_type=", fred.filetype
    )
    if (GET(fred.request)$status_code == 200) {
      fred.response <- fred.request %>%
        GET() %>%
        content(., "text") %>%
        fromJSON()
    } else NA
  }
)
```

W kolejnym kroku sprawdzono, czy wszystkie elementy listy są różne od NA. Jeśli warunek ten był spełniony, do listy `df` zapisywano różne atrybuty analizowanej zmiennej.

```
if (all(!is.na(fred.output))) {
  df <- list(
    category = fred.output$category$categories$name,
    release  = fred.output$release$releases,
    tags     = fred.output$tags$tags,
    series   = fred.output$series$seriess,
    obs      = fred.output$obs$observations %>%
      .[, -c(1, 2)]
  )
}
```

Wśród tych atrybutów są wartości obs zmiennej. Można zatem pokazać ich pierwszy wykres (rys. 2.35). Ramka danych df$obs zawiera dwie kolumny: date oraz value.

```
ts.plot(df$obs$value)
```



Rysunek 2.35. Produkt krajowy brutto dla Polski – FRED API – przypadek 1

Dla ułatwienia zapisu ramkę danych df$obs przypisano do zmiennej y.

```
y <- df$obs
```

Dokładna analiza typów danych w obiekcie y wskazuje, że dalsze operacje wymagają zmiany ich typów. W tym celu zbudowano funkcję modify, która zamienia daty obserwacji na klasy Date, a wartości na liczby zmiennoprzecinkowe. Dodatkowo wartości PKB przeliczono na miliardy poprzez podzielenie przez 1000.

```
modify <- function(x) {
  x[, "date"] <- as.Date(x[, "date"])
  x[, "value"] <- as.numeric(x[, "value"]) %>%
    { ./1000 }
  return(x)
}
```

Przy podanych warunkach zastosowano funkcję modify do zmiennej y.

```
y <- y %>% modify(.)
head(y)
```

```
##         date    value
## 1 1995-01-01  81.6206
## 2 1995-04-01  85.5793
## 3 1995-07-01  90.3861
## 4 1995-10-01  89.0557
## 5 1996-01-01 100.8792
## 6 1996-04-01 105.9969
```

W następnym kroku został utworzony wykres `ggplot` zmiennej `y`, który uwzględnia tytuły zawarte w funkcji `titles`.

```
titles <- function() {
  title <- paste(
    df$series$title,
    df$series$units_short,
    df$series$seasonal_adjustment_short,
    sep = ", "
  ) %>%
    paste0(
      .,
      " (", df$series$id, ")"
    )
  subtitle <- paste0(
    "Link: ",
    df$release$link
  )
  caption <- paste0(
    title,
    ", source: FRED"
  )
  return(list(
    title = title,
    subtitle = subtitle,
    caption = caption
  ))
}
```

Funkcję należy wywołać bez parametrów, aby utworzyć tytuły.

```
ti <- titles()
title <- ti$title
subtitle <- ti$subtitle
caption <- ti$caption
```

Kod generowania wykresu ma postać funkcji p z argumentem x. Argument x oznacza ramkę danych zawierającą daty i wartości zmiennej.

```
p <- function(x) {
  ggplot(x, aes(date, value)) +
    geom_line() +
    labs(
      title = title,
      subtitle = subtitle,
      caption = caption
    ) +
    theme_bw() +
    theme(
      plot.title = element_text(
        hjust = 0.5,
        size = 10
      ),
      plot.subtitle = element_text(hjust = 0.5),
      plot.caption = element_text(hjust = 0.5)
    )
}
```

Można teraz pokazać kolejny wykres zmiennej CPMNACSCAB1GQPL, tj. produktu krajowego brutto (PKB) dla Polski (rys. 2.36).

```
p(y)
```

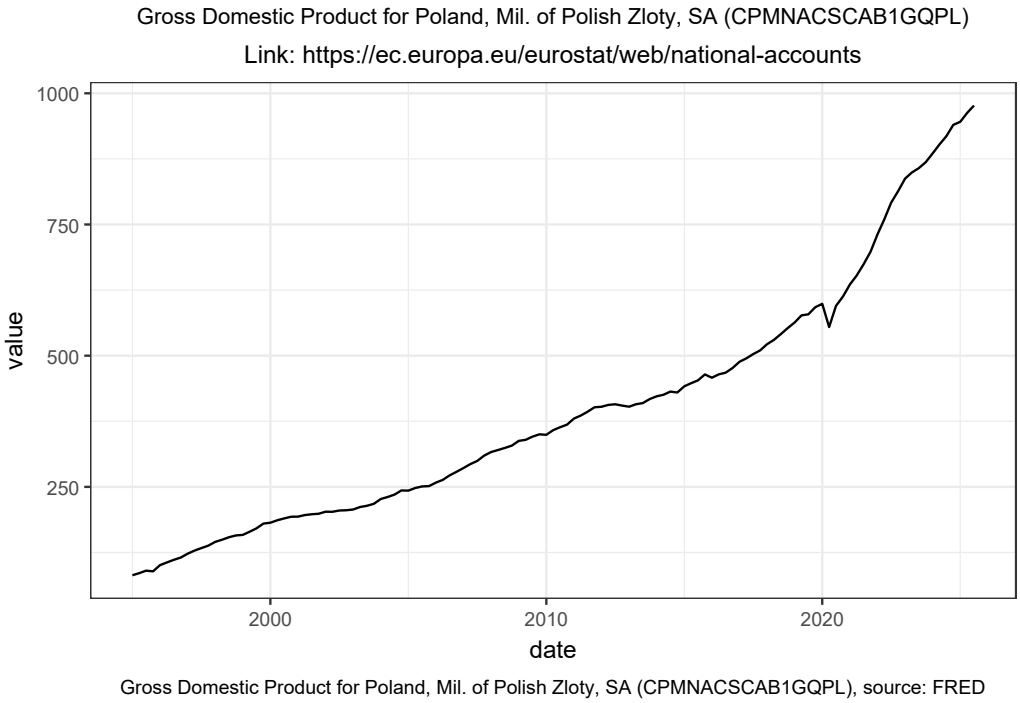
Powyższą procedurę API można uprościć korzystając ze specjalistycznej biblioteki fredr. Najpierw należy ustawić klucz dostępu do interfejsu API za pomocą funkcji fredr_set_key.

```
fredr_set_key(fred.apikey)
```

Znając identyfikator zmiennej, w tym przypadku CPMNACSCAB1GQPL, można w prosty sposób pobrać jej wartości za pomocą funkcji fredr.

```
df <- fredr(
  series_id = id
) %>%
  .[, c(1, 3)]
```

W bibliotece fredr dostępna jest również funkcja fredr_endpoints, która umożliwia wyświetlenie wszystkich końcówek interfejsu FRED – także tych, które wcześniej zdefiniowano w zmiennej fred.task.



Rysunek 2.36. Produkt krajowy brutto dla Polski – FRED API – przypadek 2

```
endpoints <- fredr_endpoints %>%
  { cbind(
    .[, 1:2],
    str_trunc(.$note, 40, side = "right")
  ) } %>%
  set_colnames(c("endpoint", "type", "note"))
head(endpoints[, 1]) %>%
  str_trunc(., 15, side = "right") %>%
  data.frame()
```

```
##                .
## 1   fred/category
## 2 fred/categor...
## 3 fred/categor...
## 4 fred/categor...
## 5 fred/categor...
## 6 fred/categor...
```

Pobranie atrybutów zmiennej o identyfikatorze CPMNACSCAB1GQPL jest teraz proste. Wystarczy zastosować funkcję `fredr_request`, przekazując parametry `endpoint` oraz `series_id`.

```
df <- lapply(
  fred.task,
  function(x) {
    fredr_request(
      endpoint = x,
      series_id = id
    )
  }
)
```

Na podstawie ramki `df` utworzono następnie ramkę `y`, której wartości zmodyfikowano za pomocą wcześniej przygotowanej funkcji `modify`.

```
y <- df$obs %>%
  .[, -c(1, 2)] %>%
  as.data.frame() %>%
  modify(.)
```

Można teraz wygenerować ostateczny wykres `ggplot` dla zmiennej CPMNACSCAB1GQPL za pomocą funkcji `p` (rys. 2.37).

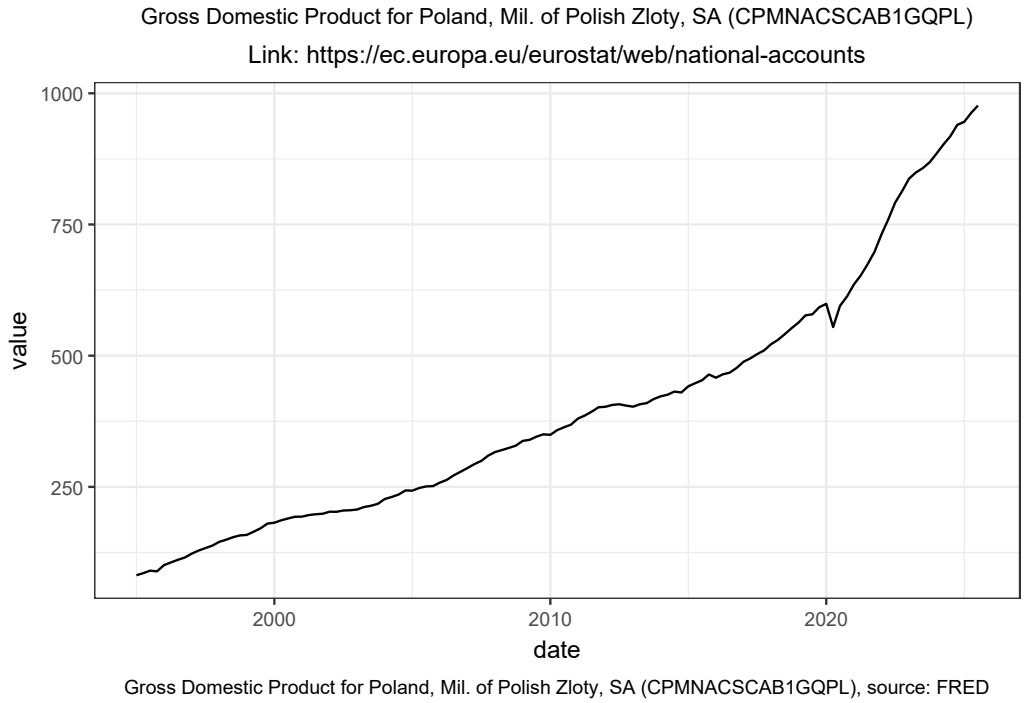
```
p(y)
```

Jak widać, zastosowanie biblioteki `fredr` znacząco upraszcza pobranie wartości i atrybutów zmiennych z bazy danych FRED.

W kolejnym podrozdziale zostanie przedstawione zagadnienie automatyzacji zadań przeglądarki internetowej w celu pobierania danych ze stron internetowych. Omówione zostaną przykłady praktycznego wykorzystania języka R do pozyskiwania i przetwarzania publicznie dostępnych danych statystycznych, co stanowi ważny element współczesnych badań empirycznych.

2.3.2. Automatyzacja zadań przeglądarki internetowej

Wśród wielu zastosowań programu R znajdują się: przetwarzanie i analiza danych, statystyka oraz ekonometria. Jednym z zastosowań w analizie dużych zbiorów danych jest automatyczne pobieranie informacji ze stron internetowych (*web scraping*). W razie wątpliwości co do legalności takiego sposobu pobierania informacji należy uzyskać opinię prawną lub odwołać się do regulaminu danej strony. Podany poniżej przykład ma charakter akademicki i służy pobraniu publicznie dostępnych



Rysunek 2.37. Produkt krajowy brutto dla Polski – FRED API – przypadek 3

danych statystycznych ze strony internetowej Głównego Urzędu Statystycznego[26]. Rozbudowany przykład automatycznego pobrania publicznie dostępnych artykułów na potrzeby prowadzenia badań naukowych z serwisu `Google Scholar` podano w książce Wdowiński (2020).

Automatyczne pobranie danych ze strony internetowej nie zawsze jest możliwe. Jest to szczególnie trudne, jeśli nie można wyświetlić w przeglądarce źródła strony w postaci języka `HTML`, gdyż strona jest generowana dynamicznie za pomocą np. języka `JavaScript`. Nie można wówczas odwołać się do znaczników `HTML/CSS` w celu pobrania ich zawartości. Niektóre serwisy generują również atrybuty znaczników `HTML/CSS` w sposób dynamiczny, co uniemożliwia jednokrotne przypisanie ich nazw.

Zadanie pobrania danych ze strony internetowej upraszcza się, jeśli można wykorzystać biblioteki języka `R`. W podanym przykładzie wykorzystano serwer `Selenium` (https://www.selenium.dev/), którego zadaniem jest automatyzacja działania przeglądarki internetowej. Występuje implementacja systemu `Selenium` w postaci bibliotek `RSelenium` (Harrison, 2022) i `seleniumPipes` (Harrison, 2016). Dokumentację biblioteki `RSelenium` można znaleźć na stronach internetowych:

[26] Por. https://stat.gov.pl/.

- https://github.com/ropensci/RSelenium,
- https://docs.ropensci.org/RSelenium/index.html.

Dokumentację i konfigurację RSelenium w zakresie współpracy z różnymi przeglądarkami można znaleźć na stronie internetowej:

- https://docs.ropensci.org/RSelenium/articles/saucelabs.html.

Dokumentacja biblioteki seleniumPipes znajduje się na stronach internetowych:

- https://github.com/johndharrison/seleniumPipes,
- https://rpubs.com/johndharrison/seleniumPipes-basic.

Działanie biblioteki RSelenium polega na uruchomieniu przeglądarki w izolowanym środowisku i wydawanie jej poleceń z poziomu języka R. W przykładzie zastosowano przeglądarkę Firefox. W podany sposób można testować poprawność budowy stron internetowych i pobierać z nich informacje. W bibliotece seleniumPipes został zaimplementowany operator przetwarzania potokowego. Wykorzystano również bibliotekę rvest (Wickham, 2024). Działanie tej biblioteki polega na wczytaniu treści strony internetowej z dokumentu HTML, a następnie na uporządkowaniu informacji zawartej w znacznikach klas CSS, wykorzystywanych do oznaczania kodu HTML. Funkcje biblioteki realizują zadania pobrania tekstu zawartego pomiędzy zdefiniowanymi znacznikami oraz rozmaitych atrybutów, np. łączy do stron internetowych zawartych w znacznikach href. Pobraną w ten sposób informację tekstową można umieścić w ramkach danych i przetworzyć. Jeśli strona internetowa, z której pobierane są informacje ma wiele podstron, to można wykonać iteracyjny proces pobrania w pętli. Zwykle odbywa się to poprzez wprowadzenie zmiennej części adresu strony do żądania GET w protokole HTTP/S (Wdowiński, 2020).

Dane zostaną pobrane ze strony internetowej GUS w postaci szeregów czasowych z *Banku Danych Makroekonomicznych* (https://bdm.stat.gov.pl/). W przykładzie zostanie wykorzystana dodatkowo biblioteka netstat (Condylios i in., 2022) do obsługi portów TCP. Jak zwykle rozpoczęto od wczytania niezbędnych bibliotek.

```
library(dplyr)
library(ggplot2)
library(magrittr)
library(netstat)
library(readxl)
library(rvest)
library(RSelenium)
library(seleniumPipes)
library(stringr)
```

Najpierw zdefiniowano adres strony, która zawiera dane: roczne, kwartalne i miesięczne.

```
gus.address <- "https://bdm.stat.gov.pl/"
```

Zmienna `gus.freq` zawiera ciąg znaków dla danych kwartalnych, który będzie poszukiwany na stronie.

```
gus.freq <- "KWARTALNE"
```

Za pomocą poniższego kodu uruchomiony został serwer `Selenium`. Wykorzystano domyślnie przeglądarkę `Firefox`. W przypadku przeglądarki `Google Chrome` trzeba pobrać sterownik `chromedriver`, umieścić go w dowolnym katalogu, odblokować dla niego domyślny port i umieścić wpis dostępu do programu w ścieżce systemowej `Windows`. Wersja sterownika (parametr `chromever`) musi być zgodna z wersją zainstalowanej przeglądarki `Chrome` (https://chromedriver.chromium.org/).

W przypadku programu `Google Chrome` potrzebny jest poniższy kod:

```
server <- rsDriver(
  browser = "chrome",
  chromever = "101.0.4951.41",
  port = free_port(),
  verbose = FALSE
)
```

W przypadku przeglądarki `Firefox` wystarczy zadać domyślne parametry i wykonać kod, który ją uruchamia.

```
server <- rsDriver(
  port = free_port(),
  browser = "firefox",
  extraCapabilities = list(
    "moz:firefoxOptions" = list(
      args = list('')
    )
  ),
  chromever = NULL,
  phantomver = NULL
)
```

```
## [1] "Connecting to remote server"
## $acceptInsecureCerts
## [1] FALSE
```

```
##
## $browserName
## [1] "firefox"
##
## $browserVersion
## [1] "144.0"
##
## $`moz:accessibilityChecks`
## [1] FALSE
##
## $`moz:buildID`
## [1] "20251009125714"
##
## $`moz:geckodriverVersion`
## [1] "0.36.0"
##
## $`moz:headless`
## [1] FALSE
##
## $`moz:platformVersion`
## [1] "10.0"
##
## $`moz:processID`
## [1] 17008
##
## $`moz:profile`
## [1] "C:
Users
user
AppData
Local
Temp
rust_mozprofile0AMemy"
##
## $`moz:shutdownTimeout`
## [1] 60000
##
## $`moz:webdriverClick`
## [1] TRUE
##
## $`moz:windowless`
## [1] FALSE
##
## $pageLoadStrategy
## [1] "normal"
##
## $platformName
## [1] "windows"
##
```

```
## $proxy
## named list()
##
## $setWindowRect
## [1] TRUE
##
## $strictFileInteractability
## [1] FALSE
##
## $timeouts
## $timeouts$implicit
## [1] 0
##
## $timeouts$pageLoad
## [1] 300000
##
## $timeouts$script
## [1] 30000
##
##
## $unhandledPromptBehavior
## [1] "dismiss and notify"
##
## $userAgent
## [1] "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:144.0)
 ↪  Gecko/20100101 Firefox/144.0"
##
## $webdriver.remote.sessionid
## [1] "0b5ed103-5379-4c9f-87d2-8e7c9ad80cb0"
##
## $id
## [1] "0b5ed103-5379-4c9f-87d2-8e7c9ad80cb0"
```

Następnie należy przekazać sterowanie do przeglądarki. Po wykonaniu każdej części kodu można obserwować wynik w otwartej przeglądarce `Firefox`.

```
browser <- server[["client"]]
```

Uruchomienie strony internetowej odbywa się poprzez zastosowanie metody `navigate` i przekazanie parametru w postaci adresu strony.

```
browser$navigate(gus.address)
Sys.sleep(5)
```

W oknie przeglądarki internetowej została wczytana następująca strona: https://bdm.stat.gov.pl/. Na tej stronie znajduje się odnośnik do danych kwartalnych

(KWARTALNE). Algorytm musi ten odnośnik znaleźć w treści strony i spowodować jego uruchomienie (kliknięcie). Odbywa się to za pomocą poniższego kodu.

```
browser$findElement(
  using = "xpath",
  value = paste0(
    "//*[text()='",
    gus.freq,
    "']"
  )
)$clickElement()
Sys.sleep(5)
```

W powyższym kodzie zastosowano metodę `findElement` z dwoma parametrami `using` i `value`. Wartość pierwszego parametru `xpath` oznacza, że do zlokalizowania tekstu "KWARTALNE" wykorzystano składnię standardu `XSLT` (*eXtensible Stylesheet Language Transformations*) do nawigowania wśród elementów i atrybutów dokumentów `XML`. Parametr `value` określa składnię `XPath`, która jest następująca:

- `//*` wskazanie wszystkich znaczników w dokumencie bez względu na to, gdzie się znajdują,
- [text()='KWARTALNE'] poszukiwanie tekstu KWARTALNE.

W rezultacie //*[text()='KWARTALNE'] oznacza poszukiwanie wskazanego ciągu znaków we wszystkich znacznikach na stronie internetowej. Przyjęto w przykładzie, że taki przypadek jest *jeden* i jest nim – w uproszczeniu – następujący znacznik:

```
print("<a href='#'>KWARTALNE</a>")
```

```
## [1] "<a href='#'>KWARTALNE</a>"
```

Po odnalezieniu elementu algorytm wykonuje metodę `clickElement`, która symuluje kliknięcie w link. Na załadowanie strony pozostawiono 5 sekund (`Sys.sleep(5)`)[27].

Łatwo sprawdzić, że w przeglądarce `Firefox` jest wczytana strona internetowa dla danych kwartalnych, dla których można wybrać kategorię: *Bilans płatniczy*, *Budownictwo* itd. Z tej listy zostaną wybrane *Wskaźniki cen*. Najpierw należy wybrać pole formularza dla listy rozwijanej, aby dostępne były opcje wyboru z tej listy. Poniższy kod realizuje to zadanie.

[27] Czytelnik powinien zwrócić uwagę na czas potrzebny na załadowanie strony internetowej po wykonaniu polecenia w przeglądarce. Jeśli elementy kodu strony nie zostaną poprawnie wczytane, to nie będą widoczne w algorytmie.

```
browser$findElement(
  using = "css",
  value = paste0(
    "#nazwy-tablic > ",
    "div:nth-child(1) > ",
    "button:nth-child(1)"
  )
)$clickElement()
Sys.sleep(5)
```

W powyższym kodzie w miejsce wartości `xpath` przyjęto wartość `css`, która pozwala na poszukiwanie znacznika HTML na podstawie jego lokalizacji CSS (*Cascading Style Sheets*). Tę lokalizację (CSS *selector*) należy przekazać za pomocą parametru `value`. Wartość *selektora* CSS można ustalić w przeglądarce internetowej poprzez wybranie opcji *Zbadaj*. Następnie należy pobrać selektor w opcji *Kopiuj*.

Wykonanie powyższego kodu powoduje otwarcie okna z listą dostępnych opcji wyboru. Następnie z listy należy automatycznie wybrać pozycję *Wskaźniki cen*[28].

```
browser$findElement(
  using = "css",
  value = paste0(
    ".open > ",
    "ul:nth-child(2) > ",
    "li:nth-child(18) > ",
    "a:nth-child(1)"
  )
)$clickElement()
Sys.sleep(5)
```

Można również zauważyć, że w ikonie *Ustawienia/Wybierz okres* nie jest domyślnie zaznaczony dostępny przedział próby statystycznej. Poniższy kod realizuje czynność naciśnięcia przycisku *Ustawienia*.

```
browser$findElement(
  using = "css",
  value = "#settings"
)$clickElement()
Sys.sleep(5)
```

Poniższy kod powoduje rozwinięcie opcji wyboru dostępnych w polu formularza *Wybierz okres*.

[28] Zmiana lokalizacji odpowiedniej opcji wyboru w dokumencie HTML wymagałaby zastosowania innego znacznika CSS lub zapisania bardziej złożonego kodu, który spowodowałby automatyczne przeszukanie dokumentu na obecność ciągu znaków.

```
browser$findElement(
  using = "css",
  value = paste0(
    "#years-filter > ",
    "div:nth-child(1) > ",
    "button:nth-child(1)"
  )
)$clickElement()
Sys.sleep(5)
```

Naciśnięcie przycisku *Rok* oznacza wybór wszystkich okresów dla dostępnej próby statystycznej.

```
browser$findElement(
  using = "css",
  value = ".all > span:nth-child(1)"
)$clickElement()
Sys.sleep(5)
```

Na koniec należy zamknąć otwarte okno z opcjami.

```
browser$findElement(
  using = "css",
  value = "#settingsFirstTabIndex"
)$clickElement()
Sys.sleep(5)
```

W podany powyżej sposób zostały automatycznie wykonane czynności niezbędne do wyświetlenia tablicy dla wartości kwartalnych wskaźników cen. W kolejnym kroku do listy `tab` pobrano źródło strony, a następnie tablicę z wartościami zmiennych. Ponieważ na pobranej stronie znajdowało się więcej tablic `HTML`, lista obejmuje tylko dwie pierwsze współrzędne.

```
tab <- browser$getPageSource()[[1]] %>%
  xml2::read_html() %>%
  html_table() %>%
  .[1:2]
```

Łatwo sprawdzić, że pierwsza współrzędna zawiera nagłówki tablicy *wskaźników cen*. Wymiary tej tablicy są następujące: 1, 105. Wymiary drugiej tablicy są następujące: 93, 105. Oznacza to, że w tablicy są 93 wiersze i 105 kolumn. Poniższy kod powoduje utworzenie dwóch ramek danych:

- `header` z nagłówkami tablicy,

- body z nazwami i wartościami zmiennych.

```
header <- tab %>%
  .[[1]] %>%
  do.call(cbind, .)
body <- tab %>%
  .[[2]] %>%
  do.call(cbind, .)
```

Następnie ramki danych header i body należało połączyć wierszami i oczyścić ze zbędnych informacji, w szczególności zamienić kropki (brakujące obserwacje) na wartości NA. Zmienna table.1 jest ramką danych.

```
table.1 <- header %>%
  rbind(., body) %>%
  gsub("kw\\.", "kw", .) %>%
  gsub("\\.", "NA", .) %>%
  gsub(",", ".", .) %>%
  as.data.frame()
```

Ponieważ obserwacje brakujące zostały zakodowane w tablicy nie tylko za pomocą kropki, to dodatkowo wykonany został poniższy kod.

```
table.1[table.1 == ""] <- NA
```

Czytelnik może samodzielnie wyświetlić zawartość ramki table.1. Ramka danych table.1 została następnie przygotowana do opracowania numerycznego, tj. ograniczono się wyłącznie do wartości liczbowych.

```
table.2 <- table.1 %>%
  .[-1, -c(1:2)]
```

Ramka danych table.2 została utworzona w celu sprawdzenia integralności danych z tablicą, którą można pobrać ze strony w formacie xls. Powyższa procedura jest przydatna, jeśli tablicy HTML nie towarzyszy źródło danych w postaci plików xls lub csv, które łatwo jest przetworzyć poza programem R. W celu sprawdzenia integralności danych utworzono wektor wartości średnich z wierszy ramki table.2.

```
r.mean.1 <- table.2 %>%
  as.matrix() %>%
  as.numeric(., na.rm = TRUE) %>%
  matrix(., ncol = ncol(table.2)) %>%
  rowMeans(., na.rm = TRUE)
```

```
## Warning in matrix(., ncol = ncol(table.2)): pojawiły się wartości
 ↪  NA na skutek
## przekształcenia
```

Następnie należało pobrać zbiór `xls`, zamienić na plik `xlsx` i wykonać dla niego podobne obliczenia. Łatwo zauważyć, że na stronie występuje przycisk *Zapisz*. Trzeba go wybrać a następnie pobrać plik `xls`.

```
browser$findElement(
  using = "css",
  value = paste0(
    ".settings_table > ",
    "div:nth-child(1) > ",
    "div:nth-child(1) > ",
    "div:nth-child(3) > ",
    "div:nth-child(1) > ",
    "button:nth-child(1) > ",
    "img:nth-child(1)"
  )
)$clickElement()
Sys.sleep(5)
browser$findElement(
  using = "css",
  value = paste0(
    ".open > ",
    "ul:nth-child(2) > ",
    "li:nth-child(3) > ",
    "a:nth-child(1)"
  )
)$clickElement()
Sys.sleep(5)
```

Plik *Wskaźniki cen.xls* trzeba samodzielnie przekształcić do pliku *.xlsx*, zapisać w katalogu, pobrać do ramki danych, uporządkować i obliczyć średnie w wierszach. Wcześniej należy go umieścić w odpowiednim katalogu lub podać własną ścieżkę dostępu w poniższym kodzie.

```
table.3 <- read_xlsx(
  path = paste0(
    "F:\\docs\\R\\b2\\ch2\\",
    "Wskaźniki cen.xlsx"
  )
) %>%
  .[6:98, -c(1:2)] %>%
  as.matrix() %>%
  gsub("\\.", NA, .) %>%
  gsub(",", ".", .) %>%
```

```
  as.numeric() %>%
  matrix(., ncol = ncol(table.2))
```

```
r.mean.2 <- table.3 %>%
  rowMeans(., na.rm = TRUE)
```

Pozostało tylko sprawdzić, czy wektory `r.mean.1` i `r.mean.2` są sobie równe.

```
equal <- all(r.mean.1 == r.mean.2)
equal
```

```
## [1] TRUE
```

Wartość TRUE oznacza, że otrzymano identyczne wyniki dla średnich w wierszach (w czasie) dla zmiennych. Metodę pobrania wartości zmiennych za pomocą biblioteki `RSelenium` można uznać za skuteczną. Na koniec należy zamknąć przeglądarkę i zakończyć sesję serwera.

```
browser$close()
server[['server']]$stop()
```

```
## [1] TRUE
```

W następnym podrozdziale podano podstawowe zagadnienia dotyczące programowania obiektowego w R.

2.4. Zarys programowania obiektowego

Programowanie obiektowe (*object-oriented programming, OOP*) w R pozwala na tworzenie zorganizowanego i modułowego kodu, który jest łatwiejszy w utrzymaniu i rozszerzaniu. Język R obsługuje kilka systemów obiektowych, w tym S3, S4 i R6. System S3 jest najprostszy i najbardziej elastyczny, ale mniej formalny – nie wymusza określonych struktur danych ani metod. System S4 obejmuje bardziej rygorystyczne zasady, umożliwiające definiowanie typów danych i metod w ramach formalnie zdefiniowanych klas. R6 to kolejny popularny system obiektowy w R, oparty na środowiskach. W tym systemie można definiować klasy z polami, metodami i dziedziczeniem[29]. System R6 oferuje podejście podobne do programowania obiektowego w językach takich jak Python, z obsługą pól i metod bezpośrednio w obiektach.

[29] Pola odpowiadają za przechowywanie stanu obiektu, natomiast metody reprezentują funkcje odwołujące się do tego stanu lub realizujące określone operacje. Dziedziczenie pozwala tworzyć nowe klasy na podstawie już istniejących, dzięki czemu obiekty potomne mogą wykorzystywać i rozszerzać funkcjonalność klas nadrzędnych.

Systemy referencyjne, takie jak R6, są szczególnie przydatne w projektach wymagających dynamicznych, złożonych obiektów z dużą liczbą metod i pól. Są również często używane w aplikacjach interaktywnych, takich jak `Shiny`, gdzie ważne jest szybkie i łatwe modyfikowanie stanu obiektu.

Tworzenie klas i metod w systemie S3 opiera się na przypisywaniu atrybutu `class` do obiektów oraz implementacji funkcji będących funkcjami generującymi metody. Klasy S3 są stosunkowo prostym podejściem do programowania obiektowego w R. Są oparte na systemie nieformalnym, w którym klasy są definiowane przez strukturę obiektu i przypisanie metod do tej struktury. S4 wymaga wykorzystania funkcji takich jak `setClass` – do definiowania klas – i `setMethod` – do tworzenia metod specyficznych dla określonych typów. Klasy S4 wprowadzają bardziej formalne podejście do programowania obiektowego, oferując pełną specyfikację atrybutów obiektu, typów danych oraz metod, co sprawia, że są bardziej restrykcyjne niż klasy S3. Klasy S4 oferują mechanizmy dziedziczenia i walidacji[30], dzięki czemu można tworzyć bardziej zaawansowane struktury obiektowe, co jest szczególnie pomocne w bardziej złożonych aplikacjach. R6 pozwala na definiowanie klas przy użyciu funkcji `R6Class`, umożliwiając osadzanie danych i metod wewnątrz obiektu.

Programowanie obiektowe jest szczególnie przydatne w projektach, które wymagają pracy z zastosowaniem złożonych struktur danych lub tworzenia obszernych bibliotek. Zrozumienie różnych systemów obiektowych pozwala wybrać odpowiednie narzędzia do realizacji konkretnego zadania, co znacząco poprawia efektywność kodu. Poniżej podano przykład klasy S3 `AR1` do analizy stacjonarności zmiennych (szerzej zob. podrozdział 3.2).

```
create.ar.1 <- function(phi, n) {
  if (abs(phi) >= 1) stop("Parametr 'phi' musi być w przedziale (-1,
   ↪  1) dla zachowania stacjonarności.")
  y <- numeric(n)
  for (t in 2:n) {
    y[t] <- phi * y[t - 1] + rnorm(1)
  }
  obj <- list(
    phi = phi,
    y = y
  )
  class(obj) <- "AR1"
  return(obj)
}
```

Następnie zdefiniowano metody dla klasy `AR1`:

- metodę `print`

[30] Walidacja w systemie S4 polega na automatycznym sprawdzaniu poprawności obiektów podczas ich tworzenia i modyfikacji.

```
print.AR1 <- function(obj) {
  cat("Model AR(1):\n")
  cat("phi = ", obj$phi, "\n")
  cat("Szereg y (10 początkowych wartości):\n")
  print(head(obj$y, 10))
}
```

- metodę `plot`

```
plot.AR1 <- function(obj, ...) {
  plot(
    obj$y,
    type = "l",
    main = "Szereg czasowy AR(1)",
    xlab = "t",
    ylab = "Wartość",
    ...
  )
  abline(h = 0, col = "grey", lty = 2)
}
```

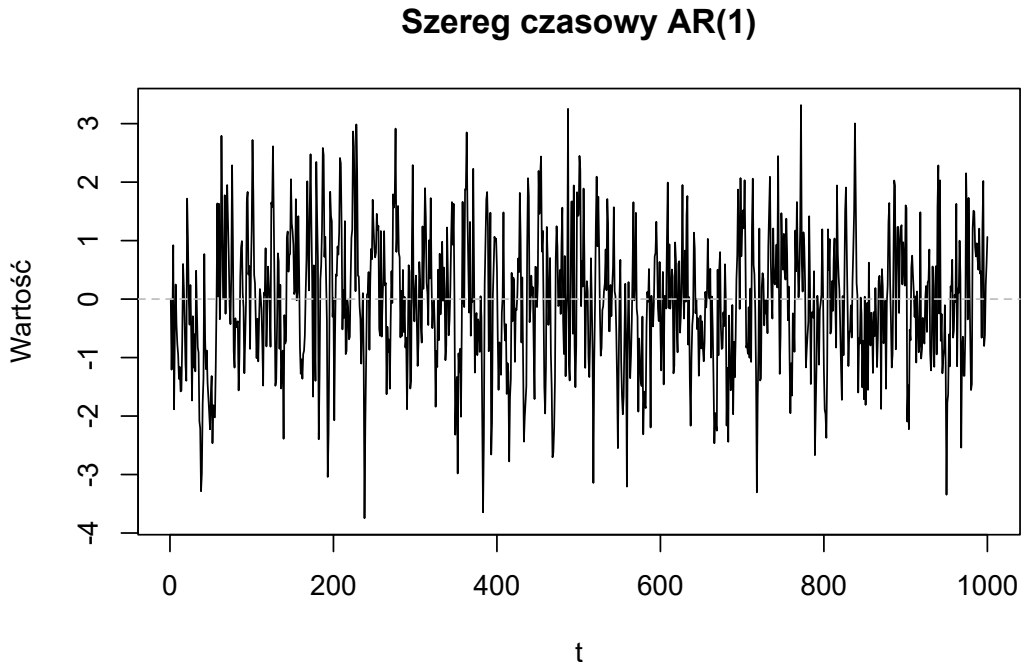
- metodę `summary`

```
summary.AR1 <- function(obj) {
  cat("Podsumowanie modelu AR(1):\n")
  cat("phi = ", obj$phi, "\n")
  cat("Liczba obserwacji = ", length(obj$y), "\n")
  cat("Średnia = ", mean(obj$y), "\n")
  cat("Wariancja = ", var(obj$y), "\n")
}
```

Poniżej podano przykład użycia klasy S3 (por. rys. 2.38).

```
set_seed()
model <- create.ar.1(phi = 0.5, n = 1000)
print(model)
```

```
## Model AR(1):
## phi =  0.5
## Szereg y (10 początkowych wartości):
##  [1]  0.0000000 -1.2070657 -0.3261036  0.9213894 -1.8850030
 ↪  -0.5133768
##  [7]  0.2493675 -0.4500562 -0.7716600 -0.9502820
```



Rysunek 2.38. Szereg czasowy AR(1) – klasa S3

```
summary(model)
```

```
## Podsumowanie modelu AR(1):
## phi =  0.5
## Liczba obserwacji =  1000
## Średnia =  -0.05336557
## Wariancja =  1.345548
```

Następnie zdefiniowano odpowiednik klasy S3 AR1 w formacie klasy S4. Podano następujące składowe[31]:

- definicja klasy

```
setClass(
  "AR1",
  slots = list(
```

[31] W systemie S4 każdy obiekt ma ściśle zdefiniowaną strukturę, która składa się ze *slotów*. Każdy slot ma: nazwę (np. ϕ, y); typ danych, który musi być przestrzegany (np. *numeric*); opcjonalną wartość domyślną ustawioną w *prototypie*.

```
    phi = "numeric",
    y = "numeric"
  ),
  prototype = list(
    phi = NA_real_,
    y = numeric(0)
  )
)
```

- konstruktor[32] dla klasy

```
create.AR1 <- function(phi, n) {
  if (abs(phi) >= 1) stop("Parametr 'phi' musi być w przedziale (-1,
   ↪  1) dla zachowania stacjonarności.")
  y <- numeric(n)
  for (t in 2:n) {
    y[t] <- phi * y[t - 1] + rnorm(1)
  }
  new("AR1", phi = phi, y = y)
}
```

- metoda show (odpowiednik `print`)

```
setMethod(
  "show",
  "AR1",
  function(object) {
    cat("Model AR(1):\n")
    cat("phi = ", object@phi, "\n")
    cat("Szereg y (pierwsze 10 wartości):\n")
    print(head(object@y, 10))
  }
)
```

- metoda `plot`

```
setMethod(
  "plot",
  "AR1",
  function(x, ...) {
    plot(
      x@y,
```

[32] Konstruktor to funkcja lub mechanizm, który tworzy nowy obiekt danej klasy oraz zapewnia, że obiekt ten ma poprawną strukturę i wartości początkowe.

```
      type = "l",
      main = "Szereg czasowy AR(1)",
      xlab = "t",
      ylab = "Wartość",
      ...
    )
    abline(h = 0, col = "grey", lty = 2)
  }
)
```

- metoda `summary`

```
setGeneric(
  "summary",
  function(object) standardGeneric("summary")
)
```

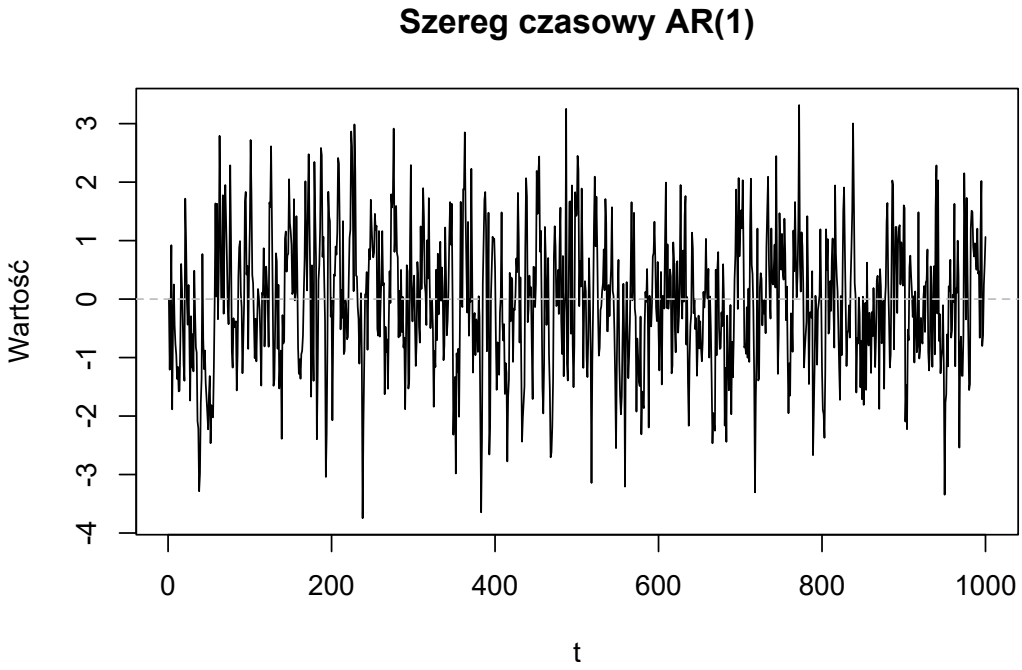
```
## [1] "summary"
```

```
setMethod(
  "summary",
  "AR1",
  function(object) {
    cat("Podsumowanie modelu AR(1):\n")
    cat("phi = ", object@phi, "\n")
    cat("Liczba obserwacji = ", length(object@y), "\n")
    cat("Średnia = ", mean(object@y), "\n")
    cat("Wariancja = ", var(object@y), "\n")
  }
)
```

Poniżej podano przykład użycia klasy S4 (por. rys. 2.39).

```
set_seed()
model <- create.AR1(phi = 0.5, n = 1000)
show(model)
```

```
## Model AR(1):
## phi =  0.5
## Szereg y (pierwsze 10 wartości):
##  [1]  0.0000000 -1.2070657 -0.3261036  0.9213894 -1.8850030
 ↪  -0.5133768
##  [7]  0.2493675 -0.4500562 -0.7716600 -0.9502820
```



Rysunek 2.39. Szereg czasowy AR(1) – klasa S4

```
summary(model)
```

```
## Podsumowanie modelu AR(1):
## phi =  0.5
## Liczba obserwacji =  1000
## Średnia =  -0.05336557
## Wariancja =  1.345548
```

Klasę S4 można rozszerzyć o walidację i dziedziczenie:

- definicja klasy S4 z walidacją

```
setClass(
  "AR1",
  slots = list(
    phi = "numeric",
    y = "numeric"
  ),
  prototype = list(
    phi = NA_real_,
```

```
    y = numeric(0)
  ),
  validity = function(object) {
    if (length(object@phi) != 1 || abs(object@phi) >= 1) {
      return("Parametr 'phi' musi być liczbą w przedziale (-1, 1).")
    }
    if (!is.numeric(object@y)) {
      return("Zmienna 'y' musi być typu numeric.")
    }
    TRUE
  }
)
```

- dodanie dziedziczenia – dziedzicząca klasa `AR1Ext` z dodatkowym slotem

```
setClass(
  "AR1Ext",
  contains = "AR1",
  slots = list(
    label = "character"
  ),
  prototype = list(
    label = "Model AR(1)"
  )
)
```

- konstruktor dla klasy `AR1Ext`

```
create.AR1Ext <- function(phi, n, label = "Model AR(1)") {
  if (abs(phi) >= 1) stop("Parametr 'phi' musi być w przedziale (-1,
   ↪  1) dla zachowania stacjonarności.")
  y <- numeric(n)
  for (t in 2:n) {
    y[t] <- phi * y[t - 1] + rnorm(1)
  }
  new("AR1Ext", phi = phi, y = y, label = label)
}
```

- zmodyfikowane metody `show` i `summary` – dodanie obsługi dodatkowego argumentu `label`

```
setMethod(
  "show",
  "AR1Ext",
```

```
  function(object) {
    cat(object@label, ":\n")
    cat("phi = ", object@phi, "\n")
    cat("Szereg y (pierwsze 10 wartości):\n")
    print(head(object@y, 10))
  }
)
```

- metoda `summary` dla `AR1Ext`

```
setMethod(
  "summary",
  "AR1Ext",
  function(object) {
    cat("Podsumowanie modelu:\n")
    cat("Label: ", object@label, "\n")
    cat("phi = ", object@phi, "\n")
    cat("Liczba obserwacji = ", length(object@y), "\n")
    cat("Średnia = ", mean(object@y), "\n")
    cat("Wariancja = ", var(object@y), "\n")
  }
)
```

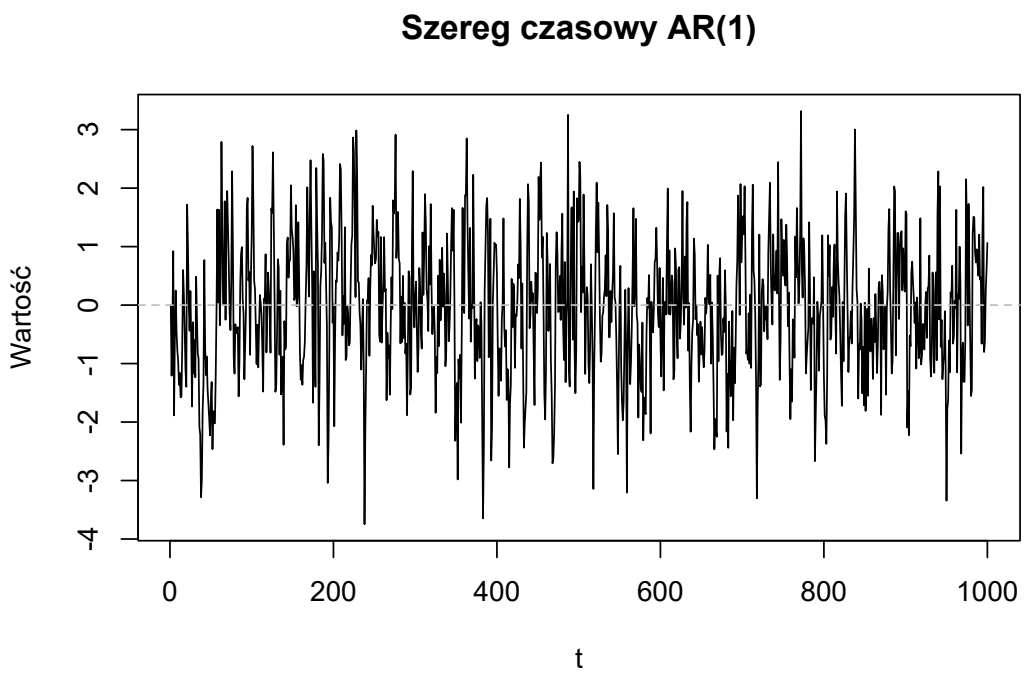
Można teraz podać przykład użycia walidacji i dziedziczenia (por. rys. 2.40).

```
set_seed()
model.1 <- create.AR1(phi = 0.5, n = 1000)
show(model.1)
```

```
## Model AR(1):
## phi =  0.5
## Szereg y (pierwsze 10 wartości):
##  [1]  0.0000000 -1.2070657 -0.3261036  0.9213894 -1.8850030
 ↪  -0.5133768
##  [7]  0.2493675 -0.4500562 -0.7716600 -0.9502820
```

```
summary(model.1)
```

```
## Podsumowanie modelu AR(1):
## phi =  0.5
## Liczba obserwacji =  1000
## Średnia =  -0.05336557
## Wariancja =  1.345548
```



Rysunek 2.40. Szereg czasowy AR(1) – klasa S4 z walidacją i dziedziczeniem – przypadek 1

- tworzenie obiektu klasy `AR1Ext`

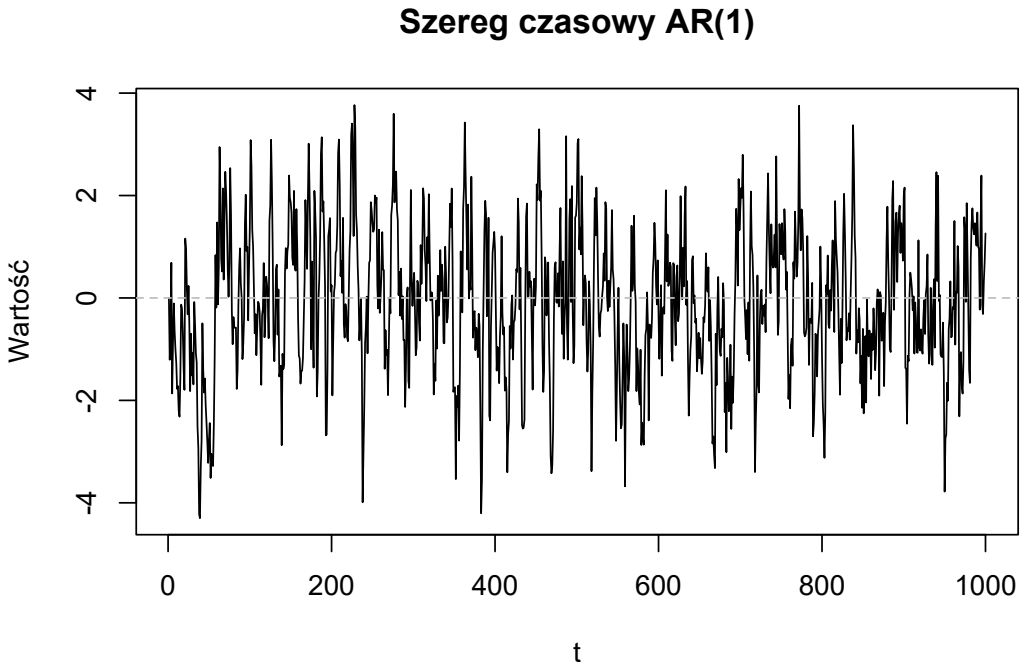
Można teraz podać przykład (por. rys. 2.41).

```
set_seed()
model.2 <- create.AR1Ext(phi = 0.7, n = 1000, label = "Model AR(1)
 ↪  z klasy AR1Ext")
show(model.2)
```

```
## Model AR(1) z klasy AR1Ext :
## phi =  0.7
## Szereg y (pierwsze 10 wartości):
##  [1]  0.0000000 -1.2070657 -0.5675168  0.6871794 -1.8646721
 ↪  -0.8761458
##  [7] -0.1072462 -0.6498123 -1.0015004 -1.2655023
```

```
summary(model.2)
```

```
## Podsumowanie modelu:
```



Rysunek 2.41. Szereg czasowy AR(1) – klasa S4 z walidacją i dziedziczeniem – przypadek 2

```
## Label:  Model AR(1) z klasy AR1Ext
## phi =  0.7
## Liczba obserwacji =  1000
## Średnia =  -0.09011307
## Wariancja =  1.975284
```

- przykład błędnego parametru `phi` (walidacja)

Wykonanie poniższego kodu spowoduje wygenerowanie komunikatu o błędzie.

```
model.3 <- create.AR1(phi = 1.5, n = 100)
```

Klasa `AR1Ext` dziedziczy wszystkie argumenty i metody klasy AR1. W klasie `AR1Ext` dodano nowy argument `label`, który można użyć do opisu modelu. Oznacza to, że rozszerzona klasa zawiera dodatkowe informacje.

Poniżej podano odpowiednik klas AR1 i `AR1Ext` w formacie klasy referencyjnej R6 uwzględniającej dziedziczenie oraz walidację.

```
library(R6)
```

```
AR1 <- R6Class(
  "AR1",
  public = list(
    phi = NULL,
    y = NULL,
    initialize = function(phi, n) {
      if (!is.numeric(phi) || length(phi) != 1 || abs(phi) >= 1) {
        stop("Parametr 'phi' musi być liczbą w przedziale (-1, 1).")
      }
      self$phi <- phi
      self$y <- numeric(n)
      for (t in 2:n) {
        self$y[t] <- self$phi * self$y[t - 1] + rnorm(1)
      }
    },
    print = function() {
      cat("Model AR(1):\n")
      cat("phi = ", self$phi, "\n")
      cat("Szereg y (pierwsze 10 wartości):\n")
      print(head(self$y, 10))
    },
    plot = function(...) {
      plot(
        self$y,
        type = "l",
        main = "Szereg czasowy AR(1)",
        xlab = "t",
        ylab = "Wartość",
        ...
      )
      abline(h = 0, col = "grey", lty = 2)
    },
    summary = function() {
      cat("Podsumowanie modelu AR(1):\n")
      cat("phi = ", self$phi, "\n")
      cat("Liczba obserwacji = ", length(self$y), "\n")
      cat("Średnia = ", mean(self$y), "\n")
      cat("Wariancja = ", var(self$y), "\n")
    }
  )
)
```

Zdefiniowano też klasę R6 `AR1Ext` (dziedziczenie po `AR1`).

```
AR1Ext <- R6Class(
  "AR1Ext",
  inherit = AR1,
  public = list(
    label = NULL,
    initialize = function(phi, n, label = "Model AR(1)") {
      super$initialize(phi, n)
      self$label <- label
    },
    print = function() {
      cat(self$label, ":\n")
      cat("phi = ", self$phi, "\n")
      cat("Szereg y (pierwsze 10 wartości):\n")
      print(head(self$y, 10))
    },
    summary = function() {
      cat("Podsumowanie modelu:\n")
      cat("Label: ", self$label, "\n")
      cat("phi = ", self$phi, "\n")
      cat("Liczba obserwacji = ", length(self$y), "\n")
      cat("Średnia = ", mean(self$y), "\n")
      cat("Wariancja = ", var(self$y), "\n")
    }
  )
)
```

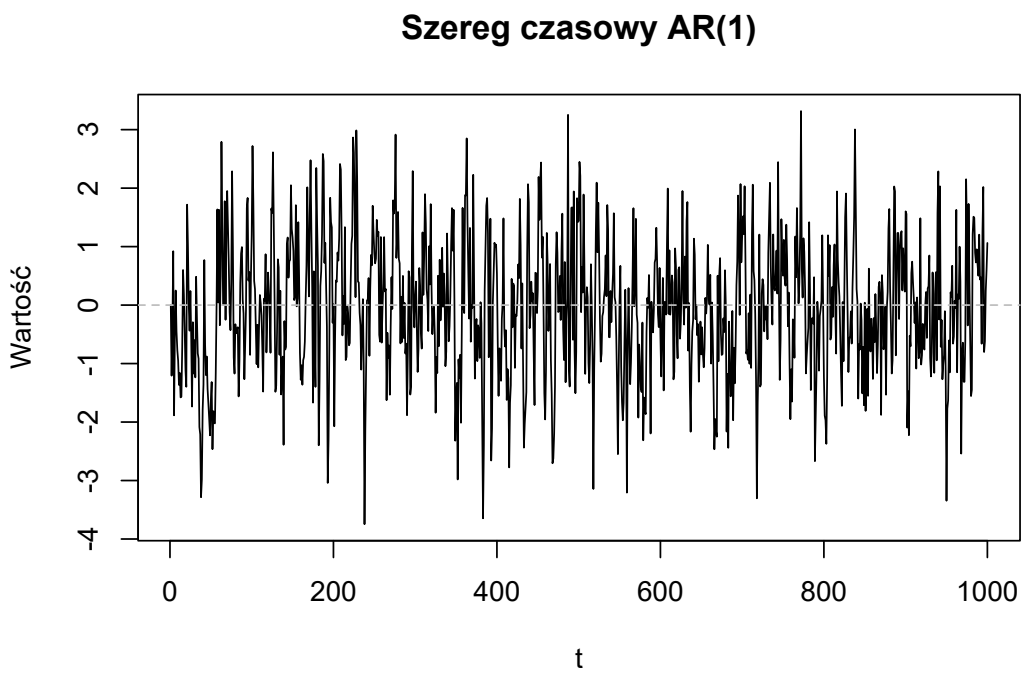
Na koniec podano przykład tworzenia i użycia obiektu R6 dla klas AR1 oraz AR1Ext (por. rys. 2.42 i 2.43).

```
set_seed()
model.1 <- AR1$new(phi = 0.5, n = 1000)
model.1$print()
```

```
## Model AR(1):
## phi =  0.5
## Szereg y (pierwsze 10 wartości):
##  [1]  0.0000000 -1.2070657 -0.3261036  0.9213894 -1.8850030
 ↪  -0.5133768
##  [7]  0.2493675 -0.4500562 -0.7716600 -0.9502820
```

```
model.1$summary()
```

```
## Podsumowanie modelu AR(1):
## phi =  0.5
## Liczba obserwacji =  1000
```



Rysunek 2.42. Szereg czasowy AR(1) – klasa R6

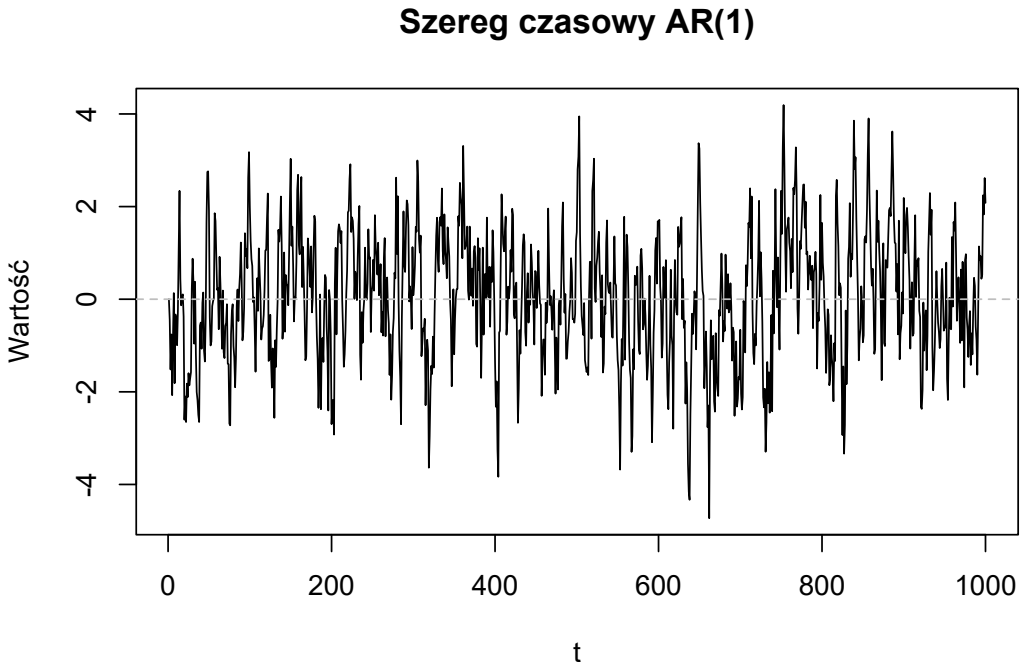
```
## Średnia =  -0.05336557
## Wariancja =  1.345548
```

```
model.2 <- AR1Ext$new(phi = 0.7, n = 1000, label = "Model AR(1)
 ↪  z klasy rozszerzonej")
model.2$print()
```

```
## Model AR(1) z klasy rozszerzonej :
## phi =  0.7
## Szereg y (pierwsze 10 wartości):
##  [1]  0.0000000 -0.4456580 -1.5172940 -0.7606391 -2.0715926
 ↪  -0.8147441
##  [7]  0.1326309 -1.8130412 -0.3302074 -0.4556373
```

```
model.2$summary()
```

```
## Podsumowanie modelu:
## Label:  Model AR(1) z klasy rozszerzonej
## phi =  0.7
```



Rysunek 2.43. Szereg czasowy AR(1) – klasa R6 rozszerzona

```
## Liczba obserwacji =  1000
## Średnia =  0.05057725
## Wariancja =  1.885833
```

Pokazano również przykład użycia błędnej wartości parametru `phi` (walidacja). Wykonanie poniższego kodu spowoduje wygenerowanie komunikatu o błędzie.

```
model.3 <- AR1$new(phi = 1.5, n = 100)
```

Parametr `phi` jest sprawdzany w konstruktorze klasy `AR1` i wyświetlany jest odpowiedni komunikat. Dziedziczona klasa `AR1Ext` automatycznie przejmuje tę walidację dzięki wywołaniu `super$initialize`. Należy podkreślić, że klasa `AR1Ext` dziedziczy wszystkie metody i pola z klasy `AR1`. W nowej klasie dodano pole `label` oraz zmodyfikowano metody `print` i `summary`, aby uwzględniały pole `label`. Klasa R6 pozwala na bardziej zwięzłe definiowanie dziedziczenia w porównaniu do klasy S4. Na zakończenie rozdziału podano zarys zagadnień związanych z tworzeniem dokumentów `R/Markdown`.

2.5. Dynamiczne dokumenty R/Markdown

Dokumenty R/Markdown służą do tworzenia dynamicznych raportów łączących kod R, tekst i wyniki obliczeń w jednym pliku. Umożliwiają automatyczne generowanie dokumentów w różnych formatach (HTML, PDF, DOCX) bez konieczności ręcznego kopiowania wyników. Niniejsza książka została zapisana w postaci dokumentu R/Markdown (Allaire i in., 2025; Xie, 2025a). Szablon dokumentu R/Markdown można wygenerować w RStudio.

```
File -> New File -> R Markdown...
```

Następnie należy wybrać opcję Dokument i podać tytuł, autora, datę dokumentu oraz wynikowy format pliku: HTML, PDF lub Word. Otrzymuje się wówczas dokument o następującej strukturze:

````
---
title: "Dokument"
author: "Piotr Wdowiński"
date: "2024-12-08"
output: pdf_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

## R/Markdown

This is an R Markdown document. Markdown is a simple formatting syntax
 ↪  for authoring HTML, PDF, and MS Word documents. For more details
 ↪  on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that
 ↪  includes both content as well as the output of any embedded R code
 ↪  chunks within the document. You can embed an R code chunk like
 ↪  this:

```{r cars}
summary(cars)
```

## Including Plots

You can also embed plots, for example:

```{r pressure, echo=FALSE}
````
````

```
plot(pressure)
```

Note that the `echo=FALSE` parameter was added to the code chunk to
 ↪ prevent printing of the R code that generated the plot.
```

Dokument kompiluje się do odpowiedniego formatu za pomocą przycisku Knit w RStudio. Należy zauważyć, że kompilacja do formatu PDF wymaga zainstalowania na komputerze dodatkowych programów:

- dystrybucji LaTeX (https://miktex.org/) w pełnej wersji,
- PERL (https://strawberryperl.com/).

Jeśli po zainstalowaniu MiKTeX nie można wygenerować dokumentu PDF, warto skorzystać z biblioteki tinytex (Xie, 2025c)[33].

```
install.packages('tinytex')
tinytex::install_tinytex()
# to uninstall TinyTeX, run tinytex::uninstall_tinytex()
```

TinyTeX to niestandardowa dystrybucja LaTeX oparta na TeX Live, która zajmuje mało miejsca na dysku, ale działa dobrze w większości przypadków, szczególnie dla użytkowników programu R. Jeśli napotka się na problem brakujących bibliotek LaTeX, biblioteka tinytex samodzielnie je zainstaluje. W większości przypadków użytkownicy R nie muszą wykonywać dodatkowych czynności. Warto przy tym pamiętać, że dystrybucja ta jest najczęściej wykorzystywana podczas kompilacji dokumentów R/Markdown, których struktura składa się z kilku charakterystycznych elementów:

1. Dokument otwiera *preambuła*. Może ona zawierać elementy sterujące i nazwy bibliotek języka LaTeX.

```
---
title: "Dokument"
author: "Piotr Wdowiński"
date: "2024-12-08"
output: pdf_document
---
```

2. Kod R (chunk) osadza się w znacznikach. Kod jest wykonywany podczas kompilacji dokumentu za pomocą biblioteki knitr, zaś jego wynik osadzany dynamicznie w dokumencie. Dostępne są szczegółowe argumenty (opcje), za pomocą

[33] Por. https://yihui.org/tinytex/.
```

których kontroluje się sposób wykonania i pokazania kodu w dokumencie. Szerzej por.:

- https://bookdown.org/yihui/rmarkdown/),
- https://bookdown.org/yihui/rmarkdown-cookbook/,
- https://yihui.org/knitr/options/.

````
```{r cars}
summary(cars)
```
````

Należy pamiętać, że element kodu (chunk) w dokumencie R/Markdown może zawierać nazwę, np. name. Ta nazwa ułatwia organizowanie kodu i poprawianie błędów, gdyż jest ona wyświetlana podczas kompilacji dokumentu.

````
```{r name}
# code
```
````

3. W dokumencie można osadzać tekst wraz z wynikami wykonanego kodu, w tym rysunki.

````
```{r pressure, echo=FALSE}
plot(pressure)
```
````

4. W dokumencie można uwzględniać rozdziały i podrozdziały niższego rzędu.

```
# Rozdział 1
## Podrozdział 1
## Podrozdział 2
```

5. Bardziej zaawansowane dokumenty można tworzyć za pomocą biblioteki bookdown (Xie, 2025a). Por. również:

- https://bookdown.org/,
- https://github.com/rstudio/bookdown-demo.

6. Należy zauważyć, że przygotowanie dokumentu R/Markdown do kompilacji w formatach HTML lub PDF obejmuje następujące główne różnice:

- wybór *silnika* generującego (*renderującego*):
  - HTML – Dokument R/Markdown jest generowany przy użyciu *silnika* `html_document`. Można go skonfigurować za pomocą różnych opcji, takich jak styl CSS, wtyczki JavaScript czy dostosowanie układu strony. HTML jest bardziej elastyczny pod względem interaktywności (np. wykresów).
  - PDF – Do generowania pliku PDF używany jest silnik `pdf_document`, który wymaga dodatkowych zależności, takich jak TeX (np. LaTeX). PDF jest mniej elastyczny pod względem interaktywności, ale pozwala na tworzenie profesjonalnie wyglądających dokumentów o stałym układzie.
- stylizacja:
  - HTML – Nadawanie stylów jest oparte głównie na formacie CSS. Można dostosować wygląd strony, dodając własne arkusze stylów CSS. Dodatkowo można używać wtyczek JavaScript (np. do obsługi interaktywnych wykresów).
  - PDF – Nadawanie stylów jest bardziej związane z systemem LaTeX, który używa klas dokumentów i szablonów. Można dostosować wygląd przy pomocy specjalnych poleceń LaTeX, takich jak `\usepackage{}`, jednak zmiany są mniej elastyczne w porównaniu do HTML.
- formatowanie i układ:
  - HTML – Dokumenty HTML są bardziej elastyczne, umożliwiają dynamiczne dostosowywanie układu (np. zmiana szerokości kolumn, responsywność) w zależności od urządzenia. Układ może być prostszy, ale i bardziej interaktywny.
  - PDF – W przypadku PDF dokument jest formatowany z zachowaniem stałego układu stron. LaTeX pozwala na precyzyjne dostosowanie układu, ale nie jest tak elastyczny jak HTML w kontekście responsywności.
- biblioteki:
  - HTML – Możliwość użycia bibliotek, takich jak `plotly`, `leaflet` i innych, które pozwalają na tworzenie interaktywnych elementów (np. interaktywnych wykresów, map itd.).
  - PDF – PDF nie obsługuje interaktywnych wykresów i map. Wszystkie wykresy są generowane w formie statycznej, ale mają wysoką jakość druku.
- zależności:
  - HTML – Wymaga się jedynie instalacji programu R i biblioteki `rmarkdown`, a generowanie dokumentu odbywa się za pomocą przeglądarki.
  - PDF – Wymaga się zainstalowanego systemu LaTeX (np. TeX Live, MikTeX), który jest używany do kompilacji dokumentu. Często mogą wystąpić problemy z brakującymi bibliotekami LaTeX, które trzeba zainstalować.
- wydajność:

  - HTML – Generowanie HTML jest szybkie i nie wymaga wielu zasobów, ponieważ nie jest konieczne generowanie skomplikowanego układu.
  - PDF – Generowanie dokumentu PDF może pochłaniać więcej zasobów, zwłaszcza przy dużych dokumentach, ponieważ wymaga przetwarzania z udziałem języka LaTeX.
- tablice i wykresy:
  - HTML – Tablice i wykresy są generowane bezpośrednio na stronie internetowej i mogą być interaktywne. Można dostosować ich wygląd przy pomocy znaczników CSS.
  - PDF – Tablice i wykresy są generowane w formie statycznej. LaTeX pozwala na precyzyjne dostosowanie tablic i wykresów, ale nie uwzględnia ich interaktywności.

Podsumowując, format HTML oferuje większą elastyczność, interaktywność i łatwiejszą personalizację wyglądu dokumentu, podczas gdy PDF zapewnia profesjonalny wygląd dokumentu o stałym układzie, z możliwością zaawansowanego formatowania dzięki językowi LaTeX.

Na zakończenie tej części książki warto zaznaczyć, że wcześniejsze podrozdziały stanowiły wprowadzenie do podstaw języka R, jego struktur danych, funkcji, możliwości wizualizacji oraz automatyzacji zadań. Umiejętności te stanowią fundament pracy analitycznej i są niezbędne do realizacji bardziej zaawansowanych zadań badawczych.

W kolejnej części książki zostanie przedstawione zagadnienie modelowania ekonometrycznego – jednego z kluczowych obszarów zastosowań programu R. Zostaną omówione podstawowe pojęcia z zakresu estymacji i testowania liniowych modeli ekonometrycznych, które pozwalają na ilościową analizę zależności gospodarczych i weryfikację hipotez ekonomicznych na podstawie danych empirycznych.

Rozdział 3

Modelowanie ekonometryczne

3.1. Wprowadzenie

Program R może być wykorzystywany do przeprowadzania zaawansowanych obliczeń. Jednym z głównych obszarów jego zastosowań jest *ekonometria*. Za pomocą metod i modeli ekonometrycznych można mierzyć zależności ekonomiczne. Modele ekonometryczne stanowią istotny element badań stosowanych, także w innych dziedzinach. Znajdują zastosowanie wszędzie tam, gdzie zachodzi potrzeba oszacowania związków pomiędzy zmiennymi na podstawie danych pochodzących z próby statystycznej.

Model ekonometryczny to probabilistyczne ujęcie zależności ekonomicznej, uzyskane poprzez sformalizowanie teorii ekonomii w ramach modelu statystycznego (Greene, 2003). Budowa modelu obejmuje kilka etapów:

1. Specyfikację – określenie zmiennych wchodzących w skład modelu, postaci funkcji łączącej te zmienne oraz przyjęcie założeń dotyczących jego struktury stochastycznej.
2. Określenie sposobu pomiaru zmiennych i zebranie materiału statystycznego dla poszczególnych zmiennych.
3. Oszacowanie parametrów.
4. Weryfikację merytoryczną i statystyczną, czyli ocenę znaku i wartości oszacowanych parametrów, testowanie hipotez statystycznych, a także określenie stopnia dopasowania modelu do danych empirycznych.
5. Ocenę modelu – przyjęcie lub odrzucenie modelu i ewentualne skierowanie go do dalszych prac.
6. Zastosowanie modelu w praktyce gospodarczej.

Biorąc powyższe pod uwagę, model ekonometryczny można określić jako narzędzie służące do weryfikacji hipotez ekonomicznych. Przez hipotezę ekonomiczną rozumie się relację łączącą zmienne występujące w systemie ekonomicznym. Model ekonometryczny jest modelem stochastycznym, w którym występuje *składnik losowy*, odzwierciedlający błędy specyfikacji modelu. W modelu tym obecna jest niepewność związana z nieznanym charakterem relacji ekonomicznej, która objawia się w postaci błędów szacunków parametrów modelu.

Modele ekonometryczne dzieli się na modele jedno- i wielorównaniowe. Modele jednorównaniowe opisują pojedynczą relację ekonomiczną, natomiast modele wielorównaniowe – wiele takich relacji. W modelach jednorównaniowych występuje relacja stochastyczna, natomiast w modelach wielorównaniowych – relacje stochastyczne, często uzupełniane relacjami tożsamościowymi, które nie mają charakteru stochastycznego. W takich przypadkach model wielorównaniowy również ma charakter stochastyczny.

Jednym z istotniejszych celów budowy modelu ekonometrycznego jest ocena kształtowania się relacji ekonomicznych w horyzoncie *ex post* oraz *ex ante*. Ocena *ex post* służy analizie mechanizmów kształtowania się zjawisk ekonomicznych w ujęciu historycznym, natomiast ocena *ex ante* umożliwia prognozowanie przebiegu tych zjawisk w przyszłości na podstawie danych historycznych. Jakość prognostyczną modelu można oceniać zarówno w ujęciu *ex post*, jak i *ex ante*, przy zastosowaniu odpowiednich mierników (Milo, 2002).

W kolejnych podrozdziałach zostaną omówione podstawowe zagadnienia:

- procesów stochastycznych,
- jednorównaniowych modeli ekonometrycznych w ujęciu klasycznej analizy regresji,
- testowania poprawności specyfikacji modelu ekonometrycznego.

Podrozdziały 3.2.1 i 3.2.2 powstały na podstawie książki Wdowiński (2010).

3.2. Procesy stochastyczne

3.2.1. Modele

Podstawowym pojęciem wykorzystanym w analizie szeregów czasowych będzie proces stochastyczny. Przez proces stochastyczny – nazywany dalej procesem – rozumie się rodzinę rzeczywistych zmiennych losowych, indeksowaną przez t, gdzie t oznacza czas (Charemza i Deadman, 1997). Proces oznacza się jako zbiór $\{X_t\}$. Elementy $X_1, X_2, \ldots, X_t$ procesu $\{X_t\}$ są zmiennymi losowymi. Zmienna losowa X_t dla ustalonego $t \in T$ nazywana jest wartością procesu w chwili t. Wartości

przyjmowane przez zmienne losowe X_t nazywane są stanami procesu i tworzą przestrzeń stanów. W dalszej części proces $\{X_t\}$ zostanie oznaczony symbolem X_t. Procesy zwykle oznacza się symbolami X_t, Y_t.

Niech T będzie podzbiorem zbioru liczb rzeczywistych. Jeśli T jest zbiorem dyskretnym (np. podzbiorem zbioru liczb całkowitych), mówi się o procesach w czasie dyskretnym. Jeśli natomiast T jest przedziałem – to o procesach w czasie ciągłym. Milo (1990) zaproponował, aby oznaczenie X_t stosować do procesów w czasie dyskretnym, a oznaczenie $X(t)$ – do procesów w czasie ciągłym.

Z procesem wiąże się pojęcie *stacjonarności*. Niech wartość oczekiwana procesu X_t wynosi μ, wariancja σ^2, natomiast kowariancja $cov(X_t, X_{t-j})$ wynosi η_j. Proces X_t jest *słabo* stacjonarny, tj. w ograniczeniu do średnich, wariancji i kowariancji, jeśli:

$$\begin{aligned} E(X_t) &= \mu = const \\ var(X_t) &= \sigma^2 = const \\ cov(X_t, X_{t-j}) &= \eta_j \end{aligned} \tag{3.1}$$

Z (3.1) wynika, że wartość oczekiwana i wariancja procesu są stałe w czasie, natomiast kowariancja pomiędzy dwoma momentami obserwacji zależy wyłącznie od odstępu czasowego $t - j$ między nimi, a nie od samych momentów obserwacji. Proces uznaje się za *niestacjonarny*, jeśli którykolwiek z tych warunków nie jest spełniony.

Proces niestacjonarny można zilustrować następującym przykładem (Charemza i Deadman, 1997). Rzuca się symetryczną i nieobciążoną monetą. Jeśli wypadnie reszka, idzie się krok w prawo. Jeśli orzeł, krok w lewo. Taki proces można opisać za pomocą ciągu zmiennych losowych Z_t. Każda zmienna losowa przyjmuje dwie wartości:

$$Z_t = \begin{cases} -1 & \text{z prawd. } 0.5 \\ +1 & \text{z prawd. } 0.5 \end{cases} \tag{3.2}$$

Każda zmienna losowa Z_t ma wartość oczekiwaną równą zeru, $E(Z_t) = 0$, wariancję równą jeden, $var(Z_t) = 1$. Zmienne losowe Z_t są niezależne, zaś proces ruchu zgodnie ze wskazaniami rzutu monetą można opisać w następujący sposób:

$$\begin{aligned} X_1 &= Z_1 \\ X_2 &= X_1 + Z_2 \\ X_3 &= X_2 + Z_3 \\ &\dots \\ X_t &= X_{t-1} + Z_t \end{aligned} \tag{3.3}$$

Wówczas:

$$E(X_t) = 0$$
$$var(X_t) = t$$

gdyż:

$$var(X_1) = var(Z_1) = 1$$
$$var(X_2) = var(X_1) + var(Z_2) = 1 + 1 = 2 \text{ itd.}$$

Widać, że wariancja procesu X_t jest liniową funkcją czasu, zatem ten proces jest niestacjonarny. Równanie (3.3) określa niestacjonarny proces *błądzenia losowego* (*random walk*). Jeśli moneta jest obciążona, to dodatkowo średnia procesu Z_t jest różna od zera, zaś średnia procesu X_t staje się funkcją czasu t i proces jest nadal niestacjonarny. Kolejnym przykładem procesu niestacjonarnego jest:

$$X_t = \mu + X_{t-1} + Z_t, \quad \mu \neq 0 \tag{3.4}$$

Proces (3.4) określa się mianem procesu błądzenia losowego z *przesunięciem* (*dryfem*) (*random walk with drift*), co oznaczałoby krok np. w prawo przed każdym rzutem monetą.

Następnym pojęciem jest *szereg czasowy*. Szeregiem czasowym $x_t = [x(t_1), \ldots, x(t_n)]'$ nazywa się uporządkowany względem t ciąg próbkowych realizacji zmiennych losowych $X(t_1), \ldots, X(t_n)$. Innymi słowy, szeregiem czasowym $x_t \in R^n$ określa się próbkową realizację rzeczywistego procesu losowego X_t (Milo, 1990; Osińska, 2006). W przypadku dyskretnym szereg czasowy odpowiadający procesowi X_t można zapisać w postaci (Milo, 1990):

$$x_{t_{t=1}}^{n} = x_1, \ldots, x_n \tag{3.5}$$

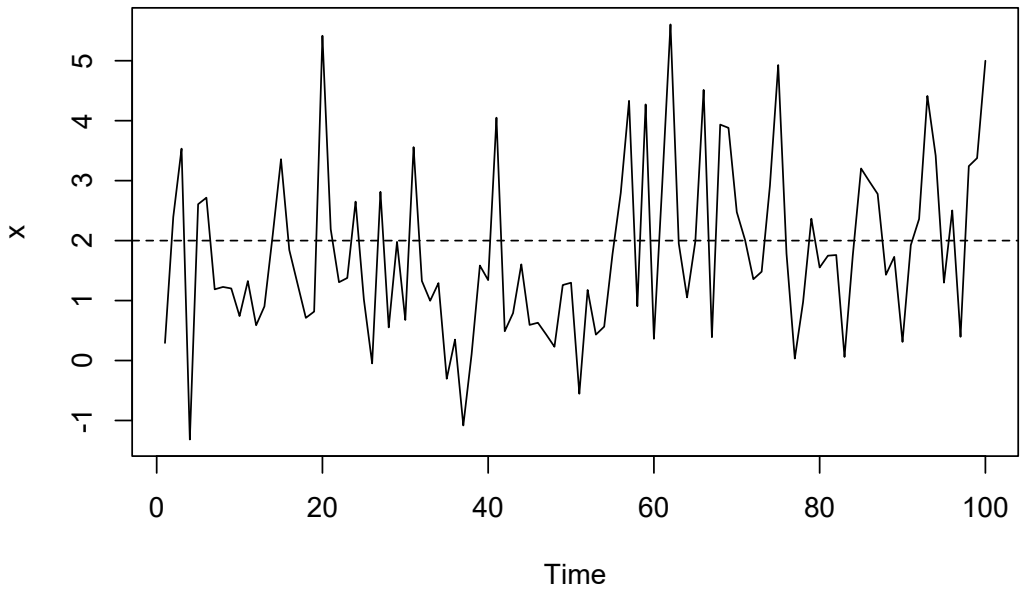
Dla uproszczenia proces stochastyczny będzie utożsamiany z szeregiem czasowym. Przyjęto, że symbol ϵ_t oznacza szereg niezależnych zmiennych losowych ciągłych o jednakowym rozkładzie ze średnią równą zeru, tj. w miejsce Z_t stosuje się ϵ_t. Proces ϵ_t jest *białym szumem*. Jeśli ten proces ma dodatkowo rozkład normalny, to nazywa się go *gaussowskim* białym szumem (Hamilton, 1994). Taki proces ma skończoną wartość oczekiwaną i wariancję. Ponadto przyjęto symbol x_t na oznaczenie szeregu czasowego w miejsce procesu o symbolu X_t.

Poniżej podano przykład szeregu stacjonarnego, $x_t \sim N(2, 2)$, wygenerowany w programie R (rys. 3.1). Na początku analizy wczytano biblioteki, w tym `latex2exp`

(Meschiari, 2022), która zawiera funkcję TeX do transformacji kodu LaTeX do postaci wyrażenia stosowanego w opisie osi wykresów.

```
library(dplyr)
library(latex2exp)
library(magrittr)
```

```
set_seed()
white.noise <- rnorm(
  n = 100,
  mean = 2,
  sd = sqrt(2)
) %>%
    data.frame(x = .) %$%
    ts.plot(x)
abline(h = 2, col = "black", lty = "dashed")
```



Rysunek 3.1. Gaussowski proces białego szumu

Do rys. 3.1 dodano linię prostą za pomocą funkcji abline, przedstawiającą wartość średnią szeregu x_t. Należy zwrócić uwagę, że w funkcji rnorm podano odchylenie standardowe sd nie zaś wariancję.

Jak wspomniano wcześniej, przykładem procesu niestacjonarnego jest proces błądzenia losowego lub proces błądzenia losowego z przesunięciem. Postać tego ostatniego można opisać wzorem (Greene, 2003):

$$x_t = \mu + x_{t-1} + \epsilon_t \tag{3.6}$$

gdzie ϵ_t to wcześniej opisany proces gaussowskiego białego szumu, tj. ciąg niezależnych zmiennych losowych o jednakowym rozkładzie, $\epsilon_t \sim \text{iid}(0, \sigma_\epsilon^2)$. Wówczas:

$$x_t = \sum_{i=0}^{t}(\mu + \epsilon_i) = \mu t + \sum_{i=0}^{t}\epsilon_i \tag{3.7}$$

Jak wynika z (3.7), wartość oczekiwana i wariancja procesu x_t są funkcjami czasu. Zatem proces x_t jest niestacjonarny. Jednak proces:

$$z_t = x_t - x_{t-1} = \mu + \epsilon_t \tag{3.8}$$

jest procesem stacjonarnym o wartości oczekiwanej $E(z_t) = \mu$ i wariancji $var(z_t) = \sigma_\epsilon^2$.

Poniżej podano przykład procesu niestacjonarnego (rys. 3.2). Oś Y opisano za pomocą równania LaTeX.

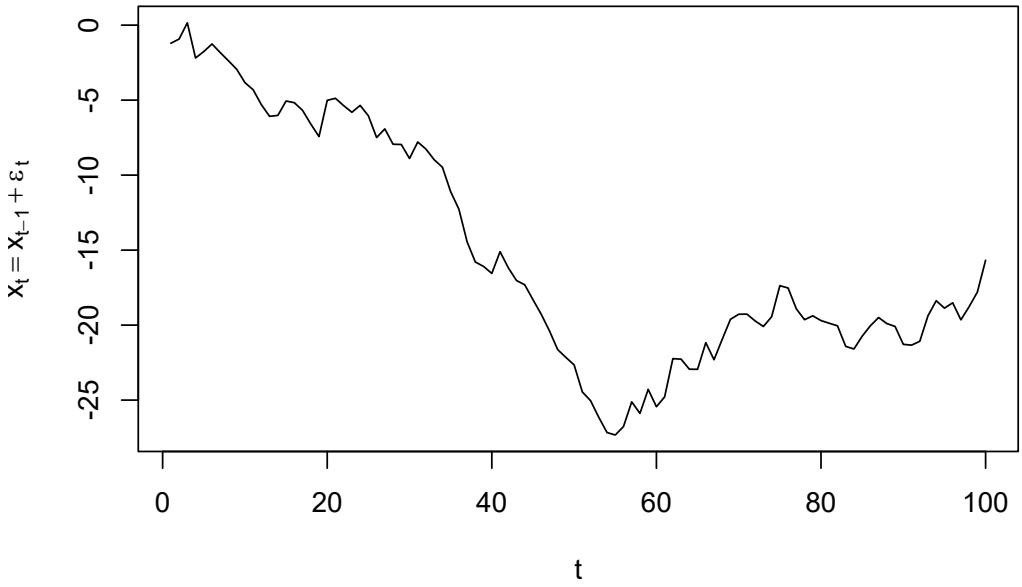
```
set_seed()
nonst <- rnorm(
  n = 100,
  mean = 0,
  sd = 1
) %>%
    cumsum() %>%
    data.frame(x = .) %$%
    ts.plot(
        x,
        xlab = "t",
        ylab = TeX("$x_t=x_{t-1}+\\epsilon_t$")
    )
```

Można wyróżnić następujące przykłady innych procesów stochastycznych (Tsay, 2010):

1. **Liniowy**

$$x_t = \mu + \sum_{i=0}^{\infty}\phi_i\epsilon_{t-i} \tag{3.9}$$

gdzie μ jest średnią procesu x_t, $\phi_0 = 1$, $\epsilon_t \sim \text{iid}(0, \sigma_\epsilon^2)$. Jeśli x_t jest procesem słabo stacjonarnym, to:



Rysunek 3.2. Proces niestacjonarny

$$E(x_t) = \mu, \quad var(x_t) = \sigma_\epsilon^2 \sum_{i=0}^{\infty} \phi_i^2 \tag{3.10}$$

2. **Autoregresja pierwszego rzędu z wyrazem wolnym** – AR(1)

$$x_t = \phi_0 + \phi_1 x_{t-1} + \epsilon_t \tag{3.11}$$

gdzie: $\epsilon_t \sim \text{iid}(0, \sigma_\epsilon^2)$.

Wiadomo ponadto, że:

$$E(x_t|x_{t-1}) = \phi_0 + \phi_1 x_{t-1}, \quad var(x_t|x_{t-1}) = var(\epsilon_t) = \sigma_\epsilon^2 \tag{3.12}$$

Jeśli x_t jest szeregiem słabo stacjonarnym, to:

$$E(x_t) = \frac{\phi_0}{1-\phi_1}, \quad var(x_t) = \frac{\sigma_\epsilon^2}{1-\phi_1^2} \tag{3.13}$$

Z powyższego wynika warunek konieczny i wystarczający słabej stacjonarności szeregu AR(1):

$$-1 < \phi_1 < 1 \tag{3.14}$$

Poniżej przedstawiono przykład szeregu AR(1) dla przyjętych parametrów o postaci $x_t = 0.2 + 0.8x_{t-1} + \epsilon_t$ (rys. 3.3).

```
mean <- 1
ar   <- 0.8
```

Zastosowano funkcję `arima.sim` z biblioteki `stats`. Ta funkcja zawiera wiele argumentów, z których najważniejsze to:

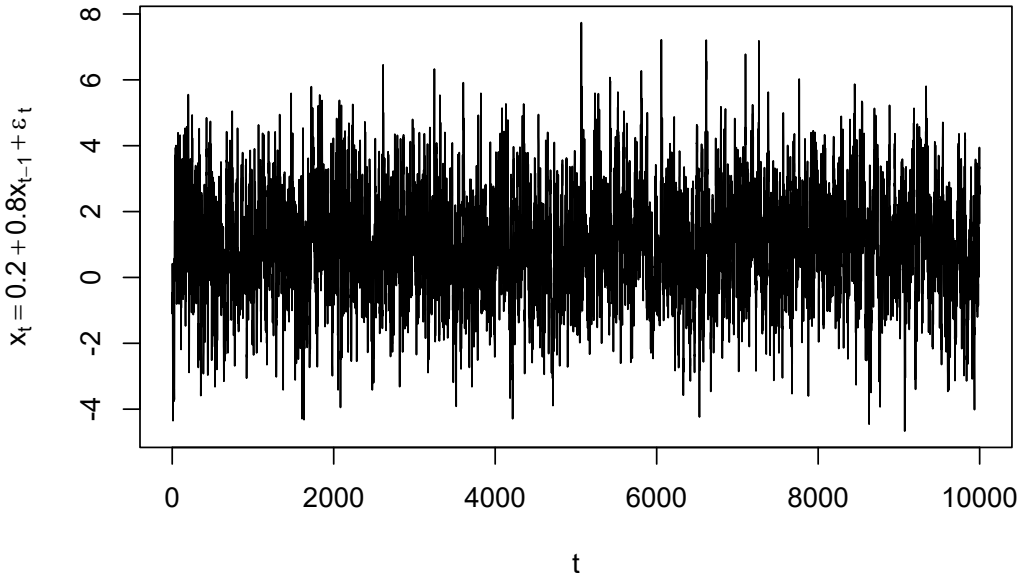
- `model` – lista parametrów modelu,
- `n` – liczba obserwacji.

```
set_seed()
ar.1 <- arima.sim(
    model = list(
      order = c(1, 0, 0),
      ar = ar),
    n = 10^6
) %>%
  {.+mean}
mean(ar.1)
```

```
## [1] 1.002233
```

```
ar.1[1:10000] %>%
    data.frame(x = .) %$%
    ts.plot(
        x,
        xlab = "t",
        ylab = TeX(
          paste0(
            "$x_t=",
            mean*(1-ar),
            "+",
            ar,
            "x_{t-1}+\\epsilon_t$"
          )
        )
    )
```

Warto zwrócić uwagę, że w powyższym kodzie do szeregu AR(1) dodano wartość 1, odpowiadającą bezwarunkowej średniej $E(x_t)$. Ponieważ wartość parametru



Rysunek 3.3. Proces stacjonarny AR(1)

$\phi_1 = 0.8$ spełnia warunek stacjonarności procesu AR(1), to $\phi_0 = E(x_t)(1 - \phi_1) = 0.2$ (wyraz wolny).

3. **Średnia ruchoma pierwszego rzędu** – MA(1)

$$x_t = \gamma_0 + \epsilon_t + \theta_1 \epsilon_{t-1} \tag{3.15}$$

gdzie: γ_0 jest stałą, $\epsilon_t \sim \text{iid}(0, \sigma_\epsilon^2)$.

Modele MA są zawsze słabo stacjonarne, gdyż są skończonymi kombinacjami liniowymi ciągów ϵ_t zmiennych losowych o jednakowym rozkładzie. Ponadto:

$$E(x_t) = \gamma_0, \quad var(x_t) = \sigma_\epsilon^2 + \theta_1^2 \sigma_\epsilon^2 = (1 + \theta_1^2)\sigma_\epsilon^2 \tag{3.16}$$

Należy zauważyć, że ϵ_t i ϵ_{t-1} są nieskorelowane. Poniżej podano przykład szeregu MA(1) dla przyjętych parametrów o postaci $x_t = 0.2 + \epsilon_t - 0.6\epsilon_{t-1}$ (rys. 3.4).

```
mean <-  0.2
ma   <- -0.6
```

Ponownie zastosowano funkcję `arima.sim`.

```
set_seed()
ma.1 <- arima.sim(
    model = list(
      order = c(0, 0, 1),
      ma = ma),
    n = 10^6
) %>%
  {.+mean}
mean(ma.1)
```

```
## [1] 0.2001772
```

```
    ma.1[1:1000] %>%
      data.frame(x = .) %$%
    ts.plot(
        x,
        xlab = "t",
        ylab = TeX(
          paste0(
            "$x_t=",
            mean,
            "+\\epsilon_t",
            ma,
            "\\epsilon_{t-1}$"
          )
      )
    )
```

Należy zwrócić uwagę, że do szeregu MA(1) dodano wartość 0.2 (wyraz wolny szeregu x_t).

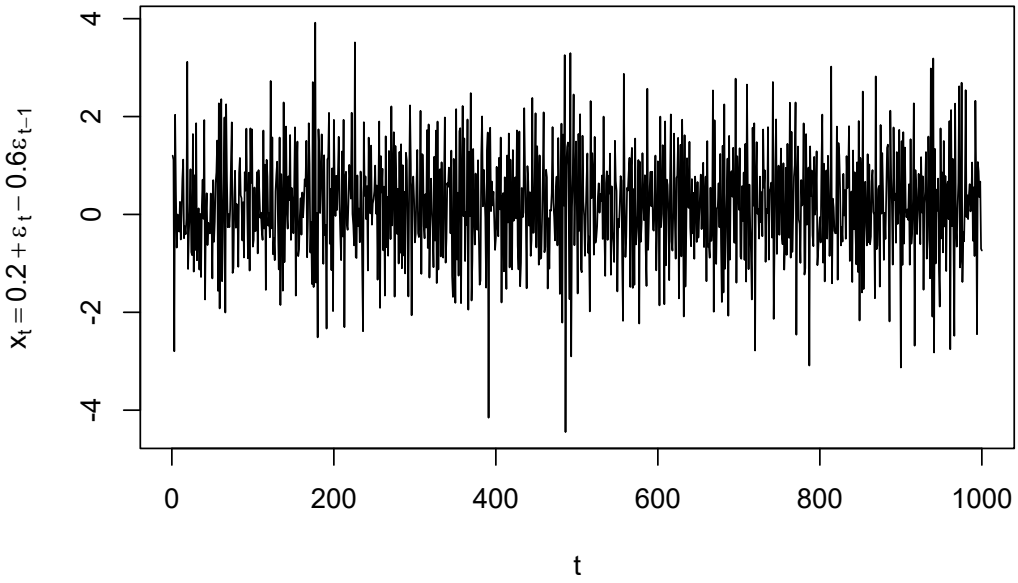
4. **ARMA(1, 1)**

$$x_t - \phi_1 x_{t-1} = \phi_0 + \epsilon_t + \theta_1 \epsilon_{t-1} \tag{3.17}$$

czyli proces łączący składnik autoregresyjny (AR) (rzędu pierwszego) i składnik średniej ruchomej (MA) (rzędu pierwszego), gdzie $\epsilon_t \sim \text{iid}(0, \sigma_\epsilon^2)$ oraz $\phi_1 \neq \theta_1$. Warunkiem słabej stacjonarności procesu jest spełnienie nierówności $|\phi_1| < 1$. Jeśli x_t jest procesem słabo stacjonarnym, to:

$$E(x_t) = \frac{\phi_0}{1 - \phi_1} \tag{3.18}$$

Wyraz wolny ϕ_0 kształtuje wartość oczekiwaną procesu, natomiast nie wpływa na jego wariancję. Przy założeniu stacjonarności x_t wariancja procesu jest dana wzorem:



Rysunek 3.4. Proces stacjonarny MA(1)

$$var(x_t) = \frac{(1 + 2\phi_1\theta_1 + \theta_1^2)\sigma_\epsilon^2}{1 - \phi_1^2} \tag{3.19}$$

Poniżej podano przykład szeregu ARMA(1, 1) o postaci $x_t - 0.8x_{t-1} = 0.2 + \epsilon_t - 0.6\epsilon_{t-1}$ (rys. 3.5).

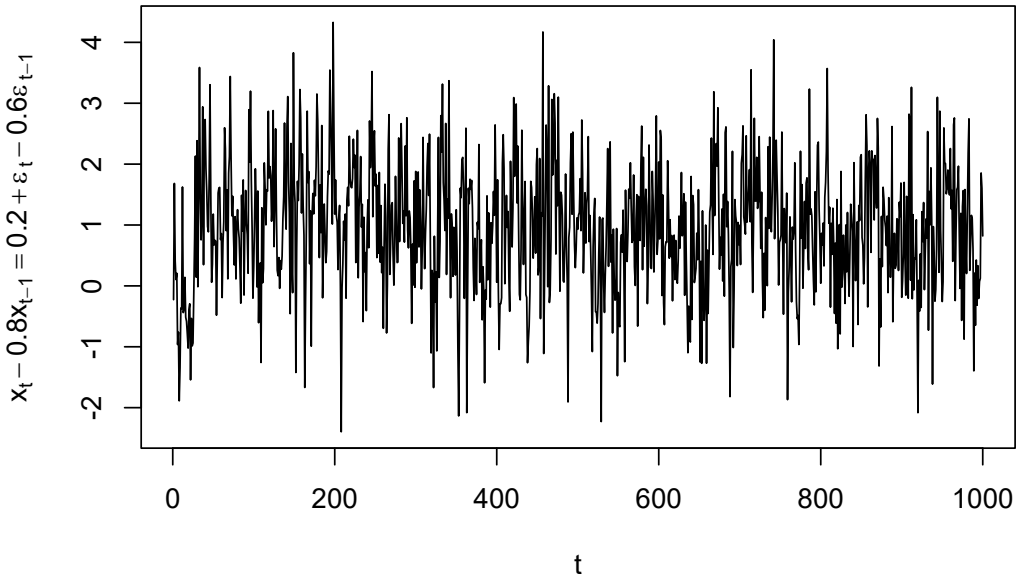
```
mean <-  1
ar   <-  0.8
ma   <- -0.6
```

Ponownie zastosowano funkcję `arima.sim`.

```
set_seed()
arma.1.1 <- arima.sim(
    model = list(
      order = c(1, 0, 1),
      ar = ar,
      ma = ma),
    n = 10^6
) %>%
  {.+mean}
mean(arma.1.1)
```

```
## [1] 1.000897
```

```
arma.1.1[1:1000] %>%
  data.frame(x = .) %$%
ts.plot(
    x,
    xlab = "t",
    ylab = TeX(
      paste0(
        "$x_t-", ar,
        "x_{t-1}=",
        mean*(1-ar),
        "+\\epsilon_t",
        ma,
        "\\epsilon_{t-1}$"
      )
    )
)
```



Rysunek 3.5. Proces stacjonarny ARMA(1,1)

W powyższym kodzie należy zwrócić uwagę, że do szeregu ARMA(1, 1) dodano wartość 1, czyli bezwarunkową średnią $E(x_t)$ szeregu. Ponieważ parametr $\phi_1 = 0.8$ spełnia warunek stacjonarności szeregu ARMA(1, 1), to $\phi_0 = E(x_t)(1 - \phi_1) = 0.2$ (wyraz wolny). Stanowi to identyczne warunki jak dla podanego wcześniej szeregu AR(1).

Należy również wspomnieć, że model błądzenia losowego jest szczególnym przypadkiem modelu AR(1) dla $\phi_0 = 0$ i $\phi_1 = 1$. Ilustrację procesu niestacjonarnego błądzenia losowego podano wcześniej (rys. 3.2).

W następnym podrozdziale zostaną omówione zagadnienia związane z testowaniem stacjonarności szeregów czasowych.

3.2.2. Stacjonarność

Szereg x_t jest niestacjonarny, zintegrowany w stopniu pierwszym ($x_t \sim I(1)$), jeśli jego pierwsze różnice tworzą szereg stacjonarny. Taki szereg można również przedstawić w postaci:

$$(1 - L)x_t = \mu + \epsilon_t \tag{3.20}$$

gdzie L oznacza operator opóźnienia, tj. $Lx_t = x_{t-1}$.

Równanie charakterystyczne dla szeregu opisanego równaniem (3.20) ma jeden pierwiastek równy jeden. Oznacza to, że szereg zawiera pierwiastek jednostkowy, a zatem jest niestacjonarny. W związku z tym testowanie stacjonarności szeregów czasowych sprowadza się do weryfikacji hipotezy o istnieniu pierwiastków jednostkowych. Najczęściej stosowanym narzędziem w tym celu jest test Dickey'a–Fullera (DF) (Dickey i Fuller, 1979). Dla ilustracji tego testu należy posłużyć się modelem autoregresji pierwszego rzędu AR(1):

$$x_t = \gamma x_{t-1} + \epsilon_t \tag{3.21}$$

gdzie: $\epsilon_t \sim N(0, \sigma_\epsilon^2)$, $cov(\epsilon_t, \epsilon_s) = 0$ dla $t \neq s$.

Wśród własności modelu (3.21) można wyróżnić:

$$\begin{aligned} E(x_t|x_{t-1}) &= \gamma x_{t-1} \\ var(x_t|x_{t-1}) &= var(\epsilon_t) = \sigma_\epsilon^2 \end{aligned} \tag{3.22}$$

Można wykazać (Tsay, 2010), że jeśli $|\gamma| < 1$, to proces (3.21) jest słabo stacjonarny. Wówczas wartość oczekiwana, wariancja i autokowariancje przyjmują wartości skończone.

W teście DF dla procesu losowego (3.21) stawia się następujące hipotezy:

$$\begin{aligned} H_0 &: \gamma = 1 \\ H_1 &: \gamma < 1 \end{aligned} \tag{3.23}$$

Można zbudować dwie statystyki testowe przy założeniu prawdziwości hipotezy zerowej:

- tradycyjną statystykę t:

$$DF = \frac{\hat{\gamma} - 1}{S(\hat{\gamma})} \tag{3.24}$$

dla której przy podejmowaniu decyzji przyjmuje się inne wartości krytyczne – ogólnie rzecz biorąc wyższe – niż w rozkładzie t-Studenta[1]. Wartości krytyczne testu DF zostały obliczone symulacyjnie metodami Monte Carlo przez Dickey'a i Fullera (Fuller, 1976; Dickey i Fuller, 1981).

- statystykę:

$$DF_{\gamma} = T(\hat{\gamma} - 1) \tag{3.25}$$

którą porównuje się z wartościami krytycznymi z opracowania Fuller (1976).

Jeśli wartość statystyki testowej jest mniejsza od odpowiedniej wartości krytycznej przy ustalonym poziomie istotności (np. $\alpha = 0.01$, tj. 1%), to hipotezę zerową odrzuca się na rzecz hipotezy alternatywnej. Wówczas przyjmuje się, że szereg czasowy jest stacjonarny.

Jeśli zakłócenia ϵ_t podlegają autokorelacji, należy zastosować rozszerzony test Dickey'a–Fullera (ADF), wynikający z modelu AR(p):

$$x_t = \gamma_0 + \gamma_1 x_{t-1} + \gamma_2 x_{t-2} + \cdots + \gamma_p x_{t-p} + \epsilon_t \tag{3.26}$$

Test ADF uwzględniający stałą oraz trend liniowy można zapisać w postaci tzw. regresji pomocniczej:

$$\Delta x_t = \mu + \beta t + \tilde{\gamma} x_{t-1} + \sum_{i=1}^{p-1} \phi_i \Delta x_{t-i} + \epsilon_t \tag{3.27}$$

gdzie:

$$\begin{aligned} \tilde{\gamma} &= \left(\sum_{i=1}^{p} \gamma_i \right) - 1 \\ \phi_i &= -\sum_{j=i+1}^{p} \gamma_j \quad \text{dla } i = 1, \ldots, p-1 \end{aligned} \tag{3.28}$$

[1] Inne wartości krytyczne wynikają z tego, że asymptotyczny rozkład statystyki DF nie jest rozkładem t-Studenta.

Wykorzystując model (3.27) stawia się dwie hipotezy, przy czym hipoteza zerowa jest hipotezą łączną:

$$\begin{aligned} H_0 &: \beta = \tilde{\gamma} = 0 \\ H_1 &: \tilde{\gamma} < 0 \end{aligned} \tag{3.29}$$

Do weryfikacji tych hipotez stosuje się statystykę DF, której wartości krytyczne – podobnie jak w klasycznym teście DF – odbiegają od wartości z rozkładu t-Studenta. W teście ADF konieczne jest ustalenie rzędu opóźnienia p. Najczęściej dokonuje się tego za pomocą kryterium informacyjnego Akaike'a (AIC) (Akaike, 1973) lub kryterium bayesowskiego Schwarza (BIC) (Schwarz, 1978).

Inny test służący do testowania stacjonarności szeregów czasowych został zaproponowany w opracowaniu Kwiatkowski i in. (1992). Test KPSS (Kwiatkowski–Phillips–Schmidt–Shin) różni się od podanych wcześniej ze względu na hipotezy badawcze. W tym teście hipoteza zerowa zakłada stacjonarność szeregu czasowego. Niech dany będzie model (Lütkepohl i Krätzig, 2004):

$$y_t = x_t + z_t \tag{3.30}$$

gdzie x_t jest procesem losowym, takim, że:

$$x_t - x_{t-1} = \upsilon_t \tag{3.31}$$

gdzie $\upsilon_t \sim \text{iid}(0, \sigma_\upsilon^2)$, natomiast z_t jest szeregiem stacjonarnym niezależnym od υ_t. W sytuacji, gdy x_t nie jest szeregiem błądzenia losowego, szereg y_t jest stacjonarny wokół trendu. W związku z tym formułuje się hipotezy:

$$\begin{aligned} H_0 &: \sigma_\upsilon^2 = 0 \\ H_1 &: \sigma_\upsilon^2 > 0 \end{aligned}$$

Kwiatkowski i in. (1992) zaproponowali następującą statystykę:

$$\text{KPSS} = \frac{\sum_{t=1}^{T} S_t^2}{T^2 \hat{\sigma}^2} \tag{3.32}$$

gdzie: $S_t = \sum_{j=1}^{t} \hat{\omega}_j$, $\hat{\omega}_t = y_t - \bar{y}$, natomiast $\hat{\sigma}^2$ jest zgodnym estymatorem długookresowej wariancji składnika losowego z_t, tj. $\hat{\sigma}^2 = \lim_{T\to\infty} \frac{1}{T} var(\sum_{t=1}^{T} z_t)$.

Kwiatkowski i in. (1992) obliczyli asymptotyczny rozkład statystyki KPSS przy założeniu, że $\upsilon_t \sim N(0, \sigma_\upsilon^2)$ i $z_t \sim N(0, \sigma_z^2)$ oraz podali jego wartości krytyczne. Hipotezę

zerową odrzuca się, jeśli wartość statystyki testowej jest większa od wartości krytycznej. W opracowaniu źródłowym podano również wartości krytyczne dla przypadku występowania trendu deterministycznego w procesie generowania y_t.

W następnym podrozdziale zostanie przedstawiona analiza empiryczna stacjonarności stóp procentowych i indeksów giełdowych.

3.2.3. Analiza empiryczna

W przedstawionej poniżej empirycznej analizie stacjonarności wykorzystano następujące zmienne:

- stopy procentowe WIBOR rynku międzybankowego (1- i 3-miesięczne),
- stopę oprocentowania kredytu mieszkaniowego dla gospodarstw domowych,
- indeksy: WIG, WIG20 i WIG-Banki dla Giełdy Papierów Wartościowych w Warszawie (GPW).

Definicje zmiennych podano w tabl. 3.1 i 3.2.

Tablica 3.1. Stopy procentowe – definicje zmiennych

Nazwa	Definicja	Próba	Liczba obs.	Źródło
MM1M	1-miesięczna stopa WIBOR, w proc.	1995M1-2024M10	358	Eurostat
MM3M	3-miesięczna stopa WIBOR, w proc.	1995M1-2024M10	358	Eurostat
IHM	Stopa oprocentowania wolumenu kredytu mieszkaniowego dla gospodarstw domowych, w proc.	2005M1-2024M9	237	NBP

Tablica 3.2. Indeksy giełdowe – definicje zmiennych

Nazwa	Definicja	Próba	Liczba obs.	Źródło
WIG	Indeks dla GPW w Warszawie, w punktach	2005M1-2024M11	239	stooq.pl
WIG20	Indeks największych spółek dla GPW w Warszawie, w punktach	2005M1-2024M11	239	stooq.pl
WIG-Banki	Indeks spółek sektora bankowego dla GPW w Warszawie, w punktach	2005M1-2024M11	239	stooq.pl

Stopy procentowe stanowią średnie miesięczne, natomiast indeksy giełdowe podano na koniec miesiąca. W celu przeprowadzenia analizy danych statystycznych rozpoczęto od wczytania odpowiednich bibliotek.

```
library(dplyr)
library(ggplot2)
library(ggthemes)
library(kableExtra)
library(knitr)
library(magrittr)
library(readxl)
library(stargazer)
library(tidyverse)
library(xts)
```

Warto zwrócić uwagę na następujące biblioteki:

- `stargazer` (tablicowanie wyników estymacji modeli ekonometrycznych) (Hlavac, 2022),
- `kableExtra` (rozszerzenie funkcji `kable` z biblioteki `knitr`),
- `ggthemes` (rozszerzenie funkcji graficznych biblioteki `ggplot2`) (Arnold, 2024),
- `tidyverse` (zestaw bibliotek do przetwarzania danych).

Dane statystyczne dla stóp procentowych pobrano ze zbioru `data_IR.xlsx` za pomocą biblioteki `readxl` i zapisano w ramce danych `ir.0`.

```
ir.0 <- paste0(
  "F:\\docs\\R\\b2\\ch3\\",
  "data_IR.xlsx"
) %>%
  read_excel() %>%
  as.data.frame()
head(ir.0)
```

```
##         date  mm1m  mm3m ihm
## 1 1995-01-01 28.04 28.87  NA
## 2 1995-02-01 28.76 29.11  NA
## 3 1995-03-01 30.05 30.32  NA
## 4 1995-04-01 29.40 29.67  NA
## 5 1995-05-01 28.01 28.17  NA
## 6 1995-06-01 27.42 27.56  NA
```

Ponieważ w zbiorze danych `ir.0` występują brakujące obserwacje `NA`, trzeba je pominąć za pomocą funkcji `na.omit`. Ponadto, ciągi znaków zawierające daty obserwacji znajdujące się w pierwszej kolumnie `ir.0` (np. `1995-01-01`) należy zamienić na właściwy format daty za pomocą funkcji `as.Date`.

Tablica 3.3. Wartości stóp procentowych

date	mm1m	mm3m	ihm
2005-01-01	6.63	6.63	7.95
2005-02-01	6.56	6.54	8.10
2005-03-01	6.37	6.15	7.93
2005-04-01	5.93	5.78	7.66
2005-05-01	5.55	5.48	7.58
2005-06-01	5.37	5.22	7.49
2005-07-01	4.88	4.68	7.00
2005-08-01	4.77	4.67	6.82
2005-09-01	4.57	4.51	6.76
2005-10-01	4.61	4.55	6.40

```
ir.1 <- ir.0 %>%
  na.omit()
ir.1$date <- as.Date(ir.1$date)
head(ir.1)
```

```
##           date mm1m mm3m  ihm
## 121 2005-01-01 6.63 6.63 7.95
## 122 2005-02-01 6.56 6.54 8.10
## 123 2005-03-01 6.37 6.15 7.93
## 124 2005-04-01 5.93 5.78 7.66
## 125 2005-05-01 5.55 5.48 7.58
## 126 2005-06-01 5.37 5.22 7.49
```

Wyniki modyfikacji zapisano do nowej ramki danych `ir.1`, której dziesięć pierwszych wierszy zawiera tabl. 3.3. Tablica została wygenerowana za pomocą funkcji `kable` z biblioteki `knitr` (Xie, 2025b).

```
ir.1 %>%
  .[1:10, ] %>%
  kable(
    format = "latex",
    digits = 2,
    booktabs = TRUE,
    row.names = FALSE,
    caption = "Wartości stóp procentowych",
    label = "inwlhw"
  ) %>%
  kable_styling(full_width = TRUE)
```

Należy zwrócić uwagę na następujące argumenty funkcji `kable`:

Tablica 3.4. Statystyki opisowe stóp procentowych – kable

mm1m	mm3m	ihm
Min. :0.180	Min. :0.210	Min. :2.270
1st Qu.:1.650	1st Qu.:1.720	1st Qu.:3.680
Median :3.620	Median :4.010	Median :5.640
Mean :3.459	Mean :3.578	Mean :5.182
3rd Qu.:4.770	3rd Qu.:4.970	3rd Qu.:6.400
Max. :7.080	Max. :7.430	Max. :8.100

- x – macierz lub ramka zawierająca dane,
- format – format wynikowy kodu dla tablicy (latex, html, markdown, pandoc lub rst),
- digits – liczba cyfr znaczących po przecinku,
- booktabs – generowanie kodu dla wysokiej jakości tablic do druku (TRUE),
- row.names – wł./wył. nazw wierszy,
- caption – tytuł tablicy,
- label – nazwa odwołania do tablicy w kodzie dokumentu R/Markdown; np. odwołanie do tabl. 3.7 w kodzie R/Markdown wykonuje się za pomocą polecenia \ref{tab:kkhfvl}.

Dla poszczególnych zmiennych podano wartości statystyk opisowych w tabl. 3.4 i 3.5. Pierwsza tablica została utworzona za pomocą funkcji kable, natomiast druga za pomocą funkcji stargazer z biblioteki o tej samej nazwie.

```
ir.1 %>%
  select(-date) %>%
  summary() %>%
  kable(
    format = "latex",
    digits = 2,
    booktabs = TRUE,
    row.names = FALSE,
    caption = "Statystyki opisowe stóp procentowych --
     ↪  \\texttt{kable}",
    label = "kslbog"
  ) %>%
    kable_styling(
      full_width = TRUE
    )
```

W pierwszym przypadku statystyki opisowe należy wygenerować za pomocą funkcji `summary`, natomiast w drugim wystarczy podać nazwę zbioru danych, gdyż zewnętrzna funkcja `stargazer` domyślnie przyjmuje, iż chodzi o wygenerowanie statystyk opisowych (tabl. 3.5).

```
ir.1 %>%
  stargazer(
    type = "latex",
    title = "Statystyki opisowe stóp procentowych --
     ↪  \\texttt{stargazer}",
    label = "mipnjo",
    style = "aer",
    align = TRUE,
    header = FALSE,
    flip = TRUE,
    digits = 2,
    digits.extra = 2,
    float = TRUE
  )
```

Tablica 3.5. Statystyki opisowe stóp procentowych – `stargazer`

Statistic	$mm1m$	$mm3m$	ihm
N	237	237	237
Mean	3.46	3.58	5.18
St. Dev.	1.93	1.97	1.60
Min	0.18	0.21	2.27
Max	7.08	7.43	8.10

Poprawne wygenerowanie tablic wymaga ustawienia znacznika `results='asis'` w deklaracji kodu (`code chunk`). Warto zwrócić uwagę na argument `style` w funkcji `stargazer`. Jest on odpowiedzialny za format zapisu tablicy. W podanym przykładzie zapisano `style = 'aer'`, co oznacza, że przyjęty został format czasopisma *American Economic Review*. Dokumentacja biblioteki `stargazer` zawiera opis formatów innych znanych czasopism.

Przebieg zmiennych w czasie można następnie przedstawić na rysunku. W tym celu warto ponownie wykorzystać bibliotekę `ggplot2`. Podstawy pracy w tej bibliotece zostały podane w podrozdziale 2.2.10. Podano w nim, że jest możliwe przedstawienie wielu linii wraz z legendą na jednym wykresie. W tym celu wygodnie jest przedstawić zbiór danych `ir.1` w postaci stosu. Można to zrobić za pomocą funkcji `gather` z biblioteki `tidyverse`. Wcześniej utworzono pomocniczy zbiór danych (`ir.2`).

```
ir.2 <- ir.1 %>%
  gather(
    key = "variable",
    value = "value",
    -date
  )
```

W ten sposób wszystkie zmienne zostały pogrupowane w stos uporządkowany w czasie. W tabl. 3.6 pokazano pierwszych dwadzieścia wartości tych zmiennych.

```
ir.2 %>%
  .[1:20, ] %>%
  kable(
    format = "latex",
    digits = 2,
    booktabs = TRUE,
    row.names = FALSE,
    caption = "Wartości stóp procentowych w postaci stosu",
    label = "rkhbzd"
  ) %>%
  kable_styling(
    full_width = TRUE
  )
```

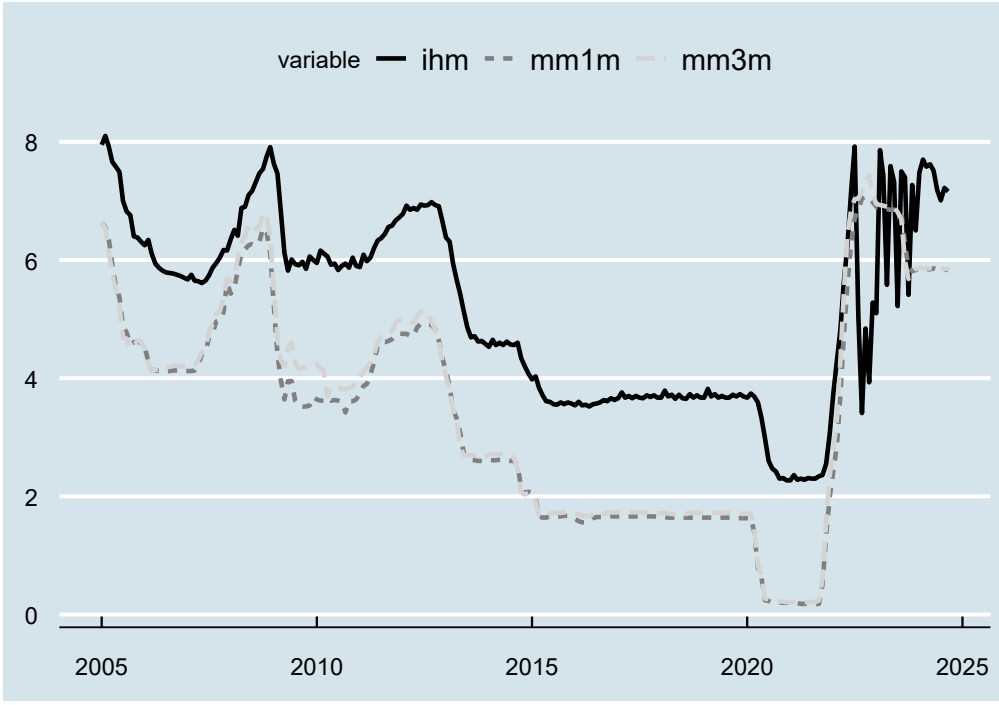
Dla takich zbiorów danych można utworzyć wykresy stóp procentowych w bibliotece `ggplot2` (rys. 3.6).

```
ir.2 %>%
  ggplot(aes(x = date, y = value)) +
  geom_line(
    aes(
        color = variable,
        linetype = variable
    ),
    size = 1
  ) +
  scale_color_manual(
    values = c(
        '#000000',
        '#808080',
        '#D3D3D3'
    )
  ) +
  theme_economist(
    base_size = 10,
    base_family = "sans",
```

Tablica 3.6. Wartości stóp procentowych w postaci stosu

date	variable	value
2005-01-01	mm1m	6.63
2005-02-01	mm1m	6.56
2005-03-01	mm1m	6.37
2005-04-01	mm1m	5.93
2005-05-01	mm1m	5.55
2005-06-01	mm1m	5.37
2005-07-01	mm1m	4.88
2005-08-01	mm1m	4.77
2005-09-01	mm1m	4.57
2005-10-01	mm1m	4.61
2005-11-01	mm1m	4.62
2005-12-01	mm1m	4.60
2006-01-01	mm1m	4.52
2006-02-01	mm1m	4.29
2006-03-01	mm1m	4.14
2006-04-01	mm1m	4.13
2006-05-01	mm1m	4.12
2006-06-01	mm1m	4.12
2006-07-01	mm1m	4.12
2006-08-01	mm1m	4.12

```
    horizontal = TRUE,
    dkpanel = FALSE
  ) +
  theme(
    axis.title.x = element_blank(),
    axis.title.y = element_blank()
  )
```



Rysunek 3.6. Stopy procentowe

Warto zwrócić uwagę na funkcję `theme_economist` z biblioteki `ggthemes`. Ta warstwa funkcji `ggplot` jest odpowiedzialna za wyświetlenie wykresu według formatu dziennika *The Economist*.

Podobną analizę można przeprowadzić dla indeksów giełdowych. Dane statystyczne pobrano ze strony internetowej https://stooq.pl/, odpowiednio z adresów:

- WIG: https://stooq.pl/q/d/l/?s=wig&d1=20050101&d2=20241130&i=m,
- WIG20: https://stooq.pl/q/d/l/?s=wig20&d1=20050101&d2=20241130&i=m,
- WIG-BANKI: https://stooq.pl/q/d/l/?s=wig_banki&d1=20050101&d2=20241130&i=m.

Dla wygody zdefiniowano wektor znakowy index.

```
index <- c(
  "wig",
  "wig20",
  "wig_banki"
)
```

W podanej poniżej lokalizacji path zapisano zbiory źródłowe pobrane ze strony stooq.pl.

```
path <- "F:\\docs\\R\\b2\\ch3\\gpw\\"
```

W podanym poniżej algorytmie źródłowe wartości indeksów giełdowych są zapisywane do katalogu path. Jeśli czynność zapisu została już raz wykonana, to tworzony jest jeden zbiór danych w postaci arkusza Excel. Należy zatem sprawdzić, czy w katalogu path znajdują się źródłowe pliki .csv.

```
files <- path %>%
  paste0(index, ".csv")
files
```

```
## [1] "F:
docs
R
b2
ch3
gpw
wig.csv"
## [2] "F:
docs
R
b2
ch3
gpw
wig20.csv"
## [3] "F:
docs
R
b2
ch3
gpw
wig_banki.csv"
```

Jeśli pliki .csv nie znajdują się w podanym katalogu, to odpowiednie dane źródłowe są pobierane ze strony stooq.pl. Nazwy indeksów są pobierane z obiektu index i przekazywane do adresu HTTPS.

```
if (!all(file.exists(files))) {
  address <- lapply(
    index,
    function(x) paste0(
      "https://stooq.pl/q/d/l/?s=",
      x,
      "&d1=20050101&d2=20241130",
      "&i=m"
    )
  ) %>%
    unlist
    lapply(
      1:length(address),
      function(x) address[x] %>%
        download.file(
          destfile = paste0(path, index[x], ".csv"),
          method = "auto"
        )
      )
}
```

Następnie pliki `.csv` są pobierane do ramek danych za pomocą funkcji `read.csv` i umieszczane na liście `gpw.0`. W ramkach danych znajdują się tylko kolumny `Data` oraz `Zamkniecie`. Ostatecznie kolumna `Zamkniecie` otrzymuje nazwę indeksu.

```
gpw.0 <- lapply(
  1:length(files),
  function(x) {
    df <- path %>%
      paste0(index[x], ".csv") %>%
      read.csv(stringsAsFactors = FALSE) %>%
      select(Data, Zamkniecie) %>%
      set_colnames(c("date", index[x]))
  }
)
```

W następnym kroku procedury powstaje ramka danych `gpw.1`, która stanowi połączenie (`left join`) ramek danych `gpw.0` według kolumny `date`. Wykorzystano w tym celu funkcję `reduce` z biblioteki `purrr` (Wickham i Henry, 2023), która jest częścią ekosystemu `tidyverse`. Jej wynikiem jest rekurencyjne zastosowanie funkcji na elementach wektora lub listy. Funkcja `reduce` pobiera pierwszy i drugi element z `.x`, stosuje na nich funkcję `.f`, a wynik traktuje jako nową wartość początkową do zastosowania z kolejnym elementem. Proces ten jest kontynuowany, aż wszystkie elementy `.x` zostaną wykorzystane. Ta funkcja okazuje się pomocna, gdy zachodzi potrzeba iteracyjnej redukcji zbiorów danych. Jest szczególnie przydatna w przetwarzaniu list i innych złożonych struktur danych.

Tablica 3.7. Wartości indeksów giełdowych

date	wig	wig20	wig_banki
2024-02-29	81944.56	2418.10	12835.19
2024-03-31	82745.58	2436.05	13524.76
2024-04-30	84569.65	2476.29	13670.81
2024-05-31	86315.26	2485.53	13141.05
2024-06-30	88613.67	2561.27	13817.94
2024-07-31	84345.70	2421.49	13194.97
2024-08-31	84868.25	2412.16	13301.72
2024-09-30	83274.20	2324.13	12458.20
2024-10-31	79550.32	2205.47	12082.62
2024-11-30	79369.82	2191.12	11894.17

```
gpw.1 <- gpw.0 %>%
  reduce(~ left_join(.x, .y, by = "date"))
```

Dla zachowania zgodności typów danych kolumna `date` jest ponownie przetwarzana do formatu *daty*.

```
gpw.1$date <- as.Date(gpw.1$date)
```

W tabl. 3.7 podano ostatnich 10 obserwacji indeksów giełdowych.

```
gpw.1 %>%
  tail(., 10) %>%
  kable(
    format = "latex",
    digits = 2,
    booktabs = TRUE,
    row.names = FALSE,
    caption = "Wartości indeksów giełdowych",
    label = "kkhfvl"
  ) %>%
  kable_styling(
    full_width = TRUE
  )
```

W tabl. 3.8 podano wartości statystyk opisowych dla indeksów giełdowych.

Tablica 3.8. Statystyki opisowe indeksów giełdowych – `kable`

wig	wig20	wig_banki
Min. :21691	Min. :1372	Min. : 2533
1st Qu.:43552	1st Qu.:2052	1st Qu.: 5784
Median :50468	Median :2310	Median : 6643
Mean :51661	Mean :2327	Mean : 6813
3rd Qu.:59669	3rd Qu.:2486	3rd Qu.: 7721
Max. :88614	Max. :3878	Max. :13818

```
gpw.1 %>%
  select(-date) %>%
  summary() %>%
  kable(
    format = "latex",
    digits = 2,
    booktabs = TRUE,
    row.names = FALSE,
    caption = "Statystyki opisowe indeksów giełdowych --
     ↪  \\texttt{kable}",
    label = "lhssoi"
  ) %>%
  kable_styling(
    full_width = TRUE
  )
```

W tabl. 3.9 podano wartości statystyk opisowych dla indeksów giełdowych według stylizacji z biblioteki `stargazer`.

```
gpw.1 %>%
  select(-date) %>%
  stargazer(
    type = "latex",
    title = "Statystyki opisowe indeksów giełdowych --
     ↪  \\texttt{stargazer}",
    label = "axghzd",
    style = "aer",
    align = TRUE,
    header = FALSE,
    flip = TRUE,
    digits = 2,
    digits.extra = 2,
    float = TRUE
  )
```

Tablica 3.9. Statystyki opisowe indeksów giełdowych – `stargazer`

Statistic	wig	$wig20$	wig_banki
N	239	239	239
Mean	51,661.39	2,326.89	6,813.18
St. Dev.	12,808.91	458.77	1,986.12
Min	21,690.80	1,372.47	2,533.36
Max	88,613.67	3,877.62	13,817.94

Poniższy kod korzysta z funkcji `gather` (z biblioteki `tidyr`), która służy do przekształcenia danych zawartych w ramce danych `gpw.1` do formatu stosu.

```
gpw.2 <- gpw.1 %>%
  gather(
    key = "variable",
    value = "value",
    -date
  )
head(gpw.2)
```

```
##         date variable   value
## 1 2005-01-31      wig 25993.0
## 2 2005-02-28      wig 28294.5
## 3 2005-03-31      wig 27268.1
## 4 2005-04-30      wig 25813.6
## 5 2005-05-31      wig 26744.4
## 6 2005-06-30      wig 28332.1
```

Obiekt `gpw.2` został wykorzystany do zbudowania tablicy z wartościami zmiennych (tabl. 3.10).

```
gpw.2 %>%
  .[1:20, ] %>%
  kable(
    format = "latex",
    digits = 2,
    booktabs = TRUE,
    row.names = FALSE,
    caption = "Wartości indeksów giełdowych w postaci stosu",
    label = "kejzzh"
  ) %>%
  kable_styling(
    full_width = TRUE
  )
```

Tablica 3.10. Wartości indeksów giełdowych w postaci stosu

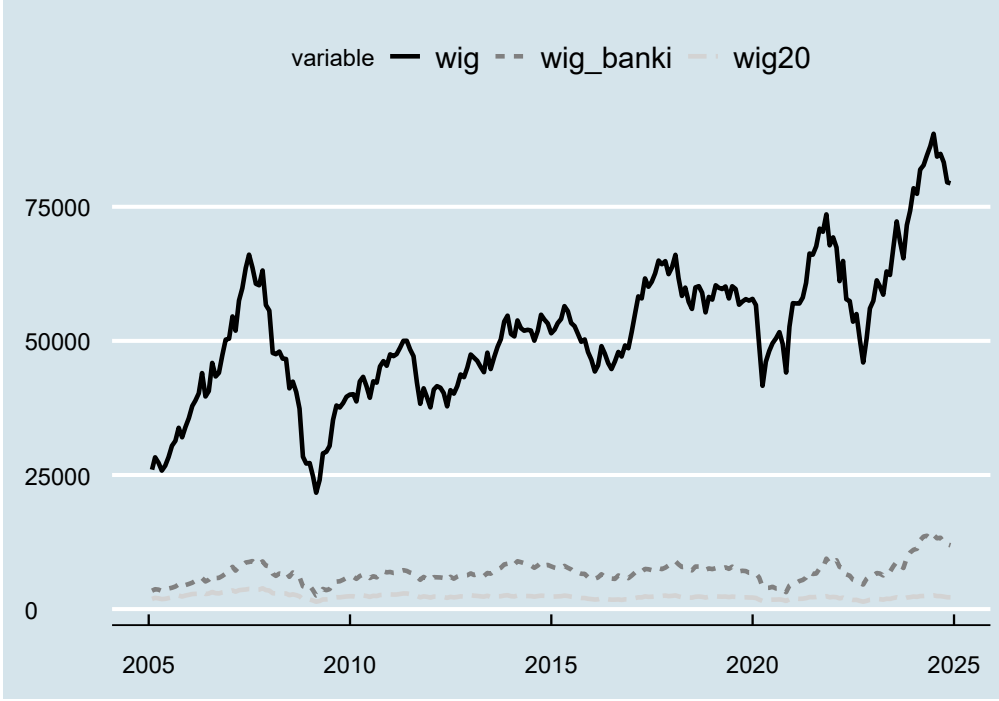
date	variable	value
2005-01-31	wig	25993.0
2005-02-28	wig	28294.5
2005-03-31	wig	27268.1
2005-04-30	wig	25813.6
2005-05-31	wig	26744.4
2005-06-30	wig	28332.1
2005-07-31	wig	30448.3
2005-08-31	wig	31364.3
2005-09-30	wig	33801.2
2005-10-31	wig	32024.4
2005-11-30	wig	33926.1
2005-12-31	wig	35600.8
2006-01-31	wig	37854.9
2006-02-28	wig	38854.4
2006-03-31	wig	40220.3
2006-04-30	wig	43998.6
2006-05-31	wig	39632.2
2006-06-30	wig	40644.6
2006-07-31	wig	45894.9
2006-08-31	wig	43363.3

Następnie pokazano wykres indeksów giełdowych za pomocą funkcji `ggplot` (rys. 3.7).

```
gpw.2 %>%
  ggplot(aes(x = date, y = value)) +
  geom_line(
    aes(
        color = variable,
        linetype = variable
    ),
    size = 1
  ) +
  scale_color_manual(
    values = c(
        '#000000',
        '#808080',
        '#D3D3D3'
    )
  ) +
  theme_economist(
    base_size = 10,
    base_family = "sans",
    horizontal = TRUE,
    dkpanel = FALSE
  ) +
  theme(
    axis.title.x = element_blank(),
    axis.title.y = element_blank()
  )
```

Kod rys. 3.7 można również przekształcić w taki sposób, aby wykresy były umieszczone w wierszach (rys. 3.8).

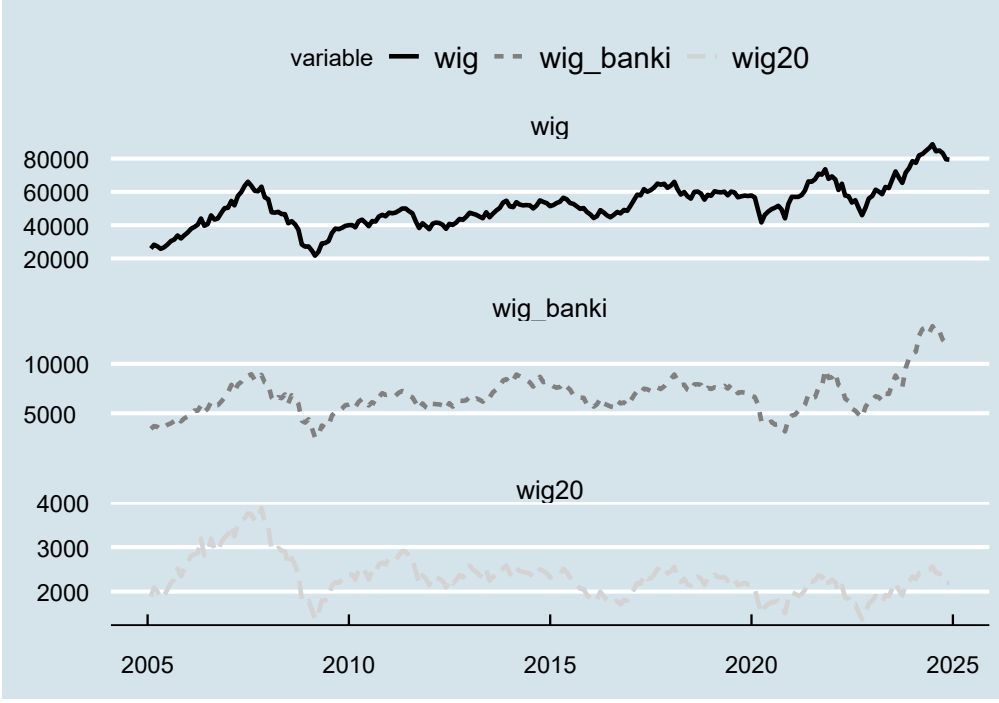
```
gpw.2 %>%
  ggplot(aes(x = date, y = value)) +
  geom_line(
    aes(
      color = variable,
      linetype = variable
    ),
    size = 1
  ) +
  scale_color_manual(
    values = c(
      '#000000',
      '#808080',
      '#D3D3D3'
    )
```



Rysunek 3.7. Indeksy giełdowe – przypadek 1

```
) +
theme_economist(
  base_size = 10,
  base_family = "sans",
  horizontal = TRUE,
  dkpanel = FALSE
) +
theme(
  axis.title.x = element_blank(),
  axis.title.y = element_blank(),
  strip.text = element_text(size = 12),
  panel.spacing = unit(1, "lines")
) +
facet_wrap(~variable, ncol = 1, scales = "free_y")
```

Następnym etapem analizy jest przekształcenie zbioru danych `ir.1` dla stóp procentowych do postaci szeregów czasowych za pomocą funkcji `xts` z biblioteki o tej samej nazwie.



Rysunek 3.8. Indeksy giełdowe – przypadek 2

```
ir.1.xts <- xts(
    ir.1[, -1],
    order.by = ir.1[, 1]
)
head(ir.1.xts)
```

```
##            mm1m mm3m  ihm
## 2005-01-01 6.63 6.63 7.95
## 2005-02-01 6.56 6.54 8.10
## 2005-03-01 6.37 6.15 7.93
## 2005-04-01 5.93 5.78 7.66
## 2005-05-01 5.55 5.48 7.58
## 2005-06-01 5.37 5.22 7.49
```

Postać obiektu `ir.1.xts` pokazana za pomocą funkcji `kable` – bez kolumny dat obserwacji – jest następująca (tabl. 3.11):

```
kable(
    ir.1.xts[1:5, ],
```

Tablica 3.11. Wartości stóp procentowych w postaci szeregów czasowych

mm1m	mm3m	ihm
6.63	6.63	7.95
6.56	6.54	8.10
6.37	6.15	7.93
5.93	5.78	7.66
5.55	5.48	7.58

```
    "latex",
    digits = 2,
    booktabs = TRUE,
    row.names = FALSE,
    caption = "Wartości stóp procentowych w postaci szeregów
     ↪  czasowych",
    label = "azwypl"
) %>%
kable_styling(
    full_width = TRUE
)
```

Dla poszczególnych zmiennych zbioru `ir.1.xts` przeprowadzono test ADF. Odpowiednie funkcje można znaleźć w bibliotece `urca` (Pfaff, 2008). Należy zatem rozpocząć od wczytania tej biblioteki.

```
library(urca)
```

Za pomocą poniższego kodu wykonano obliczenia dla testu ADF i utworzono tablicę z wynikami. W tablicy zawarto decyzję dotyczącą rzędu zintegrowania zmiennych. Najpierw utworzono niezbędne zmienne, w których zostaną zapisane wyniki.

```
type <- c(
    "none",
    "drift",
    "trend"
)
ur <- ur.tab <- adf.tab <- header <- list()
note <- paste0(
    "Symbole oznaczają: ",
    "I(1) niestacjonarność, ",
    "I(0) stacjonarność, ***, **, ",
    "* statystyczną istotność na ",
    "poziomie odpowiednio 1%, 5% i 10%."
)
```

Wykonano obliczenia przyjmując za podstawę równanie (3.27). Przyjęto następujące warianty:

- none – bez wyrazu wolnego μ i parametru trendu β,
- drift – bez parametru trendu β,
- trend – z wyrazem wolnym μ i parametrem trendu β.

W poniższej pętli for wykonano obliczenia dla trzech wspomnianych przypadków testu ADF. Wyniki zostały zapisane do listy ur.

```
for (i in type) {
    ur[[i]] <- lapply(
        1:ncol(ir.1.xts),
        function(x) {
            ur.df(
                y = ir.1.xts[, x],
                type = i,
                selectlags = "BIC"
            )
        }
    )
    ur.tab[[i]] <- data.frame(
        variable = colnames(ir.1.xts)
    )
}
```

W kolejnej zagnieżdżonej pętli for pobrano wybrane wielkości z listy ur i umieszczono na liście ur.tab. Następnie dla każdego rodzaju testu type utworzono listę adf.tab z wynikami testu.

```
for (i in type) {
    for (j in 1:ncol(ir.1.xts)) {
        ur.tab[[i]][j, 2] <- ur[[i]][[j]]@teststat[1, 1]
        ur.tab[[i]][j, 3] <- ur[[i]][[j]]@cval[1, 1]
        ur.tab[[i]][j, 4] <- ur[[i]][[j]]@cval[1, 2]
        ur.tab[[i]][j, 5] <- ur[[i]][[j]]@cval[1, 3]
        if (ur.tab[[i]][j, 2] < ur.tab[[i]][j, 3]) {
            ur.tab[[i]][j, 6] <- "I(0) ***"
        }
        if (ur.tab[[i]][j, 2] > ur.tab[[i]][j, 3]
            & ur.tab[[i]][j, 2] < ur.tab[[i]][j, 4]) {
            ur.tab[[i]][j, 6] <- "I(0) **"
        }
        if (ur.tab[[i]][j, 2] > ur.tab[[i]][j, 4]
            & ur.tab[[i]][j, 2] < ur.tab[[i]][j, 5]) {
            ur.tab[[i]][j, 6] <- "I(0) *"
```

```
        }
        if (ur.tab[[i]][j, 2] > ur.tab[[i]][j, 5]) {
            ur.tab[[i]][j, 6] <- "I(1)"
        }
    }
    adf.tab[[i]] <- ur.tab[[i]][, c(-3, -4, -5)]
    colnames(adf.tab[[i]]) <- c(
        "zmienna",
        "statystyka",
        "decyzja"
    )
    adf.tab[[i]][, "statystyka"] <- round(
        adf.tab[[i]][, "statystyka"],
        digits = 2
    )
    header[[i]] <- c(" " = 1, i = 2)
    names(header[[i]]) <- c(" ", i)
}
```

Pierwsza współrzędna listy `adf.tab` (dla przypadku `none`) ma postać.

```
adf.tab[[1]]
```

```
##   zmienna statystyka decyzja
## 1    mm1m      -1.17    I(1)
## 2    mm3m      -1.16    I(1)
## 3     ihm      -0.77    I(1)
```

Druga współrzędna listy `adf.tab` (dla przypadku `drift`) ma postać.

```
adf.tab[[2]]
```

```
##   zmienna statystyka decyzja
## 1    mm1m      -2.30    I(1)
## 2    mm3m      -2.34    I(1)
## 3     ihm      -2.18    I(1)
```

Trzecia współrzędna listy `adf.tab` (dla przypadku `trend`) ma postać.

```
adf.tab[[3]]
```

```
##   zmienna statystyka decyzja
## 1    mm1m      -2.07    I(1)
## 2    mm3m      -2.14    I(1)
## 3     ihm      -1.71    I(1)
```

Każdą tablicę można sformatować za pomocą funkcji `kable`. Wyjaśnia to poniższy kod dla przypadku testu z wyrazem wolnym (`drift`) (tabl. 3.12).

```
kable(
    adf.tab[[2]],
    digits = 2,
    format = "latex",
    caption = "Wyniki testu ADF dla stóp procentowych -- wyraz wolny",
  label = "eqrvhb",
    booktabs = TRUE,
    longtable = FALSE,
    linesep = ""
) %>%
kable_styling(
    full_width = FALSE,
    latex_options = "hold_position"
) %>%
add_header_above(
    header = header[[2]]
) %>%
footnote(
    note,
    threeparttable = TRUE
)
```

Tablica 3.12. Wyniki testu ADF dla stóp procentowych – wyraz wolny

	drift	
zmienna	statystyka	decyzja
mm1m	-2.30	I(1)
mm3m	-2.34	I(1)
ihm	-2.18	I(1)

Note:
Symbole oznaczają: I(1) niestacjonarność, I(0) stacjonarność, ***, **, * statystyczną istotność na poziomie odpowiednio 1%, 5% i 10%.

Na podstawie podanych wyników dla testu ADF, w szczególności zawartych w tabl. 3.12, należy przyjąć, że szeregi stóp procentowych są niestacjonarne.

Następnie wykonano test KPSS. Wykorzystano bibliotekę `tseries` (Trapletti i Hornik, 2024), w której jest zawarta funkcja `kpss.test`. Ta funkcja realizuje obliczenia dla

modelu (3.30) i statystyki (3.32), z możliwością uwzględnienia trendu deterministycznego w wariancie testu z trendem (`Trend`). Przyjęto następujące argumenty funkcji `kpss.test`:

- `Level` – bez trendu,
- `Trend` – z trendem.

Za pomocą poniższego kodu wykonano obliczenia dla testu KPSS i utworzono tablicę z wynikami. W tablicy zawarto decyzję dotyczącą rzędu zintegrowania zmiennych. Najpierw utworzono niezbędne zmienne, w których zostaną zapisane wyniki.

```
kpss.type <- c(
    "Level",
    "Trend"
)
kpss <- kpss.tab <- kpss.test.tab <- kpss.header <- list()
kpss.note <- paste0(
    "Symbole oznaczają: ",
    "I(1) niestacjonarność, ",
    "I(0) stacjonarność, ***, **, ",
    "* statystyczną istotność na ",
    "poziomie odpowiednio 1%, 5% i 10%."
)
```

Najpierw została wczytana biblioteka.

```
library(tseries)
```

W poniższej pętli `for` wykonano obliczenia dla dwóch wspomnianych przypadków testu KPSS. Wyniki zostały zapisane do listy `kpss`.

```
kpss.type <- c("Level", "Trend")
for (i in kpss.type) {
    kpss[[i]] <- lapply(
        1:ncol(ir.1.xts),
        function(x) {
            kpss.test(
                ir.1.xts[, x],
                null = i
            )
        }
    )
    kpss.tab[[i]] <- data.frame(
        variable = colnames(ir.1.xts)
    )
}
```

W kolejnych pętlach `for` pobrano wartości z listy `kpss` i umieszczono na liście `kpss.tab`. Następnie dla każdego rodzaju testu `type` utworzono listę `kpss.tab` z wynikami testu KPSS.

```
for (i in c("Level")) {
    for (j in 1:ncol(ir.1.xts)) {
        kpss.tab[[i]][j, 2] <- kpss[[i]][[j]]$statistic[[1]]
        kpss.tab[[i]][j, 3] <- kpss[[i]][[j]]$p.value
        if (kpss.tab[[i]][j, 2] <= 0.347) {
            kpss.tab[[i]][j, 4] <- "I(0) ***"
        }
        if (kpss.tab[[i]][j, 2] <= 0.463
            & kpss.tab[[i]][j, 2] > 0.347) {
            kpss.tab[[i]][j, 4] <- "I(0) **"
        }
        if (kpss.tab[[i]][j, 2] <= 0.739
            & kpss.tab[[i]][j, 2] > 0.463) {
            kpss.tab[[i]][j, 4] <- "I(0) *"
        }
        if (kpss.tab[[i]][j, 2] > 0.739) {
            kpss.tab[[i]][j, 4] <- "I(1)"
        }
    }
    colnames(kpss.tab[[i]]) <- c(
        "zmienna",
        "statystyka",
        "p-value",
        "decyzja"
    )
    kpss.tab[[i]][, c("statystyka", "p-value")] <- round(
        kpss.tab[[i]][, c("statystyka", "p-value")],
        digits = 2
    )
    kpss.header[[i]] <- c(" " = 1, i = 3)
    names(kpss.header[[i]]) <- c(" ", i)
}
```

```
for (i in c("Trend")) {
    for (j in 1:ncol(ir.1.xts)) {
        kpss.tab[[i]][j, 2] <- kpss[[i]][[j]]$statistic[[1]]
        kpss.tab[[i]][j, 3] <- kpss[[i]][[j]]$p.value
        if (kpss.tab[[i]][j, 2] <= 0.119) {
            kpss.tab[[i]][j, 4] <- "I(0) ***"
        }
        if (kpss.tab[[i]][j, 2] <= 0.146
            & kpss.tab[[i]][j, 2] > 0.119) {
            kpss.tab[[i]][j, 4] <- "I(0) **"
        }
```

```
        if (kpss.tab[[i]][j, 2] <= 0.216
            & kpss.tab[[i]][j, 2] > 0.146) {
            kpss.tab[[i]][j, 4] <- "I(0) *"
        }
        if (kpss.tab[[i]][j, 2] > 0.216) {
            kpss.tab[[i]][j, 4] <- "I(1)"
        }
    }
    colnames(kpss.tab[[i]]) <- c(
        "zmienna",
        "statystyka",
        "p-value",
        "decyzja"
    )
    kpss.tab[[i]][, c("statystyka", "p-value")] <- round(
        kpss.tab[[i]][, c("statystyka", "p-value")],
        digits = 2
    )
    kpss.header[[i]] <- c(" " = 1, i = 3)
    names(kpss.header[[i]]) <- c(" ", i)
}
```

Pierwsza współrzędna listy `kpss.tab` (dla przypadku `Level`) ma postać.

```
kpss.tab[[1]]
```

```
##   zmienna statystyka p-value decyzja
## 1    mm1m       1.03    0.01    I(1)
## 2    mm3m       1.06    0.01    I(1)
## 3     ihm       1.74    0.01    I(1)
```

Druga współrzędna listy `kpss.tab` (dla przypadku `Trend`) ma postać.

```
kpss.tab[[2]]
```

```
##   zmienna statystyka p-value decyzja
## 1    mm1m       0.59    0.01    I(1)
## 2    mm3m       0.59    0.01    I(1)
## 3     ihm       0.53    0.01    I(1)
```

Każdą tablicę można sformatować za pomocą funkcji `kable`. Wyjaśnia to poniższy kod dla przypadku `Level`.

```
kable(
    kpss.tab[[1]],
    digits = 2,
    format = "latex",
    caption = "Wyniki testu KPSS dla stóp procentowych -- wyraz
     ↪  wolny",
  label = "xsknvz",
    booktabs = TRUE,
    longtable = FALSE,
    linesep = ""
) %>%
kable_styling(
    full_width = FALSE,
    latex_options = "hold_position"
) %>%
add_header_above(
    header = kpss.header[[1]]
) %>%
footnote(
    kpss.note,
    threeparttable = TRUE
)
```

Tablica 3.13. Wyniki testu KPSS dla stóp procentowych – wyraz wolny

	Level		
zmienna	statystyka	p-value	decyzja
mm1m	1.03	0.01	I(1)
mm3m	1.06	0.01	I(1)
ihm	1.74	0.01	I(1)

Note:
Symbole oznaczają: I(1) niestacjonarność, I(0) stacjonarność, ***, **, * statystyczną istotność na poziomie odpowiednio 1%, 5% i 10%.

Wyniki zawarte w tabl. 3.13 potwierdzają wyniki testu ADF. Na podstawie przeprowadzonych testów ADF i KPSS można stwierdzić, że wszystkie zmienne zbioru `ir.1.xts` są niestacjonarne.

Podobną analizę można przeprowadzić dla indeksów giełdowych. W tym celu ramka danych `gpw.1` została przekształcona do postaci szeregów czasowych za pomocą funkcji `xts`.

Tablica 3.14. Wartości indeksów giełdowych w postaci szeregów czasowych

wig	wig20	wig_banki
25993.0	1887.41	3418.59
28294.5	2092.33	3700.61
27268.1	1998.33	3625.79
25813.6	1858.99	3366.51
26744.4	1917.09	3597.23

```
gpw.1.xts <- xts(
    gpw.1[, -1],
    order.by = gpw.1[, 1]
)
head(gpw.1.xts)
```

```
##                wig   wig20 wig_banki
## 2005-01-31 25993.0 1887.41   3418.59
## 2005-02-28 28294.5 2092.33   3700.61
## 2005-03-31 27268.1 1998.33   3625.79
## 2005-04-30 25813.6 1858.99   3366.51
## 2005-05-31 26744.4 1917.09   3597.23
## 2005-06-30 28332.1 2047.30   3833.37
```

Postać obiektu `gpw.1.xts` pokazana za pomocą funkcji `kable` – bez kolumny dat obserwacji – jest następująca (tabl. 3.14):

```
kable(
    gpw.1.xts[1:5, ],
    "latex",
    digits = 2,
    booktabs = TRUE,
    row.names = FALSE,
    caption = "Wartości indeksów giełdowych w postaci szeregów
     ↪  czasowych",
    label = "clnsha"
) %>%
kable_styling(
    full_width = TRUE
)
```

Dla poszczególnych zmiennych zbioru `gpw.1.xts` przeprowadzono test ADF. Za pomocą poniższego kodu wykonano obliczenia i utworzono tablicę z wynikami. W tablicy zawarto decyzję dotyczącą rzędu zintegrowania zmiennych. Najpierw utworzono niezbędne zmienne.

```
ur <- ur.tab <- adf.tab <- header <- list()
```

Wykonano obliczenia przyjmując za podstawę równanie (3.27) i warianty testu:

- none,
- drift,
- trend.

W poniższej pętli for wykonano obliczenia dla trzech przypadków testu ADF. Wyniki zostały ponownie zapisane do listy ur.

```
for (i in type) {
    ur[[i]] <- lapply(
        1:ncol(gpw.1.xts),
        function(x) {
            ur.df(
                y = gpw.1.xts[, x],
                type = i,
                selectlags = "BIC"
            )
        }
    )
    ur.tab[[i]] <- data.frame(
        variable = colnames(gpw.1.xts)
    )
}
```

W kolejnej zagnieżdżonej pętli for pobrano wybrane wielkości z listy ur i umieszczono w liście ur.tab. Następnie dla każdego rodzaju testu type utworzono listę adf.tab z wynikami testu.

```
for (i in type) {
    for (j in 1:ncol(gpw.1.xts)) {
        ur.tab[[i]][j, 2] <- ur[[i]][[j]]@teststat[1, 1]
        ur.tab[[i]][j, 3] <- ur[[i]][[j]]@cval[1, 1]
        ur.tab[[i]][j, 4] <- ur[[i]][[j]]@cval[1, 2]
        ur.tab[[i]][j, 5] <- ur[[i]][[j]]@cval[1, 3]
        if (ur.tab[[i]][j, 2] < ur.tab[[i]][j, 3]) {
            ur.tab[[i]][j, 6] <- "I(0) ***"
        }
        if (ur.tab[[i]][j, 2] > ur.tab[[i]][j, 3]
            & ur.tab[[i]][j, 2] < ur.tab[[i]][j, 4]) {
            ur.tab[[i]][j, 6] <- "I(0) **"
        }
        if (ur.tab[[i]][j, 2] > ur.tab[[i]][j, 4]
```

```
            & ur.tab[[i]][j, 2] < ur.tab[[i]][j, 5]) {
            ur.tab[[i]][j, 6] <- "I(0) *"
        }
        if (ur.tab[[i]][j, 2] > ur.tab[[i]][j, 5]) {
            ur.tab[[i]][j, 6] <- "I(1)"
        }
    }
    adf.tab[[i]] <- ur.tab[[i]][, c(-3, -4, -5)]
    colnames(adf.tab[[i]]) <- c(
        "zmienna",
        "statystyka",
        "decyzja"
    )
    adf.tab[[i]][, "statystyka"] <- round(
        adf.tab[[i]][, "statystyka"],
        digits = 2
    )
    header[[i]] <- c(" " = 1, i = 2)
    names(header[[i]]) <- c(" ", i)
}
```

Pierwsza współrzędna listy `adf.tab` (none) ma postać.

```
adf.tab[[1]]
```

```
##     zmienna statystyka decyzja
## 1       wig       0.72    I(1)
## 2     wig20      -0.39    I(1)
## 3 wig_banki       0.47    I(1)
```

Druga współrzędna listy `adf.tab` (`drift`) ma postać.

```
adf.tab[[2]]
```

```
##     zmienna statystyka decyzja
## 1       wig      -1.60    I(1)
## 2     wig20      -2.34    I(1)
## 3 wig_banki      -1.69    I(1)
```

Trzecia współrzędna listy `adf.tab` (`trend`) ma postać.

```
adf.tab[[3]]
```

```
##     zmienna statystyka decyzja
## 1       wig      -2.58    I(1)
## 2     wig20      -2.95    I(1)
## 3 wig_banki      -2.08    I(1)
```

Każdą tablicę można sformatować za pomocą funkcji `kable`. Wyjaśnia to poniższy kod dla przypadku testu z wyrazem wolnym (`drift`) (tabl. 3.15).

```
kable(
    adf.tab[[2]],
    digits = 2,
    format = "latex",
    caption = "Wyniki testu ADF dla indeksów giełdowych -- wyraz
     ↪  wolny",
  label = "awoebe",
    booktabs = TRUE,
    longtable = FALSE,
    linesep = ""
) %>%
kable_styling(
    full_width = FALSE,
    latex_options = "hold_position"
) %>%
add_header_above(
    header = header[[2]]
) %>%
footnote(
    note,
    threeparttable = TRUE
)
```

Tablica 3.15. Wyniki testu ADF dla indeksów giełdowych – wyraz wolny

	drift	
zmienna	statystyka	decyzja
wig	-1.60	I(1)
wig20	-2.34	I(1)
wig_banki	-1.69	I(1)

Note:
Symbole oznaczają: I(1) niestacjonarność, I(0) stacjonarność, ***, **, * statystyczną istotność na poziomie odpowiednio 1%, 5% i 10%.

Na podstawie podanych wyników dla testu ADF, w szczególności zawartych w tabl. 3.15, należy przyjąć, że szeregi indeksów giełdowych są niestacjonarne.

Następnie wykonano obliczenia dla testu KPSS za pomocą poniższego kodu i utworzono tablicę z wynikami. W tablicy zawarto decyzję dotyczącą rzędu zintegrowania zmiennych. Najpierw utworzono niezbędne zmienne, w których zostaną zapisane wyniki.

```
kpss <- kpss.tab <- kpss.test.tab <- kpss.header <- list()
```

W poniższej pętli `for` wykonano obliczenia dla dwóch przypadków testu KPSS. Wyniki zostały ponownie zapisane do listy `kpss`.

```
kpss.type <- c("Level", "Trend")
for (i in kpss.type) {
    kpss[[i]] <- lapply(
        1:ncol(gpw.1.xts),
        function(x) {
            kpss.test(
                gpw.1.xts[, x],
                null = i
            )
        }
    )
    kpss.tab[[i]] <- data.frame(
        variable = colnames(gpw.1.xts)
    )
}
```

W kolejnych pętlach `for` pobrano wartości z listy `kpss` i umieszczono na liście `kpss.tab`. Następnie dla każdego rodzaju testu `type` utworzono listę `kpss.tab` z wynikami testu KPSS.

```
for (i in c("Level")) {
    for (j in 1:ncol(gpw.1.xts)) {
        kpss.tab[[i]][j, 2] <- kpss[[i]][[j]]$statistic[[1]]
        kpss.tab[[i]][j, 3] <- kpss[[i]][[j]]$p.value
        if (kpss.tab[[i]][j, 2] <= 0.347) {
            kpss.tab[[i]][j, 4] <- "I(0) ***"
        }
        if (kpss.tab[[i]][j, 2] <= 0.463
            & kpss.tab[[i]][j, 2] > 0.347) {
            kpss.tab[[i]][j, 4] <- "I(0) **"
        }
        if (kpss.tab[[i]][j, 2] <= 0.739
            & kpss.tab[[i]][j, 2] > 0.463) {
            kpss.tab[[i]][j, 4] <- "I(0) *"
        }
        if (kpss.tab[[i]][j, 2] > 0.739) {
```

```
            kpss.tab[[i]][j, 4] <- "I(1)"
        }
    }
    colnames(kpss.tab[[i]]) <- c(
        "zmienna",
        "statystyka",
        "p-value",
        "decyzja"
    )
    kpss.tab[[i]][, c("statystyka", "p-value")] <- round(
        kpss.tab[[i]][, c("statystyka", "p-value")],
        digits = 2
    )
    kpss.header[[i]] <- c(" " = 1, i = 3)
    names(kpss.header[[i]]) <- c(" ", i)
}
```

```
for (i in c("Trend")) {
    for (j in 1:ncol(gpw.1.xts)) {
        kpss.tab[[i]][j, 2] <- kpss[[i]][[j]]$statistic[[1]]
        kpss.tab[[i]][j, 3] <- kpss[[i]][[j]]$p.value
        if (kpss.tab[[i]][j, 2] <= 0.119) {
            kpss.tab[[i]][j, 4] <- "I(0) ***"
        }
        if (kpss.tab[[i]][j, 2] <= 0.146
            & kpss.tab[[i]][j, 2] > 0.119) {
            kpss.tab[[i]][j, 4] <- "I(0) **"
        }
        if (kpss.tab[[i]][j, 2] <= 0.216
            & kpss.tab[[i]][j, 2] > 0.146) {
            kpss.tab[[i]][j, 4] <- "I(0) *"
        }
        if (kpss.tab[[i]][j, 2] > 0.216) {
            kpss.tab[[i]][j, 4] <- "I(1)"
        }
    }
    colnames(kpss.tab[[i]]) <- c(
        "zmienna",
        "statystyka",
        "p-value",
        "decyzja"
    )
    kpss.tab[[i]][, c("statystyka", "p-value")] <- round(
        kpss.tab[[i]][, c("statystyka", "p-value")],
        digits = 2
    )
    kpss.header[[i]] <- c(" " = 1, i = 3)
    names(kpss.header[[i]]) <- c(" ", i)
```

```
}
```

Pierwsza współrzędna listy `kpss.tab` (`Level`) ma postać.

```
kpss.tab[[1]]
```

```
##     zmienna statystyka p-value decyzja
## 1       wig        2.8    0.01    I(1)
## 2     wig20        1.5    0.01    I(1)
## 3 wig_banki        1.0    0.01    I(1)
```

Druga współrzędna listy `kpss.tab` (`Trend`) ma postać.

```
kpss.tab[[2]]
```

```
##     zmienna statystyka p-value  decyzja
## 1       wig       0.14    0.05  I(0) **
## 2     wig20       0.10    0.10 I(0) ***
## 3 wig_banki       0.15    0.05   I(0) *
```

Każdą tablicę można sformatować za pomocą funkcji `kable`. Wyjaśnia to poniższy kod dla przypadku testu `Level`.

```
kable(
    kpss.tab[[1]],
    digits = 2,
    format = "latex",
    caption = "Wyniki testu KPSS dla indeksów giełdowych -- wyraz
     ↪  wolny",
  label = "ewvuvw",
    booktabs = TRUE,
    longtable = FALSE,
    linesep = ""
) %>%
kable_styling(
    full_width = FALSE,
    latex_options = "hold_position"
) %>%
add_header_above(
    header = kpss.header[[1]]
) %>%
footnote(
    kpss.note,
    threeparttable = TRUE
)
```

Tablica 3.16. Wyniki testu KPSS dla indeksów giełdowych – wyraz wolny

	Level		
zmienna	statystyka	p-value	decyzja
wig	2.8	0.01	I(1)
wig20	1.5	0.01	I(1)
wig_banki	1.0	0.01	I(1)

Note:
Symbole oznaczają: I(1) niestacjonarność, I(0) stacjonarność, ***, **, * statystyczną istotność na poziomie odpowiednio 1%, 5% i 10%.

Wyniki dla testu KPSS tylko częściowo potwierdzają wyniki testu ADF (tabl. 3.16). Jednak ze względu na charakter procesów pokazanych na rys. 3.7 należy przyjąć wariant testu KPSS z wyrazem wolnym i bez trendu (`Level`). Na podstawie przeprowadzonych testów ADF i KPSS można stwierdzić, że wszystkie zmienne zbioru `gpw.1.xts` są niestacjonarne.

W następnym podrozdziale zostanie przedstawiony zarys metody regresji oraz testowania poprawności specyfikacji jednorównaniowego liniowego modelu ekonometrycznego.

3.3. Regresja i testowanie

3.3.1. Regresja liniowa

Modele ekonometryczne znajdują szerokie zastosowanie w praktyce. Służą do opisu zależności przyczynowo-skutkowych pomiędzy zmiennymi. Do głównych etapów modelowania ekonometrycznego – wymienionych w podrozdziale 3.1 – zalicza się:

1. Specyfikację modelu:
   - wybór zmiennej objaśnianej,
   - wybór zmiennych objaśniających,
   - wybór postaci funkcyjnej (liniowej lub nieliniowej),
   - przyjęcie założeń dotyczących struktury stochastycznej (m.in. rozkład, średnia, wariancja, autokorelacja składnika losowego).
2. Utworzenie bazy danych statystycznych:
   - wybór źródła danych (np. GUS, KNF, NBP, Eurostat, IMF itd.),

- weryfikacja poprawności danych (m.in. identyfikacja obserwacji nietypowych),
- kwantyfikacja zmiennych, np.:
  - przeliczenie danych z cen bieżących na ceny stałe,
  - konwersja indeksów jednopodstawowych na indeksy łańcuchowe lub odwrotnie,
  - obliczenie przyrostów lub temp wzrostu,
  - wyznaczenie relacji pomiędzy zmiennymi (np. wydajność pracy jako stosunek produkcji do zatrudnienia),
  - usunięcie sezonowości, trendu i cykliczności,
  - określenie tendencji długookresowej (np. z wykorzystaniem filtru Hodricka–Prescotta, analizy kointegracji lub funkcji trendu).

3. Estymację (szacowanie) parametrów modelu.

4. Weryfikację merytoryczno-statystyczną modelu:

   - ocena zgodności znaku ocen parametrów z teorią dotyczącą zmiennej objaśnianej,
   - statystyczna weryfikacja hipotez (istotność parametrów, stabilność parametrów, autokorelacja i heteroskedastyczność składnika losowego, normalność rozkładu składnika losowego, poprawność funkcyjna modelu),
   - wyznaczenie przedziałów ufności dla parametrów.

5. Ocenę jakości prognostycznej modelu:

   - analiza dopasowania modelu do danych empirycznych (współczynnik determinacji, statystyka punktów zwrotnych),
   - obliczenie i interpretacja błędów prognoz *ex post* i *ex ante*,
   - określenie przedziałów ufności dla prognoz.

6. Wykorzystanie modelu w praktyce:

   - analiza kształtowania się zmiennych w czasie,
   - prognozowanie wartości zmiennych,
   - formułowanie rekomendacji dla polityki gospodarczej.

Jednorównaniowy liniowy model ekonometryczny (model *regresji prostej*) można zapisać w następujący sposób:

$$y_t = \alpha_0 + \alpha_1 X_t + \epsilon_t \tag{3.33}$$

gdzie:

y_t – zmienna objaśniana (np. wydatki konsumpcyjne),

X_t – zmienna objaśniająca (np. dochód rozporządzalny),
α_0 i α_1 – *nieznane* parametry modelu,
ϵ_t – składnik losowy (zakłócenie), dla którego $\epsilon_t \sim N(0, \sigma_\epsilon^2)$, czyli $E(\epsilon_t) = 0$, $E(\epsilon_t' \epsilon_t) = \sigma_\epsilon^2$ oraz $cov(\epsilon_t, \epsilon_s) = 0$ dla $t \neq s$,
t – indeks czasu, $t = 1, 2, \ldots, T$, gdzie T oznacza liczebność próby (liczbę obserwacji statystycznych).

Równanie (3.33) nazywa się funkcją regresji w populacji, która reprezentuje prawdziwy związek pomiędzy zmiennymi y i X. Stanowi on tzw. proces generowania danych (*data generating process*, DGP). Jeśli nieznane parametry α_0 i α_1 zostaną oszacowane na podstawie próby statystycznej, a następnie zastąpione ocenami $\hat{\alpha}_0$ i $\hat{\alpha}_1$, to otrzyma się funkcję regresji w próbie:

$$y_t = \hat{\alpha}_0 + \hat{\alpha}_1 X_t + \hat{\epsilon}_t \tag{3.34}$$

Składnik losowy ϵ odgrywa bardzo ważną rolę w modelu ekonometrycznym. Można wskazać trzy podstawowe przesłanki uwzględnienia składnika losowego (Maddala, 2006):

1. Nieprzewidywalny czynnik losowości w zachowaniach ludzi, pogoda, wypadki losowe itp.
2. Wpływ pominiętych zmiennych – w związku z tym, że nie można uwzględnić wszystkich czynników mających wpływ na zmienną y.
3. Błąd pomiaru zmiennej y.

Parametr α_0 nazywa się wyrazem wolnym, natomiast α_1 parametrem strukturalnym (kierunkowym prostej). Parametr α_0 jest związany ze zmienną tożsamościowo równą 1, tj. $X_{0t} \equiv 1$, gdzie $t = 1, 2, \ldots, T$.

W notacji macierzowej model (3.33) można zapisać następująco:

$$y = X\alpha + \epsilon \tag{3.35}$$

gdzie:

y – wektor $T \times 1$ wartości zmiennej objaśnianej,
X – macierz $T \times k$ wartości zmiennych objaśniających,
α – wektor $k \times 1$ parametrów,
ϵ – wektor $T \times 1$ składników losowych,
k – liczba szacowanych parametrów.

Równanie (3.35) dla jednej zmiennej objaśniającej można zapisać w równoważnej postaci macierzowej:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_T \end{bmatrix} = \begin{bmatrix} 1 & X_1 \\ 1 & X_2 \\ \vdots & \vdots \\ 1 & X_T \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_T \end{bmatrix} \tag{3.36}$$

Model (3.36) służy do opisu relacji pomiędzy zmienną objaśnianą (zależną) y a zmienną objaśniającą (niezależną) X. W najprostszym przypadku przyjmuje się, że relacja ta jest *liniowa*.

Składnik losowy ϵ oznacza losowe zakłócenie. Wynika on z faktu, że nie można uwzględnić wszystkich czynników objaśniających zmienną y. Wpływ pozostałych czynników realizuje się w postaci składnika losowego. Zadaniem ekonometryka jest oszacowanie wektora parametrów α przy następujących założeniach (Greene, 2003):

1. Relacja łącząca zmienną y z X jest liniowa.
2. Wartość oczekiwana składnika losowego dla t-tej obserwacji nie zależy od wartości zmiennych objaśniających, $E(\epsilon_t|X) = 0$.
3. Każdy składnik losowy ϵ_t, $t = 1, 2, \ldots, T$, ma tę samą skończoną wariancję (jest *homoskedastyczny*), $var(\epsilon_t|X) = \sigma_\epsilon^2$.
4. Każdy składnik losowy ϵ_t jest nieskorelowany ze składnikiem losowym ϵ_s, $cov(\epsilon_t, \epsilon_s|X) = 0$ dla wszystkich $t \neq s$.
5. Analizę regresji przeprowadza się warunkowo względem obserwowanych wartości X.
6. Składnik losowy ma rozkład normalny, $\epsilon_t \sim N(0, \sigma_\epsilon^2)$.

Warunki (2)–(4) stanowią tzw. schemat (zestaw) założeń Gaussa–Markowa. Założenia (3) i (4) oznaczają, że $E(\epsilon\epsilon'|X) = \sigma_\epsilon^2 I$, gdzie I jest macierzą jednostkową stopnia T.

Parametry modelu (3.33) można oszacować na podstawie próby statystycznej za pomocą metody najmniejszych kwadratów (MNK) (dokładnie: metody najmniejszej *sumy* kwadratów). W ten sposób otrzymuje się wektor $\hat{\alpha}$ *ocen* (estymatorów) nieznanych parametrów α. MNK oznacza poszukiwanie minimum funkcji (Verbeek, 2017):

$$\begin{aligned} S(\alpha) &= \sum_{t=1}^{T} \epsilon_t^2 = \epsilon'\epsilon = (y - X\alpha)'(y - X\alpha) \\ &= y'y - 2y'X\alpha + \alpha'X'X\alpha \end{aligned}$$

Estymator MNK spełnia warunek:

$$\frac{\delta S}{\delta \alpha}\Big|_{\hat{\alpha}} = -2(X'y - X'X\hat{\alpha}) = 0 \tag{3.37}$$

Na podstawie warunku (3.37) otrzymuje się wyrażenie:

$$X'X\hat{\alpha} = X'y \tag{3.38}$$

Ostatecznie estymator MNK ma postać (pod warunkiem nieosobliwości macierzy $X'X$):

$$\hat{\alpha} = (X'X)^{-1}X'y \tag{3.39}$$

Na podstawie (3.35) i (3.39) łatwo zauważyć, że wektor *reszt* (realizacji składnika losowego) $\hat{\epsilon}$ ma postać:

$$\hat{\epsilon} = y - X\hat{\alpha} = y - \hat{y} \tag{3.40}$$

Z warunków (3.37) i (3.40) wynika, że:

$$X'(y - X\hat{\alpha}) = X'\hat{\epsilon} = 0 \tag{3.41}$$

Warunek (3.41) oznacza, że każda kolumna macierzy X jest *ortogonalna* względem wektora reszt modelu. Wartości teoretyczne $\hat{y}$ są związane z wartościami rzeczywistymi y za pomocą relacji:

$$\hat{y} = X\hat{\alpha} = X(X'X)^{-1}X'y = Hy \tag{3.42}$$

gdzie macierz $H = X(X'X)^{-1}X'$ o wymiarach $T \times T$ transformuje wektor wartości *rzeczywistych* na wektor wartości *teoretycznych*.

Przy założeniach (2)–(4), estymator MNK $\hat{\alpha}$ dla wektora parametrów α, jest najlepszym estymatorem w klasie nieobciążonych estymatorów liniowych (BLUE, *best linear unbiased estimator*). Wariancja estymatora $\hat{\alpha}$ jest następująca:

$$var(\hat{\alpha}|X) = \sigma_\epsilon^2(X'X)^{-1} \tag{3.43}$$

Wariancja σ_ϵ^2 jest nieznana i trzeba ją zastąpić jej *szacunkiem*. Nieobciążonym estymatorem wariancji σ_ϵ^2 jest s^2 w postaci:

$$s^2 = \frac{1}{T-k}\hat{\epsilon}'\hat{\epsilon} \tag{3.44}$$

gdzie k oznacza liczbę szacowanych parametrów modelu (3.35).

Na mocy równań (3.43) oraz (3.44), estymator wariancji wektora $\hat{\alpha}$ wyraża się następującym wzorem:

$$\widehat{var}(\hat{\alpha}|X) = s^2(X'X)^{-1} \tag{3.45}$$

W oparciu o estymator wariancji (3.45) można określić *precyzję* szacunku parametrów modelu. Niech c_{ii} oznacza element diagonalny (i,i) macierzy $(X'X)^{-1}$, a $\hat{\alpha}_i$ – i-tą współrzędną wektora ocen parametrów. Wówczas estymatorem wariancji oceny $\hat{\alpha}_i$ jest wyrażenie s^2c_{ii}. Pierwiastek kwadratowy z tej wariancji nazywany jest *błędem standardowym* estymatora $\hat{\alpha}_i$ i oznaczany jako $S(\hat{\alpha}_i) = s\sqrt{c_{ii}}$.

Na mocy założeń (3), (4) i (6), składniki losowe ϵ_t stanowią *ciąg niezależnych zmiennych losowych o jednakowym rozkładzie*, ze średnią równą zero i wariancją σ_ϵ^2. Na podstawie założeń (2) i (6), estymator MNK $\hat{\alpha}$ ma rozkład normalny ze średnią α i wariancją $\sigma_\epsilon^2(X'X)^{-1}$:

$$\hat{\alpha} \sim N(\alpha, \sigma_\epsilon^2(X'X)^{-1}) \tag{3.46}$$

Podane powyżej warunki stanowią podstawę do przeprowadzenia w podrozdziale 3.3.2 wnioskowania statystycznego dotyczącego jakości szacunków modelu (3.35).

3.3.2. Testowanie specyfikacji modeli

Testowanie hipotez statystycznych stanowi kluczowy element analizy w modelach ekonometrycznych. Polega ono na ocenie, czy zaobserwowane dane empiryczne są zgodne z określonym założeniem dotyczącym populacji, z której pochodzą. W kontekście ekonometrii hipotezy najczęściej odnoszą się do parametrów modelu, takich jak współczynniki regresji czy wariancja składnika losowego.

Proces testowania polega na sformułowaniu hipotezy zerowej (H_0), która reprezentuje domyślne założenie, oraz hipotezy alternatywnej (H_1), konkurencyjnej wobec H_0. Następnie, na podstawie dostępnej próby statystycznej, oblicza się wartość statystyki testowej, która pozwala ocenić stopień zgodności danych z hipotezą H_0.

Fundamentalną rolę odgrywa poziom istotności, który określa prawdopodobieństwo popełnienia błędu pierwszego rodzaju, tj. odrzucenia prawdziwej hipotezy zerowej. W praktyce powszechnie stosuje się m.in. test *t* (*t*-Studenta), test F oraz test χ^2.

Wynik testu ocenia się na podstawie *p*-wartości (*p-value*), która określa krańcowy poziom istotności testu statystycznego – najmniejszy poziom istotności, przy którym hipotezę zerową można jeszcze odrzucić. Jeśli *p*-wartość jest mniejsza od przyjętego poziomu istotności, hipotezę zerową H_0 odrzuca się na rzecz hipotezy alternatywnej H_1.

Statystyczne testowanie hipotez – przedstawione w podrozdziałach 3.3.2.1-3.3.2.6 – dostarcza narzędzi do podejmowania decyzji na podstawie danych, co ma kluczowe znaczenie zarówno dla weryfikacji teorii ekonomicznych, jak i oceny jakości modeli ekonometrycznych.

3.3.2.1. Normalność

Test Shapiro–Wilka należy do najczęściej stosowanych testów statystycznych służących do oceny normalności rozkładu składnika losowego (Shapiro i Wilk, 1965). Celem testu jest sprawdzenie, czy badana próbka pochodzi z populacji o rozkładzie normalnym, co stanowi istotne założenie wielu metod statystycznych.

Test opiera się na analizie korelacji pomiędzy uporządkowanymi wartościami empirycznymi a odpowiadającymi im wartościami teoretycznymi wynikającymi z rozkładu normalnego. Wynik testu wyrażany jest za pomocą statystyki W, przy czym wartości bliskie jedności wskazują na zgodność z rozkładem normalnym.

Hipotezy statystyczne w teście Shapiro–Wilka mają postać:

H_0 : próbka pochodzi z populacji o rozkładzie normalnym
H_1 : próbka nie pochodzi z populacji o rozkładzie normalnym

Postać statystyki testowej – opartej na ilorazie dwóch różnych estymatorów wariancji σ^2 – jest następująca (Spanos, 1986):

$$W = \frac{b^2}{s^2} \tag{3.47}$$

gdzie:

$$b = \sum_{t=1}^{n} a_{T-t+1}(x_{(T-t+1)} - x_{(t)}) \tag{3.48}$$

$x_{(t)}$ – posortowane rosnąco wartości testowanej zmiennej, takie że $x_{(t-1)} \leq x_{(t)}$ dla $t = 2, 3, \ldots, T$,
$n = \frac{T}{2}$ – jeśli T jest liczbą parzystą lub $n = \frac{T-1}{2}$, jeśli T jest liczbą nieparzystą,
a_t – wartości tablicowe podane w opracowaniu Shapiro i Wilk (1965) dla $T \leq 50$, a następnie rozszerzone w opracowaniu Royston (1995) dla $3 \leq T \leq 5000$,
s^2 – wariancja próbkowa zmiennej x_t.

Test Jarque'a–Bery (Jarque i Bera, 1987) jest popularnym testem statystycznym służącym do oceny, czy dane mają rozkład normalny[2]. Test opiera się na analizie współczynników skośności (*skewness*) oraz kurtozy (*kurtosis*) próby, które – w przypadku rozkładu normalnego – powinny przyjmować wartości charakterystyczne dla niego: skośność równą 0 oraz kurtozę równą 3.

[2] Por. również test normalności Doornika–Hansena (Doornik i Hansen, 2008).

Wynik testu wyrażany jest za pomocą statystyki LM, której wysoka wartość wskazuje na istotne odchylenia danych od normalności. Test ten znajduje szerokie zastosowanie, zwłaszcza w analizach ekonometrycznych, ze względu na swoją prostotę i przejrzystość interpretacyjną.

Hipotezy statystyczne w teście Jarque'a–Bery są takie same jak w przypadku testu Shapiro–Wilka. Statystyka testowa, oznaczana jako LM, ma postać:

$$LM = T\left[\left(\sqrt{b_1}\right)^2/6 + (b_2 - 3)^2/24\right] \tag{3.49}$$

gdzie:

$\sqrt{b_1} = \hat{\mu}_3/\hat{\mu}_2^{3/2}$,
$b_2 = \hat{\mu}_4/\hat{\mu}_2^2$,
$\hat{\mu}_j = \sum_{t=1}^{T}(\hat{\epsilon}_t - \bar{\hat{\epsilon}})^j/T$,
$\bar{\hat{\epsilon}} = \sum_{t=1}^{T}\hat{\epsilon}_t/T$.

Skośność

Dla próbki zawierającej T wartości $x_1, x_2, \ldots, x_T$, skośność γ_1 jest obliczana jako:

$$\gamma_1 = \frac{\frac{1}{T}\sum_{t=1}^{T}(x_t - \bar{x})^3}{\left(\frac{1}{T}\sum_{t=1}^{T}(x_t - \bar{x})^2\right)^{3/2}} \tag{3.50}$$

gdzie:

x_t – t-ta wartość w próbce,
$\bar{x}$ – średnia dla próbki,
T – liczebność próbki.

Wzór ten można uprościć do:

$$\gamma_1 = \frac{m_3}{m_2^{3/2}} \tag{3.51}$$

gdzie:

$m_2 = \frac{1}{T}\sum_{t=1}^{T}(x_t - \bar{x})^2$ – drugi moment centralny (wariancja),
$m_3 = \frac{1}{T}\sum_{t=1}^{T}(x_t - \bar{x})^3$ – trzeci moment centralny.

Kurtoza

Dla próbki zawierającej T wartości $x_1, x_2, \ldots, x_T$, kurtoza γ_2 jest obliczana jako:

$$\gamma_2 = \frac{\frac{1}{T}\sum_{t=1}^{T}(x_t - \bar{x})^4}{\left(\frac{1}{T}\sum_{t=1}^{T}(x_t - \bar{x})^2\right)^2} \tag{3.52}$$

gdzie:

x_t – t-ta wartość w próbce,
$\bar{x}$ – średnia dla próbki,
T – liczebność próbki.

Wzór ten można uprościć do:

$$\gamma_2 = \frac{m_4}{m_2^2} \tag{3.53}$$

gdzie:

$m_2 = \frac{1}{T}\sum_{t=1}^{T}(x_t - \bar{x})^2$ – drugi moment centralny,
$m_4 = \frac{1}{T}\sum_{t=1}^{T}(x_t - \bar{x})^4$ – czwarty moment centralny.

W praktyce często używa się tzw. kurtozy nadwyżkowej (*excess kurtosis*), definiowanej jako:

$$\gamma_2^* = \gamma_2 - 3 \tag{3.54}$$

Kurtoza nadwyżkowa wyraża, o ile kurtoza danego rozkładu odbiega od kurtozy rozkładu normalnego, która wynosi 3. To właśnie tę wartość uwzględnia się w statystyce testowej Jarque'a–Bery, co sprawia, że test opiera się na porównaniu kurtozy nadwyżkowej z wartością zero.

Wartości kurtozy nadwyżkowej interpretuje się następująco:

- kurtoza równa zero wskazuje na rozkład zbliżony do normalnego (mezokurtoza),
- kurtoza dodatnia oznacza, że rozkład ma bardziej *spiczasty* kształt z cięższymi ogonami (leptokurtoza),
- kurtoza ujemna oznacza rozkład bardziej *spłaszczony* z lżejszymi ogonami (platykurtoza).

Wzory (3.50)–(3.53) służą do oceny stopnia asymetrii oraz kurtozy. W szczególności kurtoza informuje, czy dane wykazują tendencję do występowania wartości skrajnych w porównaniu do rozkładu normalnego.

Statystyki $\sqrt{b_1}$ i b_2 dla rozkładu próbkowego nazywa się odpowiednio współczynnikami skośności i kurtozy. W związku z tym statystyka LM testu Jarque'a–Bery służy do sprawdzenia zgodności tych współczynników z wartościami charakterystycznymi dla rozkładu normalnego. Statystyka LM ma asymptotyczny rozkład $\chi^2(2)$, ponieważ jest sumą kwadratów dwóch asymptotycznie niezależnych zmiennych losowych o rozkładzie normalnym standaryzowanym.

3.3.2.2. Heteroskedastyczność

W schemacie założeń Gaussa–Markowa przyjmuje się, że każdy składnik losowy ϵ_t, $t = 1, 2, \ldots, T$ ma tę samą skończoną wariancję $var(\epsilon_t|X) = \sigma^2_\epsilon$. Uchylenie tego założenia oznacza, że składnik losowy jest *heteroskedastyczny*. W takiej sytuacji estymator MNK pozostaje nieobciążony, jednak klasyczny estymator jego macierzy kowariancji jest obciążony. Prowadzi to do zniekształcenia wartości błędów standardowych, a w konsekwencji do niepoprawnych wniosków w testach istotności parametrów. Dlatego testowanie heteroskedastyczności stanowi standardową procedurę podczas wnioskowania statystycznego w modelach ekonometrycznych. W przykładach empirycznych w podrozdziale 3.3.3 wnioskowanie przeprowadzono na podstawie trzech testów:

- Goldfelda–Quandta (Goldfeld i Quandt, 1965),
- Breuscha–Pagana (Breusch i Pagan, 1979; Verbeek, 2017),
- White'a (White, 1980).

Test Goldfelda–Quandta służy do sprawdzenia, czy wariancja składnika losowego zmienia się w zależności od poziomu zmiennej objaśniającej. Analiza polega na porównaniu wariancji między dwiema grupami danych, a wynik testu jest wyrażany za pomocą statystyki F, która wskazuje, czy występuje statystycznie istotna różnica między tymi wariancjami.

W związku z powyższym test Goldfelda–Quandta jest znany jako test ilorazu wariancji. W tym teście wymaga się uprzedniego uporządkowania danych rosnąco według zmiennej, o której przypuszcza się, że wpływa na zmienność składnika losowego, a następnie ich podziału na trzy grupy. Grupa pierwsza składa się z pierwszych T_1 obserwacji z wariancją σ^2_1, grupa druga z ostatnich T_2 obserwacji z wariancją σ^2_2, a grupa trzecia z $T_3 = T - T_1 - T_2$ obserwacji środkowych. Grupę środkową pomija się w analizie, aby zwiększyć kontrast między porównywanymi grupami i zminimalizować efekt przejściowy. Hipotezy badawcze są następujące:

$$H_0 : \sigma^2_1 = \sigma^2_2$$
$$H_1 : \sigma^2_2 > \sigma^2_1$$

Następnie stosuje się MNK w każdej grupie danych i otrzymuje odrębnie sumy kwadratów reszt SSR_1 i SSR_2 oraz szacunki wariancji reszt $s^2_1 = SSR_1/(T_1 - k)$ i $s^2_2 = SSR_2/(T_2 - k)$. Przy założeniu, że składniki losowe ϵ_t tworzą ciąg niezależnych zmiennych losowych o rozkładzie normalnym, iloraz wyrażeń SSR_j/σ^2_j ma rozkład $\chi^2(T_j - k)$, gdzie $j = 1, 2$, a k jest liczbą szacowanych parametrów. Wówczas iloraz:

$$\frac{s_2^2/\sigma_2^2}{s_1^2/\sigma_1^2} \tag{3.55}$$

może zostać wykorzystany do konstrukcji statystyki testowej, ponieważ przy założeniu hipotezy zerowej $\sigma_1^2 = \sigma_2^2$ powyższy iloraz upraszcza się do stosunku dwóch niezależnych estymatorów wariancji, który ma rozkład F.

Zatem przy założeniu prawdziwości hipotezy zerowej statystyka testowa ma postać:

$$F = \frac{s_2^2}{s_1^2} \sim F(T_2 - k, T_1 - k) \tag{3.56}$$

Hipotezę zerową odrzuca się, jeżeli wartość statystyki F jest większa od wartości krytycznej, tzn. gdy statystyka F przyjmuje dostatecznie duże wartości.

Test Breuscha–Pagana opiera się na analizie reszt modelu, badając zależność między kwadratami reszt a zmiennymi objaśniającymi. Celem testu jest sprawdzenie, czy wariancja składnika losowego pozostaje stała, czy też zmienia się w zależności od wartości zmiennych objaśniających. Wynik testu jest wyrażany za pomocą statystyki ξ, która pozwala na ocenę, czy istnieje istotna różnica w wariancji składnika losowego.

W celu przedstawienia testu Breuscha–Pagana trzeba odwołać się do równań (3.33) i (3.34). Należy zauważyć, że w równaniu (3.34) składnik losowy ϵ_t został zastąpiony resztą $\hat{\epsilon}_t$, tj. realizacją składnika losowego w próbie statystycznej. W tym podejściu zakłada się, że wariancja składnika losowego jest funkcją zmiennych objaśniających modelu (3.33). Ponadto, asymptotyczny rozkład statystyki testowej Breuscha–Pagana zależy od normalności rozkładu zakłóceń ϵ_t modelu. Tym samym test jest wrażliwy na odchylenia składnika losowego od rozkładu normalnego. Aby temu przeciwdziałać, Koenker (1981) zaproponował wersję testu odporną na naruszenie założenia normalności. W teście Breuscha–Pagana stawia się następujące hipotezy:

H_0 : wariancja składnika losowego jest jednorodna
H_1 : wariancja składnika losowego jest niejednorodna

Test Breuscha–Pagana polega na wykonaniu szacunków następującej regresji pomocniczej:

$$\hat{\epsilon}_t^2 = \gamma_0 + \gamma_1 X_t + \nu_t \tag{3.57}$$

gdzie $\nu_t \sim \text{iid}(0, \sigma_\nu^2)$, oraz na testowaniu hipotezy, że $\gamma_1 = 0$. Sprawdzian testu ma postać:

$$\xi = TR^2 \tag{3.58}$$

gdzie:

T – liczba obserwacji w regresji pomocniczej,
R^2 – wartość współczynnika determinacji w regresji pomocniczej.

Przy założeniu prawdziwości hipotezy H_0, statystyka testowa ma rozkład χ^2 z liczbą stopni swobody równą liczbie regresorów w równaniu pomocniczym (z wyłączeniem wyrazu wolnego). W powyższym przykładzie jest to rozkład $\chi^2(1)$. Hipotezę zerową odrzuca się, jeśli zbyt duża część wariancji reszt zostaje wyjaśniona przez zmienne objaśniające.

Kolejnym testem na heteroskedastyczność jest test White'a. Hipotezy statystyczne są takie same jak w teście Breuscha–Pagana. Test ten zakłada bardziej ogólną postać zależności wariancji od zmiennych objaśniających, uwzględniając również ich kwadraty i iloczyny. Test White'a służy do wykrywania różnych form heteroskedastyczności, które mogą pogarszać własności estymatora MNK. Test polega na wykonaniu szacunków następującej regresji pomocniczej:

$$\hat{\epsilon}_t^2 = \gamma_0 + \gamma_1 X_t + \gamma_2 X_t^2 + \nu_t \tag{3.59}$$

gdzie $\nu_t \sim \text{iid}(0, \sigma_\nu^2)$, oraz na testowaniu hipotezy, że $\gamma_1 = \gamma_2 = 0$. Sprawdzian testu ma postać:

$$\xi = TR^2 \tag{3.60}$$

gdzie:

T – liczba obserwacji w regresji pomocniczej,
R^2 – wartość współczynnika determinacji w regresji pomocniczej.

Przy założeniu prawdziwości hipotezy H_0 statystyka testowa ma asymptotyczny rozkład χ^2 z liczbą stopni swobody równą liczbie regresorów w równaniu pomocniczym (z wyłączeniem wyrazu wolnego). W omawianym przypadku jest to rozkład $\chi^2(2)$. Hipotezę zerową odrzuca się, jeśli zbyt duża część wariancji reszt zostaje wyjaśniona przez dodatkowe zmienne objaśniające.

3.3.2.3. Autokorelacja

Autokorelacja składnika losowego powoduje, że nie są spełnione założenia schematu Gaussa–Markowa. W takim przypadku błędy standardowe estymatorów oraz statystyki testowe mają niepożądane własności. Nawet w dużych próbach własności statystyk testowych nie ulegają poprawie, tj. nie zachowują się zgodnie z przewidywaniami teorii asymptotycznej.

Autokorelacja składnika losowego oznacza, że nie jest spełnione założenie o braku korelacji błędów specyfikacji ϵ_t w dwóch różnych momentach czasu (Wooldridge, 2020):

$$cor(\epsilon_t, \epsilon_s|X) \neq 0 \quad \text{dla } t \neq s \tag{3.61}$$

Autokorelację pierwszego rzędu, AR(1), składnika losowego można przedstawić w następujący sposób:

$$\epsilon_t = \rho\epsilon_{t-1} + \eta_t, \quad t = 1, 2, ..., T \tag{3.62}$$

$$|\rho| < 1 \tag{3.63}$$

gdzie: $\eta_t \sim \text{iid}(0, \sigma_\eta^2)$.

Jeśli zignoruje się autokorelację składnika losowego ϵ_t, to szacunek wariancji estymatora $\hat{\alpha}$ będzie zwykle obciążony, czyli niezgodny z rzeczywistą wariancją, gdy $\rho \neq 0$. Jeśli $\rho > 0$, co często występuje w zastosowaniach ekonomicznych, to wariancja estymatora jest zaniżona względem *prawdziwej* wariancji. Dla wysokich wartości współczynnika autokorelacji ρ obciążenie wariancji estymatora może być znaczne. W warunkach autokorelacji nie powinno stosować się statystyk t, F i LM, ponieważ ich teoretyczne rozkłady są wówczas niepoprawne.

W celu testowania autokorelacji składnika losowego w analizach empirycznych, przedstawionych w rozdziale 3.3.3, zastosowano cztery testy:

- Durbina–Watsona (Durbin i Watson, 1950),
- Boxa–Pierce'a (Box i Pierce, 1970),
- Ljung–Boxa (Ljung i Box, 1978),
- Breuscha–Godfrey'a (Breusch, 1978; Godfrey, 1978).

Test Durbina–Watsona służy do testowania autokorelacji rzędu pierwszego – AR(1) – składników losowych w oparciu o reszty modelu ekonometrycznego. Sprawdzian testu ma postać:

$$DW = \frac{\sum_{t=2}^{T}(\hat{\epsilon}_t - \hat{\epsilon}_{t-1})^2}{\sum_{t=1}^{T}\hat{\epsilon}_t^2} \tag{3.64}$$

Można pokazać, że:

$$DW \approx 2(1 - \hat{\rho}) \tag{3.65}$$

gdzie $\hat{\rho}$ jest estymatorem MNK parametru ρ otrzymanym na podstawie reszt $\hat{\epsilon}_t$ i $\hat{\epsilon}_{t-1}$.

Durbin i Watson (1950) otrzymali rozkład statystyki DW, który zależy od założeń klasycznego modelu regresji, w tym m.in. od *normalności* rozkładu składnika losowego. Ponadto, rozkład tej statystyki zależy od wartości zmiennych niezależnych, ich liczby, wielkości próby statystycznej oraz obecności wyrazu wolnego w regresji. Zwykle przyjmuje się następujące hipotezy w teście Durbina–Watsona:

$$H_0 : \rho = 0$$
$$H_1 : \rho > 0$$

Jeśli $\hat{\rho} \approx 0$, to $DW \approx 2$. Podobnie, jeśli $\hat{\rho} > 0$, to $DW < 2$[3]. Test Durbina–Watsona polega na porównaniu wartości statystyki testowej z dwiema wartościami krytycznymi – d_L (dolną) i d_U (górną). Jeśli $DW < d_L$, to należy odrzucić H_0 na rzecz H_1. Jeśli $DW > d_U$, to nie ma podstaw do odrzucenia H_0. Wyniki testu są niekonkluzywne, jeśli $d_L \leq DW \leq d_U$. Podany przedział nazywa się przedziałem *niekonkluzywności* testu.

Jak wskazano wyżej, test Durbina–Watsona ma pewne ograniczenia:

1. Występowanie przedziału niekonkluzywności.
2. Trudności w obliczeniu dokładnych wartości krytycznych.
3. Konieczność uwzględnienia wyrazu wolnego w regresji.
4. Brak zastosowania dla regresji *dynamicznych*, tj. z opóźnioną zmienną objaśnianą.

W celu przedstawienia testów Boxa–Pierce'a i Ljung–Boxa trzeba wprowadzić pojęcie funkcji autokorelacyjnej (Box i in., 2015). Niech dany będzie wektor reszt $\hat{\epsilon} = [\hat{\epsilon}_1, \hat{\epsilon}_2, \dots, \hat{\epsilon}_T]'$. Funkcja autokorelacyjna ma postać:

$$\hat{\rho}_s = \frac{\gamma_s}{\gamma_0} \tag{3.66}$$

gdzie: $\gamma_s = \frac{1}{T}\sum_{t=s+1}^{T} \hat{\epsilon}_t \hat{\epsilon}_{t-s}$, $s = 0, 1, 2, \dots, S$, przy czym $S < T/4$.

Stąd γ_s jest autokowariancją reszt. Dla $s = 0$ otrzymuje się autokowariancję dla opóźnienia zero, co oznacza autokowariancję $\hat{\epsilon}_t$ i $\hat{\epsilon}_t$, tj. oszacowanie wariancji procesu $\hat{\epsilon}_t$.

Gdyby model był poprawny, a reszty były generowane przy założeniu *prawdziwych* wartości parametrów, to proces $\hat{\epsilon}_t$ byłby ciągiem niezależnych zmiennych losowych. Wówczas ciąg pierwszych m próbkowych współczynników autokorelacji

[3] Dla przypadku, gdy statystyka Durbina–Watsona przyjmuje wartość większą niż 2, czyli w sytuacji ujemnej autokorelacji składników losowych, por. np. Durbin i Watson (1950); Greene (2003); Verbeek (2017).

$\hat{\rho} = [\hat{\rho}_1, \hat{\rho}_2, \dots, \hat{\rho}_m]'$ dla dużego T miałby wielowymiarowy rozkład normalny. Dla współczynników autokorelacji $\hat{\rho}_s$, gdzie $s = 1, \dots, m$, formułuje się następnie hipotezy dotyczące niezależności reszt w kolejnych opóźnieniach.

W testach Boxa–Pierce'a i Ljung–Boxa hipotezy są następujące:

$$\begin{aligned} &H_0 : \rho_1 = \rho_2 = \dots = \rho_m = 0 \\ &H_1 : \rho_i \neq 0, \quad \text{dla wybranego } i \in 1, \dots, m \end{aligned} \tag{3.67}$$

Hipoteza H_0 jest hipotezą łączną. W teście Boxa–Pierce'a stosuje się następującą statystykę Q:

$$Q = T \sum_{s=1}^{m} \hat{\rho}_s^2 \sim \chi^2(m) \tag{3.68}$$

W teście Ljung–Boxa, który ma lepsze własności w małych próbach, stosuje się zmodyfikowaną statystykę Q:

$$Q^* = T(T+2) \sum_{s=1}^{m} \frac{\hat{\rho}_s^2}{T-s} \sim \chi^2(m) \tag{3.69}$$

Jeśli testy Boxa–Pierce'a i Ljung–Boxa stosuje się do reszt modelu, to statystyki Q i Q^* mają asymptotyczny rozkład $\chi^2(m-k)$, gdzie $m \approx log(T)$, zaś k oznacza liczbę szacowanych parametrów regresji.

Testem alternatywnym dla testu Durbina–Watsona jest test Breuscha–Godfrey'a. Test ten nie ma ograniczeń testu Durbina–Watsona. Można go również łatwo zastosować w praktyce. Test Breuscha–Godfrey'a jest testem *LM* dla następujących hipotez:

$$\begin{aligned} &H_0 : \rho_1 = 0 \\ &H_1 : \rho_1 \neq 0 \end{aligned} \tag{3.70}$$

Test Breuscha–Godfrey'a polega na wykonaniu regresji reszt $\hat{\epsilon}_t$ względem reszt $\hat{\epsilon}_{t-1}$ oraz pozostałych regresorów modelu:

$$\hat{\epsilon}_t = \delta_0 + \delta_1 \hat{\epsilon}_{t-1} + \delta_2 X_t + \nu_t \tag{3.71}$$

gdzie: $\nu_t \sim \text{iid}(0, \sigma_\nu^2)$.

Test polega na sprawdzeniu, czy parametr δ_1 jest statystycznie istotny. Sprawdzianem testu jest statystyka:

$$\xi = TR^2 \sim \chi^2(1) \tag{3.72}$$

gdzie T jest liczbą obserwacji w regresji pomocniczej, zaś R^2 wartością współczynnika determinacji w regresji pomocniczej. Test Breuscha–Godfrey'a można uogólnić na przypadki autokorelacji wyższych rzędów. Dla autokorelacji drugiego rzędu należy uwzględnić dwa opóźnienia reszt w regresji pomocniczej, tj. $\hat{\epsilon}_{t-1}$ i $\hat{\epsilon}_{t-2}$.

3.3.2.4. Postać funkcyjna

Test RESET (*Regression Specification Error Test*) jest używany do wykrywania błędów specyfikacji w modelu regresji, takich jak pominięcie istotnych zmiennych objaśniających czy błędna postać funkcyjna relacji między zmiennymi. Test polega na sprawdzeniu, czy w modelu rozszerzonym o nieliniowe transformacje wartości teoretycznych występują istotne statystycznie efekty sugerujące błędną specyfikację. Z kolei test Harvey'a–Colliera jest stosowany w celu identyfikacji problemów z liniowością w modelu regresji, badając, czy reszty rekursywne są losowe i niezależne od zmiennych objaśniających. Oba testy są przydatne w diagnozowaniu nieprawidłowości w postaci funkcyjnej modelu ekonometrycznego, które mogą prowadzić do błędnych wniosków.

W celu testowania postaci funkcyjnej modelu (3.33) zastosowano wspomniane dwa testy:

- RESET (Ramsey, 1969; Kleiber i Zeileis, 2008; Brooks, 2019),
- Harvey'a–Colliera (Harvey i Collier, 1977; Harvey, 1990; Kianifard i Swallow, 1996; Kleiber i Zeileis, 2008).

Założenie o niezależności składnika losowego ϵ_t od zmiennych objaśniających X, tj. $E(\epsilon_t|X) = 0$ jest warunkiem zgodności estymatora MNK. W modelu (3.33) przyjęto, że relacja łącząca zmienne y i X jest liniowa względem parametrów i może być reprezentowana przez liniową funkcję. Jeśli postać funkcji jest błędnie wyspecyfikowana i relacja łącząca zmienne jest nieliniowa, to nie można założyć niezależności składnika losowego od zmiennych objaśniających. Liniowość związku można weryfikować za pomocą testu RESET Ramsey'a.

Test polega na oszacowaniu parametrów regresji pomocniczej, w której do modelu dodaje się wyższe potęgi wartości teoretycznych $\hat{y}_t$ – zwykle kwadraty i sześciany – obliczone na podstawie pierwotnego modelu (3.33).

$$y_t = \beta_1 + \beta_2 \hat{y}_t^2 + \beta_3 \hat{y}_t^3 + \cdots + \beta_p \hat{y}_t^p + \alpha X_t + \nu_t \tag{3.73}$$

gdzie: $\nu_t \sim \text{iid}(0, \sigma_\nu^2)$.

Zadaniem kolejnych potęg wartości teoretycznych $\hat{y}_t$ jest uchwycenie nieliniowości różnego typu. Można zauważyć, że potęgi wartości teoretycznych zawierają również potęgi i kombinacje zmiennych objaśniających. Sprawdzianem w teście RESET może być statystyka *LM*:

$$\xi = TR^2 \sim \chi^2(p-1) \tag{3.74}$$

gdzie T jest liczbą obserwacji w regresji pomocniczej, natomiast R^2 jest wartością współczynnika determinacji dla regresji pomocniczej, a $(p-1)$ oznacza liczbę dodatkowo dołączonych składników $\hat{y}_t^2, \dots, \hat{y}_t^p$.

Jeśli wartość statystyki testowej jest większa od wartości krytycznej rozkładu χ^2, to należy odrzucić hipotezę zerową, że liniowa postać funkcji jest poprawna.

W teście Harvey'a–Colliera użyto reszt rekursywnych w celu testowania potencjalnych form nieliniowości. Reszty rekursywne (sekwencyjne) otrzymuje się poprzez wielokrotne szacowanie parametrów modelu przy jednoczesnym zwiększaniu liczby obserwacji w próbie statystycznej. Reszty rekursywne dla modelu (3.33) można zapisać w następujący sposób (Kianifard i Swallow, 1996). Niech X_{j-1} oznacza macierz $(j-1) \times k$ początkowych obserwacji dla zmiennych objaśniających. Jeśli $(j-1) \geq k$ oraz macierz $X'_{j-1}X_{j-1}$ jest nieosobliwa, to można oszacować parametry za pomocą MNK:

$$\hat{\alpha}_{j-1} = (X'_{j-1}X_{j-1})^{-1}X'_{j-1}y_{j-1} \tag{3.75}$$

gdzie y_{j-1} oznacza wektor pierwszych $j-1$ współrzędnych wektora y, zaś $X = [x'_1, \dots, x'_T]'$.

Następnie wyznacza się jednookresową prognozę $\hat{y}_j = x'_j\hat{\alpha}_{j-1}$. Błąd prognozy jest równy $y_j - \hat{y}_j$, natomiast wariancja tego błędu wynosi $s^2[1 + x'_j(X'_{j-1}X_{j-1})^{-1}x_j]$, gdzie $s^2 = \hat{\epsilon}'\hat{\epsilon}/(T-k)$. Na tej podstawie oblicza się reszty rekursywne:

$$w_j = \frac{y_j - \hat{y}_j}{[1 + x'_j(X'_{j-1}X_{j-1})^{-1}x_j]^{1/2}} \tag{3.76}$$

gdzie: $j = k+1, \dots, T$.

Każda reszta w_j jest standaryzowaną resztą po dodaniu jednej obserwacji do zbioru danych. Takie reszty zostały wykorzystane w teście Harvey'a–Colliera. Przy założeniu modelu (3.33) reszty w_j mają rozkład normalny ze średnią równą zero. Sprawdzianem testu jest statystyka:

$$t = \frac{\bar{w}}{\hat{\sigma}\sqrt{T-k}} \sim t_{T-k-1} \tag{3.77}$$

gdzie: $\hat{\sigma}^2 = \sum_{j=k+1}^{T}(w_j - \bar{w})^2/(T-k-1)$, zaś $\bar{w} = \sum_{j=k+1}^{T} w_j/(T-k)$.

Należy zwrócić uwagę, że w przypadku regresji wielorakiej dane statystyczne są sortowane w porządku rosnącym lub malejącym względem wybranej zmiennej objaśniającej, dla której występuje podejrzenie, że jest źródłem błędu specyfikacji. W przypadku regresji prostej tę zasadę stosuje się odpowiednio. Decyzję na podstawie sprawdzianu testu podejmuje się, stosując dwustronny test *t*.

3.3.2.5. Istotność parametrów

Test istotności *t*-Studenta (test *t*) jest jednym z najczęściej stosowanych testów statystycznych w analizie regresji liniowej oraz przy porównywaniu średnich między grupami. Jego celem jest ocena, czy współczynniki regresji w modelu ekonometrycznym są statystycznie istotnie różne od zera.

Postać hipotez testu *t* jest następująca:

$$\begin{aligned} H_0&: \ \alpha = 0 \\ H_1&: \ \alpha \neq 0 \end{aligned} \tag{3.78}$$

Hipoteza zerowa H_0 zakłada, że współczynnik regresji jest równy zeru, czyli brak jest związku między zmienną objaśniającą a zmienną objaśnianą. Hipoteza alternatywna H_1 określa, że współczynnik ten jest różny od zera.

Statystykę testową oblicza się jako stosunek różnicy między oszacowanym współczynnikiem regresji a jego wartością wynikającą z hipotezy zerowej (zwykle równą zeru) do błędu standardowego tego estymatora:

$$t = \frac{\hat{\alpha} - 0}{SE(\hat{\alpha})} \tag{3.79}$$

gdzie:

$\hat{\alpha}$ – oszacowanie współczynnika regresji,
$SE(\hat{\alpha})$ – błąd standardowy tego oszacowania.

Wariancję estymatora $\hat{\alpha}$ opisano za pomocą wyrażenia (3.45). Błędy standardowe poszczególnych ocen parametrów to pierwiastki kwadratowe z elementów diagonalnych macierzy wariancji. Oznacza to, że błąd standardowy estymatora $\hat{\alpha}$ zależy zarówno od rozrzutu składnika losowego (czyli zmienności niewyjaśnionej przez model), jak i od informacji zawartej w zmiennych objaśniających. Im większa precyzja estymacji (czyli im mniejsza wartość $\hat{\sigma}^2$ i większa informacja w danych), tym mniejszy błąd standardowy oraz większa wartość statystyki *t*, co zwiększa szanse na odrzucenie hipotezy zerowej.

Statystyka testowa *t* ma rozkład Studenta z $T-k$ stopniami swobody, gdzie:

T – liczba obserwacji w próbie,
k – liczba oszacowanych parametrów w modelu (w tym wyrazu wolnego).

Wynik testu ocenia się na podstawie *p*-wartości lub wartości krytycznej rozkładu *t*-Studenta (przyjęto, że poziom istotności wynosi 0,05, czyli 5%).

$$\text{Odrzuca się } H_0, \text{ jeśli p-val} < 0.05 \quad \text{lub} \quad |t| > t_{0.05/2,T-k} \tag{3.80}$$

Jeśli *p*-wartość jest mniejsza niż założony poziom istotności, to hipotezę zerową się odrzuca. Jeśli *p*-wartość jest większa lub równa poziomowi istotności, to nie ma podstaw do odrzucenia hipotezy zerowej – współczynnik regresji uznaje się wówczas za statystycznie nieistotny.

Test istotności oparty jest na rozkładzie *t*-Studenta, który zależy od pojęcia liczby stopni swobody, dlatego ich zrozumienie jest niezbędne dla poprawnej interpretacji testu.

Stopnie swobody to liczba niezależnych obserwacji, które mogą swobodnie się zmieniać, przy jednoczesnym spełnieniu określonych warunków statystycznych (np. zachowania średniej). Pojęcie to jest kluczowe w estymacji parametrów oraz wnioskowaniu statystycznym.

1. **Stopnie swobody w estymacji parametrów**

W przypadku estymacji parametrów, liczba stopni swobody to liczba niezależnych wartości, które pozostają po uwzględnieniu liczby oszacowanych parametrów. Na przykład dla próby o wielkości T, jeśli szacuje się wartość średnią (jedno parametryczne oszacowanie), liczba stopni swobody wynosi $T - 1$, ponieważ po obliczeniu średniej jedna z obserwacji jest liniowo zależna od pozostałych.

2. **Stopnie swobody w rozkładzie *t*-Studenta**

W przypadku testu *t*, liczba stopni swobody odnosi się do liczby niezależnych danych pomniejszonej o liczbę oszacowanych parametrów. Liczba ta wpływa na kształt rozkładu *t*-Studenta – im jest większa, tym bardziej rozkład ten przypomina rozkład normalny. Dla próbki o T obserwacjach i k szacowanych parametrach liczba stopni swobody wynosi $T - k$. Ponieważ liczba stopni swobody wpływa na kształt rozkładu, to tym samym wpływa na decyzję o odrzuceniu hipotezy zerowej, ponieważ przy mniejszej liczbie stopni swobody rozkład *t*-Studenta ma dłuższe ogony, co zwiększa niepewność estymacji.

Test *t* znajduje szerokie zastosowanie w analizie ekonometrycznej, gdyż pozwala na ocenę istotności wpływu poszczególnych zmiennych objaśniających na zmienną objaśnianą w modelu.

3.3.2.6. Ocena jakości dopasowania do danych

Błąd średni równania

Błąd średni równania (MSE, *Mean Squared Error*) jest jedną z najczęściej stosowanych miar oceny jakości dopasowania modelu do danych empirycznych. Błąd MSE mierzy średni kwadrat różnicy między rzeczywistymi wartościami zmiennej objaśnianej a wartościami teoretycznymi (przewidywanymi przez model). Miara ta jest oparta na resztach modelu (realizacjach składnika losowego).

$$MSE = \frac{1}{T}\sum_{t=1}^{T}(y_t - \hat{y}_t)^2 \tag{3.81}$$

gdzie:

y_t – rzeczywiste wartości zmiennej objaśnianej (zależnej),
$\hat{y}_t$ – przewidywane (teoretyczne) wartości zmiennej objaśnianej,
T – liczba obserwacji.

MSE jest miarą błędu popełnianego przy przewidywaniu wartości zmiennej objaśnianej na podstawie wartości zmiennych objaśniających. Niższa wartość MSE wskazuje na lepsze dopasowanie modelu do danych, ponieważ oznacza, że różnice między rzeczywistymi a przewidywanymi wartościami są mniejsze. Wyższa wartość MSE świadczy natomiast o gorszym dopasowaniu modelu i większych błędach prognozy. Błąd MSE jest szczególnie przydatny w ocenie modeli, ponieważ jest łatwy do obliczenia oraz interpretacji.

Należy zauważyć, że MSE ma tę samą jednostkę, co zmienna objaśniana podniesiona do kwadratu. Dla łatwiejszej interpretacji często stosuje się pierwiastek kwadratowy z MSE, zwany błędem średnim standardowym (RMSE, *Root Mean Squared Error*).

$$RMSE = \sqrt{MSE} \tag{3.82}$$

RMSE wyrażony jest w tych samych jednostkach co zmienna objaśniana, co czyni go bardziej intuicyjnym w interpretacji. Należy zauważyć, że błąd MSE ma pewne wady. Ponieważ jest oparty na kwadratach różnic, jest wrażliwy na duże błędy (tzw. wartości odstające lub nietypowe), co może prowadzić do zniekształconej oceny jakości modelu.

Współczynnik determinacji

Współczynnik determinacji R^2 jest jedną z najważniejszych miar oceny dopasowania modelu do danych empirycznych. Określa on, jaka część zmienności zmiennej objaśnianej jest wyjaśniona przez zmienność zmiennych objaśniających w modelu. Wartość R^2 mieści się w przedziale od 0 do 1, gdzie wartość 1 oznacza, że model idealnie objaśnia zmienność w danych empirycznych, a wartość 0 wskazuje na brak

takiego wyjaśnienia. Należy zaznaczyć, że przedział $R^2 \in \langle 0, 1 \rangle$ ma zastosowanie wyłącznie w przypadku, gdy w modelu uwzględniono wyraz wolny. W modelach bez wyrazu wolnego R^2 może przyjmować wartości ujemne.

Dekompozycja wariancji całkowitej zmiennej objaśnianej y_t stanowi podstawę interpretacji współczynnika determinacji R^2 w klasycznym modelu regresji liniowej. W przypadku danych w postaci szeregów czasowych z T obserwacjami, wariancję całkowitą zmiennej objaśnianej y_t można zdekomponować następująco:

$$\underbrace{\sum_{t=1}^{T}(y_t - \bar{y})^2}_{\text{wariancja całkowita (SST)}} = \underbrace{\sum_{t=1}^{T}(\hat{y}_t - \bar{y})^2}_{\text{wariancja wyjaśniona (SSR)}} + \underbrace{\sum_{t=1}^{T}(y_t - \hat{y}_t)^2}_{\text{wariancja resztowa (SSE)}} \tag{3.83}$$

gdzie:

y_t – wartość rzeczywista zmiennej objaśnianej w chwili t,
$\hat{y}_t$ – wartość teoretyczna zmiennej objaśnianej w chwili t,
$\bar{y} = \frac{1}{T}\sum_{t=1}^{T} y_t$ – średnia arytmetyczna zmiennej y_t,
SST – suma kwadratów odchyleń wartości zmiennej y_t od średniej,
SSR – suma kwadratów odchyleń wartości zmiennej $\hat{y}_t$ od średniej (wyjaśniona zmienność),
SSE – suma kwadratów reszt (niewyjaśniona zmienność).

Współczynnik R^2 w kontekście regresji liniowej wyraża się jako stosunek sumy kwadratów regresji do całkowitej sumy kwadratów.

$$R^2 = \frac{\text{SSR}}{\text{SST}} = 1 - \frac{\text{SSE}}{\text{SST}} \tag{3.84}$$

Współczynnik $R^2 \in \langle 0, 1 \rangle$ informuje, jaka część zmienności y_t została wyjaśniona przez model. Wartości bliższe 1 świadczą o lepszym dopasowaniu modelu do danych empirycznych. Współczynnik R^2 mierzy więc, jaki odsetek zmienności obserwowanej w danych jest wyjaśniony przez model.

W przypadku regresji wielorakiej wartość R^2 może być myląca, ponieważ zawsze nie maleje wraz z dodawaniem kolejnych zmiennych objaśniających, nawet jeśli nie mają one istotnego wpływu. W takich przypadkach często stosuje się skorygowany współczynnik determinacji R^2_{adj}.

$$R^2_{\text{adj}} = 1 - (1 - R^2)\,\frac{T-1}{T-k-1} \tag{3.85}$$

gdzie:

T – liczba obserwacji,
k – liczba zmiennych objaśniających w modelu (bez wyrazu wolnego).

Skorygowany współczynnik determinacji uwzględnia *karę* za dodawanie zbędnych zmiennych, dzięki czemu jest bardziej wiarygodną miarą jakości dopasowania. Współczynnik determinacji i jego wersja skorygowana są szeroko stosowane w analizach ekonometrycznych. Pozwalają na ocenę dopasowania modelu do danych. W interpretacji należy jednak zawsze uwzględniać zarówno liczbę zmiennych w modelu, jak i wartość współczynnika skorygowanego. Należy pamiętać, że wysoka wartość współczynnika determinacji nie gwarantuje poprawnej specyfikacji modelu ani zależności przyczynowej.

W podrozdziale 3.3.3 zostanie przedstawiona analiza empiryczna obejmująca indeksy branżowe dla GPW w Warszawie, stopy procentowe, inflację w Polsce oraz ceny samochodów elektrycznych na podstawie danych Komisji Europejskiej.

3.3.3. Analiza empiryczna

3.3.3.1. Model indeksu branżowego GPW WIG-Banki

Modelowanie indeksów giełdowych jest jednym z kluczowych zagadnień ekonometrii finansowej, ponieważ umożliwia lepsze zrozumienie dynamiki rynku kapitałowego, ocenę ryzyka inwestycyjnego oraz stosowanie strategii portfelowych. Indeksy takie jak WIG, WIG20 czy mWIG40 dla GPW stanowią syntetyczną miarę kondycji rynku, a ich analiza wymaga zastosowania zaawansowanych narzędzi statystycznych.

W literaturze dominuje podejście oparte na modelach szeregów czasowych, w szczególności ARIMA – uogólnionych modelach autoregresyjnych i średnich ruchomych dla niestacjonarnych szeregów czasowych (*AutoRegressive Integrated Moving Average*), GARCH – standardowych modelach ekonometrii finansowej służących do opisu i prognozowania zmienności szeregów czasowych – i ich wariantach (*Generalized AutoRegressive Conditional Heteroskedasticity*), które dobrze opisują warunkową zmienność (Bollerslev, 1986; Hamilton, 1994). Model GARCH pozwala na modelowanie wariancji warunkowej, tj. wariancji zależnej od przeszłych wartości wariancji i błędów losowych, co umożliwia uchwycenie zjawiska grupowania zmienności (*volatility clustering*) typowego dla danych finansowych. Współcześnie coraz częściej stosuje się również metody hybrydowe oraz uczenie maszynowe, łączące klasyczne techniki ekonometryczne z algorytmami sztucznej inteligencji (Zhang i in., 1998; Fischer i Krauss, 2018).

Jak wspomniano, jednym z najbardziej rozpowszechnionych podejść do analizy indeksów giełdowych jest zastosowanie modeli zmienności warunkowej z rodziny GARCH. Modele te służą nie tylko do opisu zmienności, lecz także do analizy ryzyka, oceny efektywności rynku oraz prognozowania stóp zwrotu (Fiszeder, 2003, 2009). Wykorzystanie danych o wysokiej częstotliwości, np. pięciominutowych, poprawia jakość prognoz, szczególnie przy zastosowaniu miar takich jak zmienność zrealizowana (Doman, 2004). Badania wskazują, że w modelach GARCH-M można uchwycić premię za ryzyko oraz asymetryczne reakcje rynku na szoki informacyjne

(Fiszeder i Kwiatkowski, 2005). Model GARCH-M (*GARCH-in-Mean*) rozszerza standardową postać modelu GARCH poprzez wprowadzenie warunkowej wariancji lub jej pierwiastka bezpośrednio do równania średniej, co umożliwia analizę wpływu zmienności na oczekiwaną stopę zwrotu. Dzięki temu można szacować zależność między ryzykiem a oczekiwanym zyskiem zgodnie z teorią premii za ryzyko. W analizie mikrostruktury rynku dekompozycja *spreadu bid-ask* na składowe związane z asymetrią informacji oraz nierównowagą podaży i popytu dostarcza cennych informacji o mechanizmach działania rynków wschodzących (Miłobędzki i Nowak, 2022).

Alternatywą dla klasycznych modeli zmienności są podejścia uwzględniające nieliniowość i zmienne reżimy rynkowe. Modele progowe TAR (*Threshold Autoregression*) i M-TAR (*Momentum Threshold Autoregression*) pozwalają wykrywać asymetrię w reakcjach indeksów na wzrosty oraz spadki cen, co może pozostać niezauważone w modelach liniowych (Miłobędzki, 2006). Model TAR różnicuje dynamikę szeregu w zależności od przekroczenia określonego progu, natomiast M-TAR wykorzystuje zmiany kierunku jako kryterium przełączania między reżimami. Oba modele są użyteczne w analizie sytuacji, gdy reakcje na spadki cen są silniejsze niż na wzrosty (lub odwrotnie), co wskazuje na obecność nieliniowej asymetrii. Bardziej zaawansowane są modele przełącznikowe oparte na łańcuchach Markowa, pozwalające na modelowanie zmiennych struktur ryzyka i nieobserwowalnych stanów rynku (Kwiatkowski, 2010). Efekty nieliniowe badano także przy użyciu wykładników Lapunowa i testów BDS (Doman i Doman, 2004), potwierdzając, że rynek akcji w Polsce jest złożony i odbiega od klasycznych założeń modelowych.

Opisane wyżej zjawiska lokalne na GPW nie funkcjonują w oderwaniu od szerszego kontekstu międzynarodowego. Badania pokazują istotne powiązania między rynkiem polskim a giełdami w USA i Wielkiej Brytanii. Li i Majerowska (2008), stosując asymetryczny model GARCH-BEKK, wykazały istnienie efektów przenoszenia zmienności z rynków rozwiniętych, choć ich siła była ograniczona, a czynniki krajowe dominowały. Wyniki te wskazują na potencjał dywersyfikacyjny rynków Europy Środkowo-Wschodniej. Fałdziński i in. (2012) analizowali przenoszenie ryzyka między rynkami, wykorzystując teorię ekstremalnych wartości (*EVT*) (Embrechts i in., 1997) i test przyczynowości w ryzyku (Hong i Liu, 2011), rozszerzony o miary *Expected Shortfall* (*ES*) (Acerbi i Tasche, 2002) oraz *Spectral Risk Measure* (*SRM*) (Acerbi, 2002). Badanie objęło 36 indeksów giełdowych i 19 kursów walutowych z lat 2006–2011, wykazując asymetrię w rozprzestrzenianiu się strat i zysków – straty przenosiły się szybciej. *SRM* była najskuteczniejszą miarą w identyfikowaniu katastrofalnych zdarzeń finansowych. Analiza potwierdziła przydatność zaawansowanych miar ryzyka w identyfikowaniu systemowego zagrożenia i zjawiska zarażania (*contagion*) między rynkami. Modele MGARCH oraz VAR-GARCH-BEKK pozwalają analizować transmisję szoków i zmienności (Fiszeder i Bruzda, 2001; Bukowski i Gowers, 2016). Modele z rodziny MGARCH (*Multivariate GARCH*) umożliwiają jednoczesne modelowanie zmienności wielu szeregów czasowych wraz z kowariancjami, co jest kluczowe w badaniach nad współzależnościami rynkowymi. Jednym z popularnych przedstawicieli tej klasy jest model VAR-GARCH-BEKK, który łączy wektorową autoregresję (VAR)

z modelem GARCH w specyfikacji BEKK (Baba i in., 1990). Model ten pozwala uchwycić dynamiczną transmisję zmienności oraz zależności między rynkami w czasie. Transmisja ta ma istotne znaczenie dla strategii inwestycyjnych, gdyż wpływa na skuteczność prognoz w analizie portfelowej (Brzeszczyński, 2018).

W kontekście ryzyka systematycznego istotne są także wyniki opracowań Wdowiński (2004) oraz Wdowiński i Wrzesiński (2004), w których przeanalizowano determinanty rynkowego ryzyka *beta* w Polsce w latach 1996–2002, traktując je jako parametr zmienny w czasie, estymowany w regresji indeksów WIG i WIG20 względem głównych zagranicznych indeksów giełdowych. Badanie obejmowało modele czynników monetarnych i realnych, w których zmienne objaśniające – takie jak dochód, wydajność pracy, bilans handlowy, deficyt budżetowy, stopa procentowa i kurs złotego – wyrażono względem gospodarki USA. Uzyskane wyniki wskazują, że czynniki monetarne, zwłaszcza kurs walutowy i stopa procentowa, w większym stopniu wyjaśniały zmienność *beta* ryzyka niż czynniki realne. W uzupełnieniu, Wdowiński i Zglińska-Pietrzak (2005), wykorzystując model GARCH oraz procedurę oceny jakości predykcyjnej Fair i Shiller (1990), przeanalizowali wpływ indeksów NYSE (DJIA i NASDAQ) oraz indeksów europejskich (DAX i FTSE) na indeks WIG. Wyniki wskazały, że rynki amerykańskie miały relatywnie większą siłę wyjaśniania zmian indeksu WIG niż rynki europejskie. Z kolei Wdowiński i Małecka (2010), testując obecność efektów asymetrycznych w indeksach giełdowych rynków rozwiniętych i Europy Środkowej, zastosowali podejście Engle i Ng (1993) w ramach modeli ARCH/GARCH. Analiza potwierdziła, że reakcja zmienności na negatywne informacje rynkowe była silniejsza niż na informacje pozytywne, co uzasadnia stosowanie asymetrycznych modyfikacji modeli GARCH.

Równolegle do powyższych podejść, istotną rolę odgrywają czynniki makroekonomiczne w modelowaniu indeksów giełdowych. W badaniach opartych na modelu VECM potwierdzono istnienie długookresowych zależności między indeksem WIG a zmiennymi takimi jak PKB, inflacja, podaż pieniądza czy indeks CRB (Fiszeder i Rowiński, 2012). VECM (*Vector Error Correction Model*) jest rozszerzeniem modelu VAR dla zmiennych skointegrowanych, umożliwiając jednoczesne modelowanie krótkookresowej dynamiki i długookresowej równowagi. W artykule Osowska i Wójcik (2024) przeanalizowano wpływ treści komunikatów FOMC na reakcje rynków finansowych, wykorzystując dane z lat 2001–2023 oraz model *FinBERT* do oceny sentymentu (Araci, 2019). Prognozowano reakcje indeksu S&P 500 oraz kursu EUR/USD, stosując zarówno regresję liniową, jak i nieliniowe algorytmy uczenia maszynowego, takie jak *SVR* (Drucker i in., 1997), *Random Forest* (Breiman, 2001) i *XGBoost* (Chen i Guestrin, 2016). Wykazano, że sentyment miał istotną moc predykcyjną, szczególnie w przypadku niespodziewanych decyzji Fed, a modele nieliniowe lepiej oddawały złożoność reakcji rynków. Pandemia COVID-19 osłabiła trafność prognoz z powodu zwiększonej niepewności. Analizy prognozowalności stóp zwrotu wskazują, że wskaźniki wyceny fundamentalnej – np. *forward* P/E czy *price-to-book* – mają dużą wartość prognostyczną w długim okresie (Gajdka i Pietraszewski, 2019), a oczekiwania inwestorów poprawiają trafność prognoz (Osińska, 1999). Warto podkreślić, że uwzględnienie oczekiwań inwestorów jako zmiennych wyprzedzających poprawia

trafność modeli prognostycznych, jednocześnie podważając hipotezę rynku efektywnego (Osińska, 1999). Majerowska i Spigarska (2021), wykorzystując modele panelowe zgodne z MSSF (International Accounting Standards Board, 2023), wykazały istotny wpływ wskaźnika ROA na rentowność akcji spółek sektora finansowego w latach 2000–2018. Gostkowska-Drzewicka i Majerowska (2018) potwierdziły istnienie efektu sektorowego, a Gostkowska-Drzewicka i Majerowska (2017) wykazały, że współczynnik *beta* (Sharpe, 1964; Lintner, 1965) sektora bankowego wzrastał w dekoniunkturze, a przewaga banków pod względem niższego ryzyka występowała jedynie przed kryzysem finansowym. Podkreślono również znaczenie cyklu gospodarczego w ocenie stabilności sektora finansowego. W opracowaniu Gostkowska-Drzewicka i Majerowska (2022), stosując dynamiczne modele panelowe GMM, potwierdzono efekt dostosowania zadłużenia do poziomu docelowego oraz istotny wpływ rentowności, wielkości, płynności i możliwości inwestycyjnych.

Ważnym obszarem badań jest analiza mikrostruktury rynku i związanych z nią anomalii. *Price clustering* – preferowanie cen kończących się na 0, 5 lub 00 – wskazuje na nielosowy rozkład cen (Miłobędzki, 2020). Zmienność wolumenu transakcji często wyprzedza zmiany cen, co sugeruje istnienie informacji zawartych w obrocie (Miłobędzki, 2009). Dane o wysokiej częstotliwości ujawniają efekty śródzienne i sezonowe, np. efekt dnia tygodnia, sprzeczne z hipotezą rynku efektywnego (Strawiński i Ślepaczuk, 2008). Analizy sektorowe, np. branży tekstylno-odzieżowej, wskazują, że niektóre sektory mogą oferować lepszy profil ryzyka i zysku niż szeroki rynek (Gajdka i Schabek, 2018). Osińska i in. (2011), stosując modelowanie równań strukturalnych (SEM) (Hair i in., 2010), wykazali, że cechy psychologiczne inwestorów, zwłaszcza optymizm i skłonność do ryzyka, miały większe znaczenie niż czynniki rynkowe.

Wnioski płynące z powyższych badań wskazują, że prawidłowe modelowanie indeksów giełdowych wymaga znajomości klasycznych narzędzi ekonometrycznych oraz integracji podejść nieliniowych, bayesowskich, mikrostrukturalnych i fundamentalnych. Rynek kapitałowy w Polsce, jako przykład rynku wschodzącego, charakteryzuje się dużą złożonością i dynamizmem, co czyni go wartościowym obiektem badań.

Przedmiotem rozważań będzie model z jedną zmienną objaśniającą w postaci:

$$y_t = \alpha_0 + \alpha_1 X_t + \epsilon_t \tag{3.86}$$

gdzie:

y_t – indeks WIG-Banki (tempo wzrostu w proc.),
X_t – indeks WIG (tempo wzrostu w proc.),
α_0 i α_1 – parametry modelu,
ϵ_t – składnik losowy, $\epsilon_t \sim N(0, \sigma_\epsilon^2)$.

Model (3.86) nazywa się jednowskaźnikowym modelem Sharpe'a (Sharpe, 1963). Przed przystąpieniem do wykonania obliczeń należy wczytać odpowiednie biblioteki.

```
library(DescTools)
library(dplyr)
library(ggplot2)
library(ggthemes)
library(gridExtra)
library(kableExtra)
library(knitr)
library(lmtest)
library(magrittr)
library(readxl)
library(reshape2)
library(stargazer)
library(xts)
```

Punktem wyjścia dla zbioru danych statystycznych będzie obiekt `gpw.1.xts` wykorzystany w podrozdziale 3.2.3 za pomocą biblioteki `xts`. Na podstawie tego zbioru utworzono obiekt `y.reg` (oraz jego kopię `y.reg.2`), który zawiera dwie zmienne modelu – `wig_banki` i `wig` w postaci tempa wzrostu.

```
y.reg <- y.reg.2 <- gpw.1.xts[, c("wig_banki", "wig")] %>%
log() %>%
diff(
    .,
    differences = 1
) %>%
na.omit() %>%
{. * 100} %>%
round(., digits = 2)
```

Początkowe wiersze zbioru `y.reg` są następujące:

```
head(y.reg)
```

```
##            wig_banki   wig
## 2005-02-28      7.93  8.48
## 2005-03-31     -2.04 -3.69
## 2005-04-30     -7.42 -5.48
## 2005-05-31      6.63  3.54
## 2005-06-30      6.36  5.77
## 2005-07-31      3.74  7.20
```

Na początku warto pokazać indywidualne wykresy obu zmiennych (rys. 3.9) oraz wykres rozrzutu (*scatter plot*) (rys. 3.10). W tym celu trzeba najpierw utworzyć odpowiednią ramkę danych (`y.reg.df`).

```
y.reg.df <- data.frame(y.reg) %>%
cbind(date = as.Date(rownames(.)), .)
head(y.reg.df)
```

```
##                  date wig_banki   wig
## 2005-02-28 2005-02-28      7.93  8.48
## 2005-03-31 2005-03-31     -2.04 -3.69
## 2005-04-30 2005-04-30     -7.42 -5.48
## 2005-05-31 2005-05-31      6.63  3.54
## 2005-06-30 2005-06-30      6.36  5.77
## 2005-07-31 2005-07-31      3.74  7.20
```

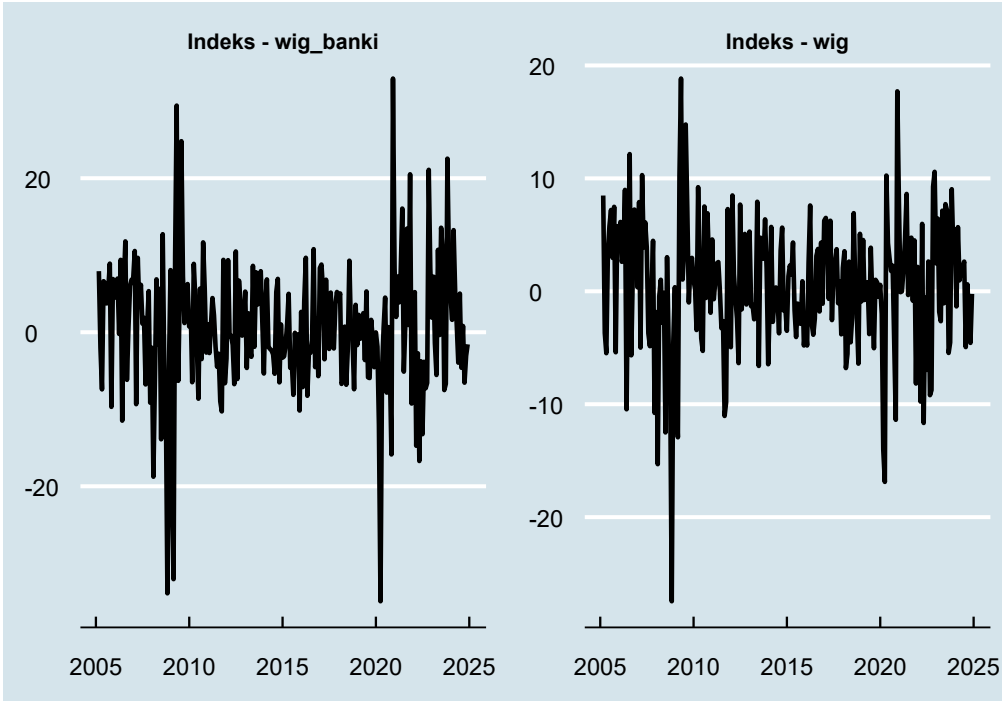
W pierwszej kolumnie obiektu `y.reg.df` znajdują się daty obserwacji, które zostaną umieszczone na osi X. Wykresy poszczególnych zmiennych zostaną zapisane do listy za pomocą wcześniej omówionej funkcji `lapply`.

```
y.reg.plot <- lapply(
    c("wig_banki", "wig"),
    function(z) {
        z <- as.name(z)
        ggplot(
            y.reg.df, aes(x = date)
        ) +
        geom_line(
            aes_string(y = z),
            size = 1
        ) +
        theme_economist(
            base_size = 10,
            base_family = "sans",
            horizontal = TRUE,
            dkpanel = FALSE
        ) +
        theme(
            plot.title = element_text(
                hjust = 0.5,
                size = 10
            ),
            axis.title.x = element_blank(),
            axis.title.y = element_blank()
        ) +
        labs(
            title = paste0("Indeks - ", z)
        )
    }
)
```

W przypadku powyższego kodu warto zauważyć, że w warstwie wykresu `geom_line` zawarto oznaczenie wartości osi Y w postaci polecenia `aes_string` a nie `aes` jak w poprzednich przykładach. Jest to spowodowane tym, że nazwa zmiennej w postaci znakowej (*string*) jest przekazywana do funkcji `lapply` jako argument `z`. Nie jest to zatem zmienna `z`, która ma być umieszczona na wykresie, lecz *uchwyt* do nazwy zmiennej. Jeśli zatem tworzy się wiele wykresów – np. w pętli – to nazwy zmiennych do warstwy *aesthetics* trzeba przekazać za pomocą polecenia `aes_string`. Warto również zwrócić uwagę na inny element wykresu – szablon wykresu (`theme`) stylizowany według wykresów magazynu *The Economist* (`theme_economist`).

Ze względu na to, że wykresy zostały umieszczone na liście, to do utworzenia rysunku zbiorczego zastosowano funkcję `marrangeGrob` z jednym wierszem i dwiema kolumnami (rys. 3.9).

```
marrangeGrob(
    y.reg.plot,
    nrow = 1,
    ncol = 2,
    top = NULL
)
```



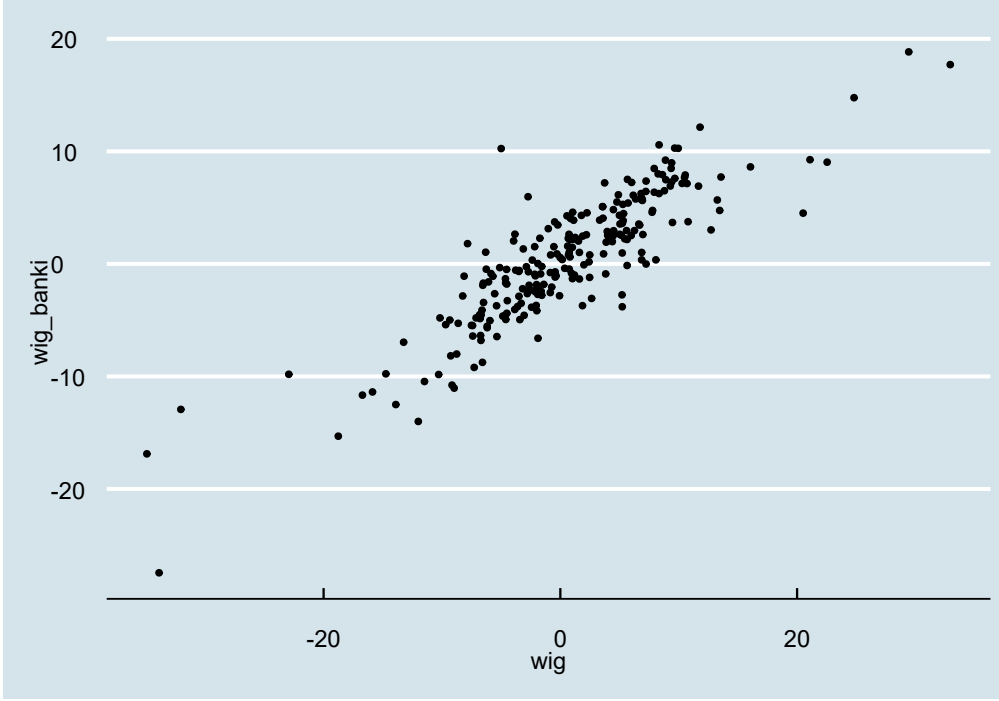
Rysunek 3.9. Rysunek wykresów indeksów WIG-Banki i WIG

Następnie utworzono wykres rozrzutu indeksów giełdowych, na którym zmienna `wig` jest zmienną niezależną (oś X), natomiast `wig_banki` zmienną zależną (oś Y). Zastosowano również szablon *The Economist* (rys. 3.10).

```
scatter <- ggplot(
    y.reg, aes(x = wig_banki, y = wig)
) +
geom_point(
    size = 1
) +
theme_economist(
    base_size = 10,
    base_family = "sans",
    horizontal = TRUE,
    dkpanel = FALSE
) +
labs(
    x = "wig",
    y = "wig_banki"
)
scatter
```

Do rysunku 3.10 można dodać określone elementy. Jednym z nich jest linia regresji (por. równanie 3.86) wraz z przedziałem ufności (rys. 3.11), którą dodaje się za pomocą warstwy `geom_smooth` z oznaczeniem metody `lm` (regresja liniowa).

```
scatter.1 <- ggplot(
    y.reg, aes(x = wig, y = wig_banki)
) +
geom_point(
    size = 1
) +
geom_smooth(
    method = "lm",
    colour = "#808080"
) +
theme_economist(
    base_size = 10,
    base_family = "sans",
    horizontal = TRUE,
    dkpanel = FALSE
) +
labs(
    x = "wig",
    y = "wig_banki"
)
scatter.1
```

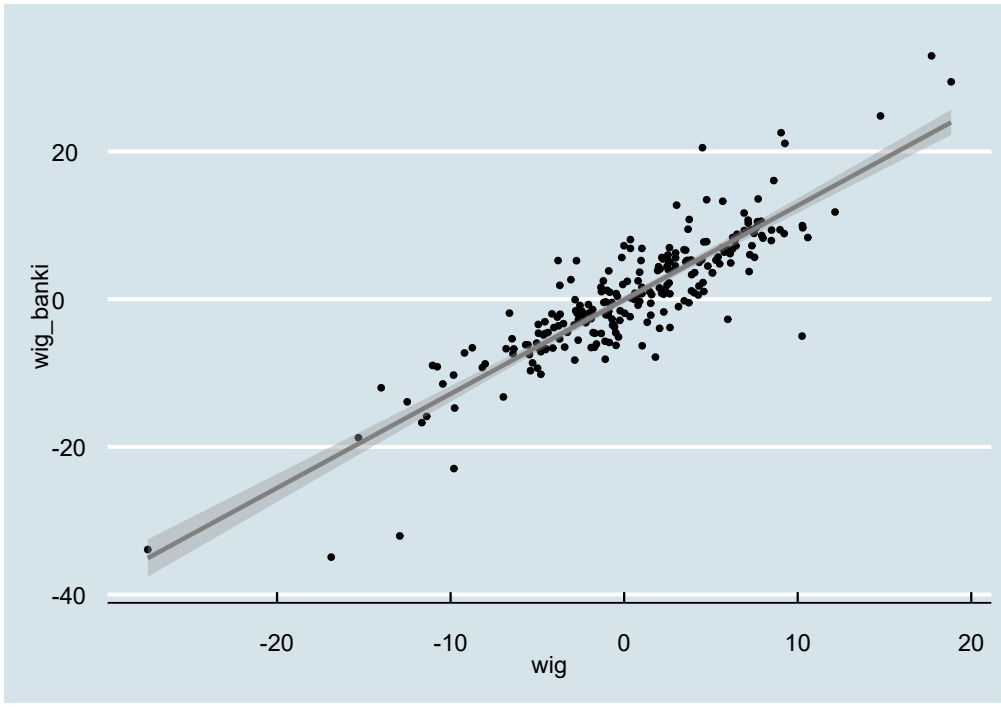


Rysunek 3.10. Wykres rozrzutu dla indeksów giełdowych

```
## `geom_smooth()` using formula = 'y ~ x'
```

Można również utworzyć złożony rysunek, na którym znajdą się wykresy – punktowy i histogram – dla zmiennych brzegowych. W tym celu trzeba utworzyć odrębne wykresy – obiekty `x.hist`, `y.hist` i pomocniczo `empty.plot`.

```
x.hist <- ggplot(
    y.reg, aes(x = wig)
) +
geom_histogram(
    bins = 30
) +
theme(legend.position = "none") +
theme_economist(
    base_size = 10,
    base_family = "sans",
    horizontal = TRUE,
    dkpanel = FALSE
)
```



Rysunek 3.11. Wykres rozrzutu dla indeksów giełdowych z linią regresji

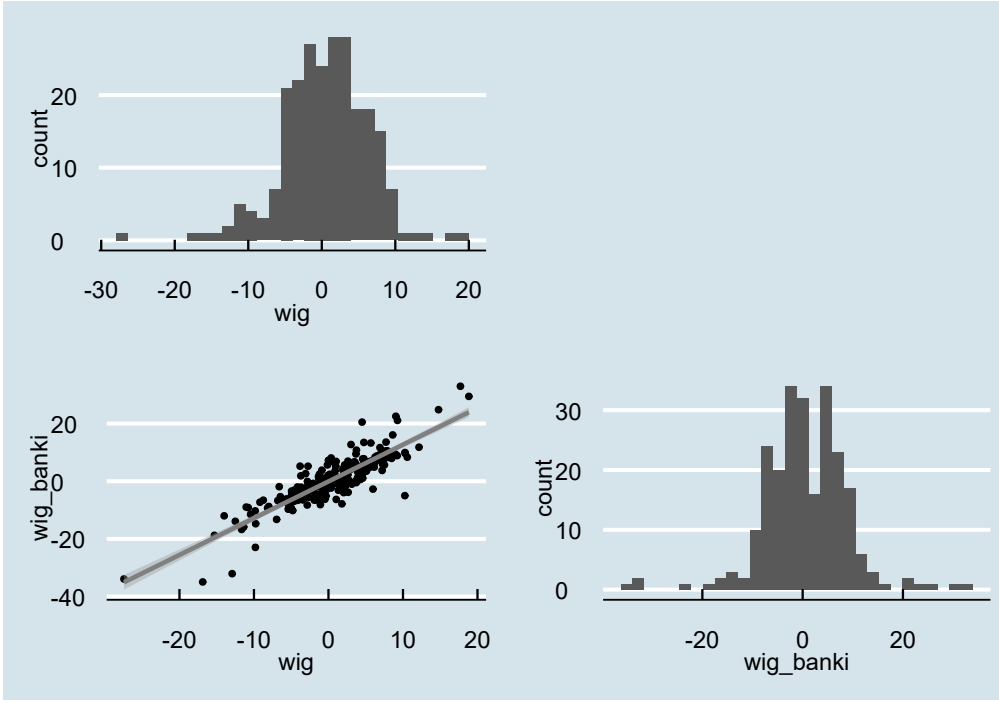
```
y.hist <- ggplot(
    y.reg, aes(x = wig_banki)
) +
geom_histogram(
    bins = 30
) +
theme(legend.position = "none") +
theme_economist(
    base_size = 10,
    base_family = "sans",
    horizontal = TRUE,
    dkpanel = FALSE
)

empty.plot <- ggplot() +
theme(legend.position = "none") +
theme_economist() +
theme(
    axis.line.x = element_blank()
)
```

Następnie wszystkie wykresy należy umieścić na jednym rysunku, w tym rysunek dla obiektu `scatter.1` (rys. 3.12).

```
grid.arrange(
    x.hist,
    empty.plot,
    scatter.1,
    y.hist,
    ncol = 2,
    nrow = 2
)
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



Rysunek 3.12. Wykres rozrzutu dla indeksów giełdowych z linią regresji oraz histogramem

Na podstawie rys. 3.11 można stwierdzić, że w zbiorze wartości tempa wzrostu indeksów `wig` i `wig_banki` występują obserwacje nietypowe (odstające). Ich obecność może mieć znaczący wpływ na wyniki estymacji modelu (3.86) i dlatego usunięto te obserwacje ze zbioru `y.reg`. Nie zawsze jest to jednak uzasadnione, zwłaszcza gdy analiza dotyczy wartości znajdujących się w *ogonach* rozkładu (nietypowych). Takie

obserwacje odstające można zauważyć również na histogramach na rys. 3.12. Do *korekty* rozkładu zastosowano bibliotekę `DescTools` (Signorell, 2024). Zawarta w niej funkcja `Winsorize` umożliwia ograniczenie wpływu obserwacji najbardziej oddalonych od średniej (Hastings i in., 1947). Procedura polega na zastąpieniu obserwacji odstających poniżej/powyżej dolnego/górnego percentyla wartością najmniejszą/największą w pozostałym zbiorze. Dla przejrzystości kodu można obliczyć kwantyle przed wywołaniem funkcji `Winsorize`. Wykonanie procedury przedstawia poniższy kod.

```
y.reg$wig <- Winsorize(
    y.reg$wig,
    quantile(y.reg$wig, probs = c(0.05, 0.95),
    na.rm = FALSE)
)
y.reg$wig_banki <- Winsorize(
    y.reg$wig_banki,
    quantile(y.reg$wig_banki, probs = c(0.05, 0.95),
    na.rm = FALSE)
)
```

Po wykonaniu korekty rozkładu otrzymano obraz graficzny pokazany na rys. 3.13.

```
scatter.2 <- ggplot(
    y.reg, aes(x = wig, y = wig_banki)
) +
geom_point(
    size = 1
) +
geom_smooth(
    method = "lm",
    colour = "#808080"
) +
theme_economist(
    base_size = 10,
    base_family = "sans",
    horizontal = TRUE,
    dkpanel = FALSE
) +
labs(
    x = "wig",
    y = "wig_banki"
)

x.hist <- ggplot(
    y.reg, aes(x = wig)
) +
geom_histogram(
```

```
    bins = 30
) +
theme(legend.position = "none") +
theme_economist(
    base_size = 10,
    base_family = "sans",
    horizontal = TRUE,
    dkpanel = FALSE
)

y.hist <- ggplot(
    y.reg, aes(x = wig_banki)
) +
geom_histogram(
    bins = 30
) +
theme(legend.position = "none") +
theme_economist(
    base_size = 10,
    base_family = "sans",
    horizontal = TRUE,
    dkpanel = FALSE
)

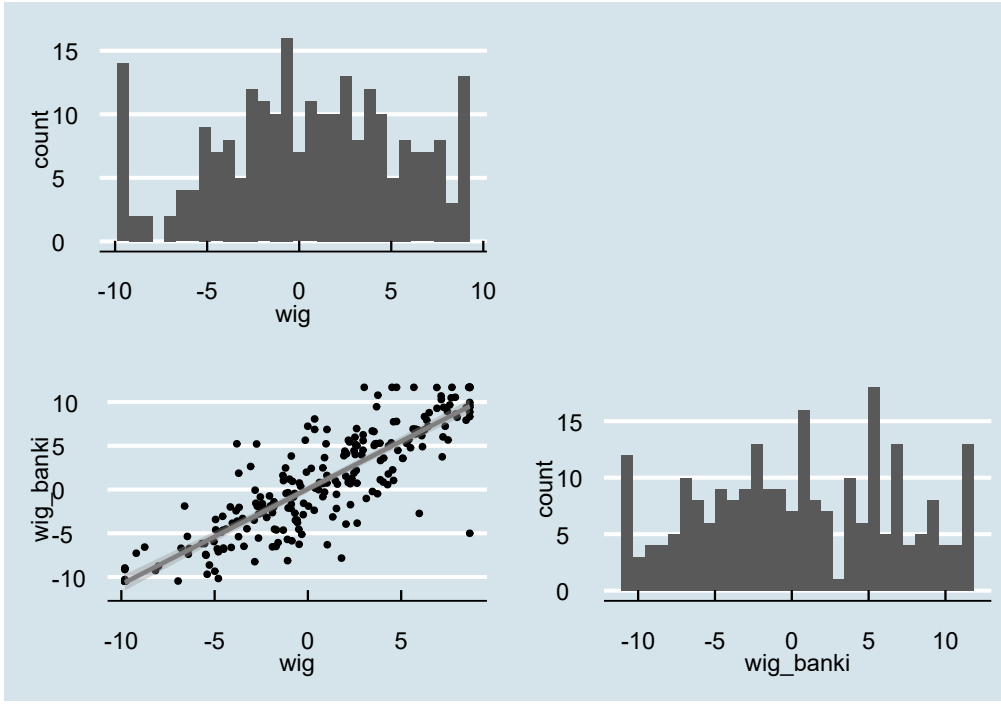
empty.plot <- ggplot() +
theme(legend.position = "none") +
theme_economist() +
theme(
    axis.line.x = element_blank()
)
```

```
grid.arrange(
    x.hist,
    empty.plot,
    scatter.2,
    y.hist,
    ncol = 2,
    nrow = 2
)
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

Na podstawie rys. 3.13 można przyjąć, że na skutek zastosowania korekty rozkładu zmiennych wyraźnie zmniejszyła się rola obserwacji odstających.

Z rys. 3.9 wynika, że największa zmienność indeksów giełdowych wystąpiła w czasie kryzysu finansowego w latach 2008–2010. Dlatego, w celu uzyskania bardziej



Rysunek 3.13. Wykres rozrzutu dla indeksów giełdowych z linią regresji oraz histogramami po korekcie rozkładu

wiarygodnych szacunków modelu Sharpe'a (3.86), dodatkowo, oprócz korekty rozkładu, wyodrębniono podpróbę statystyczną począwszy od roku 2010 do lipca 2019 r. W związku z tym utworzono nowy zbiór danych statystycznych `y.reg.1` oraz zbiór pomocniczy `y.reg.df.1`.

```
y.reg.1 <- y.reg %>%
    .["2010-01-01/2019-07-01"]
y.reg.df.1 <- data.frame(y.reg.1) %>%
    cbind(date = as.Date(rownames(.)), .)
head(y.reg.1)
```

```
##            wig_banki    wig
## 2010-01-31      2.42  0.180
## 2010-02-28     -6.49 -3.430
## 2010-03-31      8.88  8.674
## 2010-04-30      4.39  1.980
## 2010-05-31     -1.99 -4.160
## 2010-06-30     -8.63 -5.280
```

Wartości statystyk opisowych skorygowanych temp wzrostu indeksów `wig` i `wig_banki`, zawartych w zbiorze `y.reg.df.1`, przedstawiono w tabl. 3.17.

```
stargazer(
    y.reg.df.1[, c("wig", "wig_banki")],
    type = "latex",
    title = paste0(
        "Statystyki opisowe tempa wzrostu indeksów giełdowych -- ",
        "\\texttt{stargazer}"
    ),
    label = "qreodp",
    style = "aer",
    align = TRUE,
    header = FALSE,
    flip = TRUE,
    digits = 2,
    digits.extra = 2,
    float = TRUE
)
```

Tablica 3.17. Statystyki opisowe tempa wzrostu indeksów giełdowych – `stargazer`

Statistic	*wig*	*wig_banki*
N	114	114
Mean	0.36	0.26
St. Dev.	4.17	5.26
Min	−9.81	−10.28
Max	8.67	11.67

Można teraz przejść do estymacji modelu (3.86) za pomocą MNK. W tym celu zostanie użyta funkcja `lm` z biblioteki `stats` (R Core Team, 2025).

```
model <- lm(
    wig_banki ~ wig,
    data = y.reg.1
)
```

W powyższym kodzie wykorzystano utworzony wcześniej zbiór danych `y.reg.1`, który zawiera dwie zmienne – `wig` i `wig_banki` – w postaci temp wzrostu – w podpróbie statystycznej. Utworzony obiekt `model` jest listą, która zawiera wartości zmiennych oraz szacunki parametrów i struktury stochastycznej modelu. Złożoną strukturę tego obiektu można poznać za pomocą funkcji `str`.

```
str(model)
```

Najprostszym sposobem na pokazanie wyników estymacji jest zastosowanie funkcji `summary`.

```
summary(model)
```

```
##
## Call:
## lm(formula = wig_banki ~ wig, data = y.reg.1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.8090 -1.8363  0.1396  1.6669  9.5109
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.13486    0.25132  -0.537    0.593
## wig          1.08819    0.06029  18.050   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.673 on 112 degrees of freedom
## Multiple R-squared:  0.7442, Adjusted R-squared:  0.7419
## F-statistic: 325.8 on 1 and 112 DF,  p-value: < 2.2e-16
```

Aby uzyskać lepsze efekty prezentacji wyników można uprzednio utworzyć odpowiednią tablicę na podstawie listy `model` – np. z ocenami parametrów – sformatować ją i pokazać za pomocą funkcji `kable` z biblioteki `knitr`.

```
kable(
    model$coefficients,
    "latex",
    digits = 2,
    booktabs = TRUE,
    row.names = FALSE,
    caption = "Wyniki estymacji modelu"
) %>%
kable_styling(
    full_width = TRUE
)
```

Można również wykorzystać bibliotekę `stargazer` i funkcję o tej samej nazwie.

```
stargazer(
    model,
    type = "latex",
    title = "Wyniki estymacji modelu indeksów giełdowych w formacie
     ↪  AER -- \\texttt{stargazer}",
    label = "ruwoum",
    style = "aer",
    align = TRUE,
    header = FALSE,
    flip = TRUE
)
```

Tablica 3.18. Wyniki estymacji modelu indeksów giełdowych w formacie AER – `stargazer`

	wig_banki
wig	1.088^{***}
	(0.060)
Constant	-0.135
	(0.251)
Observations	114
R^2	0.744
Adjusted R^2	0.742
Residual Std. Error	2.673 (df = 112)
F Statistic	325.811^{***} (df = 1; 112)
Notes:	***Significant at the 1 percent level.
	**Significant at the 5 percent level.
	*Significant at the 10 percent level.

Warto zwrócić uwagę na argument `style`, który decyduje o formatowaniu tablicy zgodnie z wybranym szablonem. Biblioteka `stargazer` wspiera wiele formatów prezentacji wyników stosowanych w renomowanych czasopismach, np. *American Economic Review* (`aer`), *American Journal of Political Science* (`ajps`), *Quarterly Journal of Economics* (`qje`). W zaprezentowanym kodzie wykorzystano szablon *American Economic Review*. Zmiana na format *Quarterly Journal of Economics* jest natychmiastowa.

```
stargazer(
    model,
    type = "latex",
    title = "Wyniki estymacji modelu indeksów giełdowych w formacie
     ↪  QJE -- \\texttt{stargazer}",
```

```
    label = "nnbwmh",
    style = "qje",
    align = TRUE,
    header = FALSE,
    flip = TRUE
)
```

Tablica 3.19. Wyniki estymacji modelu indeksów giełdowych w formacie QJE – `stargazer`

	wig_banki
wig	1.088^{***}
	(0.060)
Constant	-0.135
	(0.251)
N	114
R^2	0.744
Adjusted R^2	0.742
Residual Std. Error	2.673 (df = 112)
F Statistic	325.811^{***} (df = 1; 112)
Notes:	***Significant at the 1 percent level.
	**Significant at the 5 percent level.
	*Significant at the 10 percent level.

W tabl. 3.18 i 3.19 podano wyniki estymacji modelu, który po oszacowaniu parametrów ma postać:

$$\hat{y}_t = -0.1349 + 1.0882 X_t \tag{3.87}$$

Ocena $\widehat{\alpha}_1 = 1.0882$ jest miarą ryzyka sektora bankowego względem ryzyka rynkowego. Model poddano weryfikacji zarówno merytorycznej, jak i statystycznej. W celu weryfikacji hipotez zastosowano różne testy statystyczne z pomocą biblioteki `lmtest` (Hothorn i in., 2022).

Na początku przeprowadzono test normalności rozkładu składnika losowego. Przeprowadzenie takiego testu jest istotne, ponieważ odrzucenie hipotezy o normalności może prowadzić do błędnych wniosków w klasycznych testach istotności oraz utrudniać ocenę własności składnika losowego, takich jak heteroskedastyczność i autokorelacja.

Do testowania normalności wykorzystano dwa testy:

- Shapiro–Wilka (Shapiro i Wilk, 1965; Pearson i in., 1977; Domański i Wagner, 1992; Royston, 1995),
- Jarque'a–Bery (Jarque i Bera, 1980, 1987; Bera i Jarque, 1981).

Przed przystąpieniem do testowania warto sporządzić wykres rozkładu reszt $\hat{\epsilon}_t$ modelu (rys. 3.14).

```
resid <- data.frame(
    resid = model$residuals
)
ggplot(
    resid, aes(x = resid)
) +
geom_histogram(
    bins = 30
) +
theme(legend.position = "none") +
theme_economist(
    base_size = 10,
    base_family = "sans",
    horizontal = TRUE,
    dkpanel = FALSE
)
```

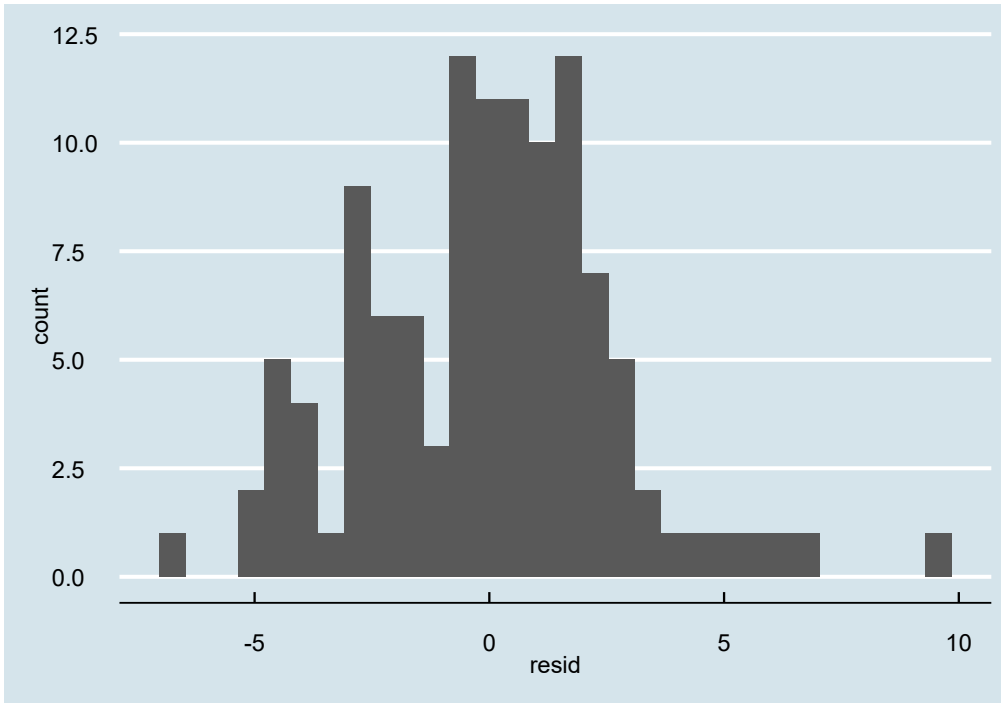
Wartość statystyki testowej (3.47) można obliczyć za pomocą funkcji `shapiro.test` z biblioteki `stats` (zob. Dodatek B).

```
shapiro.test(model$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  model$residuals
## W = 0.98003, p-value = 0.08643
```

Wartość statystyki testowej wynosi 0.98, natomiast *p*-wartość (`p-value`) wynosi 0.09. Nie ma zatem podstaw do odrzucenia hipotezy zerowej na poziomie istotności 0,05, co oznacza, że rozkład składnika losowego nie odbiega istotnie od rozkładu normalnego.

Test Jarque'a–Bery jest uwzględniony w wielu bibliotekach R. Do wykonania testu wykorzystano bibliotekę `tseries`, w której występuje funkcja `jarque.bera.test` (zob. Dodatek B).



Rysunek 3.14. Rozkład reszt modelu indeksów giełdowych

```
library(tseries)
```

Ta funkcja zostanie zastosowana do reszt obiektu `model`.

```
jarque.bera.test(model$residuals)
```

```
##
##  Jarque Bera Test
##
## data:  model$residuals
## X-squared = 5.1282, df = 2, p-value = 0.07699
```

Statystyka w teście Jarque'a–Bery ma wartość 5.13, zaś *p*-wartość (`p-value`) wynosi 0.08. Na podstawie tego testu można stwierdzić, że nie ma podstaw do odrzucenia hipotezy o normalności rozkładu składnika losowego na poziomie istotności 0,05.

Obliczenia zostały powtórzone po zastosowaniu wzorów na skośność, kurtozę i statystykę Jarque'a–Bery.

1. **Skośność i kurtoza**

Należy obliczyć współczynniki skośności i kurtozy.

2. **Statystyka Jarque'a–Bery**

$$\text{LM} = \text{JB} = T \cdot \frac{b_1}{6} + T \cdot \frac{(b_2 - 3)^2}{24} \tag{3.88}$$

gdzie T to liczba obserwacji, b_1 to skośność podniesiona do kwadratu, a b_2 to kurtoza.

3. **Wynik**

Porównanie wyników uzyskanych z funkcji `jarque.bera.test` z ręcznie obliczoną wartością statystyki Jarque'a–Bery i *p*-wartością.

```
data <- model$residuals
T <- length(data)
x.bar <- mean(data)
s2 <- mean((data - x.bar)^2)
s3 <- mean((data - x.bar)^3)
s4 <- mean((data - x.bar)^4)
b1 <- (s3^2) / (s2^3)
b2 <- s4 / (s2^2)
JB <- T * (b1 / 6 + ((b2 - 3)^2) / 24)
JB.p.value <- 1 - pchisq(JB, df = 2)
JB.test <- jarque.bera.test(model$residuals)
cat("Statystyka JB (obliczenia własne): ", JB, "\n")
```

```
## Statystyka JB (obliczenia własne):  5.12825
```

```
cat("P-value (obliczenia własne): ", JB.p.value, "\n")
```

```
## P-value (obliczenia własne):  0.07698652
```

```
cat("Statystyka JB (biblioteka): ", JB.test$statistic, "\n")
```

```
## Statystyka JB (biblioteka):  5.12825
```

```
cat("P-value (biblioteka): ", JB.test$p.value, "\n")
```

```
## P-value (biblioteka):  0.07698652
```

Następnie przeprowadzono test heteroskedastyczności Goldfelda–Quandta składnika losowego na podstawie funkcji `gqtest`. Wyniki testowania przy pomocy biblioteki `lmtest` są następujące:

```
gqtest(
  formula,
  point = 0.5,
  fraction = 0,
  alternative = c("greater", "two.sided", "less"),
  order.by = NULL,
  data = list()
)
```

Do funkcji `gqtest` został przekazany obiekt `model`.

```
gqtest(model)
```

```
##
##  Goldfeld-Quandt test
##
## data:  model
## GQ = 1.8228, df1 = 55, df2 = 55, p-value = 0.01391
## alternative hypothesis: variance increases from segment 1 to 2
```

Wartość statystyki testowej jest równa 1.82, co przy obliczonej *p*-wartości wynoszącej 0.01 oznacza, że należy odrzucić hipotezę H_0 na poziomie istotności 0,05.

Wyniki testu Breuscha–Pagana z uwzględnieniem poprawki Koenker (1981) są następujące:

```
bptest(model)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  model
## BP = 1.0425, df = 1, p-value = 0.3072
```

Wartość statystyki testowej jest równa 1.04, co przy *p*-wartości wynoszącej 0.31 oznacza, że nie ma podstaw do odrzucenia hipotezy H_0. Zatem nie występują podstawy do stwierdzenia, że składnik losowy jest heteroskedastyczny.

Aby przeprowadzić test White'a na heteroskedastyczność, można użyć biblioteki `lmtest` i funkcji `bptest`, która standardowo wykonuje test Breuscha–Pagana, ale można ją dostosować do testu White'a tworząc dodatkową formułę dla argumentu `varformula` w funkcji `bptest`.

```
white.test <- bptest(
    model,
    ~ wig + I(wig^2),
    data = y.reg.1
)
white.test
```

```
##
##  studentized Breusch-Pagan test
##
## data:  model
## BP = 2.2837, df = 2, p-value = 0.3192
```

Wartość statystyki testowej jest równa 2.28, co przy krańcowym poziomie istotności testu `p-value` (*p*-wartość) wynoszącym 0.32 oznacza, że nie ma podstaw do odrzucenia hipotezy H_0. Zatem również test White'a nie wskazuje na heteroskedastyczność składnika losowego w modelu.

Poniżej przedstawiono sposób obliczenia statystyki testowej White'a, stosując wzory na sprawdzian testu. W tym celu zdefiniowano własną funkcję `white.test.manual`.

```
white.test.manual <- function(model) {
  residuals <- residuals(model)
  X <- model.matrix(model)
  Z <- X[, -1]
  Z <- cbind(Z, Z^2)
  for (i in 1:(ncol(Z) - 1)) {
    for (j in (i + 1):ncol(Z)) {
      Z <- cbind(Z, Z[, i] * Z[, j])
    }
  }
  aux.model <- lm(residuals^2 ~ Z)
  n <- nrow(X)
  R2 <- summary(aux.model)$r.squared
  test.stat <- n*R2
  df <- ncol(Z)
  p.value <- pchisq(test.stat, df, lower.tail = FALSE)
```

```
  result <- list(
    statistic = test.stat,
    p.value = p.value
  )
  return(result)
}
```

Wyniki testowania są następujące:

```
result <- white.test.manual(model)
result$statistic
```

```
## [1] 2.356807
```

```
result$p.value
```

```
## [1] 0.5017257
```

Wartość statystyki testowej White'a wynosi: 2.3568, *p*-wartość to: 0.501726. Ponieważ *p*-wartość wynosi 0.501726 i jest większa od typowego poziomu istotności (0,05), brak jest podstaw do odrzucenia hipotezy zerowej o homoskedastyczności. Oznacza to, że model nie wykazuje istotnych oznak heteroskedastyczności składnika losowego według testu White'a.

Następnie przeprowadzono test autokorelacji składnika losowego. Poniżej podano wyniki testu Durbina–Watsona otrzymane przy wykorzystaniu biblioteki `lmtest`. Z logiką testu Durbina–Watsona można zapoznać się pod następującym adresem:

- https://github.com/cran/lmtest/blob/master/R/dwtest.R.

```
dwtest(model)
```

```
##
##  Durbin-Watson test
##
## data:  model
## DW = 2.1672, p-value = 0.817
## alternative hypothesis: true autocorrelation is greater than 0
```

Sprawdzian testu wynosi 2.17, *p*-wartość wynosi 0.82. Wynik testu wskazuje, że nie ma podstaw do odrzucenia hipotezy zerowej. Należy zatem przyjąć, że w modelu

nie występuje statystycznie istotna autokorelacja pierwszego rzędu w składniku losowym.

Następnie przeprowadzono test Boxa–Pierce'a. Implementacja tego testu znajduje się w bibliotece `stats`.

```
Box.test(
    model$residuals,
    type = "Box-Pierce"
)
```

```
##
##  Box-Pierce test
##
## data:  model$residuals
## X-squared = 0.88299, df = 1, p-value = 0.3474
```

W teście Boxa–Pierce'a wartość statystyki testowej jest równa 0.88, co przy *p*-wartości wynoszącej 0.35 oznacza, że nie ma podstaw do odrzucenia hipotezy H_0. Należy przyjąć, że w modelu nie występuje statystycznie istotna autokorelacja pierwszego rzędu w składniku losowym.

Implementację testu Boxa–Pierce'a można znaleźć na następującej stronie internetowej:

- https://github.com/wch/r-source/blob/trunk/src/library/stats/R/ts-tests.R.

```
Box.test <- function(x, lag = 1, type=c("Box-Pierce", "Ljung-Box"),
 ↪  fitdf=0)
{
    if (NCOL(x) > 1)
        stop ("x is not a vector or univariate time series")
    DNAME <- deparse1(substitute(x))
    lag <- as.integer(lag)
    type <- match.arg(type)
    cor <- acf(
      x,
      lag.max = lag,
      plot = FALSE,
      na.action = na.pass
    )
    n <- sum(!is.na(x))
    PARAMETER <- c(df = lag-fitdf)
    obs <- cor$acf[2:(lag+1)]
    if (type=="Box-Pierce")
    {
```

```
        METHOD <- "Box-Pierce test"
        STATISTIC <- n*sum(obs^2)
        PVAL <- 1-pchisq(STATISTIC, lag-fitdf)
    }
    else
    {
        METHOD <- "Box-Ljung test"
        STATISTIC <- n*(n+2)*sum(1/seq.int(n-1, n-lag)*obs^2)
        PVAL <- 1-pchisq(STATISTIC, lag-fitdf)
    }
    names(STATISTIC) <- "X-squared"
    structure(list(
      statistic = STATISTIC,
      parameter = PARAMETER,
      p.value = PVAL,
      method = METHOD,
      data.name = DNAME
    ),
    class = "htest")
}
```

Następnie przeprowadzono test Ljung–Boxa.

```
Box.test(
    model$residuals,
    type = "Ljung-Box"
)
```

```
##
##  Box-Ljung test
##
## data:  model$residuals
## X-squared = 0.90643, df = 1, p-value = 0.3411
```

W teście Ljung–Boxa wartość statystyki testowej jest równa 0.91, co przy krańcowym poziomie istotności `p-value` równym 0.34 oznacza, że nie ma podstaw do odrzucenia hipotezy H_0. Ponownie należy przyjąć, że w modelu nie występuje statystycznie istotna autokorelacja pierwszego rzędu w składniku losowym.

Następnie wykonano testowanie autokorelacji pierwszego rzędu na podstawie testu Breuscha–Godfrey'a przy zastosowaniu biblioteki `lmtest` i poniższego kodu:

- https://github.com/cran/lmtest/blob/master/R/bgtest.R.

```
bgtest <- function(formula, order = 1, order.by = NULL, type =
 ↪  c("Chisq", "F"), data = list(), fill = 0)
{
  dname <- paste(deparse(substitute(formula)))
  if(!inherits(formula, "formula")) {
    X <- if(is.matrix(formula$x))
      formula$x
    else model.matrix(terms(formula), model.frame(formula))
      y <- if(is.vector(formula$y))
      formula$y
    else model.response(model.frame(formula))
  } else {
      mf <- model.frame(formula, data = data)
      y <- model.response(mf)
      X <- model.matrix(formula, data = data)
  }
  if(!is.null(order.by))
  {
    if(inherits(order.by, "formula")) {
      z <- model.matrix(order.by, data = data)
      z <- as.vector(z[,ncol(z)])
    } else {
      z <- order.by
    }
    X <- as.matrix(X[order(z),])
    y <- y[order(z)]
  }
  n <- nrow(X)
  k <- ncol(X)
  order <- 1:order
  m <- length(order)
  resi <- lm.fit(X,y)$residuals
  Z <- sapply(order, function(x) c(rep(fill, length.out = x),
 ↪  resi[1:(n-x)]))
  if(any(na <- !complete.cases(Z))) {
    X <- X[!na, , drop = FALSE]
    Z <- Z[!na, , drop = FALSE]
    y <- y[!na]
    resi <- resi[!na]
    n <- nrow(X)
  }
  auxfit <- lm.fit(cbind(X,Z), resi)
  cf <- auxfit$coefficients
  vc <- chol2inv(auxfit$qr$qr) * sum(auxfit$residuals^2) /
 ↪  auxfit$df.residual
  names(cf) <- colnames(vc) <- rownames(vc) <- c(colnames(X),
   ↪  paste("lag(resid)", order, sep = "_"))
  switch(match.arg(type),
  "Chisq" = {
```

```
    bg <- n * sum(auxfit$fitted.values^2)/sum(resi^2)
    p.val <- pchisq(bg, m, lower.tail = FALSE)
    df <- m
    names(df) <- "df"
  },
  "F" = {
    uresi <- auxfit$residuals
    bg <- ((sum(resi^2) - sum(uresi^2))/m) / (sum(uresi^2) / (n-k-m))
    df <- c(m, n-k-m)
    names(df) <- c("df1", "df2")
    p.val <- pf(bg, df1 = df[1], df2 = df[2], lower.tail = FALSE)
  })
  names(bg) <- "LM test"
  RVAL <- list(
    statistic = bg,
    parameter = df,
    method = paste("Breusch-Godfrey test for serial correlation of
     ↪  order up to", max(order)),
    p.value = p.val,
    data.name = dname,
    coefficients = cf,
    vcov = vc
  )
  class(RVAL) <- c("bgtest", "htest")
  return(RVAL)
}
vcov.bgtest <- function(object, ...) object$vcov
df.residual.bgtest <- function(object, ...) if(length(df <-
 ↪  object$parameter) > 1L) df[2] else NULL
```

Powyższy kod definiuje funkcję `bgtest` z biblioteki `lmtest`, czyli test Breuscha–Godfrey'a na autokorelację składnika losowego w modelu . Szczegółowe wyjaśnienie tego kodu znajduje się poniżej.

1. **Określenie argumentów funkcji**

- `formula`: formuła opisująca model, np. `y ~ x1 + x2`.
- `order`: maksymalny rząd autokorelacji (liczba opóźnień do przetestowania, domyślnie 1).
- `order.by`: zmienna według której mają być uporządkowane dane (opcjonalnie).
- `type`: rodzaj testu, *Chisq* (domyślnie) lub *F*.
- `data`: lista danych do wykorzystania w modelu.
- `fill`: wartość do wypełniania braków przy tworzeniu opóźnień, domyślnie 0.

2. **Pobranie danych i przygotowanie zmiennych**

- dname przechowuje nazwę formuły.
- Jeśli formula nie jest typem formula, zmienne X i y są wyodrębniane bezpośrednio. W przeciwnym razie tworzone są obiekty mf, y i X z użyciem model.frame i model.matrix.

3. **Sortowanie** order.by

- Jeśli order.by jest formułą, tworzy się macierz z, której ostatnia kolumna jest wykorzystywana do sortowania X i y.
- Dane X i y są sortowane według order(z).

4. **Przygotowanie opóźnionych reszt**

- resi przechowuje reszty oszacowane przez lm.fit dla modelu X, y.
- Macierz Z przechowuje opóźnione wartości reszt, np. lag(resid)_1, lag(resid)_2, itd., utworzone dla każdego opóźnienia z obiektu order.

5. **Usuwanie braków w danych**

- Sprawdzane są brakujące wartości (NA), a wiersze z brakami są usuwane z X, y, Z i resi.

6. **Osadzenie regresji pomocniczej**

- Model pomocniczy (regresja reszt względem X i Z) jest tworzony w auxfit.

7. **Obliczanie wartości statystyki testowej i *p*-wartości**

- Zależnie od type wybierany jest test *Chi-kwadrat* lub *F-test*
- *Chi-kwadrat*: liczba obserwacji n jest mnożona przez stosunek sumy kwadratów wartości teoretycznych do sumy kwadratów reszt, *p*-wartość jest obliczana z rozkładu χ^2.
- *F-test*: obliczany jest iloraz wariancji reszt, a następnie *p*-wartość z rozkładu *F*.

8. **Zwracanie wyników**

- Wynik RVAL zawiera:
  - statistic: wartość statystyki testowej,
  - parameter: liczba stopni swobody,
  - method: etykieta testu,
  - p.value: *p*-wartość,

  - `data.name`: nazwa danych,
  - `coefficients` i `vcov`: współczynniki i macierz kowariancji.

9. **Określenie dodatkowych funkcji pomocniczych**

- `vcov.bgtest`: zwraca macierz kowariancji.
- `df.residual.bgtest`: zwraca liczbę stopni swobody reszt dla testu *F*.

Wyniki testowania według powyższego kodu są następujące:

```
bgtest(model)
```

```
##
##  Breusch-Godfrey test for serial correlation of order up to 1
##
## data:  model
## LM test = 0.88698, df = 1, p-value = 0.3463
```

W teście Breuscha–Godfrey'a wartość statystyki testowej jest równa 0.89, co przy krańcowym poziomie istotności `p-value` równym 0.35 oznacza, że również w tym przypadku nie ma podstaw do odrzucenia hipotezy H_0.

Kolejnym testem jest test RESET przy założeniu, że w regresji pomocniczej występuje druga i trzecia potęga $\hat{y}_t$. Test RESET jest zaimplementowany w bibliotece `lmtest`. Por. również następujący kod:

- https://github.com/cran/lmtest/blob/master/R/resettest.R.

```
reset <- resettest <- function(formula, power = 2:3,
  type = c("fitted", "regressor", "princomp"), data = list(),
  vcov = NULL, ...)
{
  dname <- paste(deparse(substitute(formula)))
  if(!inherits(formula, "formula")) {
    mf <- model.frame(formula)
    X <- if(is.matrix(formula$x))
      formula$x
      else model.matrix(terms(formula), mf)
    y <- if(is.vector(formula$y))
      formula$y
      else model.response(mf)
  } else {
    mf <- model.frame(formula, data = data)
    y <- model.response(mf)
    X <- model.matrix(formula, data = data)
```

```
  }
  y <- scale(y, center = attr(terms(mf), "intercept") > 0L)
  k <- ncol(X)
  n <- nrow(X)
  type <- match.arg(type)
  switch(type,
    "fitted" = {
      y.hat <- lm.fit(X,y)$fitted.values
      Z <- matrix(t(sapply(y.hat, "^", power)), nrow=n)
    },
    "regressor" = {
      Z <- as.matrix(mf[,which(!sapply(mf,is.factor))[-1]])
      Z <- matrix(as.vector(t(sapply(as.vector(Z), "^", power))),
↪   nrow=n)
    },
    "princomp" = {
      Z <- as.matrix(mf[,which(!sapply(mf,is.factor))[-1]])
      pc1 <- as.matrix(eigen(cov(Z))$vectors)[,1]
      pc1 <- as.vector(Z %*% pc1)
      Z <- matrix(t(sapply(pc1, "^", power)), nrow=n)
    })
  if(is.null(vcov)) {
    XZ <- cbind(X, Z)
    q <- ncol(Z)
    res1 <- lm.fit(X,y)$residuals
    res2 <- lm.fit(XZ,y)$residuals
    res1 <- sum(res1^2)
    res2 <- sum(res2^2)
    df1 <- q
    df2 <- n-(k+q)
    reset <- (df2/df1) * ((res1 - res2) / res2)
    df <- c(df1, df2)
    pval <- as.vector(pf(reset, df1, df2, lower.tail = FALSE))
  } else {
    reset <- waldtest(lm(y ~ 0 + X + Z), "Z", vcov = vcov, ...)
    df <- abs(as.numeric(reset[2L, 2L:1L]))
    pval <- as.numeric(reset[2L, 4L])
    reset <- as.numeric(reset[2L, 3L])
  }
  names(reset) <- "RESET"
  names(df) <- c("df1", "df2")
  RVAL <- list(
    statistic = reset,
    parameter = df,
    method = "RESET test",
    p.value = pval,
    data.name = dname
  )
  class(RVAL) <- "htest"
```

```
  return(RVAL)
}
```

Powyższy kod definiuje funkcję `resettest`, za pomocą której wykonuje się test RESET. Szczegółowe wyjaśnienie tego kodu znajduje się poniżej.

1. **Określenie argumentów funkcji**

- `formula`: formuła opisująca model, np. `y ~ x1 + x2`.
- `power`: wektor potęg (domyślnie `2:3`), do których podnoszone są wartości teoretyczne lub regresory w celu utworzenia dodatkowych zmiennych.
- `type`: typ zmiennych do przetestowania, możliwe wartości to `"fitted"` (wartości teoretyczne), `"regressor"` (regresory) i `"princomp"` (główna składowa regresorów).
- `data`: lista danych do wykorzystania w modelu.
- `vcov`: funkcja kowariancji używana w alternatywnej wersji testu (opcjonalnie).
- `...`: dodatkowe argumenty dla funkcji `waldtest`, jeśli `vcov` jest podane.

2. **Pobranie danych i przygotowanie zmiennych**

- `dname` przechowuje nazwę formuły.
- Jeśli `formula` nie jest typem `formula`, tworzone są `X` i `y` z `formula` (lub `model.matrix` oraz `model.response`).
- Jeśli `formula` jest typem `formula`, zmienne `X` i `y` są pobierane przy pomocy `model.matrix` i `model.response`.
- `y` jest standaryzowany, jeśli model zawiera wyraz wolny (co jest sprawdzane przy użyciu `attr(terms(mf), "intercept")`).

3. **Ustawianie i wybór typu zmiennych** `type`

- `type`: typ zmiennych, które będą podnoszone do potęg.
- `"fitted"`: tworzenie `Z` przez podnoszenie do potęgi wartości teoretycznych `y.hat` (z `lm.fit(X, y)`).
- `"regressor"`: tworzenie `Z` z macierzy regresorów, z pominięciem zmiennych czynnikowych (factor).
- `"princomp"`: obliczanie pierwszej głównej składowej `pc1` na podstawie regresorów `Z`, a następnie podnoszenie jej do potęg.

4. **Obliczanie wartości statystyki testowej**

- Jeśli vcov jest NULL (podstawowa wersja testu), to:
  - Tworzy się rozszerzony model XZ z X i Z.
  - Oblicza się reszty res1 dla modelu bazowego oraz res2 dla modelu rozszerzonego.
  - Statystyka testowa reset oznacza stosunek reszt obliczony na podstawie zmniejszenia sumy kwadratów reszt.
  - df1 i df2 reprezentują stopnie swobody.
  - pval to *p*-wartość z rozkładu *F* na podstawie reset.
- Jeśli vcov jest podane (alternatywna wersja testu), to:
  - waldtest (test Walda) jest stosowany do oceny istotności Z.
  - df i pval są pobierane z wyniku waldtest.

5. **Zwracanie wyników**

- RVAL to lista zawierająca wyniki testu:
  - statistic: wartość statystyki testowej,
  - parameter: liczba stopni swobody,
  - method: etykieta testu,
  - p.value: *p*-wartość,
  - data.name: nazwa danych.
- Klasa RVAL to "htest", co pozwala na wykorzystanie standardowych funkcji do wyświetlania wyników testów w R.

Wyniki testowania według powyższego kodu są następujące:

```
resettest(model)
```

```
##
##  RESET test
##
## data:  model
## RESET = 1.8037, df1 = 2, df2 = 110, p-value = 0.1695
```

Wartość statystyki testowej wynosi 1.8, natomiast *p*-wartość wynosi 0.17, co oznacza, że nie ma podstaw do odrzucenia hipotezy zerowej o związku liniowym pomiędzy indeksami giełdowymi wig_banki i wig. Należy zauważyć, że hipoteza alternatywna testu RESET nie określa w żaden sposób nieliniowej postaci funkcji.

Następnie podano wyniki testu Harvey'a–Colliera, którego implementacja występuje w bibliotece lmtest. Por. również następujący kod:

- https://github.com/cran/lmtest/blob/master/R/harvtest.R.

```
harvtest <- function(formula, order.by=NULL, data=list())
{
  dname <- paste(deparse(substitute(formula)))

  if(!inherits(formula, "formula")) {
    X <- if(is.matrix(formula$x))
      formula$x
      else model.matrix(terms(formula), model.frame(formula))
    y <- if(is.vector(formula$y))
      formula$y
      else model.response(model.frame(formula))
  } else {
    mf <- model.frame(formula, data = data)
    y <- model.response(mf)
    X <- model.matrix(formula, data = data)
  }
  k <- ncol(X)
  n <- nrow(X)
  rec.res <- function(X, y)
  {
    n <- nrow(X)
    q <- ncol(X)
    w <- rep(0,(n-q))
    Xr1 <- X[1:q,,drop = FALSE]
    xr <- as.vector(X[q+1,])
    X1 <- chol2inv(qr.R(qr(Xr1)))
    fr <- as.vector((1 + (t(xr) %*% X1 %*% xr)))
    betar <- X1 %*%t(Xr1)%*% y[1:q]
    w[1] <- (y[q+1] - t(xr) %*% betar)/sqrt(fr)
    for(r in ((q+2):n))
    {
      X1 <- X1 - (X1 %*% outer(xr, xr) %*% X1)/fr
      betar <- betar + X1 %*% xr * w[r-q-1]*sqrt(fr)
      xr <- as.vector(X[r,])
      fr <- as.vector((1 + (t(xr) %*% X1 %*% xr)))
      w[r-q] <- (y[r] - t(xr) %*% betar)/sqrt(fr)
    }
    return(w)
}
  if(!is.null(order.by))
  {
    if(inherits(order.by, "formula")) {
      z <- model.matrix(order.by, data = data)
      z <- as.vector(z[,ncol(z)])
    } else {
      z <- order.by
    }
```

```
    X <- as.matrix(X[order(z),])
    y <- y[order(z)]
  }
  resr <- rec.res(X,y)
  sigma <- sqrt(var(resr)*(length(resr)-1)/(n-k-1))
  resr <- resr / sigma
  harv <- abs(sum(resr)/sqrt(n-k))/sqrt(var(resr))
  names(harv) <- "HC"
  df <- n-k-1
  names(df) <- "df"
  RVAL <- list(
    statistic = harv,
    parameter = df,
    method = "Harvey-Collier test",
    p.value= 2 * pt(harv, n-k-1,lower.tail=FALSE),
    data.name=dname
  )
  class(RVAL) <- "htest"
  return(RVAL)
}
```

Powyższy kod definiuje funkcję `harvtest`, czyli test Harvey'a–Colliera, służący do oceny poprawności specyfikacji funkcji regresji liniowej. Test ten bada liniowy charakter zależności między zmiennymi. Szczegółowe wyjaśnienie tego kodu znajduje się poniżej.

1. **Określenie argumentów funkcji**

   - `formula`: formuła opisująca model, np. `y ~ x1 + x2`.
   - `order.by`: opcjonalna zmienna, według której mogą być uporządkowane dane.
   - `data`: lista danych do wykorzystania w modelu.

2. **Pobranie danych i przygotowanie zmiennych**

   - `dname` przechowuje nazwę formuły.
   - Jeśli `formula` nie jest obiektem typu `formula`, zmienne `X` i `y` są wyodrębniane bezpośrednio, a w przeciwnym razie tworzone są z `model.frame`.
   - `X` to macierz regresorów, a `y` to wektor zmiennej zależnej.

3. **Zdefiniowanie funkcji** `rec.res`

   - Funkcja `rec.res` oblicza tzw. reszty rekurencyjne, które są wykorzystywane w teście Harvey'a–Colliera.

- Dla pierwszych `q` obserwacji (`Xr1`) funkcja estymuje początkowe wartości `betar` (wektora współczynników regresji) i `fr` (skalar mierzący niepewność).
- Następnie iteracyjnie dodaje się kolejne obserwacje, aktualizując `betar` i `fr`, by uzyskać reszty rekurencyjne `w`.

4. **Sortowanie** `order.by`

- Jeśli `order.by` jest formułą, to tworzona jest zmienna `z` na podstawie tej formuły. W przeciwnym razie `z` to wartość `order.by`.
- Dane `X` i `y` są sortowane według `z`.

5. **Obliczenie wartości statystyki testowej**

- `resr` to reszty rekurencyjne obliczone przez `rec.res`.
- `sigma` to szacunek odchylenia standardowego reszt rekurencyjnych.
- `resr` jest standaryzowane przez `sigma`.
- `harv` to statystyka testowa Harvey'a–Colliera, obliczana jako wartość bezwzględna sumy `resr`, podzielona przez odchylenie standardowe.

6. **Obliczenie *p*-wartości**

- `p.value` jest obliczane jako dwukrotność dystrybuanty rozkładu *t*-Studenta dla `harv`, przy założeniu `df = n-k-1` stopni swobody.

7. **Zwracanie wyników**

- `RVAL` to lista zawierająca:
  - `statistic`: wartość statystyki testowej,
  - `parameter`: liczba stopni swobody,
  - `method`: etykieta testu,
  - `p.value`: *p*-wartość,
  - `data.name`: nazwa danych.
- `RVAL` jest zwracany jako obiekt klasy `"htest"`.

Wyniki testowania według powyższego kodu są następujące:

```
harvtest(model)
```

```
##
##  Harvey-Collier test
##
## data:  model
## HC = 0.1617, df = 111, p-value = 0.8718
```

Można zauważyć, że wartość statystyki testowej wynosi 0.16, co przy krańcowym poziomie istotności `p-value` równym 0.87 oznacza, że nie ma podstaw do odrzucenia hipotezy o liniowości związku.

Ostatecznie został przeprowadzony test istotności ocen parametrów na podstawie testu *t*. W tym celu można odwołać się to tablic 3.18 i 3.19.

```
t.stats <- summary(model)$coefficients[, "t value"]
p.values <- summary(model)$coefficients[, "Pr(>|t|)"]
print(t.stats)
```

```
## (Intercept)         wig
##  -0.5366343  18.0502332
```

```
print(p.values)
```

```
##  (Intercept)          wig
## 5.925846e-01 6.072709e-35
```

Interpretację otrzymanych wielkości można zapisać w postaci kodu podanego poniżej.

```
coef.summary <- summary(model)$coefficients
t.stats <- coef.summary[, "t value"]
p.values <- coef.summary[, "Pr(>|t|)"]
for (i in seq_along(t.stats)) {
  name <- rownames(coef.summary)[i]
  t.val <- round(t.stats[i], 4)
  p.val <- p.values[i]
  if (p.val < 0.05) {
    cat(paste0("Współczynnik dla zmiennej '", name, "' jest
     ↪  statystycznie istotny (t = ", t.val, ", p = ", round(p.val,
     ↪  4), ").\n"))
  } else {
    cat(paste0("Współczynnik dla zmiennej '", name, "' nie jest
     ↪  statystycznie istotny (t = ", t.val, ", p = ", round(p.val,
     ↪  4), ").\n"))
  }
}
```

```
## Współczynnik dla zmiennej '(Intercept)' nie jest statystycznie
 ↪  istotny (t = -0.5366, p = 0.5926).
## Współczynnik dla zmiennej 'wig' jest statystycznie istotny (t =
 ↪  18.0502, p = 0).
```

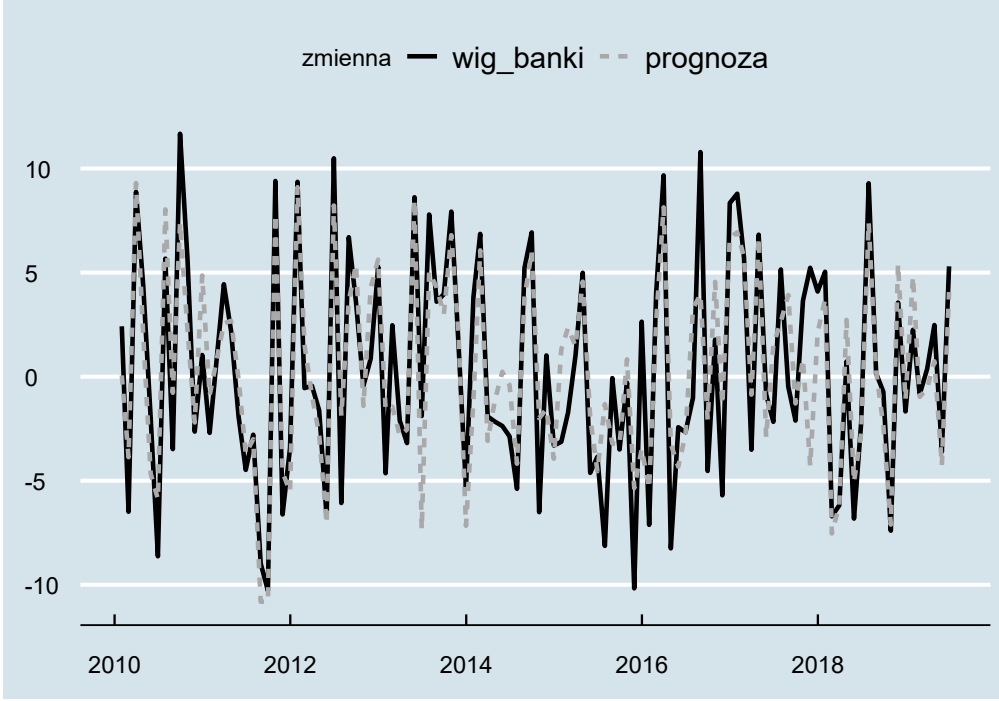
Pokazano również wykres dopasowania danych *teoretycznych* (modelowych) do danych *empirycznych* (rzeczywistych). W tym celu utworzono ramkę danych `fit`.

```
fit <- data.frame(
    y.reg.df.1[,c("date", "wig_banki")],
    prognoza = model$fitted.values
) %>%
melt(
    .,
    id.vars = "date",
    value.name = "value"
) %>%
set_colnames(c(
    "date",
    "zmienna",
    "value"
    )
)
```

Wykres dopasowania pokazano na rys. 3.15.

```
ggplot(
    fit, aes(x = date)
) +
geom_line(aes(
    y = value,
    color = zmienna,
    linetype = zmienna
    ),
    size = 1
) +
theme(legend.position = "none") +
theme_economist(
    base_size = 10,
    base_family = "sans",
    horizontal = TRUE,
    dkpanel = FALSE
) +
theme(
    axis.title.x = element_blank(),
    axis.title.y = element_blank()
) +
scale_color_manual(
    values = c("#000000", "#A9A9A9")
)
```

W podobny sposób jak wyżej można podać interpretację współczynnika determinacji R^2, czyli dopasowania modelu do danych.



Rysunek 3.15. Wykres dopasowania w modelu indeksów giełdowych

```
r2 <- summary(model)$r.squared
adj.r2 <- summary(model)$adj.r.squared
cat(paste0("Wartość R2 wynosi ", round(r2, 3), ", co oznacza, że model
 ↪  wyjaśnia około ", round(r2 * 100, 1), "% zmienności zmiennej
 ↪  'y'.\n"))
```

```
## Wartość R2 wynosi 0.744, co oznacza, że model wyjaśnia około 74.4%
 ↪  zmienności zmiennej 'y'.
```

```
cat(paste0("Skorygowany R2 wynosi ", round(adj.r2, 2), ", co
 ↪  uwzględnia liczbę zmiennych w modelu i wielkość próby.\n"))
```

```
## Skorygowany R2 wynosi 0.74, co uwzględnia liczbę zmiennych w modelu
 ↪  i wielkość próby.
```

```
if (r2 >= 0.9) {
  cat("Model jest bardzo dobrze dopasowany do danych.\n")
} else if (r2 >= 0.7) {
```

```
  cat("Model jest dobrze dopasowany do danych.\n")
} else if (r2 >= 0.5) {
  cat("Model jest przeciętnie dopasowany do danych.\n")
} else {
  cat("Model jest słabo dopasowany do danych - możliwe, że nie
   ↪  uwzględniono istotnych zmiennych objaśniających.\n")
}
```

```
## Model jest dobrze dopasowany do danych.
```

Na koniec przeprowadzono interpretację oceny parametru kierunkowego w oszacowanym modelu Sharpe'a.

```
slope <- coef(model)[2]
print(slope)
```

```
##      wig
## 1.088186
```

Ocena 1.09 oznacza, że wzrost indeksu WIG o 1% wiązał się przeciętnie ze wzrostem indeksu WIG-Banki o 1.09% (zob. również Wdowiński i in. (1997)). Na podstawie przeprowadzonych testów można stwierdzić, że jakość statystyczna równania (3.86) jest wysoka.

3.3.3.2. Model prognostyczny indeksów branżowych GPW

W tym podrozdziale oszacowano jednowskaźnikowe modele *prognostyczne* według Sharpe (1963) dla indeksów branżowych GPW w Warszawie, uwzględniając indeks WIG-Banki opisany w podrozdziale 3.3.3.1. Na początku analizy wczytano niezbędne biblioteki.

```
library(bizdays)
library(broom)
library(cowplot)
library(dplyr)
library(ggplot2)
library(gridExtra)
library(kableExtra)
library(knitr)
library(lmtest)
library(lubridate)
library(magrittr)
library(purrr)
```

```
library(reshape2)
library(tidyr)
library(tidyverse)
library(tseries)
library(writexl)
library(xts)
```

Zdefiniowano również ścieżkę dostępu do plików.

```
path <- "F:\\docs\\R\\b2\\ch3\\"
```

Wektor `index` zawiera nazwy indeksów giełdowych.

```
index <- c(
  "wig",
  "wig_banki",
  "wig_budow",
  "wig_chemia",
  "wig_energ",
  "wig_gornic",
  "wig_gry",
  "wig_info",
  "wig_leki",
  "wig_media",
  "wig_moto",
  "wig_nrchom",
  "wig_odziez",
  "wig_paliwa",
  "wig_spozyw"
)
```

Ścieżka `path` zawiera pliki `.csv`, w których znajdują się dane statystyczne dla poszczególnych indeksów giełdowych. W celu określenia nazw plików utworzono zmienną `files`.

```
files <- path %>%
  paste0(index, ".csv")
files
```

```
##  [1] "F:
docs
R
b2
ch3
wig.csv"
```

```
##  [2] "F:
docs
R
b2
ch3
wig_banki.csv"
##  [3] "F:
docs
R
b2
ch3
wig_budow.csv"
##  [4] "F:
docs
R
b2
ch3
wig_chemia.csv"
##  [5] "F:
docs
R
b2
ch3
wig_energ.csv"
##  [6] "F:
docs
R
b2
ch3
wig_gornic.csv"
##  [7] "F:
docs
R
b2
ch3
wig_gry.csv"
##  [8] "F:
docs
R
b2
ch3
wig_info.csv"
##  [9] "F:
docs
R
b2
ch3
wig_leki.csv"
```

```
## [10] "F:
docs
R
b2
ch3
wig_media.csv"
## [11] "F:
docs
R
b2
ch3
wig_moto.csv"
## [12] "F:
docs
R
b2
ch3
wig_nrchom.csv"
## [13] "F:
docs
R
b2
ch3
wig_odziez.csv"
## [14] "F:
docs
R
b2
ch3
wig_paliwa.csv"
## [15] "F:
docs
R
b2
ch3
wig_spozyw.csv"
```

Jeśli pliki `.csv` nie występują w lokalizacji `path`, to są pobierane ze strony internetowej https://stooq.pl.

```
if (!all(file.exists(files))) {
  address <- lapply(
    index,
    function(x) paste0(
      "https://stooq.pl/q/d/l/?s=",
      x,
      "&i=d"
    )
```

```
  ) %>%
    unlist
  lapply(
    1:length(address),
    function(x) address[x] %>%
      download.file(
        destfile = paste0(path, index[x], ".csv"),
        method = "auto"
      )
    )
}
```

Następnie utworzono zmienną `df.0`, która jest listą zawierającą ramki danych pobranych z plików `.csv`. Każda ramka danych zawiera dwie kolumny: datę (od 1991-04-16 do 2024-11-22) i indeks.

```
df.0 <- lapply(
  1:length(files),
  function(x) {
    df <- path %>%
      paste0(index[x], ".csv") %>%
      read.csv(stringsAsFactors = FALSE) %>%
      select(Data, Zamkniecie) %>%
      set_colnames(c("date", index[x]))
  }
)
```

Następnie zmieniono nazwy elementów listy `df.0`.

```
names(df.0) <- index
names(df.0)[1] <- c("wig_wig")
names(df.0$wig_wig) <- c("date", "wig_wig")
names(df.0)
```

```
##  [1] "wig_wig"    "wig_banki"  "wig_budow"  "wig_chemia"
 ↪  "wig_energ"
##  [6] "wig_gornic" "wig_gry"    "wig_info"   "wig_leki"
 ↪  "wig_media"
## [11] "wig_moto"   "wig_nrchom" "wig_odziez" "wig_paliwa"
 ↪  "wig_spozyw"
```

Kolejnym etapem analizy jest sprawdzenie jakości danych statystycznych (*data quality check*). Sprawdzono następujące własności:

- liczbę obserwacji (wszystkich, brakujących, zerowych i ujemnych),

- duplikaty,
- statystyki opisowe,
- obserwacje nietypowe.

Następnie utworzono funkcję dq, która zwraca tablicę z wynikami analizy powyższych cech.

```
dq <- function(x) {
  metrics <- x %>%
    summarize(across(where(is.numeric), list(
      n_tot = ~length(.x),
      n_mis = ~sum(is.na(.x)),
      n_zer = ~sum(.x == 0, na.rm = TRUE),
      n_neg = ~sum(.x < 0, na.rm = TRUE),
      n_dup = ~sum(duplicated(x$date))
    ),
    .names = "{.col}_{.fn}"))
  stats <- x %>%
    summarize(across(where(is.numeric), list(
      mean   = ~mean(.x, na.rm = TRUE),
      median = ~median(.x, na.rm = TRUE),
      min    = ~min(.x, na.rm = TRUE),
      max    = ~max(.x, na.rm = TRUE),
      q1     = ~quantile(.x, 0.25, na.rm = TRUE),
      q3     = ~quantile(.x, 0.75, na.rm = TRUE)
    ),
    .names = "{.col}_{.fn}")) %>%
    mutate_all(function(x) round(x, 2))
  outliers <- x %>%
    summarize(across(where(is.numeric), ~{
      q1 <- quantile(.x, 0.25, na.rm = TRUE)
      q3 <- quantile(.x, 0.75, na.rm = TRUE)
      iqr <- q3 - q1
      sum(.x < (q1 - 1.5 * iqr) | .x > (q3 + 1.5 * iqr), na.rm = TRUE)
    },
    .names = "{.col}_outliers"))
  stats_table <- bind_cols(
    metrics, stats, outliers
  )
  tab <- stats_table %>%
    pivot_longer(
      cols = everything(),
      names_to = c("statystyka", ".value"),
      names_sep = "_"
    )
  tab$statystyka <- c(
    "Liczba obs.",
    "Liczba brakujących obs.",
```

```
    "Liczba obs. = 0",
    "Liczba obs. < 0",
    "Liczba duplikatów",
    "Średnia",
    "Mediana",
    "Minimum",
    "Maksimum",
    "Q1",
    "Q3",
    "Liczba nietypowych obs."
  )
  return(tab)
}
```

Funkcja dq została zastosowana do listy `df.0`. Najpierw utworzono listę zawierającą tablice z funkcji dq. Każda tablica zawiera dwie kolumny: `statystyka` i nazwę indeksu. Następnie wszystkie tablice zostały połączone w jedną tablicę według kolumny `statystyka`. Do tego celu wykorzystano funkcje `reduce` i `left_join`. Wyniki zapisano do zmiennej `dq.0`.

```
dq.0 <- lapply(
  df.0,
  function(x) dq(x)
) %>%
  reduce(~ left_join(.x, .y, by = "statystyka"))
```

Tablica `dq.0` zawiera zestawienie cech każdego indeksu ze zbioru `index`. Można ją przedstawić w przejrzystej postaci – w formacie *landscape* – za pomocą funkcji `kable` (tabl. 3.20).

```
dq.1 <- dq.0 %>%
  t() %>%
  as.data.frame() %>%
  set_colnames(.[1, ]) %>%
  .[-1, ] %>%
  mutate_all(function(x) as.numeric(x))
```

```
n.col <- dq.1 %>%
  ncol()
dq.1 %>%
  kable(
    booktabs = TRUE,
    format = "latex",
    align = rep("c", n.col),
    caption = "Zestawienie cech indeksów ze zbioru \\texttt{index}",
```

```
    label = "tvwvrb"
  ) %>%
  kable_styling(
    latex_options = c("scale_down"),
    font_size = 9,
    position = "center"
  ) %>%
  landscape()
```

Tablica 3.20. Zestawienie cech indeksów ze zbioru index

	Liczba obs.	Liczba brakujących obs.	Liczba obs. = 0	Liczba obs. < 0	Liczba duplikatów	Średnia	Mediana	Minimum	Maksimum	Q1	Q3	Liczba nietypowych obs.
wig	7958	0	0	0	0	37504.86	41032.71	635.30	89414.00	16163.83	53931.43	0
banki	7958	0	0	0	0	4852.44	5418.83	63.53	13992.51	1870.60	7094.22	0
budow	7958	0	0	0	0	3033.41	2184.69	63.53	12592.70	1470.12	4022.95	502
chemia	7958	0	0	0	0	5983.22	4362.66	63.53	17282.32	1609.71	9887.03	0
energ	3726	0	0	0	0	3041.87	2894.99	999.33	4803.82	2397.69	3754.79	0
gornic	3434	0	0	0	0	4092.81	4097.86	1567.96	6871.19	3533.37	4665.61	51
gry	1976	0	0	0	0	17298.04	15803.85	5175.40	41876.27	12554.41	19998.99	166
info	7958	0	0	0	0	1858.11	1447.27	63.53	5559.42	1169.83	2076.47	1117
leki	1975	0	0	0	0	4706.45	5180.28	2461.49	8984.20	3187.44	5794.92	0
media	7958	0	0	0	0	3368.68	3204.39	63.53	9513.11	1616.38	4599.90	14
moto	1975	0	0	0	0	5599.96	5390.86	2297.59	9568.81	3895.05	6677.82	0
nrchom	7958	0	0	0	0	2135.80	1881.87	63.53	6584.06	1409.74	2614.78	357
odziez	1975	0	0	0	0	6556.80	6107.08	2573.37	11528.69	5352.80	7553.78	39
paliwa	7958	0	0	0	0	3339.42	2829.44	63.53	8678.64	1612.26	4798.09	0
spozyw	7958	0	0	0	0	2473.42	2381.92	63.53	4978.96	1388.40	3503.89	0

Po analizie jakości danych statystycznych można przejść do dalszej analizy. Ponieważ lista `df.0` zawiera indywidualne ramki danych dla indeksów giełdowych, należy je połączyć w jeden zbiór danych. W tym celu ponownie wykorzystano funkcje `reduce` oraz `left_join` i utworzono ramkę `df.1`.

```
df.1 <- df.0 %>%
  reduce(~ left_join(.x, .y, by = "date"))
```

Jeśli występują braki w danych, to można zastosować funkcję `na.omit`. Następnie dla zredukowanego zbioru danych kolumna `date` została przekształcona do formatu daty za pomocą funkcji `as.Date`.

```
df.2 <- df.1 %>%
  na.omit()
df.2$date <- as.Date(df.2$date)
```

```
df.2 %>%
  head() %>%
  kable(booktabs = TRUE) %>%
  kable_styling(
    latex_options = c("hold_position", "scale_down"),
    font_size = 9,
    full_width = TRUE
  ) %>%
  landscape()
```

Dla celów dalszej analizy przekształcono ramkę `df.2` w postaci obiektu `xts` o nazwie `df.3`.

```
df.3 <- xts(df.2[, -1], order.by = df.2[, 1])
```

Następnie przekształcono zbiór danych statystycznych `df.3` z poziomów do temp wzrostu (pierwszych przyrostów logarytmów) `df.4`. W konsekwencji ramka `df.4` zawiera wielkości procentowe. Wartości dodatnie oznaczają wzrost, natomiast ujemne spadek indeksu giełdowego w chwili t względem $t-1$ (dane dzienne).

```
df.4 <- df.3 %>%
  log %>%
  diff(1) %>%
  {.*100} %>%
  round(4) %>%
  na.omit()
```

Liczba obserwacji w ramce danych `df.4` wynosi 1974.

```
obs <- 12*(1/12)*52*5
```

W dalszej analizie przyjęto podpróbę `df.5` zbioru danych `df.4` liczącą 52 tygodnie (dane są 5-dniowe), tj. liczba obserwacji wynosi 260.

```
df.5 <- df.4 %>%
  tail(obs) %>%
  as.data.frame() %>%
  cbind(date = rownames(.), .)
df.5$date <- as.Date(df.5$date)
```

Obiekt `df.5` jest ramką danych, w której jest 16 kolumn. Może się zdarzyć, że ramka `df.5` zawiera kolumny, w których występują wyłącznie brakujące wartości. Wówczas należy usunąć takie kolumny. W tym celu wykorzystano funkcje `sapply` i `is.na`.

```
na.columns <- sapply(df.5, function(x) all(is.na(x)))
na.columns
```

```
##       date    wig_wig  wig_banki  wig_budow wig_chemia  wig_energ
 ↪  wig_gornic
##      FALSE      FALSE      FALSE      FALSE      FALSE      FALSE
 ↪  FALSE
##    wig_gry   wig_info   wig_leki  wig_media   wig_moto wig_nrchom
 ↪  wig_odziez
##      FALSE      FALSE      FALSE      FALSE      FALSE      FALSE
 ↪  FALSE
## wig_paliwa wig_spozyw
##      FALSE      FALSE
```

Następnie utworzono ramkę `df.6`, z której usunięto kolumny wskazane przez wektor `na.columns` o wartościach logicznych.

```
df.6 <- df.5 %>%
  .[, !na.columns]
```

Liczba kolumn ramki `df.6` wynosi 16. Następnie została utworzona lista `df.7`, która najpierw stanowi przekształcenie do postaci stosu za pomocą funkcji `melt` a następnie do postaci listy za pomocą funkcji `split` według kolumny `variable`.

```
df.7 <- df.6 %>%
  melt(
    id.vars = "date",
    variable.name = "variable",
```

```
    value.name = "value"
  ) %>%
  split(.$variable)
```

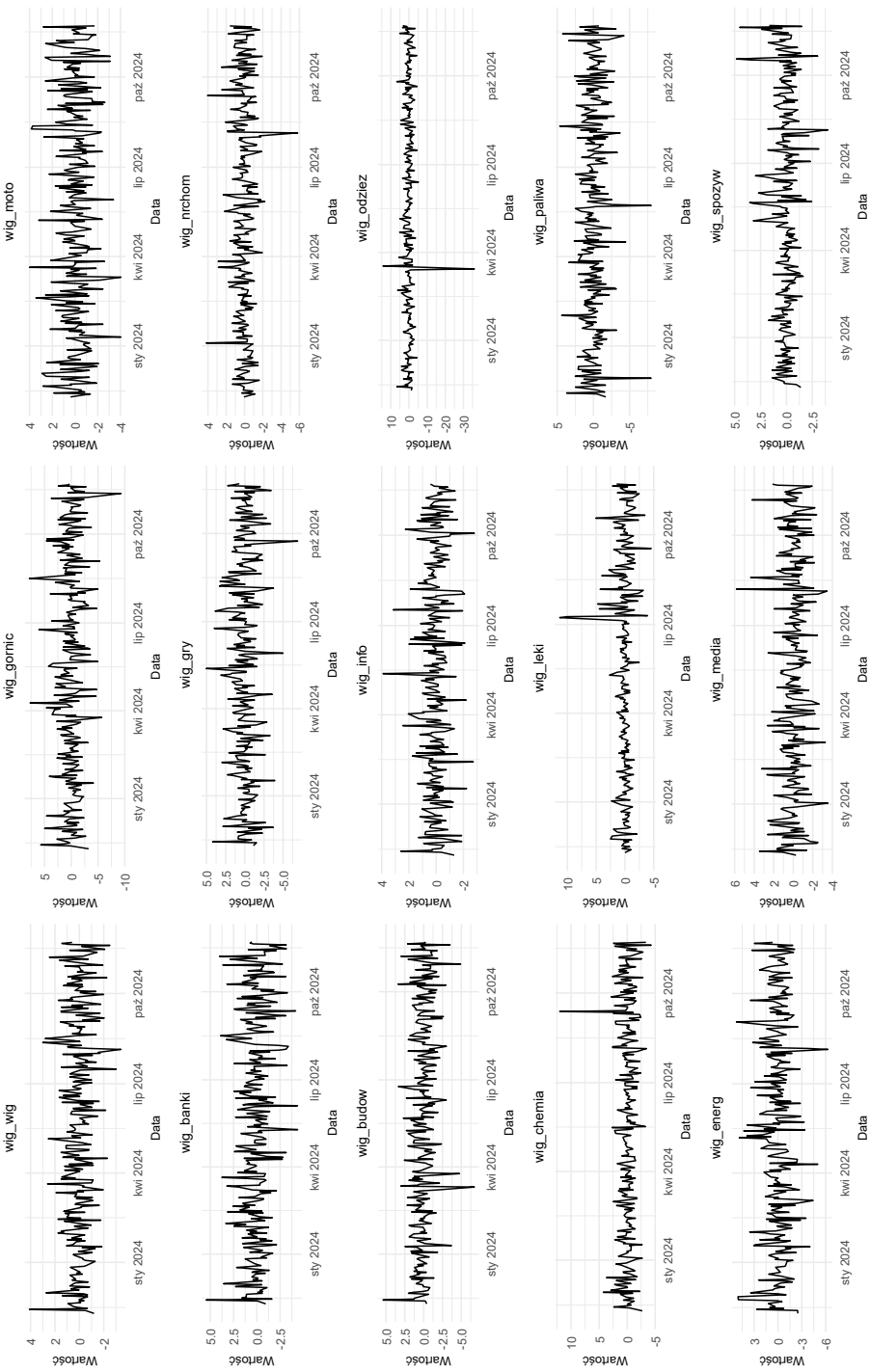
Lista `df.7` zawiera 15 współrzędnych, z których każda obejmuje ramkę liczącą 260 obserwacji i 3 kolumny: `date`, `variable` i `value`.

Lista `df.7` została wykorzystana do utworzenia listy `pl.1` wykresów dla poszczególnych indeksów.

```
pl.1 <- lapply(
  names(df.7),
  function(x) df.7[[x]] %>%
    ggplot(aes(x = date, y = value)) +
    geom_line() +
    labs(
      title = x,
      x = "Data",
      y = "Wartość"
    ) +
    theme_minimal() +
    theme(
      text = element_text(family = "Arial"),
      plot.title = element_text(
        hjust = 0.5,
        size = 10
      ),
      axis.title = element_text(size = 9),
      legend.title = element_text(size = 9)
    )
)
```

Rysunek 3.16, który przedstawia wszystkie wykresy, można uzyskać za pomocą funkcji `marrangeGrob`.

```
marrangeGrob(
  pl.1,
  nrow = length(df.7)/3,
  ncol = length(df.7)/5
)
```



Rysunek 3.16. Rysunek wykresów indeksów branżowych GPW

Następnie przeprowadzono analizę regresji dla jednowskaźnikowego modelu Sharpe'a z uwzględnieniem przesunięcia czasowego indeksu WIG o h okresów:

$$y_t = \alpha_0 + \alpha_1 X_{t-h} + \epsilon_t \tag{3.89}$$

gdzie:

y_t – branżowy indeks giełdowy (w proc.),
X_{t-h} – indeks WIG z przesunięciem o h okresów (w proc.),
ϵ_t – składnik losowy, $\epsilon_t \sim N(0, \sigma_\epsilon^2)$.

Dla celów analizy utworzono kilka zmiennych pomocniczych:

- `var.x`: zmienna objaśniająca,
- `obs`: liczba wszystkich obserwacji (ramka danych `df.6`, zakres danych: od 2023-11-10 do 2024-11-22),
- `s`: liczba obserwacji w okresie empirycznej weryfikacji prognoz,
- `obs.min`: najmniejsza liczba obserwacji, dla których będzie wykonana analiza regresji,
- `var.y`: kolejne zmienne objaśniane,
- h: horyzont prognozy.

```
var.x   <- "wig_wig"
obs     <- nrow(df.6)
s       <- ceiling(.3*obs)
obs.min <- ceiling((.3*obs)+s)
var.y   <- names(df.6)[
  -which(names(df.6) %in% c("date", var.x))
]
h       <- 0:10
```

Następnie zdefiniowano funkcję `tp`, służącą do obliczenia wartości statystyki punktów zwrotnych.

```
tp <- function(z) {
  signs <- sign(diff(z))
  c(FALSE, (diff(signs) != 0), FALSE)
}
```

Funkcja `create.calendar` z biblioteki `bizdays` (Freitas, 2024) pozwala na dostosowanie zbioru danych do struktury pięciodniowej (z wyłączeniem sobót i niedziel).

```
create.calendar(
  "calendar",
  weekdays = c("saturday", "sunday")
)
```

Poniższy kod realizuje wyznaczenie szacunków modelu (3.89), w którym indeksy branżowe `var.y` są funkcją indeksu WIG `var.x`. Obiekt `model` jest listą wielowymiarową indeksowaną wartościami wektorów `h` i `var.y`. Zatem współrzędna wyższego poziomu jest indeksowana wartościami `h` horyzontu prognozy, natomiast współrzędna niższego poziomu wartościami `var.y`, tj. kolejnymi wartościami indeksów branżowych.

```
model <- lapply(
  h,
  function(h) {
    lapply(
      var.y,
      function(y) {
        tryCatch(
          expr = {
            df.0 <- df.6[, c("date", y, var.x)]
            if (h != 0) {
              na.rows <- as.data.frame(
                matrix(NA, nrow = h, ncol = ncol(df.0))
              )
              colnames(na.rows) <- colnames(df.0)
              df.0 <- df.0 %>%
                rbind(na.rows) %>%
                set_rownames(1:nrow(.))
              biz.dates <- bizseq(
                df.0$date[nrow(df.0)-h]+1,
                df.0$date[nrow(df.0)-h]+h*2,
                cal = "calendar"
              )[1:h]
              df.0.date <- df.0$date %>%
                na.omit()
              df.0$date <- c(df.0.date, biz.dates)
              df.0$lagged.x <- lag(df.0[, var.x], h)
              mo.form <- as.formula(paste0(y, " ~ lagged.x"))
              wh.form <- as.formula(" ~ I(lagged.x^2)")
            } else {
              mo.form <- as.formula(
                paste0(y, " ~ ", var.x)
              )
              wh.form <- as.formula(
                paste0(" ~ I(", var.x, "^2)")
              )
            }
            n <- df.0 %>%
              na.omit %>%
              nrow
            if (n >= obs.min) {
              df.1 <- df.0
              df.1[(nrow(df.1)-h-s):nrow(df.1), y] <- NA
```

```
df.1 <- df.1 %>%
  na.omit
model <- lm(mo.form, data = df.1)
summary <- summary(model)
bic <- BIC(model)
jb  <- jarque.bera.test(model$residuals)$p.value
dw  <- dwtest(model)$p.value
bg  <- bgtest(model, order = 1)$p.value
wh  <- bptest(
  model,
  wh.form,
  data = df.1
)$p.value[[1]]
r2  <- 100*(summary$adj.r.squared)
actual <- df.1[, y] %>%
  as.vector
dev <- model$fitted.values %>%
  as.vector() %>%
  cbind(df.1, dev = .) %>%
  select(date, dev)
oos <- predict(model, newdata = df.0) %>%
  cbind(df.0, oos = .) %>%
  select(date, oos) %>%
  na.omit
oot <- predict(model, newdata = df.0) %>%
  cbind(df.0, oot =  .) %>%
  select(date, oot) %>%
  na.omit
df.2 <- list(df.0, dev, oos, oot) %>%
  reduce(
    left_join,
    by = "date"
  )
df.3 <- df.2 %>%
  .[, c("date", y, "dev", "oos", "oot")]
df.4 <- df.3 %>%
  mutate(
    oos = ifelse(!is.na(dev), NA, oos)
  ) %>%
  mutate(
    oot = ifelse(!is.na(.data[[y]]), NA, oot)
  ) %>%
  mutate(
    oos = ifelse(!is.na(oot), NA, oos)
  ) %>%
  mutate_if(
    is.numeric, round, 6
  )
see.dev <- df.4 %>%
```

```
                {.[, y]-.[, "dev"]} %>%
                as.vector %>%
                na.omit %>%
                {sum(.^2)/(length(.)-1)} %>%
                sqrt
              see.oos <- df.4 %>%
                {.[, y]-.[, "oos"]} %>%
                as.vector %>%
                na.omit %>%
                {sum(.^2)/(length(.)-1)} %>%
                sqrt
              df.dev <- df.4 %>%
                select(date, .data[[y]], dev) %>%
                na.omit
              actual.tp.dev <- tp(df.dev[, y])
              predic.tp.dev <- tp(df.dev[, "dev"])
              match.tp.dev <- sum(
                actual.tp.dev == predic.tp.dev,
                na.rm = TRUE
              )
              tp.stat.dev <- 100*(match.tp.dev /
↪ length(actual.tp.dev))
              df.oos <- df.4 %>%
                select(date, .data[[y]], oos) %>%
                na.omit
              actual.tp.oos <- tp(df.oos[, y])
              predic.tp.oos <- tp(df.oos[, "oos"])
              match.tp.oos <- sum(
                actual.tp.oos == predic.tp.oos,
                na.rm = TRUE
              )
              tp.stat.oos <- 100*(match.tp.oos /
↪ length(actual.tp.oos))
              coef <- summary$coefficients
              return(list(
                model = model,
                formula = mo.form,
                var.y = y,
                var.x = var.x,
                a0 = coef[1, 1],
                a1 = coef[2, 1],
                a0.p = coef[1, 4],
                a1.p = coef[2, 4],
                bic = bic,
                jb = jb,
                dw = dw,
                bg = bg,
                wh = wh,
                r2 = r2,
```

```
                see.dev = see.dev,
                see.oos = see.oos,
                tp.dev = tp.stat.dev,
                tp.oos = tp.stat.oos,
                df = df.4,
                h = h
              ))
            } else return(NULL)
          },
          error = function(e) {
            NULL
          },
          warning=function(w) {
            NULL
          },
          message = function(m) {
            NULL
          }
        )
      }
    )
})
```

Struktura powyższego kodu jest następująca:

1. Utworzenie modeli dla różnych wartości `h` (horyzont prognozy) i `var.y` (zmienne zależne). Wykorzystanie funkcji `lapply()` do iteracji, obsługa błędów za pomocą funkcji `tryCatch()`, a następnie obliczenie wartości różnych statystyk diagnostycznych i jakościowych dla dopasowanych modeli. Ustalenie okresu `oos` obejmującego fragmenty danych historycznych i okresu `oot` obejmującego okresy przyszłe, dla których nie są znane rzeczywiste wartości `y`.

2. Zastosowanie mechanizmu iteracyjnego:

   - `lapply(h, ...)` – iteruje po różnych wartościach horyzontu prognozy (`h`);
   - `lapply(var.y, ...)` – iteruje po różnych zmiennych zależnych (`y`).

3. Obsługa błędów:

   - Jeśli wystąpi błąd, ostrzeżenie lub komunikat, funkcja zwróci `NULL`, zapobiegając zatrzymaniu całej pętli.

4. Przygotowanie danych:

- *Wybór zmiennych* – pobiera kolumny `date`, `y` oraz zmienne niezależne `var.x` z danych wejściowych `df.6`;
- Dodaje `h` pustych wierszy (`NA`) na końcu ramki `df.0`, aby uwzględnić przyszłe wartości;
- Tworzy sekwencję przyszłych dat (`bizseq()`) i dodaje je do kolumny `date`;
- Oblicza *opóźnione wartości zmiennej niezależnej* (`lagged.x`), które będą używane jako predyktor w modelu;
- Tworzy *formułę modelu*: `y ~ lagged.x`;
- Tworzy *formułę testu Breuscha–Pagana* do testowania heteroskedastyczności.

5. Sprawdzenie liczby obserwacji:

- Jeśli liczba niepustych obserwacji (`n`) jest *większa lub równa* `obs.min`, szacowane są parametry modelu. W przeciwnym razie zwraca się wartość `NULL`.

6. Utworzenie modelu:

- Usuwa się końcowe obserwacje w zmiennej zależnej (`y`), aby wyznaczyć prognozę;
- Dopasowuje się model (`lm()`) i zapisuje jego podsumowanie;
- Oblicza się kryterium `BIC()`, które służy do porównywania modeli.

7. Zastosowanie testów diagnostycznych:

- *Jarque–Bera* (`jb`) – test normalności składników losowych na podstawie reszt;
- *Durbin–Watson* (`dw`) – test autokorelacji składników losowych na podstawie reszt;
- *Breusch–Godfrey* (`bg`) – test autokorelacji wyższych rzędów składników losowych na podstawie reszt;
- *Breusch–Pagan* (`wh`) – test heteroskedastyczności składników losowych na podstawie reszt.

8. Obliczenie jakości dopasowania:

- Skorygowany współczynnik determinacji R^2, pomnożony przez 100 (wyrażony w %).

9. Wyznaczenie prognoz:

- `dev` – wartości dopasowane do danych historycznych;
- `oos` – prognoza dla `df.0` (*out-of-sample*);
- `oot` – prognoza dla `df.0` (*out-of-time*).

10. Wykonanie transformacji i wyznaczenie miar jakości prognoz:

    - Usuwa `oos` dla okresów, w których występują dopasowane wartości `dev`;
    - Zaokrągla wszystkie wartości numeryczne do 6 miejsc po przecinku.

11. Wyznaczenie błędów prognoz:

    - *SEE* – miara błędu dopasowania modelu dla `dev`;
    - Podobnie dla `oos`.

12. Obliczenie wartości statystyk trafności punktów zwrotnych:

    - Oblicza procent zgodnych punktów zwrotnych dla `oot`;
    - Podobnie dla `oos`.

13. Zwrócenie wyników:

    - Przekazanie wartości listy zawierającej model, statystyki i prognozy.

Następna funkcja usuwa współrzędne `NULL` z listy model i redukuje tę listę do jednego wymiaru.

```
valid.models <- Filter(
  Negate(is.null),
  unlist(model, recursive = FALSE)
)
```

Ramka `output.0` zawiera wszystkie wyniki estymacji, z których można pobierać wartości według przyjętych kryteriów (tabl. 3.21).

```
output.0 <- data.frame(
  y = sapply(valid.models, function(x) x$var.y),
  x = sapply(valid.models, function(x) x$var.x),
  a0 = sapply(valid.models, function(x) x$a0),
  a1 = sapply(valid.models, function(x) x$a1),
  a0.p = sapply(valid.models, function(x) x$a0.p),
  a1.p = sapply(valid.models, function(x) x$a1.p),
  bic = sapply(valid.models, function(x) x$bic),
  jb = sapply(valid.models, function(x) x$jb),
  dw = sapply(valid.models, function(x) x$dw),
  bg = sapply(valid.models, function(x) x$bg),
  wh = sapply(valid.models, function(x) x$wh),
  r2  = sapply(valid.models, function(x) x$r2),
```

```
  see.dev = sapply(valid.models, function(x) x$see.dev),
  see.oos = sapply(valid.models, function(x) x$see.oos),
  tp.dev = sapply(valid.models, function(x) x$tp.dev),
  tp.oos = sapply(valid.models, function(x) x$tp.oos),
  h = sapply(valid.models, function(x) x$h)
) %>%
  mutate(across(-c(x, y, h), function(x) round(x, digits = 2)))
```

```
output.0[1:15, -2] %>%
  kable(
    booktabs = TRUE,
    format = "latex",
    caption = "Wyniki estymacji dla ramki \\texttt{output.0}",
    label = "wahlxm"
  ) %>%
  kable_styling(
    latex_options = c("scale_down"),
    font_size = 8,
    position = "center"
  ) %>%
  column_spec(1, width = "1.5cm") %>%
  landscape()
```

Tablica 3.21. Wyniki estymacji dla ramki `output.0`

y	a0	a1	a0.p	a1.p	bic	jb	dw	bg	wh	r2	see.dev	see.oos	tp.dev	tp.oos	h
wig_budow	0.10	0.71	0.30	0.00	602.99	0.00	0.76	0.46	0.97	24.61	1.23	1.05	61.33	69.62	0
wig_-chemia	-0.11	0.44	0.19	0.00	579.04	0.00	0.95	0.08	0.56	12.04	1.15	1.93	56.91	65.82	0
wig_energ	-0.05	0.90	0.68	0.00	669.15	0.00	0.26	0.54	0.36	26.53	1.48	1.25	69.06	54.43	0
wig_gornic	-0.03	1.17	0.83	0.00	729.12	0.00	0.07	0.17	0.73	30.66	1.74	1.83	58.01	67.09	0
wig_gry	0.05	0.74	0.59	0.00	620.82	0.05	0.58	0.83	0.97	24.33	1.29	1.42	65.19	58.23	0
wig_info	0.04	0.56	0.43	0.00	430.25	0.00	0.97	0.06	0.99	34.36	0.76	0.76	56.91	59.49	0
wig_leki	0.12	0.23	0.26	0.05	672.91	0.00	0.01	0.01	0.56	1.67	1.49	1.63	58.56	56.96	0
wig_media	-0.07	0.58	0.42	0.00	564.88	0.72	0.81	0.37	0.82	20.66	1.11	1.48	62.43	62.03	0
wig_moto	-0.12	0.59	0.19	0.00	598.66	0.18	0.82	0.34	0.93	18.28	1.21	1.35	67.40	53.16	0
wig_-nrchom	0.02	0.34	0.72	0.00	478.57	0.00	0.19	0.37	0.56	12.53	0.87	1.14	61.88	63.29	0
wig_odziez	-0.01	1.64	0.98	0.00	939.08	0.00	1.00	0.00	0.15	21.23	3.11	1.66	70.72	70.89	0
wig_paliwa	-0.07	0.96	0.49	0.00	625.72	0.00	0.33	0.67	0.66	34.49	1.31	1.29	74.59	77.22	0
wig_-spozyw	0.05	0.28	0.52	0.00	501.04	0.00	0.21	0.44	0.72	7.91	0.93	1.30	60.22	54.43	0
wig_banki	0.19	-0.27	0.08	0.01	651.26	0.01	0.42	0.72	0.51	2.96	1.42	1.92	61.67	55.70	1
wig_budow	0.16	-0.01	0.13	0.91	652.42	0.00	0.76	0.38	0.54	-0.55	1.42	1.54	53.89	54.43	1

Następnie można przystąpić do filtrowania ramki `output.0` do ramki `output.1`, np. dla warunku, że `tp.oos >= tp.dev` (tabl. 3.22).

```
output.1 <- output.0 %>%
  filter(
    # filtrowanie wyników
    # e.g.
    # jb > 0.05 &
    (tp.oos >= tp.dev)
  ) %>%
  arrange(h)
```

```
output.1[1:15, -2] %>%
  kable(
    booktabs = TRUE,
    format = "latex",
    caption = "Wyniki estymacji dla ramki \\texttt{output.1}",
    label = "uyjcjp"
  ) %>%
  kable_styling(
    latex_options = c("scale_down"),
    font_size = 8,
    position = "center"
  ) %>%
  column_spec(1, width = "1.5cm") %>%
  landscape()
```

Tablica 3.22. Wyniki estymacji dla ramki `output`. 1

y	a0	a1	a0.p	a1.p	bic	jb	dw	bg	wh	r2	see.dev	see.oos	tp.dev	tp.oos	h
wig_budow	0.10	0.71	0.30	0.00	602.99	0.00	0.76	0.46	0.97	24.61	1.23	1.05	61.33	69.62	0
wig_-chemia	-0.11	0.44	0.19	0.00	579.04	0.00	0.95	0.08	0.56	12.04	1.15	1.93	56.91	65.82	0
wig_gornic	-0.03	1.17	0.83	0.00	729.12	0.00	0.07	0.17	0.73	30.66	1.74	1.83	58.01	67.09	0
wig_info	0.04	0.56	0.43	0.00	430.25	0.00	0.97	0.06	0.99	34.36	0.76	0.76	56.91	59.49	0
wig_-nrchom	0.02	0.34	0.72	0.00	478.57	0.00	0.19	0.37	0.56	12.53	0.87	1.14	61.88	63.29	0
wig_odziez	-0.01	1.64	0.98	0.00	939.08	0.00	1.00	0.00	0.15	21.23	3.11	1.66	70.72	70.89	0
wig_paliwa	-0.07	0.96	0.49	0.00	625.72	0.00	0.33	0.67	0.66	34.49	1.31	1.29	74.59	77.22	0
wig_budow	0.16	-0.01	0.13	0.91	652.42	0.00	0.76	0.38	0.54	-0.55	1.42	1.54	53.89	54.43	1
wig_-chemia	-0.07	0.08	0.46	0.38	595.61	0.01	0.96	0.06	0.14	-0.13	1.22	2.14	56.11	60.76	1
wig_gry	0.13	-0.04	0.26	0.74	668.84	0.00	0.56	0.78	0.76	-0.50	1.49	1.74	55.00	65.82	1
wig_info	0.11	-0.15	0.11	0.04	499.00	0.00	0.95	0.04	0.23	1.90	0.93	0.93	63.89	64.56	1
wig_media	-0.02	0.08	0.80	0.41	604.77	0.53	0.68	0.57	0.41	-0.18	1.25	1.52	56.11	59.49	1
wig_moto	-0.07	-0.06	0.50	0.53	633.22	0.24	0.84	0.25	0.92	-0.33	1.35	1.57	53.89	63.29	1
wig_-nrchom	0.05	0.07	0.50	0.35	501.19	0.00	0.29	0.60	0.82	-0.07	0.93	1.33	57.22	58.23	1
wig_odziez	0.17	-0.33	0.53	0.22	977.23	0.00	0.99	0.00	0.72	0.29	3.51	2.18	53.89	63.29	1

Można również przyjąć filtr `output.2`[4], w którym wybiera się największą wartość współczynnika determinacji R^2 według wartości horyzontu prognozy `h` z ramki `output.1` (tabl. 3.23).

```
output.2 <- output.1 %>%
  group_by(h) %>%
  slice_max(r2, n = 1) %>%
  arrange(desc(r2))
```

```
output.2[, -2] %>%
  kable(
    booktabs = TRUE,
    format = "latex",
    caption = "Wyniki estymacji dla ramki \\texttt{output.2}",
    label = "nazbzg"
  ) %>%
  kable_styling(
    latex_options = c("scale_down"),
    font_size = 8,
    position = "center"
  ) %>%
  column_spec(1, width = "1.5cm") %>%
  landscape()
```

[4] Warto zwrócić uwagę, że do graficznego przedstawienia obiektów `output.0`, `output.1` i `output.2` w postaci obróconej – *landscape* – została wykorzystana funkcja `landscape` jako uzupełnienie funkcji `kable`.

Tablica 3.23. Wyniki estymacji dla ramki `output`. 2

y	a0	a1	a0.p	a1.p	bic	jb	dw	bg	wh	r2	see.dev	see.oos	tp.dev	tp.oos	h
wig_paliwa	-0.07	0.96	0.49	0.00	625.72	0.00	0.33	0.67	0.66	34.49	1.31	1.29	74.59	77.22	0
wig_gornic	0.02	0.43	0.88	0.01	768.65	0.01	0.19	0.48	0.83	3.79	2.01	2.38	56.74	58.23	3
wig_leki	0.18	-0.26	0.12	0.02	657.18	0.00	0.01	0.02	0.61	2.38	1.50	1.72	56.25	59.49	5
wig_info	0.11	-0.15	0.11	0.04	499.00	0.00	0.95	0.04	0.23	1.90	0.93	0.93	63.89	64.56	1
wig_media	-0.04	-0.17	0.67	0.06	576.69	0.64	0.51	0.99	0.28	1.49	1.21	1.55	59.43	62.03	6
wig_gry	0.13	-0.20	0.26	0.07	640.06	0.00	0.57	0.81	0.95	1.28	1.46	1.72	59.20	62.03	7
wig_budow	0.09	0.18	0.42	0.09	608.08	0.00	0.78	0.39	0.25	1.14	1.37	1.52	57.31	62.03	10
wig_odziez	0.08	0.37	0.77	0.17	941.67	0.00	1.00	0.00	0.88	0.52	3.53	2.21	58.38	64.56	8
wig_info	0.09	-0.09	0.19	0.20	478.85	0.00	1.00	0.00	0.61	0.39	0.93	0.91	56.40	64.56	9
wig_gornic	0.10	-0.17	0.51	0.28	786.08	0.00	0.10	0.35	0.64	0.09	2.09	2.42	58.10	63.29	2
wig_gry	0.12	-0.09	0.29	0.40	651.42	0.00	0.47	0.99	0.24	-0.17	1.46	1.72	53.11	59.49	4

Łatwo zauważyć, że najwyższa wartość współczynnika R^2 występuje dla indeksu wig_paliwa i $h = 0$. W kolejnym kroku procedury przyjęto, że wyniki dla modeli *optymalnych* pod względem prognostycznym znajdują się w ramce `output.2`. W dalszej analizie przyjęto pomocniczy obiekt `id`.

```
id <- output.2
```

Ponieważ ramka `id` zawiera wyniki estymacji dla najlepszych modeli (z punktu widzenia przyjętych wcześniej kryteriów), należy ustalić numery regresji w liście `valid.models`. Tę procedurę realizuje poniższy kod.

```
final.model.0 <- lapply(
  1:nrow(id),
  function(x) {
    which(
      sapply(
        valid.models,
        function(y) {
          round(y$see.oos, 2) == id[x, "see.oos"] &
          round(y$tp.dev, 2) == id[x, "tp.dev"] &
          round(y$tp.oos, 2) == id[x, "tp.oos"] &
          y$h == id[x, "h"]
        }
      )
    )
  }
) %>%
  unlist
final.model.0
```

```
##  [1]  12  46  77  20  92 103 141 123 132  32  61
```

Dla zbioru *najlepszych* modeli można utworzyć wykresy dopasowania prognoz `oos` i `oot` do próby statystycznej. To zadanie realizuje poniższy kod, który tworzy listę wykresów `final.model.1`.

```
final.model.1 <- lapply(
  final.model.0,
  function(x) {
    var.y <- valid.models[[x]]$var.y
    valid.models[[x]]$df %>%
      melt(
        id.vars = "date",
        variable.name = "variable",
        value.name = "value"
```

```
  ) %>%
  mutate(
    variable = factor(
      variable,
      levels = c(var.y, "dev", "oos", "oot")
    )
  ) %>%
  ggplot(
    aes(
      x = date,
      y = value,
      color = variable,
      linetype = variable
    )
  ) +
  geom_line() +
  theme_minimal() +
  labs(
    title = paste("Najlepszy model (", var.y, ")", sep = ""),
    y = var.y,
    x = "Czas"
  ) +
  scale_color_manual(
    values = c(
      setNames("black", var.y),
      "dev"  = "grey40",
      "oos"  = "grey60",
      "oot"  = "grey80"
    )
  ) +
  scale_linetype_manual(
    values = c(
      setNames("solid", var.y),
      "dev"  = "solid",
      "oos"  = "dashed",
      "oot"  = "dotted"
    )
  ) +
  scale_x_date(
    date_breaks = "3 months",
    date_labels = "%Y-%m-%d"
  ) +
  theme(
    legend.position = "top",
    axis.text.x = element_text(
      angle = 90,
      hjust = 1
    ),
    plot.title = element_text(hjust = 0.5)
```

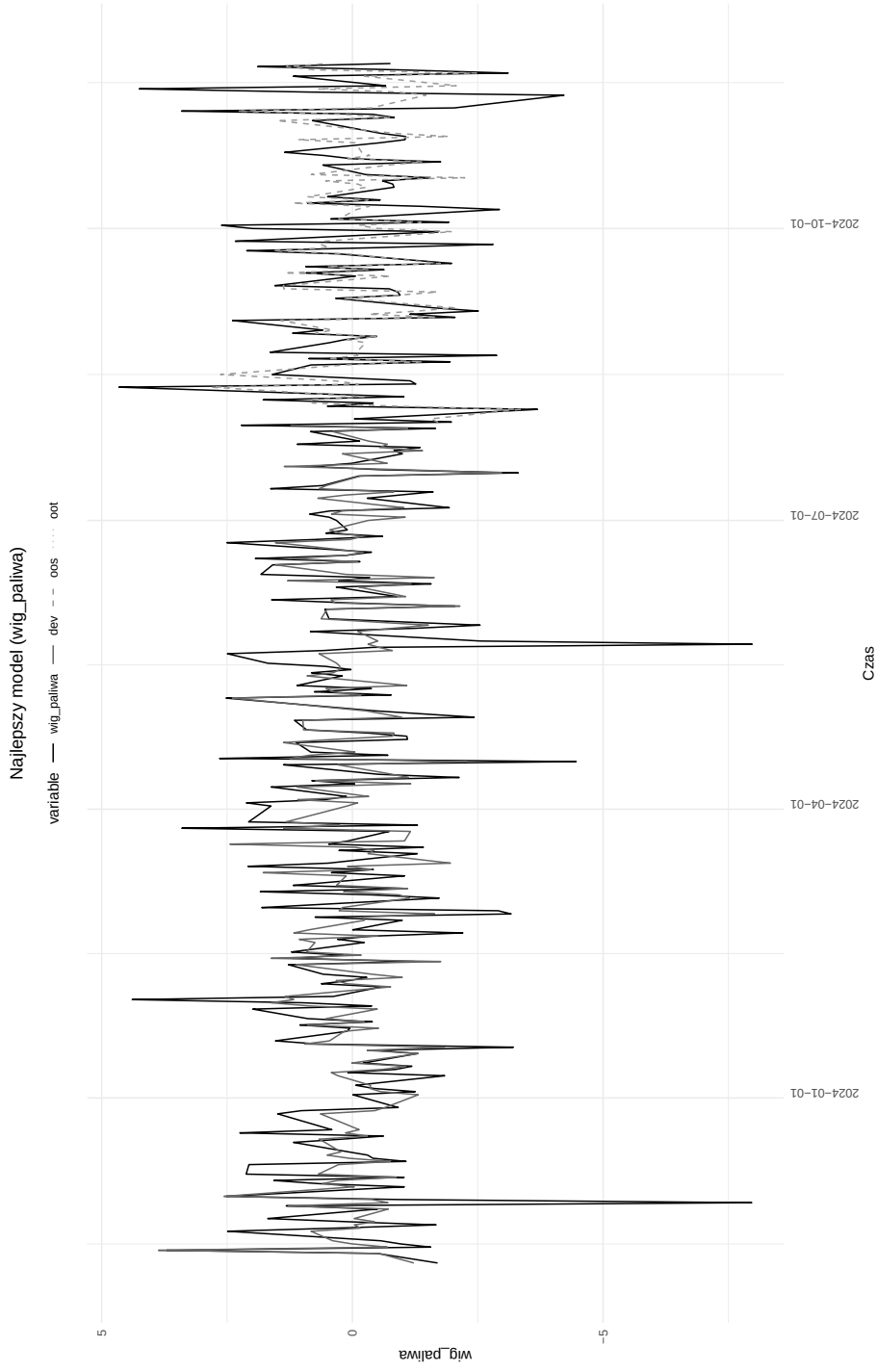
```
      )
  }
)
```

Wykresy – w postaci *landscape* – można umieścić w pętli `for` w sposób podany poniżej.

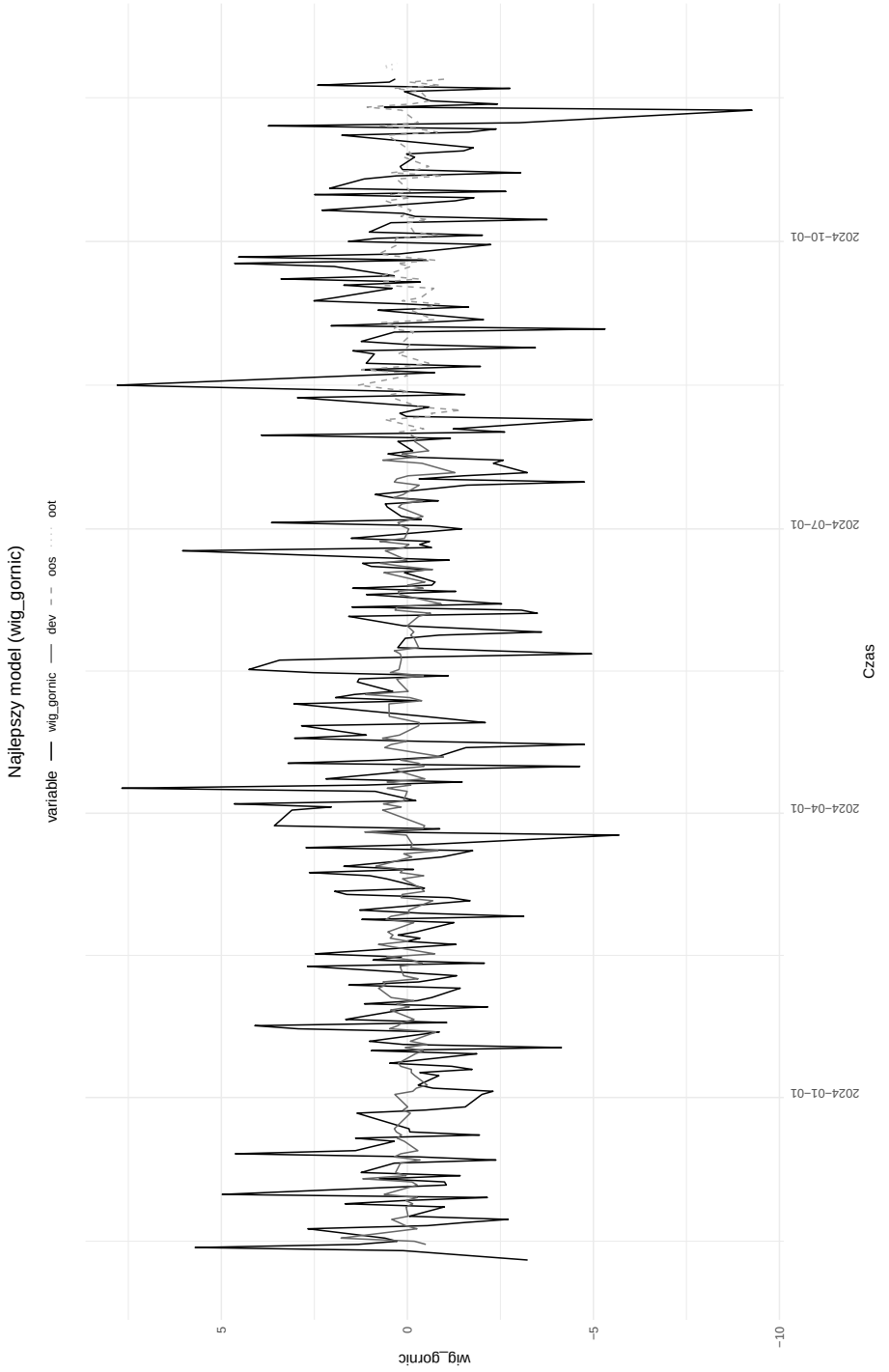
```
i.all <- seq_along(final.model.1)
rnd.str <- stri_rand_strings(
  n = length(i.all),
  length = 6,
  pattern = "[a-z]"
)
captions <- lapply(
  i.all,
  function(x) {
    var.name <- gsub(
      "_",
      "\\\\_",
      valid.models[[final.model.0[x]]]$var.y
    )
    paste0("Najlepszy model: indeks ", var.name)
  }
)
```

```
for (i in i.all) {
  ggplot2::ggsave(
    paste0("output/plots/plot_", rnd.str[i], ".pdf"),
    plot = final.model.1[[i]],
    width = 14,
    height = 9,
    units = "in"
  )
}
```

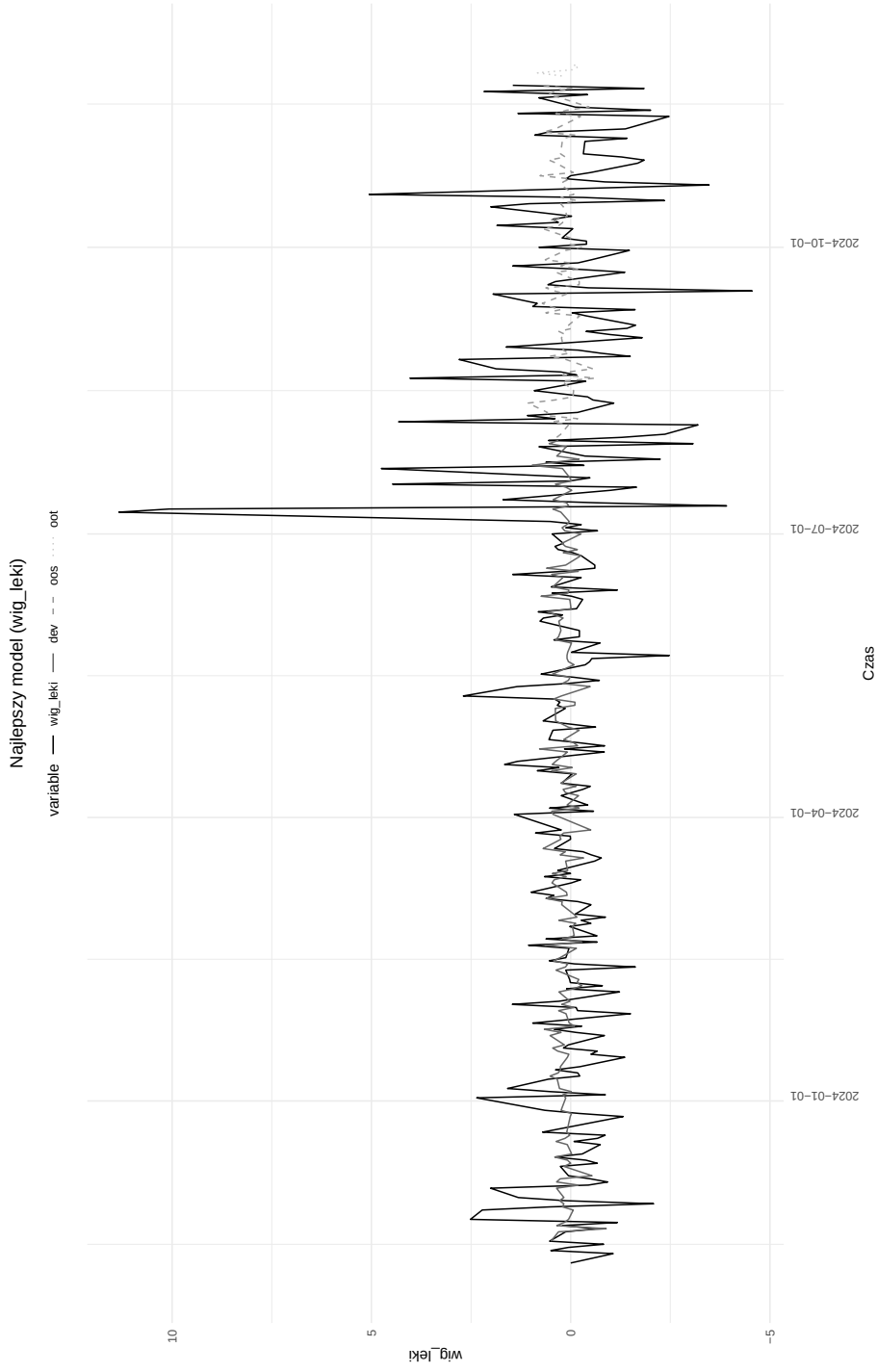
```
for (i in i.all) {
  cat("\\begin{landscape}\n")
  cat("\\begin{figure}[htbp]\n")
  cat("  \\centering\n")
  cat("  \\includegraphics[angle=0,
   ↪  width=\\linewidth]{output/plots/plot_", rnd.str[i], ".pdf}\n",
   ↪  sep = "")
  cat("  \\caption{", captions[[i]], "}\n", sep = "")
  cat("  \\label{fig:", rnd.str[i], "}\n", sep = "")
  cat("\\end{figure}\n")
  cat("\\end{landscape}\n\n")
}
```



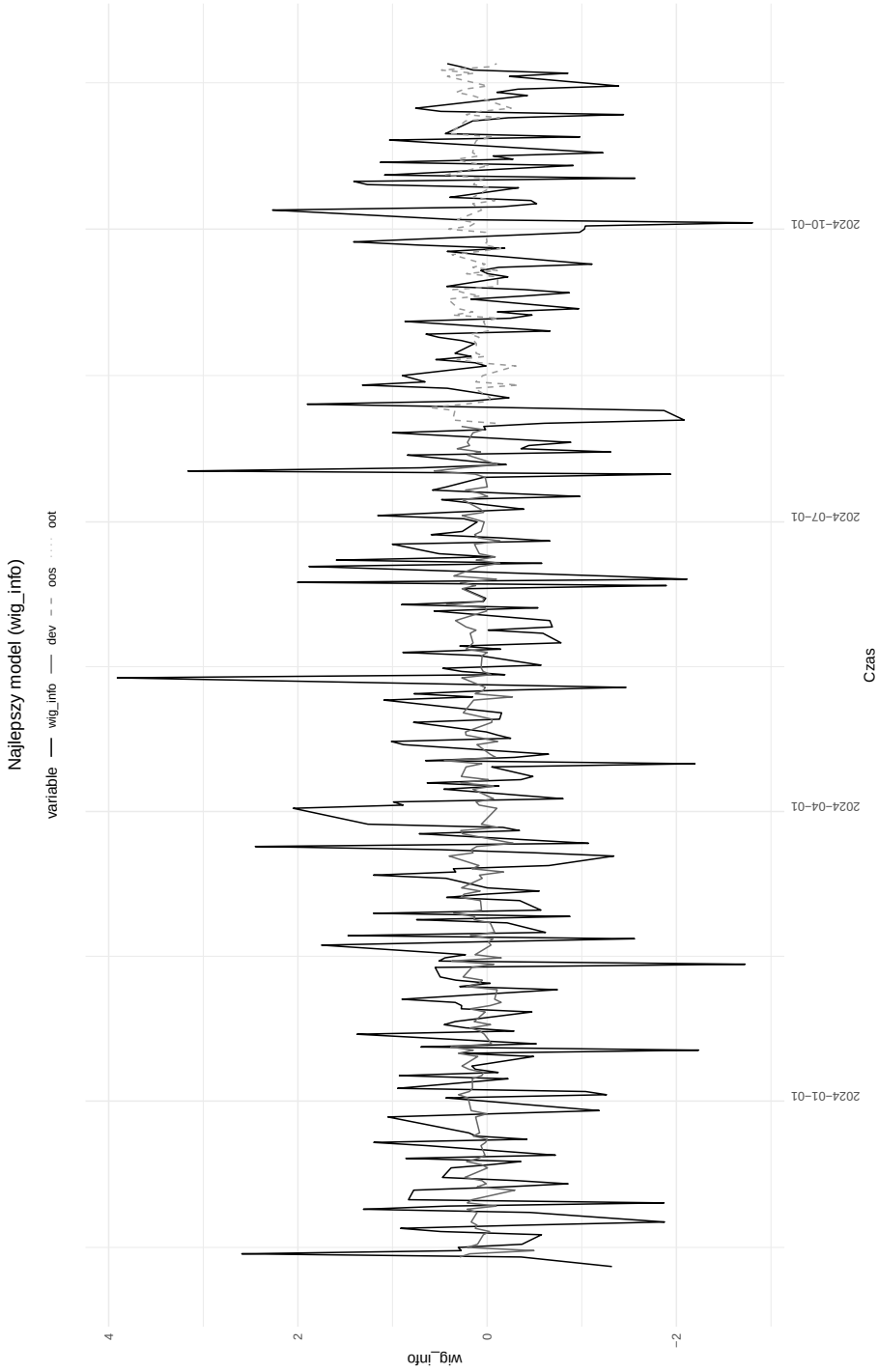
Rysunek 3.17. Najlepszy model: indeks wig_paliwa



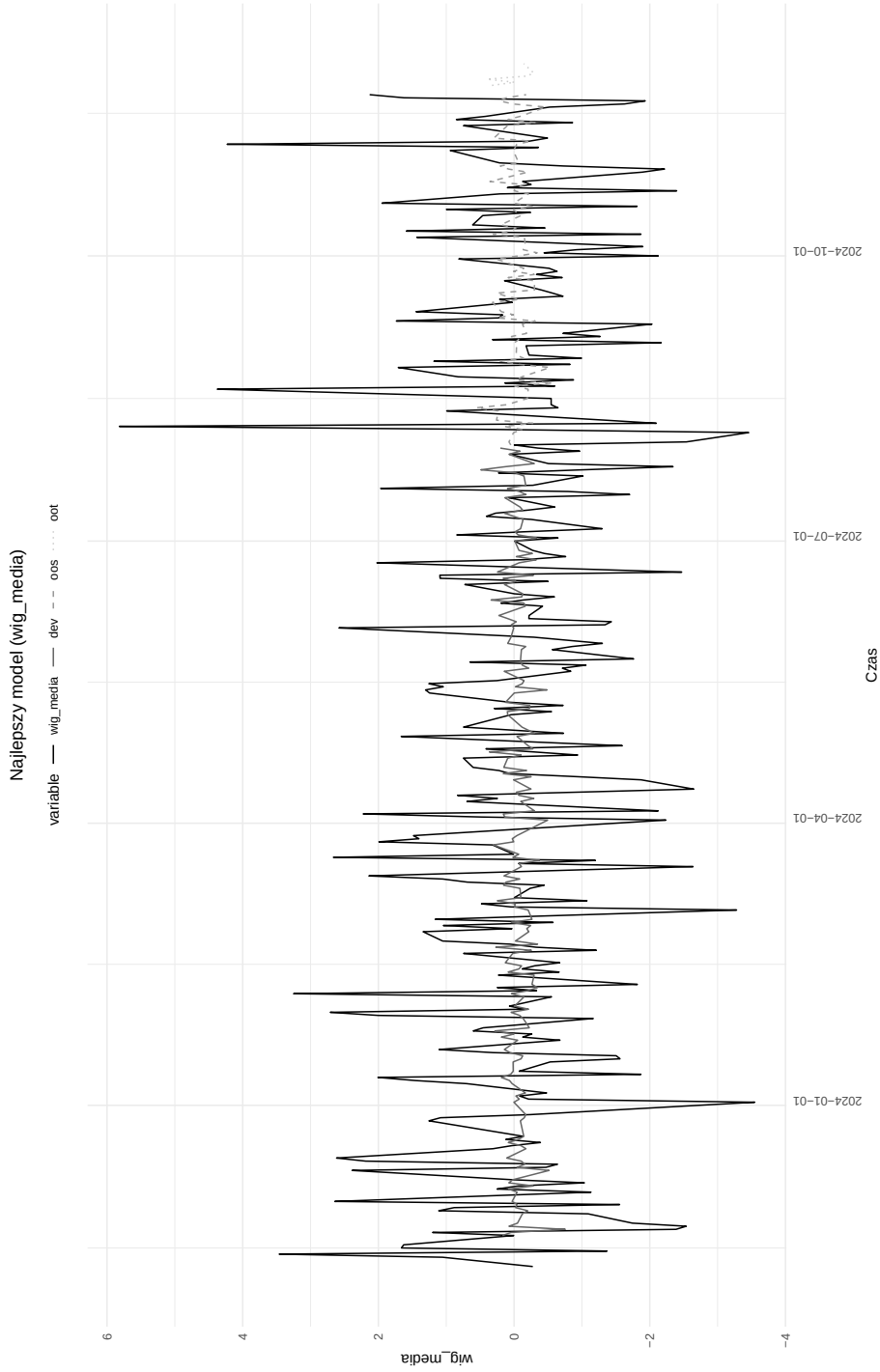
Rysunek 3.18. Najlepszy model: indeks wig_gornic



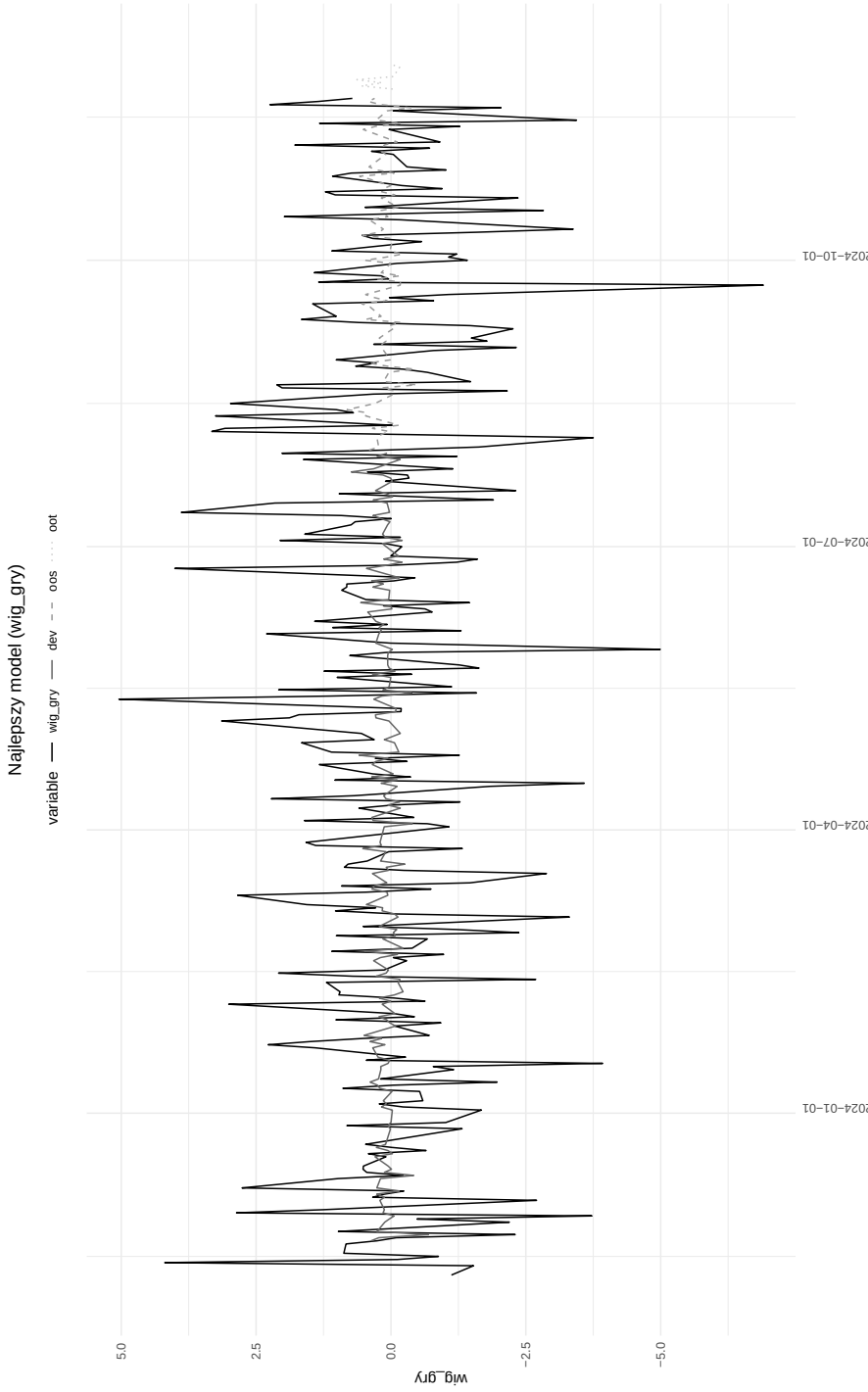
Rysunek 3.19. Najlepszy model: indeks wig_leki



Rysunek 3.20. Najlepszy model: indeks wig_info

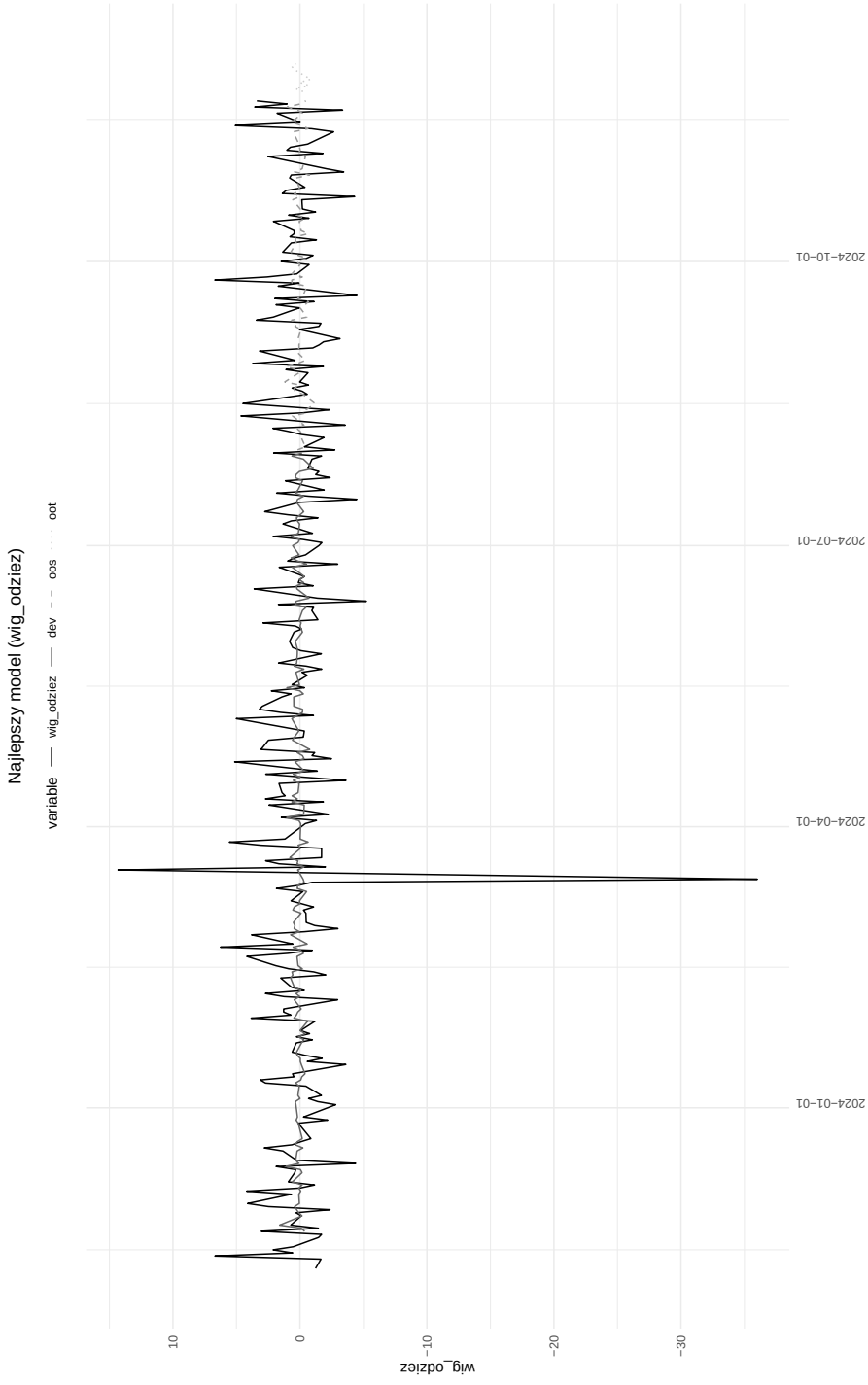


Rysunek 3.21. Najlepszy model: indeks wig_media



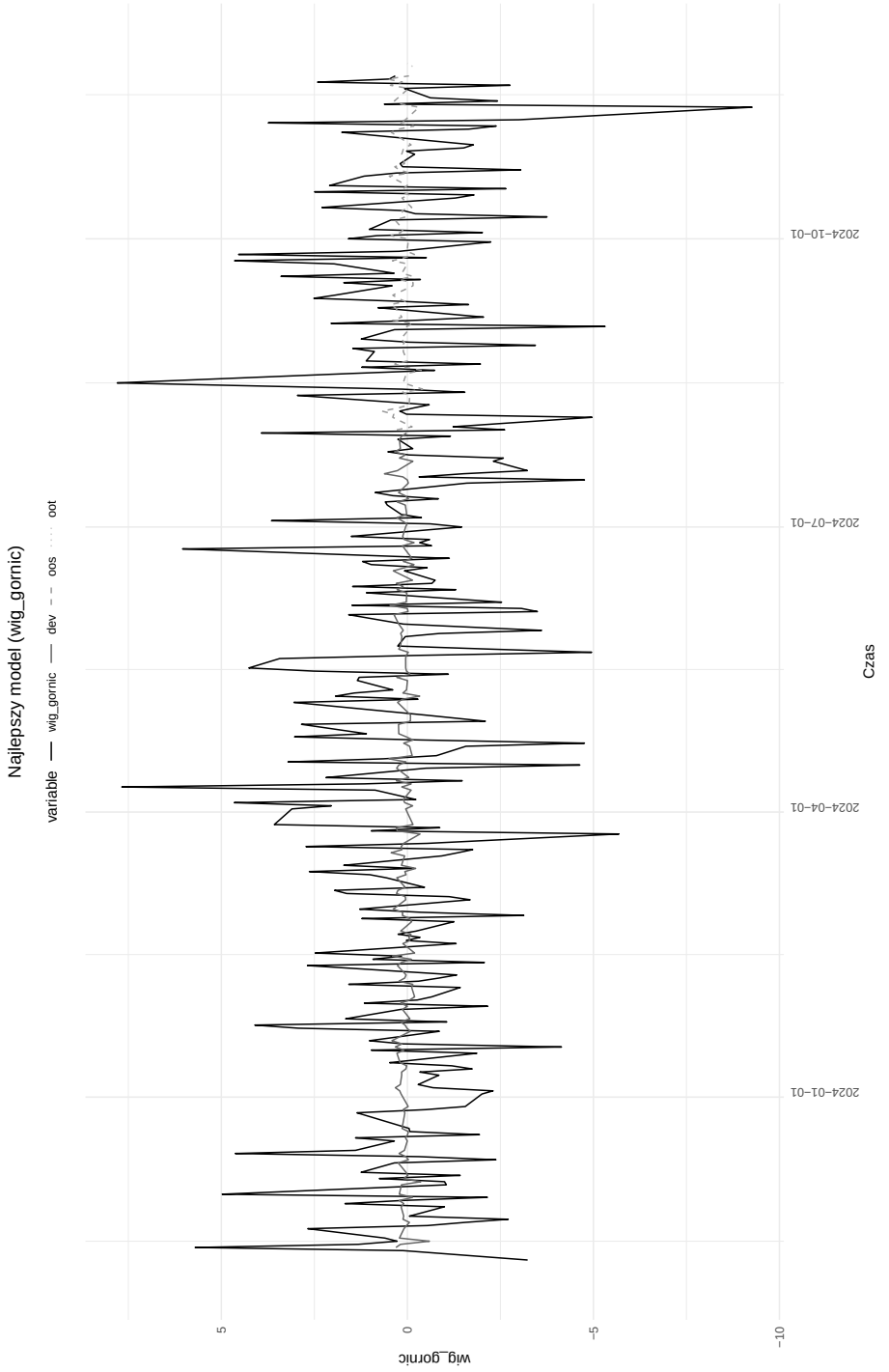
Rysunek 3.22. Najlepszy model: indeks wig_gry

Rysunek 3.23. Najlepszy model: indeks wig_budow



Rysunek 3.24. Najlepszy model: indeks wig_odziez

Rysunek 3.25. Najlepszy model: indeks wig_info



Rysunek 3.26. Najlepszy model: indeks wig_gornic

Rysunek 3.27. Najlepszy model: indeks wig_gry

Na końcu zaprezentowano sposób wygenerowania równań modeli dla indeksów giełdowych, przedstawionych na wykresach. Równania podano wraz z rzędem opóźnienia h, który określa również liczbę prognoz *ex ante* (`oot`). Równania zapisano na liście eq.

```
eq <- lapply(
  final.model.0,
  function(x) {
    m <- valid.models[[x]]
    var.y <- m$var.y %>%
      gsub("wig_", "", .) %>%
      toupper()
    var.x <- m$var.x %>%
      gsub("_wig", "", .) %>%
      toupper()
    a0 <- m$model$coefficients[[1]] %>%
      round(2)
    a1 <- m$model$coefficients[2] %>%
      round(2)
    h  <- m$h
    cat(paste0(
      "$$\\widehat{",
      var.y,
      "}_{t}=",
      a0,
      ifelse(a1>=0, paste0("+", a1), a1),
      var.x,
      ifelse(h == 0, "_{t}", paste0("_{t-", h, "}")),
      "$$",
      "\n\n"
    ))
  }
)
```

$$\widehat{PALIWA}_t = -0.07 + 0.96WIG_t$$

$$\widehat{GORNIC}_t = 0.02 + 0.43WIG_{t-3}$$

$$\widehat{LEKI}_t = 0.18 - 0.26WIG_{t-5}$$

$$\widehat{INFO}_t = 0.11 - 0.15WIG_{t-1}$$

$$\widehat{MEDIA}_t = -0.04 - 0.17WIG_{t-6}$$

$$\widehat{GRY}_t = 0.13 - 0.2WIG_{t-7}$$

$$\widehat{BUDOW}_t = 0.09 + 0.18WIG_{t-10}$$

$$\widehat{ODZIEZ}_t = 0.08 + 0.37WIG_{t-8}$$

$$\widehat{INFO}_t = 0.09 - 0.09WIG_{t-9}$$

$$\widehat{GORNIC}_t = 0.1 - 0.17WIG_{t-2}$$

$$\widehat{GRY}_t = 0.12 - 0.09WIG_{t-4}$$

Podsumowując, dla różnych indeksów branżowych uzyskano różne postacie modeli *optymalnych* pod względem prognostycznym. W przypadkach, w których $h > 0$, równania pozwalają na prognozowanie *ex ante* w horyzoncie h okresów. Czytelnik może zmodyfikować powyższy kod w celu uzyskania innych postaci modeli, zgodnie z wybranymi kryteriami jakości statystycznej (por. obiekt `output` . 1).

W kolejnym podrozdziale analizie poddane zostaną szczegółowe kanały oddziaływania polityki pieniężnej oraz ich znaczenie dla funkcjonowania systemu finansowego, co pozwoli pełniej zrozumieć, w jaki sposób zmiany stóp procentowych banku centralnego oraz na rynku międzybankowym przenikają do sfery realnej gospodarki.

3.3.3.3. Model stopy procentowej

Efekt transmisji zmian stóp procentowych stanowi kluczowy element mechanizmu polityki pieniężnej, determinując zarówno skalę, jak i tempo, w jakim zmiany stóp banku centralnego oddziałują na warunki finansowe w gospodarce. Proces ten obejmuje mechanizmy bezpośrednie, takie jak wpływ stopy referencyjnej na koszty kredytów i oprocentowanie depozytów, oraz mechanizmy pośrednie, wynikające z interakcji rynków finansowych, oczekiwań podmiotów gospodarczych i strukturalnych cech systemu bankowego.

W literaturze przedmiotu wyodrębniono dwa główne nurty badawcze: modele strukturalne, oparte na równaniach popytu i podaży kredytu w ramach teorii kosztów dostosowawczych banków, oraz podejścia ekonometryczne, w szczególności modele korekty błędem (ECM), modele progowe (TAR, M-TAR) czy ich dalsze nieliniowe rozszerzenia, które umożliwiają analizę asymetrii i zależności warunkowych w mechanizmie transmisji. W ostatnich dekadach rozwój metod dla danych panelowych,

estymacji bayesowskiej oraz modeli z parametrami zmiennymi w czasie pozwolił szczegółowo badać różnice w sile i szybkości transmisji między krajami, okresami i segmentami rynku finansowego (Smets, 1995; Taylor, 1995; Gregor i in., 2021).

Zrozumienie mechanizmu transmisji wymaga jednak odniesienia do teorii struktury terminowej stóp procentowych. W praktyce banków centralnych i instytucji finansowych szczególną rolę odgrywają modele umożliwiające spójne prognozowanie krzywej dochodowości i wycenę instrumentów pochodnych, co bezpośrednio przekłada się na ocenę skuteczności polityki pieniężnej. Modele równowagi, takie jak klasyczny model Vasicka (Vasicek, 1977), w którym stopa natychmiastowa (*spot*) jest reprezentowana przez proces stochastyczny przy założeniu braku arbitrażu, stanowią fundament analizy krzywej dochodowości. Przeglądy literatury (Cairns, 2004; Gibson i in., 2010; Wu, 2024) obejmują zarówno modele jednowskaźnikowe, jak i wielowskaźnikowe, podkreślając znaczenie metod numerycznych oraz integracji czynników makroekonomicznych.

W warunkach polskich badania empiryczne koncentrują się przede wszystkim na analizie zależności między krótko- i długoterminowymi stopami WIBOR oraz ocenie przydatności hipotezy racjonalnych oczekiwań w prognozowaniu stóp procentowych. Miłobędzki (2008), Blangiewicz i Miłobędzki (2008), Blangiewicz i Miłobędzki (2009) oraz Miłobędzki (2010) zastosowali modele VAR, VECM oraz TVECM (*Threshold VECM*), wskazując na istnienie relacji długookresowej, a także asymetrii w procesach dostosowawczych, zwłaszcza dla 3-miesięcznej stopy WIBOR. Podobne wnioski sformułowali Ziarko-Siwek i Kamiński (2003), którzy zwrócili uwagę na ograniczoną zdolność prostych modeli dla *spreadu* do prognozowania kierunku zmian krótkoterminowych stóp procentowych. Wynika stąd, że dla polskiego rynku konieczne jest uwzględnianie nieliniowości oraz efektów specyficznych dla terminów zapadalności.

Rozwinięciem tej problematyki jest również kwestia prognozowania krzywej dochodowości, która w literaturze łączona jest z oceną skuteczności polityki pieniężnej. Rubaszek (2016) oraz Kostyra i Rubaszek (2020) porównali prognostyczną skuteczność modeli według Diebold i Li (2006), prostych modeli szeregów czasowych (AR, RW) oraz metod uczenia maszynowego. Badania te wykazały, że w przypadku Polski najlepsze krótkoterminowe prognozy zapewniał model Nelsona–Siegela oparty na hipotezie oczekiwań, choć żadne z podejść nie uchwyciło długoterminowego trendu spadkowego stóp. W odniesieniu do rynku amerykańskiego przewagę w dłuższych horyzontach miały modele dynamiczne, a prognozy warunkowe – zakładające znajomość przyszłych zmiennych makroekonomicznych – istotnie poprawiały trafność prognoz krótkoterminowych.

Nie mniej istotnym obszarem badań jest identyfikacja asymetrii w dostosowaniach detalicznych stóp procentowych do zmian stóp rynkowych. Analiza Sznajderska (2013) dla Polski potwierdziła istnienie asymetrycznej kointegracji między stopami depozytów i kredytów a stopami rynku międzybankowego, uzależnionej m.in. od kierunku zmiany stóp, poziomu aktywności gospodarczej, płynności, oczekiwań i stopnia konkurencji. Podobne wyniki uzyskano w badaniach dla Włoch (Gambacorta i Iannotti, 2007), Australii (Lim, 2001) i Wielkiej Brytanii (Becker i in., 2012).

Meta-analizy (Gregor i in., 2021) dowiodły ponadto, że siła transmisji była niższa w przypadku stóp dla kredytów konsumpcyjnych i długoterminowych oraz że osłabła po kryzysie finansowym, szczególnie w gospodarkach o rosnącej otwartości handlowej i większej zmienności rynków kapitałowych.

Różnice w skuteczności transmisji stóp procentowych wynikały także z uwarunkowań strukturalnych i instytucjonalnych, a także regulacyjnych. Smets (1995) oraz Angeloni i in. (2003) wskazali, że kanał stopy procentowej działał silniej w systemach finansowych o wyższym stopniu rozwoju i większej integracji. Z kolei Papadamou i in. (2015) pokazali, że przejrzystość polityki pieniężnej w gospodarkach wschodzących wzmacniała skuteczność transmisji, zmniejszając potrzebę stosowania radykalnych zmian stóp. Leroy i Lucotte (2016) podkreślili natomiast, że różnice w strukturze rynków finansowych oraz poziomie ryzyka tłumaczyły znaczną część zróżnicowania efektu transmisji w strefie euro, co miało istotne implikacje dla reform sprzyjających integracji finansowej.

Analizując kanały transmisji, należy zauważyć, że polityka pieniężna oddziałuje na gospodarkę wielotorowo – poprzez koszt kapitału, kurs walutowy, a także efekty redystrybucyjne. Taylor (1995) oraz Brzoza-Brzezina i in. (2013) akcentują, iż skuteczność poszczególnych kanałów zależała od struktury gospodarki oraz heterogeniczności podmiotów gospodarczych. Badania Franta i in. (2014) dla Czech oraz Kliber i in. (2016) dla Polski wykazały, że efektywność polityki pieniężnej była zmienna w czasie, a okresy kryzysów finansowych znacząco ograniczały zdolność banku centralnego do kontroli krótkoterminowych stóp procentowych.

Podsumowując, mechanizm transmisji stóp procentowych stanowi proces złożony i podatny na wpływ licznych czynników makroekonomicznych, strukturalnych oraz instytucjonalnych. Modele nieliniowe lepiej opisują rzeczywiste procesy dostosowawcze niż tradycyjne modele liniowe, a o skuteczności transmisji decydują m.in. segmentacja rynku finansowego, przejrzystość banku centralnego, poziom ryzyka oraz warunki konkurencji w sektorze bankowym. Wnioski płynące z badań krajowych i międzynarodowych wskazują jednoznacznie, że efektywność polityki pieniężnej w dużej mierze zależy od jakości i szybkości transmisji stóp, co rodzi zarówno wyzwania metodologiczne dla badaczy, jak i praktyczne dla decydentów.

W tym podrozdziale zaprezentowano przykład analizy stóp oprocentowania kredytów i depozytów w zależności od stopy WIBOR 3M (3-miesięcznej stopy procentowej rynku międzybankowego). Założono, że stopa WIBOR 3M pozostaje pod silnym wpływem stopy referencyjnej banku centralnego. Jak wcześniej wskazano, analiza zmian stóp procentowych ma istotne znaczenie dla poznania mechanizmu transmisji polityki pieniężnej na sektor bankowy, akcję kredytową oraz wzrost gospodarczy (Serwa i Wdowiński, 2017). W kontekście polskim badania Marcinkowska i in. (2014) pokazały, że zaostrzenie wymogów kapitałowych i płynnościowych w sektorze bankowym może prowadzić do nieznacznego spowolnienia tempa wzrostu gospodarczego, głównie poprzez ograniczenie dostępności kredytu i wzrost jego kosztu, przy jednoczesnym zwiększeniu stabilności systemu finansowego. Analiza krótkookresowych skutków regulacji wskazała na ich wpływ na marże kredytowe i koszty pozyskania

kapitału, natomiast efekty długookresowe obejmowały zmniejszenie ryzyka wystąpienia kryzysów finansowych. We wnioskach z badań podkreślono konieczność uwzględniania zarówno kosztów prywatnych ponoszonych przez banki, jak i korzyści społecznych przy określaniu optymalnego poziomu wymogów ostrożnościowych występujących w mechanizmie transmisji.

W dalszej części podrozdziału oszacowano model z jedną zmienną objaśniającą w postaci:

$$y_t = \alpha_0 + \alpha_1 X_t + \epsilon_t \tag{3.90}$$

gdzie:

y_t – stopa procentowa ze zbioru *ECB MIR: MFI Interest Rate Statistics* (w proc.),
X_t – stopa WIBOR 3M (w proc.),
α_0 i α_1 – parametry modelu,
ϵ_t – składnik losowy, $\epsilon_t \sim N(0, \sigma_\epsilon^2)$.

ECB MIR to zbiór notowań stóp procentowych stosowanych przez monetarne instytucje finansowe, publikowanych przez Europejski Bank Centralny. Choć głównym zakresem MIR jest strefa euro, dane obejmują także wybrane kraje spoza strefy euro – w tym Polskę – które raportują według metodologii ECB. Dane obejmują m.in. stopy procentowe od nowych umów kredytowych i depozytowych oraz od istniejących sald, wykorzystywane do analizy warunków finansowania i transmisji polityki pieniężnej.

Do zgromadzenia materiału statystycznego wykorzystano bazę danych *ECB Statistical Data Warehouse*[5], która zawiera statystyki stóp procentowych *MIR: MFI Interest Rate Statistics*[6]. Do pobrania danych wykorzystano bibliotekę `ecb`. Na początku należy wczytać odpowiednie biblioteki.

```
library(dplyr)
library(ecb)
library(eurostat)
library(forcats)
library(ggplot2)
library(magrittr)
library(purrr)
library(reshape2)
library(rlist)
library(stringr)
library(tidyr)
library(tseries)
library(xts)
library(zoo)
```

[5] Por. https://data.ecb.europa.eu/.
[6] Por. https://data.ecb.europa.eu/data/datasets/MIR/data-information.

W poniższym algorytmie wykorzystano dwie dodatkowe biblioteki:

- zoo (obsługa szeregów czasowych) (Zeileis i in., 2023),
- purrr (zestaw narzędzi do obsługi funkcji i wektorów).

```
dat <- ecb.sdw("MIR.M.PL.......PLN.")
```

Parametr funkcji ecb.sdw określa identyfikator zmiennych w bazie ECB, w którym mogą występować symbole *wieloznaczne* (*wildcard*). Jeśli pomiędzy kropkami znajdują się symbole zmiennych, to ich usunięcie oznacza wprowadzenie symbolu wieloznacznego. W bazie ECB istnieje np. zmienna o identyfikatorze MIR.M.PL.B.A20.F.R.A.2240.PLN.O[7]. Usunięcie części symboli powoduje rozszerzenie zbioru zmiennych i pobranie wszystkich tych, które spełniają warunki wieloznaczne. Tę możliwość zapewniają funkcje biblioteki ecb. Identyfikator MIR.M.PL.......PLN. oznacza, że zmienne:

- pochodzą z zasobu MIR,
- mają częstotliwość miesięczną M,
- dotyczą gospodarki Polski PL,
- dotyczą stóp oprocentowania dla waluty polskiej PLN.

Funkcja ecb.sdw zostanie zdefiniowana w dalszej części podrozdziału.

```
ecb.sdw <- function(x) {
  # zwkmnc
}
```

Deklarację treści zwkmnc w funkcji ecb.sdw podano w kolejnych krokach. Zadaniem poniższego kodu jest pobranie identyfikatorów wszystkich zmiennych w ramach symbolu wieloznacznego MIR.M.PL.......PLN..

```
id <- x %>%
  ecb::get_data(., list(detail = "nodata")) %>%
  names()
```

Kolejne symbole zmiennych z wektora id zostaną przekazane do funkcji get_data z biblioteki ecb. Dane dla zmiennych pobrano do ramek danych i zapisano jako elementy listy dat.1 za pomocą funkcji lapply. Jeśli dane dla wybranej zmiennej są niedostępne, element listy otrzymuje wartość NULL. Lista dat.1 jest następnie oczyszczana z elementów o wartości NULL za pomocą funkcji list.clean z biblioteki rlist.

[7] Por. https://sdw.ecb.europa.eu/quickview.do?SERIES_KEY=124.MIR.M.PL.B.A20.F.R.A.2240.PLN.O.

```
dat.1 <- lapply(
  id,
  function(x) {
    tryCatch({
      ecb::get_data(x, list(detail = "full"))
  },
    error = function(e) {
      return(NULL)
  },
    warning=function(w) {
      return(NULL)
  })
}
) %>%
  list.clean(., function(x) is.null(x), TRUE)
```

W kolejnym kroku procedury zostały przywrócone nazwy elementów listy zgodne z symbolem pobranej zmiennej.

```
names(dat.1) <- id
```

Następnie na liście `dat.2` zapisano wybrane informacje o zmiennych, które umożliwiają sformułowanie ich nazw. Są to tytuł `title` oraz komentarz `comment_ts`, służące do opisu zmiennych w bazie ECB.

```
dat.2 <- lapply(
  1:length(dat.1),
  function(x) {
    tryCatch({
      name <- names(dat.1[[x]])
      title <- str_detect(name, "title") %>%
        { name[.] }
      comment <- str_detect(name, "comment_ts") %>%
        { name[.] }
      dat <- dat.1[[x]] %>%
        data.frame(., stringsAsFactors = FALSE) %>%
        .[, c(
          title,
          comment,
          "obstime",
          "obsvalue"
        )] %>%
          unite(
            .,
            varname,
            1:(length(title)+length(comment)),
```

```
            sep = ",",
            remove = TRUE
          )
      varname <- unique(dat$varname)
      dat <- dat %>%
        .[, -1] %>%
        set_colnames(c("time", varname))
      dat$time <- paste0(dat$time, "-01") %>%
        as.Date()
      return(dat)
  },
    error = function(w) {
      return(NULL)
  })
}
) %>%
  list.clean(., function(x) is.null(x), TRUE)
```

Struktura wywołań funkcji `lapply`, które prowadzą do utworzenia listy `dat.2`, jest następująca:

- indeksem są kolejne pozycje listy `dat.1`,
- do zmiennej `name` pobrano kolejno wszystkie cechy zmiennych z bazy ECB,
- w zmiennej `title` zapisano wszystkie cechy uwzględniające ciąg `'title'`,
- w zmiennej `comment` zapisano wszystkie cechy uwzględniające ciąg `'comment_ts'`,
- w ramce danych `dat` zapisano odpowiednio skonfigurowane kolumny zawierające tytuł, komentarz, datę obserwacji i jej wartość,
- zmieniono konstrukcję ramki danych `dat` za pomocą funkcji `unite` z biblioteki `tidyr`, aby połączyć tytuł z komentarzem w jednej kolumnie,
- rozbudowaną definicję zmiennej zapisano w zmiennej `varname`,
- z ramki `dat` usunięto zbędną kolumnę i nadano nowe nazwy kolumnom,
- specyficzny zapis daty miesięcznej obserwacji z bazy ECB zapisano jako obiekt `date`,
- na koniec należało zwrócić wartość ramki `dat` do elementu listy `dat.2` i oczyścić listę z elementów o wartościach `NULL` na wypadek, gdyby nie powiodło się pobranie danych dla części zmiennych.

W kolejnym kroku utworzono listę `dat.3`, która stanowi uporządkowanie listy `dat.2`, polegające na usunięciu szeregów o zbyt małej liczbie obserwacji.

```
dat.3 <- lapply(
  dat.2,
  function(x) {
    data.frame(x, stringsAsFactors = FALSE) %>%
```

```
      na.omit() %>%
      {
        if (nrow(.) == 0 | nrow(.) == 1) NULL
        else {
          if (nrow(.) < 5*12) NULL
          else .
  }
}
}
) %>%
  list.clean(., function(x) is.null(x), TRUE)
```

W kolejnej fazie utworzono listę `dat.4`, stanowiącej dalsze uporządkowanie listy `dat.3`, w której zachowano jedynie szeregi kończące się nie dawniej niż kwartał temu.

```
dat.4 <- lapply(
  dat.3,
  function(x) {
    last <- x$time %>%
      last
    if (last < Sys.Date()-1*90) NULL
    else x
}
) %>%
  list.clean(., function(x) is.null(x), TRUE)
```

Lista `dat.4` została przekształcona w listę `dat.5`, gdzie każdy element jest ramką danych.

```
dat.5 <- lapply(
  dat.4,
  function(x) data.frame(x, stringsAsFactors = FALSE)
)
```

W kolejnym kroku została zdefiniowana zmienna `z` zawierająca ciąg dat miesięcznych.

```
z <- as.yearmon(1995 + seq(0, 30*12)/12) %>%
  as.Date() %>%
  tibble(time = ., var = NA)
z %>% data.frame() %>% head
```

Do utworzenia obiektu `z` wykonano następujące czynności:

- utworzono sekwencję miesięcy od 1995 roku przez 30 lat (30*12 miesięcy),
- przekonwertowano ją do obiektu typu `Date`,
- utworzono ramkę danych `tibble` z dwiema kolumnami:
  - `time`: daty miesięczne od stycznia 1995 do grudnia 2025,
  - `var`: wartości NA.

Lista `dat.5` została następnie przekształcona w listę `dat.6`.

```
dat.6 <- dat.5 %>%
  list.insert(., 1, data.frame(z)) %>%
  reduce(left_join, by = "time") %>%
  .[, -2]
```

Lista `dat.6` powstała w rezultacie następujących czynności:

- Wstawienie ramki danych `z` jako pierwszego elementu listy `dat.5`.
- Wykonanie łańcuchowego lewostronnego łączenia (`left_join`) wszystkich ramek danych w tej liście według kolumny `time`.
- Usunięcie drugiej kolumny z wynikowej ramki danych – czyli kolumny `var`, która pochodziła z `z` i miała same wartości NA.

Nazwy kolumn w ramce `dat.6` zostały uporządkowane. Poniższy kod realizuje następujące czynności:

- funkcja `str_squish` usuwa nadmiarowe *białe znaki* (np. spacje) w nazwach kolumn:
  - zamienia wiele spacji w jedną,
  - usuwa spacje z początku i końca napisu.
- `str_replace_all` zamienia wszystkie ciągi kropek na jedną kropkę.
- pierwsza kolumna otrzymuje nazwę `time`, niezależnie od tego, jak nazywała się wcześniej.

```
colnames(dat.6) <- colnames(dat.6) %>%
  str_squish() %>%
  str_replace_all(., "\\.+", ".")
colnames(dat.6)[1] <- "time"
```

Kod `zwkmnc` kończy się przekazaniem ramki danych `dat.6` jako wartości funkcji `ecb.sdw`, co oznacza koniec postępowania prowadzącego do utworzenia zbioru danych.

```
return(dat.6)
```

Cały kod funkcji `ecb.sdw` podano poniżej.

```
ecb.sdw <- function(x) {
  id <- x %>%
    ecb::get_data(., list(detail = "nodata")) %>%
    names()
  dat.1 <- lapply(
    id,
    function(x) {
      tryCatch({
        ecb::get_data(
          x,
          list(detail = "full")
        )
      },
      error = function(e) {
        return(NULL)
      },
      warning=function(w) {
        return(NULL)
      })
    }
    ) %>%
      list.clean(., function(x) is.null(x), TRUE)
  names(dat.1) <- id
  dat.2 <- lapply(
    1:length(dat.1),
    function(x) {
      tryCatch({
        name <- names(dat.1[[x]])
        title <- str_detect(name, "title") %>%
          { name[.] }
        comment <- str_detect(name, "comment_ts") %>%
          { name[.] }
        dat <- dat.1[[x]] %>%
          data.frame(., stringsAsFactors = FALSE) %>%
          .[, c(
            title,
            comment,
            "obstime",
            "obsvalue"
          )] %>%
            unite(
              .,
              varname,
              1:(length(title)+length(comment)),
```

```
            sep = ",", remove = TRUE
          )
      varname <- unique(dat$varname)
      dat <- dat %>%
        .[, -1] %>%
        set_colnames(c("time", varname))
      dat$time <- paste0(dat$time, "-01") %>%
        as.Date()
      return(dat)
  },
    error = function(w) {
      return(NULL)
    })
  }
) %>%
  list.clean(., function(x) is.null(x), TRUE)
dat.3 <- lapply(
  dat.2,
  function(x) {
    data.frame(x, stringsAsFactors = FALSE) %>%
      na.omit() %>%
      {
        if (nrow(.) == 0 | nrow(.) == 1) NULL
        else {
          if (nrow(.) < 5*12) NULL
          else .
        }
      }
  }
) %>%
  list.clean(., function(x) is.null(x), TRUE)
dat.4 <- lapply(
  dat.3,
  function(x) {
    last <- x$time %>%
      last
    if (last < Sys.Date()-1*120) NULL
    else x
  }
) %>%
  list.clean(., function(x) is.null(x), TRUE)
dat.5 <- lapply(
  dat.4,
  function(x) data.frame(x, stringsAsFactors = FALSE)
)
z <- as.yearmon(1995 + seq(0, 30*12)/12) %>%
  as.Date() %>%
  tibble(time = ., var = NA)
dat.6 <- dat.5 %>%
```

```
    list.insert(., 1, data.frame(z)) %>%
    reduce(left_join, by = "time") %>%
    .[, -2]
  colnames(dat.6) <- colnames(dat.6) %>%
    str_squish() %>%
    str_replace_all(., "\\.+", ".")
  colnames(dat.6)[1] <- "time"
  return(dat.6)
}
```

Teraz można już wykorzystać funkcję `ecb.sdw` do pobrania zbioru zmiennych z bazy ECB.

```
dat.0 <- ecb.sdw("MIR.M.PL.....PLN.")
```

```
ncol(dat.0)
```

```
## [1] 125
```

Zbiór `dat.0` zawiera 125 zmiennych. Może się zdarzyć, że ten zbiór nie jest *gęsty*, tj. po usunięciu brakujących obserwacji w zmiennych, kolumna `time` zawiera luki w sekwencji dat. Tę własność sprawdzono za pomocą poniższego kodu.

```
dat.1 <- lapply(
  names(dat.0)[-1],
  function(x) {
    y <- dat.0[, x]
    non.na <- which(!is.na(y))
    if(length(non.na) == 0) {
      return(FALSE)
    } else {
      d <- diff(non.na)
      return(all(d == 1))
    }
  }
) %>%
  unlist
```

Lista `dat.1` zawiera wartości `TRUE`, jeśli kolumna zbioru `dat.0` zawiera wartości `NA` co najwyżej na początku lub na końcu wektora. Można zatem zaindeksować zbiór `dat.0` za pomocą `dat.1` i usunąć kolumny, które nie spełniają założenia *gęstości* zbioru obserwacji.

```
dat.2 <- dat.0[, c(TRUE, dat.1)]
```

Może się zdarzyć, że zmienne nie zawierają stóp procentowych, lecz inne wartości (np. wolumeny). Ponieważ w zbiorze powinny pozostać wyłącznie stopy procentowe, potrzebne jest zastosowanie kolejnego filtru. Jednak należy zauważyć, że nazwa każdej zmiennej jest długa, gdyż zawiera pełną definicję zmiennej z bazy ECB.

```
names(dat.2)[2] %>%
  str_trunc(width = 40, side = "right", ellipsis = "...")
```

```
## [1] "Bank.interest.rates.loans.to.corporat..."
```

W tej sytuacji utrudnione jest przeszukanie nazw kolumn (zmiennych) pod względem określonych (lecz trudnych do ustalenia) ciągów znaków i dlatego zastosowano filtr numeryczny. Na podstawie wartości stopy procentowej WIBOR 3M oznaczono do usunięcia te stopy procentowe ze zbioru `dat.2`, których wartości nie spełniają reguły 3σ. Najpierw pobrano wartości zmiennej WIBOR 3M z bazy Eurostat.

```
wib3m <- get_eurostat(
  "irt_st_m",
  filters = list(
    geo = "PL",
    int_rt = "IRT_M3"
  )
)
head(wib3m)
```

```
## # A tibble: 6 x 5
##   freq  int_rt geo   time       values
##   <chr> <chr>  <chr> <date>      <dbl>
## 1 M     IRT_M3 PL    1970-01-01     NA
## 2 M     IRT_M3 PL    1970-02-01     NA
## 3 M     IRT_M3 PL    1970-03-01     NA
## 4 M     IRT_M3 PL    1970-04-01     NA
## 5 M     IRT_M3 PL    1970-05-01     NA
## 6 M     IRT_M3 PL    1970-06-01     NA
```

```
wib3m <- wib3m %>%
  .[, -c(1:3)] %>%
  set_colnames(c("time", "wib3m"))
```

Następnie obliczono średnią tej stopy i wykonano filtrowanie zbioru `dat.2`.

```
options(scipen = 999)
mean <- mean(wib3m$wib3m, na.rm = TRUE)
sd   <- sd(wib3m$wib3m, na.rm = TRUE)
check.0 <- dat.2[-1] %>%
  colMeans(na.rm = TRUE) %>%
  as.vector
check.0
```

```
##  [1]      5.280000      5.375444      5.852556      5.420444
 ↪  5.672667
##  [6]      5.245000      7.207333      5.643361      5.536183
 ↪  5.369889
## [11]      5.850111      5.187778      5.991222      5.394889
 ↪  5.724000
## [16]      4.554111      6.192573      5.217925     10.594357
 ↪  10.204938
## [21]      8.746888    896.704918      4.795222      5.371444
 ↪  7.441000
## [26]      8.549344   5235.778409   4522.164773   1264.437500
 ↪  5.129886
## [31]      4.735852      5.113011   5948.669767   5868.218605
 ↪  5.406763
## [36]      5.544606      5.075778   4363.784091   1538.187500
 ↪  713.738636
## [41]      5.403295      5.097159      5.511080      5.367727
 ↪  5.642557
## [46]   6886.933333     17.655228      9.124444      9.011556
 ↪  4605.697674
## [51]     11.985726   1460.409302     10.872614    762.604651
 ↪  13.104730
## [56]      6.316266      5.517444      5.568889   3716.213953
 ↪  5.918714
## [61]   3370.437500      5.581364      7.237333   2273.232558
 ↪  1346.386364
## [66]      7.352282      7.342386  58002.182573  34307.535270
 ↪  5.255892
## [71]     10.714647      4.861080      9.612557     15.224432
 ↪  14.817159
## [76]      1.187593      1.037386 100966.456432  52241.680498
 ↪  2.875975
## [81]      3.093610      3.128921      3.014813
```

Łatwo zauważyć, że niektóre średnie wartości w kolumnach zbioru `dat.2` wskazują na inny charakter zmiennej niż stopa procentowa. Zatem ze zbioru `dat.2` usunięto kolumny zaindeksowane na zbiorze `check.1`.

```
mult <- 3
check.1 <- check.0 %>%
  { (. >= mean-mult*sd) & (. <= mean+mult*sd) }
dat.3 <- dat.2[, c(TRUE, check.1)]
```

Teraz można już połączyć ramki danych `wib3m` i `dat.3` za pomocą funkcji `merge`.

```
wib3m$time <- as.Date(wib3m$time)
dat.3$time <- as.Date(dat.3$time)
dat.ir.1 <- merge(wib3m, dat.3, by = "time")
```

Następnie utworzono obiekt `dat.ir.2` na podstawie kolumn ramki `dat.ir.1` od trzeciej do ostatniej.

```
dat.ir.2 <- lapply(
  3:ncol(dat.ir.1),
  function(x) {
    c.names <- colnames(dat.ir.1)[c(1:2, x)]
    dat <- data.frame(dat.ir.1[, 1:2], dat.ir.1[, x]) %>%
      set_colnames(c.names) %>%
      drop_na()
    dat$time <- dat$time %>%
      as.Date()
    dat <- dat %>%
      .[with(., order(time)), ]
    return(dat)
  }
)
```

Dla obiektu `dat.ir.2` wykonano następujące czynności:

- Pobranie nazw trzech kolumn: `time`, `wib3m` i aktualnej kolumny `x`.
- Utworzenie nowej ramki danych, która zawiera: `time`, drugą kolumnę z `dat.ir.1`, aktualną kolumnę `x` (czyli jedną ze zmiennych z `dat.3`).
- Nadanie kolumnom odpowiednich nazw.
- Usunięcie wierszy z brakującymi danymi.
- Konwersję kolumny `time` na typ `Date`.
- Sortowanie ramki danych po `time`.

Następnie wykonano automatyczną estymację parametrów modeli liniowych dla każdej ramki danych z listy `dat.ir.2`.

```
dat.ir.3 <- lapply(
  seq_along(dat.ir.2),
  function(i) {
    id <- i
    z <- dat.ir.2[[i]]
    y <- z[, 3]
    x <- z[, 2]
    reg <- lm(y ~ x)
    result <- list(
      reg,
      z$time,
      data.frame(
        name   = names(z)[3] %>%
          gsub("\\.", " ", .),
        a.0    = summary(reg)$coefficients[1, 1],
        a.1    = summary(reg)$coefficients[2, 1],
        t_a.0  = summary(reg)$coefficients[1, 3],
        t_a.1  = summary(reg)$coefficients[2, 3],
        R2     = summary(reg)$r.squared*100,
        JB     = jarque.bera.test(reg$residuals)$p.value,
        SEE    = summary(reg)$sigma,
        sample = paste0(first(z$time), " : ", last(z$time)),
        id     = id
      )
    )
    return(result)
  }
)
```

Obiekt `dat.ir.3` powstał w wyniku czynności:

- Dla każdej ramki danych z `dat.ir.2` szacuje się parametry modelu, w którym stopa procentowa jest funkcją stopy WIBOR 3M.
- Zapisuje się wynikowy obiekt regresji `lm`, daty i podsumowanie wyników w postaci ramki danych.

Wyniki oszacowań regresji zawarte w ramce `dat.ir.3` obejmują:

- `reg` – cały model (do ewentualnej późniejszej analizy),
- `z$time` – wektor dat,
- ramkę danych z podsumowaniem wyników:
- `name` – nazwa zmiennej objaśnianej (kolumna 3 z zamianą kropek na spacje),
- `a.0`, `a.1` – współczynniki regresji,
- `t_a.0`, `t_a.1` – statystyki *t* dla tych współczynników,
- `R2` – współczynnik determinacji,
- `JB` – *p*-wartość dla testu Jarque'a–Bery (normalności reszt),

- SEE – standardowy błąd modelu,
- sample – zakres czasowy danych użytych w estymacji,
- id – numer identyfikacyjny modelu.

Następnie zebrano wyniki oszacowań regresji z listy dat.ir.3 w jedną zwartą ramkę danych dat.ir.4, tj. x[[3]] oznacza, że pobrano trzeci element z każdego z tych podzbiorów, czyli ramkę z wynikami. Ramki danych zostały połączone wierszami (rbind) w jedną dużą ramkę danych. Następnie zmodyfikowano kolumny od 2 do 8 (a.0, a.1, t_a.0, t_a.1, R2, JB, SEE) poprzez zaokrąglenie wszystkich liczb do dwóch miejsc po przecinku.

```
dat.ir.4 <- lapply(
  dat.ir.3,
  function(x) x[[3]]
) %>%
  do.call(rbind, .) %>%
  mutate_at(2:8, function(x) round(x, 2)) %>%
  arrange(desc(R2))
```

Ramka dat.ir.4 to posortowana malejąco według kolumny R2, czytelna tablica zbiorcza wszystkich wyników oszacowań regresji. Dla tego zbioru można pokazać wybrane wyniki w postaci graficznej (rys. 3.28). To zadanie realizuje poniższy kod, który zawiera dwie osobne *mapy* dla t_a.1 i R2 wraz z dodaniem oznaczeń istotności statystycznej (*, **, ***). Nazwy zmiennych są następujące:

```
short.name <- dat.ir.4[, c("id", "name")]
short.name$name <- str_trunc(short.name$name, 60)
short.name
```

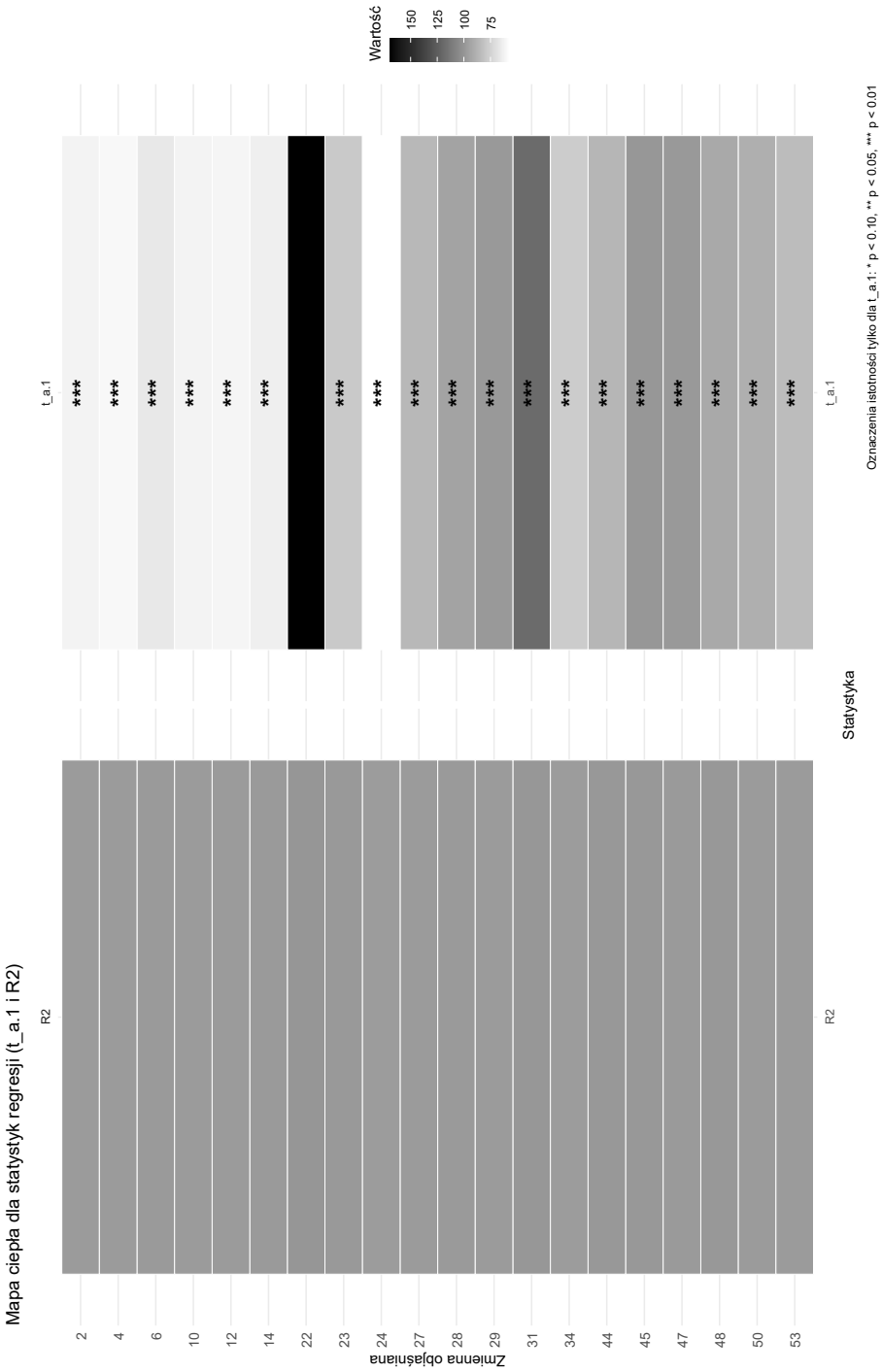
```
##    id                                                          name
## 1  22 Bank interest rates loans to corporations of up to EUR 1M...
## 2  31 Bank interest rates loans to corporations with a floating...
## 3  45 Poland Annualised agreed rate AAR Narrowly defined effect...
## 4  48 Bank interest rates loans to households for other purpose...
## 5  44 Bank interest rates loans to households for house purchas...
## 6  23 Bank interest rates loans to corporations of over EUR 1M ...
## 7  47 Bank interest rates loans to households for house purchas...
## 8  28 Bank interest rates loans to corporations of over EUR 25M...
## 9   6 Bank interest rates loans to corporations with an origina...
## 10 14 Poland Annualised agreed rate AAR Narrowly defined effect...
## 11 50 Poland Annualised agreed rate AAR Narrowly defined effect...
## 12  2 Bank interest rates loans to corporations with an origina...
## 13 10 Bank interest rates loans to corporations with an origina...
## 14 12 Bank interest rates loans to corporations with an origina...
## 15 27 Poland Annualised agreed rate AAR Narrowly defined effect...
```

```
## 16  4 Bank interest rates loans to corporations with an origina...
## 17 29 Bank interest rates loans to corporations of up to EUR 1M...
## 18 53 Poland Annualised agreed rate AAR Narrowly defined effect...
## 19 24 Poland Annualised agreed rate AAR Narrowly defined effect...
## 20 34 Bank interest rates loans to corporations of over EUR 25M...
## 21 35 Poland Annualised agreed rate AAR Narrowly defined effect...
## 22 26 Bank interest rates loans to corporations of over EUR 1M ...
## 23  1 Bank interest rates loans to corporations with an origina...
## 24 32 Poland Annualised agreed rate AAR Narrowly defined effect...
## 25 33 Bank interest rates loans to corporations of up to EUR 0 ...
## 26 51 Bank interest rates revolving loans overdrafts convenienc...
## 27 13 Poland Annualised agreed rate AAR Narrowly defined effect...
## 28  8 Poland Annualised agreed rate AAR Narrowly defined effect...
## 29 30 Poland Annualised agreed rate AAR Narrowly defined effect...
## 30 59 Bank interest rates deposits from corporations with an ag...
## 31 43 Bank interest rates loans to households for house purchas...
## 32 46 Poland Annualised agreed rate AAR Narrowly defined effect...
## 33  7 Poland Annualised agreed rate AAR Narrowly defined effect...
## 34  9 Poland Annualised agreed rate AAR Narrowly defined effect...
## 35 38 Bank interest rates loans to households for consumption n...
## 36 39 Bank interest rates loans to households for consumption p...
## 37 36 Bank interest rates loans to corporations with collateral...
## 38 61 Poland Annualised agreed rate AAR Narrowly defined effect...
## 39 60 Bank interest rates deposits from households with an agre...
## 40 49 Poland Annualised agreed rate AAR Narrowly defined effect...
## 41 15 Poland Annualised agreed rate AAR Narrowly defined effect...
## 42  5 Bank interest rates loans to households with an original ...
## 43  3 Bank interest rates loans to households with an original ...
## 44 11 Bank interest rates loans to households with an original ...
## 45 55 Bank interest rates extended credit card credit to corpor...
## 46 54 Bank interest rates revolving loans overdrafts to househo...
## 47 18 Bank interest rates loans to households for house purchas...
## 48 21 Bank interest rates loans to households for consumption o...
## 49 56 Poland Annualised agreed rate AAR Narrowly defined effect...
## 50 16 Poland Annualised agreed rate AAR Narrowly defined effect...
## 51 52 Poland Annualised agreed rate AAR Narrowly defined effect...
## 52 19 Poland Annualised agreed rate AAR Narrowly defined effect...
## 53 25 Bank interest rates loans to households renegotiations Po...
## 54 62 Poland Annualised agreed rate AAR Narrowly defined effect...
## 55 17 Poland Annualised agreed rate AAR Narrowly defined effect...
## 56 40 Bank interest rates loans to households for consumption w...
## 57 42 Bank interest rates loans to households for consumption w...
## 58 57 Bank interest rates overnight deposits from corporations ...
## 59 20 Bank interest rates loans to households for consumption o...
## 60 41 Poland Annualised agreed rate AAR Narrowly defined effect...
## 61 37 Poland Annual percentage rate of charge APRC Credit and o...
## 62 58 Poland Annualised agreed rate AAR Narrowly defined effect...
```

```
heatmap <- dat.ir.4 %>%
  .[1:20, ] %>%
  select(id, t_a.1, R2) %>%
  pivot_longer(
    cols = c(t_a.1, R2),
    names_to = "stat",
    values_to = "value"
  ) %>%
  mutate(
    signif = case_when(
      stat == "t_a.1" & abs(value) >= 2.576 ~ "***",
      stat == "t_a.1" & abs(value) >= 1.960 ~ "**",
      stat == "t_a.1" & abs(value) >= 1.645 ~ "*",
      TRUE ~ ""
    )
  )
```

Następnie utworzono wykres ggplot z użyciem funkcji facet_wrap.

```
heatmap %>%
  ggplot(
    aes(x = stat, y = fct_rev(factor(id)), fill = value)
  ) +
  geom_tile(color = "white") +
  geom_text(aes(label = signif), color = "black", size = 7, fontface =
  ↪  "bold") +
  scale_fill_gradient(
    low = "white", high = "black",
    name = "Wartość"
  ) +
  labs(
    x = "Statystyka",
    y = "Zmienna objaśniana",
    title = "Mapa ciepła dla statystyk regresji (t_a.1 i R2)",
    caption = "Oznaczenia istotności tylko dla t_a.1: * p < 0.10, **
    ↪  p < 0.05, *** p < 0.01"
  ) +
  facet_wrap(~ stat, scales = "free_x") +
  theme_minimal() +
  theme(axis.text.y = element_text(size = 10))
```



Rysunek 3.28. Mapa ciepła dla statystyk regresji

W kolejnym kroku procedury utworzono nową ramkę danych `dat.ir.5`, która zawiera tylko te modele, dla których $JB > 0.05$.

```
dat.ir.5 <- dat.ir.4 %>%
  subset(JB > 0.05) %>%
  arrange(a.1)
```

Kolumna `JB` zawiera *p*-wartości z testu Jarque'a–Bery, który sprawdza, czy reszty regresji mają rozkład normalny. Wyniki są posortowane rosnąco według współczynnika `a.1` przy zmiennej niezależnej `wib3m`.

Następnie z ramki `dat.ir.5`, w której występują tylko modele z $JB > 0.05$, pobrano indeks modelu dla największej *p*-wartości statystyki JB za pomocą funkcji `which.max`. Ten indeks posłużył do pobrania z listy `dat.ir.3` pełnego zestawu danych modelu:

- obiektu `lm`,
- daty,
- podsumowania wyników.

```
reg.ir.0 <- dat.ir.5 %>%
  .$JB %>%
  which.max()
reg.ir.1 <- dat.ir.5$id[reg.ir.0] %>%
  dat.ir.3[[.]]
```

Poniższy kod wypisuje nazwę zmiennej objaśnianej z modelu `reg.ir.1`.

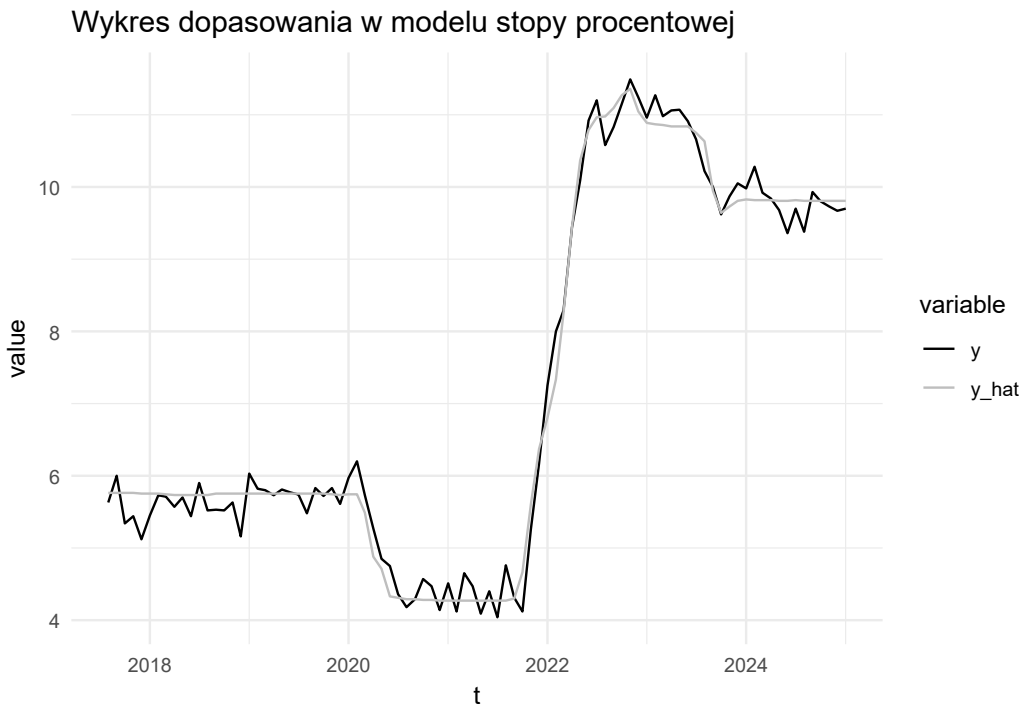
```
cat(paste("Zmienna\n", reg.ir.1[[3]]$name))
```

```
## Zmienna
##  Bank interest rates loans to households for other purposes new
 ↪  business Poland Annualised agreed rate AAR Narrowly defined
 ↪  effective rate NDER Credit and other institutions MFI except MMFs
 ↪  and central banks reporting sector Other lending excluding
 ↪  revolving loans and overdrafts convenience and extended credit
 ↪  card debt Total initial rate fixation New business coverage
 ↪  Households and non profit institutions serving households S 14 and
 ↪  S 15 sector denominated in Polish zloty
```

W ostatnim kroku procedury utworzono wykres liniowy (rys. 3.29) dla obiektu `reg.ir.1` porównujący w czasie (w próbie statystycznej od 2017-08-01 do 2025-01-01):

- rzeczywiste wartości zmiennej objaśnianej `y`,
- wartości dopasowane przez model `y_hat`.

```
data.frame(
  t     = reg.ir.1[[2]],
  y     = reg.ir.1[[1]]$model$y,
  y_hat = reg.ir.1[[1]]$fitted.values
) %>%
  melt(id.vars = c("t")) %>%
  ggplot(aes(x = t, y = value, color = variable)) +
  geom_line() +
  scale_color_manual(
    values = c("y" = "black", "y_hat" = "grey")
  ) +
  labs(
    title = "Wykres dopasowania w modelu stopy procentowej"
  ) +
  theme_minimal()
```



Rysunek 3.29. Wykres dopasowania w modelu stopy procentowej

Na rys. 3.29 można zauważyć, jak zmienna objaśniana zmieniała się w czasie w porównaniu do jej wartości szacowanych przez model ze stopą WIBOR 3M. Można także zaobserwować:

- zbieżność linii: jeśli linie y (rzeczywista) i y_hat (dopasowana) są blisko siebie, oznacza to dobre dopasowanie modelu w danym okresie.
- odchylenia: widoczne różnice między y a y_hat mogą wskazywać na okresy, w których model mniej dokładnie oddaje rzeczywistość.
- trend i dynamikę: czy model dobrze odwzorowuje ogólny trend (rosnący, malejący) zmiennej.

Rysunek 3.29 pozwala ocenić jakość *najlepszego* modelu dla stopy procentowej (ze zbioru stóp *ECB MIR: MFI Interest Rate Statistics*):

```
cat(paste("Zmienna\n", reg.ir.1[[3]]$name))
```

```
## Zmienna
##  Bank interest rates loans to households for other purposes new
 ↪  business Poland Annualised agreed rate AAR Narrowly defined
 ↪  effective rate NDER Credit and other institutions MFI except MMFs
 ↪  and central banks reporting sector Other lending excluding
 ↪  revolving loans and overdrafts convenience and extended credit
 ↪  card debt Total initial rate fixation New business coverage
 ↪  Households and non profit institutions serving households S 14 and
 ↪  S 15 sector denominated in Polish zloty
```

tj. jego dopasowanie w czasie oraz zidentyfikować okresy, w których stopa WIBOR 3M miała silny lub słaby związek z objaśnianą stopą procentową.

Powyższe rozważania dotyczące mechanizmu transmisji zmian stóp procentowych tworzą tło dla analizy kolejnego kluczowego obszaru polityki pieniężnej – inflacji. W następnym podrozdziale zostaną przedstawione główne podejścia teoretyczne i ekonometryczne wykorzystywane do badania procesów inflacyjnych. Zostaną omówione najważniejsze kierunki badań nad inflacją w Polsce, co pozwoli lepiej osadzić dalsze analizy w kontekście doświadczeń krajowych.

3.3.3.4. Model inflacji

Modelowanie inflacji stanowi jedno z kluczowych zagadnień współczesnej makroekonomii, ponieważ umożliwia nie tylko diagnozowanie mechanizmów cenotwórczych, ale również prognozowanie zmian inflacyjnych w średnim i długim okresie (Blanchard i Johnson, 2012). W praktyce badawczej często stosuje się modele ekonometryczne, w szczególności modele autoregresyjne (ARIMA) oraz wektorowej

autoregresji (VAR), które pozwalają uchwycić dynamikę procesów inflacyjnych (Stock i Watson, 2001). Istotne znaczenie przypisuje się również nowoczesnym wersjom krzywej Phillipsa, w których inflacja zależy od oczekiwań podmiotów gospodarczych oraz od luki popytowej w gospodarce (Gordon, 1997). Podejście alternatywne stanowią modele równowagi ogólnej DSGE, integrujące mikroekonomiczne podstawy zachowań podmiotów gospodarczych (agentów) z uwarunkowaniami polityki pieniężnej (Smets i Wouters, 2007). W kontekście globalnych szoków podażowych oraz niestabilności cen surowców coraz większą rolę w modelowaniu inflacji odgrywają czynniki egzogeniczne oraz modele strukturalne o wysokim stopniu złożoności (Forbes i in., 2021).

Badania nad inflacją w Polsce rozwijały się w wielu kierunkach. Początkowo koncentrowano się na analizach teoretycznych i klasycznych ujęciach ekonometrycznych, a następnie stopniowo przechodzono do modeli z kointegracją zmiennych, strukturalnych modeli wielorównaniowych, a w dalszej kolejności do nowoczesnych modeli DSGE i badań nad oczekiwaniami inflacyjnymi. Niezmiennym celem było pogłębienie wiedzy o mechanizmach kształtujących inflację oraz dostarczenie narzędzi wspierających politykę pieniężną.

Dynamiczne modele równowagi ogólnej (DSGE) stanowią szczególnie ważny krok w kierunku nowoczesnej analizy prognostycznej. Wykazano, że nawet w niewielkiej skali mogą one dostarczać prognoz zbliżonych jakością do wyników modeli VAR, a w przypadku dynamiki PKB – niekiedy nawet dokładniejszych (Rubaszek i Skrzypczyński, 2008). Z czasem udowodniono również, że nowokeynesowskie modele DSGE dobrze sprawdzają się w prognozowaniu średnio- i długookresowym, przewyższając w wielu przypadkach prognozy ekspertów (Kolasa i in., 2012). Dalsze badania pokazały, że uwzględnienie sztywności cen pozwala lepiej odwzorować krótkookresową dynamikę inflacji (Baranowski i in., 2016), a rozprawa doktorska Leszczyńska-Paczesna (2019) dowiodła praktycznego znaczenia inflacji bazowej w polityce pieniężnej poprzez zastosowanie wielosektorowego modelu DSGE gospodarki otwartej. Wreszcie, uwzględnienie w modelu DSGE długoterminowych stóp procentowych pozwoliło zidentyfikować ich stabilizującą rolę w małej gospodarce otwartej (Wesołowski, 2016).

Równolegle rozwijały się analizy kointegracyjne i modele wielorównaniowe, które skupiały się na długookresowych zależnościach między cenami, płacami i wydajnością pracy. Wykazano, że koszty pracy odgrywały główną rolę w kształtowaniu inflacji w okresie transformacji (Welfe i Majsterek, 2001, 2002), a dalsze badania potwierdziły znaczenie szoków produkcyjnych jako źródła trwałych trendów inflacyjnych (Kelm i Majsterek, 2005). Modele kointegracyjne umożliwiły także identyfikację relacji długookresowych w gospodarce, co pokazano w opracowaniu Welfe i in. (2004). Uzupełnieniem tych badań były analizy symulacyjne cen i płac, pozwalające lepiej zrozumieć wpływ kursu walutowego i stóp procentowych na gospodarkę (Wdowiński, 2002).

Kolejny nurt dotyczył rozwijania metod prognozowania inflacji. Badania wskazały na przewagę modeli czynnikowych nad tradycyjnymi modelami VAR i AR (Kotłowski,

2008) oraz na skuteczność bayesowskich modeli VAR uwzględniających sezonowość (Stelmasiak i Szafrański, 2016). Jednocześnie podkreślano, że żaden model nie jest w pełni doskonały i konieczne jest równoległe stosowanie różnych metod prognostycznych (Wójcik, 2015). Alternatywą okazały się podejścia oparte na tablicach przepływów międzygałęziowych, które w nowy sposób opisywały procesy inflacyjne (Przybyliński i Gorzałczyński, 2016). Wspomagająco działały syntetyczne wskaźniki wyprzedzające, pod warunkiem ich bieżącej aktualizacji (Szafrański, 2013), a wpływ szoków energetycznych analizowano za pomocą modeli równowagi ogólnej (Gradzewicz i in., 2024) oraz bayesowskich modeli VAR (Szafranek i in., 2024), dokumentując ich silne efekty proinflacyjne.

Szczególne miejsce w literaturze zajmują badania nad oczekiwaniami inflacyjnymi. Wykazano, że komunikacja banku centralnego istotnie kształtuje prognozy analityków (Baranowski i in., 2021), a poprawa jakości badań ankietowych jest konieczna dla skuteczniejszego wykorzystania danych w polityce pieniężnej (Janecki, 2024). Analizy wykazały także, że oczekiwania przedsiębiorstw odgrywają większą rolę w modelach nowokeynesowskich niż oczekiwania konsumentów czy analityków finansowych (Łyziak, 2016a, 2016b), a ich znaczenie potwierdzają badania krzywej Phillipsa, gdzie decydująca okazała się krajowa luka popytowa (Łyziak, 2019).

Istotnym uzupełnieniem wymienionych opracowań były badania nad pomiarem inflacji. Wykazano, że indeks CPI w Polsce raczej zaniża niż zawyża poziom inflacji, szczególnie w okresach gwałtownych zmian struktury konsumpcji, takich jak podczas pandemii (Hałka i Leszczyńska-Paczesna, 2023). Dyskusja nad rolą inflacji bazowej w polityce pieniężnej znalazła rozwinięcie w badaniach teoretycznych i empirycznych w opracowaniu Leszczyńska-Paczesna (2019).

Ważnym kierunkiem badań była też analiza inflacji w szerszym kontekście polityki gospodarczej. Wskazywano na potrzebę stopniowej konwergencji monetarnej w procesie wchodzenia do strefy euro (Orlowski, 2001), analizowano efekty polityki fiskalnej i pieniężnej w okresie transformacji (Milo i in., 2001), a także badano zależność między inflacją a kursem walutowym w kontekście globalizacji i efektu Balassy–Samuelsona (Kelm, 2010). Z kolei inflacja importowana i powiązania płacowo-cenowe okazały się kluczowe w objaśnianiu ostatnich wahań cen (Kelm i Sobiech Pellegrini, 2023). W najnowszych pracach zwrócono też uwagę na wpływ czynników politycznych i kryzysowych, które w latach 2018–2023 silnie kształtowały wzrost i inflację w Polsce (Bukowski, 2025).

Obok podejść nowoczesnych warto odnotować badania klasyczne. Hipoteza o optymalnej stopie inflacji wskazywała, że sprzyja ona długookresowemu wzrostowi, gdy kształtuje się na poziomie 3–4% (Baranowski, 2008). Rozważano także neutralność pieniądza (Brzoza-Brzezina i in., 2002) oraz związki między realnymi stopami procentowymi a inflacją, wykorzystując model *P-star* (Brzoza-Brzezina, 2002). Równolegle Welfe (2000) zaproponował dwa agregatowe modele inflacji dla Polski, które akcentowały zarówno krótkookresowe punkty zwrotne, jak i relacje długookresowe.

Ostatnim wątkiem są badania sektorowe i cząstkowe. Analizy wskazały, że niska inflacja w Polsce wynikała w dużej mierze z czynników popytowych i globalnych szoków surowcowych (Szafranek i Hałka, 2017). Wykazano także zróżnicowanie uporczywości inflacji między poszczególnymi sektorami (Hertel i Leszczyńska, 2013) oraz przydatność modeli ze zmiennymi ukrytymi do opisu inflacji z perspektywy bayesowskiej (Kwiatkowski, 2019).

Podsumowując, literatura pokazuje, że badania nad inflacją w Polsce rozwijały się wielotorowo i ewoluowały od klasycznych teorii ku nowoczesnym metodom prognostycznym. Modele DSGE okazały się wartościowym narzędziem, analizy kointegracyjne pogłębiły wiedzę o długookresowych zależnościach, a rosnące znaczenie oczekiwań inflacyjnych i jakości pomiaru cen podniosło skuteczność polityki pieniężnej. Całość tych badań dowodzi, że procesy inflacyjne w Polsce są złożone i wymagają interdyscyplinarnego podejścia, łączącego różne źródła danych i metody analizy.

W dalszej części podrozdziału oszacowano model inflacji dla indeksu HICP z jedną zmienną objaśniającą w postaci:

$$y_t = \alpha_0 + \alpha_1 X_{t-h} + \epsilon_t \tag{3.91}$$

gdzie:

y_t – indeks cen HICP (tempo wzrostu w proc.),
X_{t-h} – indeks cen kategorii wydatków konsumpcyjnych według COICOP, opóźniony o h okresów (tempo wzrostu w proc.),
α_0 i α_1 – parametry modelu,
ϵ_t – składnik losowy, $\epsilon_t \sim N(0, \sigma_\epsilon^2)$,
h – horyzont prognozy (od 1 do 12).

Indeks cen HICP to zharmonizowany indeks cen konsumpcyjnych, czyli wskaźnik inflacji opracowany przez Eurostat w celu umożliwienia porównywania poziomu i dynamiki cen konsumpcyjnych między krajami Unii Europejskiej oraz strefy euro. COICOP (*Classification of Individual Consumption by Purpose*) to Międzynarodowa Klasyfikacja Spożycia Indywidualnego według Celu, stosowana m.in. przez Eurostat, ONZ i krajowe urzędy statystyczne.

Poniżej podano kod procedury. Na początku zostały wczytane odpowiednie biblioteki[8].

[8] Warto zwrócić uwagę na rosnące znaczenie danych skanowanych (skanerowych) w modelowaniu inflacji, ponieważ umożliwiają one bardziej szczegółowe i bieżące monitorowanie zmian cen w gospodarce. Biblioteka R `PriceIndices` została opracowana w celu ułatwienia analizy takich danych poprzez obliczanie bilateralnych i multilateralnych indeksów cen oraz ich rozszerzeń. Dane skanowane wymagają wcześniejszego oczyszczenia, ujednolicenia nazw produktów, klasyfikacji i agregacji, co często wiąże się z tworzeniem specyficznych narzędzi. Dzięki szerokim możliwościom parametryzacji biblioteka `PriceIndices` jest przydatna zarówno dla praktyków, jak i naukowców, umożliwiając kompleksowe badanie dynamiki cen w oparciu o różnorodne metody indeksacji (Białek, 2022).

```
library(broom)
library(cowplot)
library(dplyr)
library(eurostat)
library(ggplot2)
library(gridExtra)
library(lmtest)
library(lubridate)
library(magrittr)
library(reshape2)
library(tidyr)
library(tidyverse)
library(tseries)
library(writexl)
library(xts)
```

W przedstawionym fragmencie kodu języka R utworzono dwie zmienne tekstowe, które służą do dalszej pracy z danymi statystycznymi, w szczególności danymi dotyczącymi cen konsumpcyjnych. W pierwszej linii utworzono zmienną name i przypisano jej wartość prc_hicp_midx. W kontekście analiz ekonomicznych lub statystycznych taki identyfikator odnosi się do zbioru danych zawierającego miesięczny harmonizowany indeks cen konsumpcyjnych (HICP), który jest standardowym miernikiem inflacji wykorzystywanym m.in. przez Eurostat. W drugiej linii utworzono zmienną y z przypisaną wartością CP00. W systematyce klasyfikacji COICOP, stosowanej przez Eurostat, kod ten odnosi się do ogólnego wskaźnika cen dla całego koszyka towarów i usług konsumpcyjnych.

```
name <- "prc_hicp_midx"
y    <- "CP00"
```

Następnie utworzono zmienną file, która przechowuje ścieżkę do pliku CSV zawierającego dane. Sprawdzono, czy plik o tej ścieżce istnieje za pomocą funkcji file.exists. Jeśli plik nie istnieje, to zostaje utworzony obiekt data.0 poprzez pobranie danych z serwisu Eurostat. Wykorzystano do tego celu funkcję get_eurostat, przekazując nazwę zbioru name oraz parametr time_format = "date", który zapewnia, że daty zostaną zwrócone w formacie typu Date. Utworzono plik CSV o wskazanej ścieżce, zapisując do niego dane z obiektu data.0. Użyto funkcji write.csv, z parametrem row.names = FALSE, co zapobiega zapisywaniu nazw wierszy w pliku. W przeciwnym przypadku, tj. gdy plik istnieje, tworzy się obiekt data.0 przez wczytanie danych z pliku CSV przy pomocy funkcji read.csv. Na koniec kolumna TIME_PERIOD zostaje konwertowana na format daty za pomocą funkcji as.Date.

```
file <- "F:\\docs\\R\\b2\\ch3\\b2_ch3_data_0.csv"
if (!file.exists(file)) {
data.0 <- name %>%
  get_eurostat(time_format = "date")
  data.0 %>%
    write.csv(
      "F:\\docs\\R\\b2\\ch3\\b2_ch3_data_0.csv",
      row.names = FALSE
    )
} else {
  data.0 <- read.csv(file)
  data.0$TIME_PERIOD <- as.Date(data.0$TIME_PERIOD)
}
```

Następnie utworzono obiekt `geo_label` poprzez wywołanie funkcji `get_eurostat_dic`. Funkcja ta pobiera słownik dla zmiennej `geo`, oznaczającej jednostki geograficzne (np. kraje, regiony). Utworzono również obiekt `coicop_label` zawierający słownik etykiet dla zmiennej `coicop`, która klasyfikuje wydatki konsumpcyjne według kategorii COICOP. Na koniec utworzono obiekt `unit_label`, który zawiera słownik jednostek miary (np. indeksy, procenty, euro) używanych w danych Eurostat.

```
geo.label <- get_eurostat_dic("geo")
coicop.label <- get_eurostat_dic("coicop")
unit.label <- get_eurostat_dic("unit")
```

W poniższym kodzie utworzono obiekt `data.1` na podstawie przekształcenia obiektu `data.0` za pomocą łańcucha operacji z użyciem operatora potokowego `%>%` i funkcji `left_join`. Kolejno dodano do zbioru `data.0` słowniki: `geo.label`, `coicop.label` i `unit.label`. Na koniec utworzono podzbiór danych przy użyciu funkcji `filter`, wybierając tylko te obserwacje, dla których:

- kod geograficzny (`geo`) jest równy `"PL"` (Polska),
- kod jednostki (`unit`) jest równy `"I15"` (np. indeks z okresem bazowym = 2015).

```
data.1 <- data.0 %>%
  left_join(geo.label, by = c("geo" = "code_name")) %>%
  rename(geo.label = full_name) %>%
  left_join(coicop.label, by = c("coicop" = "code_name")) %>%
  rename(coicop.label = full_name) %>%
  left_join(unit.label, by = c("unit" = "code_name")) %>%
  rename(unit.label = full_name) %>%
  filter(geo == "PL" & unit == "I15")
```

Następnie na podstawie zbioru `data.1` utworzono obiekt `data.2` poprzez zastosowanie funkcji `select`. Pominięto wybrane kolumny:

- geo (kod jednostki geograficznej),
- geo.label (opis jednostki geograficznej),
- unit (kod jednostki miary),
- unit.label (opis jednostki miary),
- coicop.label (opis kategorii COICOP).

W wyniku tej operacji utworzono uproszczony zbiór danych, który zawiera jedynie te kolumny, które są niezbędne do dalszej analizy.

```
data.2 <- data.1 %>%
  select(
    -geo,
    -geo.label,
    -unit,
    -unit.label,
    -coicop.label
  )
```

Sprawdzono, czy unikalna wartość w kolumnie freq w zbiorze data.2 jest równa "M", co oznacza miesięczną częstotliwość danych. Jeśli warunek ten jest spełniony, to tworzy się obiekt data.3 na podstawie obiektu data.2 poprzez usunięcie kolumny freq.

```
if (unique(data.2$freq) == "M") {
  data.3 <- data.2 %>%
    select(-freq)
}
```

Utworzono zbiór danych data.4 z przestawionymi kolumnami, co ma na celu ułatwienie dalszej analizy poprzez uporządkowanie danych.

```
data.4 <- data.3 %>%
  .[, c(2, 1, 3)]
```

Następnie powstał obiekt data.5 na podstawie obiektu data.4, tj. lista ramek danych, poprzez podzielenie zbioru data.4 według wartości w kolumnie coicop. Wykorzystano do tego celu funkcję split z argumentem .$coicop, który oznacza, że podział następuje na podstawie kategorii COICOP. W wyniku tego działania powstała lista, w której każdy element odpowiada jednej kategorii wydatków konsumpcyjnych COICOP, tj. każdy element listy zawiera dane tylko dla danej kategorii.

```
data.5 <- data.4 %>%
  split(.$coicop)
```

W wyniku poniższej operacji utworzono zmienną `oldest.date`, która reprezentuje najwcześniejszą datę występującą w zbiorze danych, niezależnie od kategorii COICOP.

```
oldest.date <- min(
  unlist(
    lapply(
      data.5,
      function(x) min(x$TIME_PERIOD)
    )
  )
) %>%
  as.Date()
oldest.date
```

```
## [1] "1996-01-01"
```

Podobnie powstała zmienna `newest.date`, która zawiera najnowszą datę obserwacji występującą w całym zbiorze danych, niezależnie od kategorii COICOP.

```
newest.date <- max(
  unlist(
    lapply(
      data.5,
      function(x) max(x$TIME_PERIOD)
    )
  )
) %>%
  as.Date()
newest.date
```

```
## [1] "2024-09-01"
```

W następnym kodzie utworzono ramkę danych `dates`, która zawiera jedną kolumnę o nazwie `TIME_PERIOD` z wartościami dat w równych, miesięcznych odstępach, obejmujących pełny zakres dostępnych danych.

```
dates <- seq(
  oldest.date,
  newest.date,
  by = "month"
) %>%
  data.frame(TIME_PERIOD = .)
```

Następnie utworzono obiekt `data.6` jako listę ramek danych, otrzymaną w wyniku zastosowania funkcji `lapply` do obiektu (listy) `data.5`. W wyniku działania funkcji `left_join` utworzono dla każdej kategorii COICOP nową ramkę danych, w której zachowana jest pełna oś czasu (na podstawie `dates`), a dane oryginalne z `x` są dopasowane tylko tam, gdzie dostępne były obserwacje (w pozostałych wierszach pojawią się wartości NA). W efekcie powstał obiekt `data.6`, czyli lista ramek danych, które zawierają pełny zakres czasowy dla każdej kategorii COICOP.

```
data.6 <- lapply(
  data.5,
  function(x) {
    left_join(dates, x, by = "TIME_PERIOD")
  }
)
```

W kolejnym kroku powstał obiekt `data.7` jako lista ramek danych przekształconych z obiektu (elementów listy) `data.6` za pomocą funkcji `lapply`. Dla każdej ramki danych `x` na liście `data.6` utworzono:

- Zmienną `name`, która zawiera unikalną (i niepustą) wartość z kolumny `coicop`. Wartość ta reprezentuje kod kategorii COICOP, do której należy dana ramka danych.
- Nową nazwę kolumny – kolumna `"values"` została przemianowana na wartość zmiennej `name`, czyli na kod COICOP.
- Końcową wersję ramki danych poprzez usunięcie kolumny `coicop` (która staje się niepotrzebna po przeniesieniu jej wartości do nagłówka kolumny z danymi).

W rezultacie powstała lista `data.7`, w której każda ramka danych zawiera kolumnę:

- z datami (`TIME_PERIOD`),
- o nazwie odpowiadającej kodowi COICOP (np. `"CP0111"`), zawierającą wartości dla tej kategorii w czasie.

Obiekt nie zawiera już kolumny `coicop`, ponieważ jej informacja została przeniesiona do nagłówka kolumny z danymi.

```
data.7 <- lapply(
  data.6,
  function(x) {
    name <- unique(na.omit(x$coicop))
    names(x)[which(names(x) == "values")] <- name
    x %>%
      select(-coicop)
  }
)
```

Następnie utworzono obiekt `data.8` poprzez iteracyjne scalenie wszystkich ramek danych znajdujących się w obiekcie (liście) `data.7`. Wykorzystano funkcję `Reduce`, która stosuje operację łączenia (`full_join`) parami, począwszy od pierwszych dwóch ramek w liście `data.7`, a następnie kolejno dołączając każdą następną. Połączenia `full_join` zastosowano na podstawie wspólnej kolumny `TIME_PERIOD`. Dzięki temu zachowano wszystkie dostępne daty nawet wtedy, gdy niektóre kategorie COICOP nie zawierały obserwacji w danym miesiącu.

```
data.8 <- Reduce(function(x, y) {
  full_join(
    x, y, by = c(
      "TIME_PERIOD" = "TIME_PERIOD"
    )
  )
}, data.7)
```

W kolejnym kroku utworzono obiekt `data.9` jako obiekt klasy `xts`, na podstawie obiektu (korpusu danych) `data.8[, -1]` – czyli wszystkich kolumn z obiektu `data.8` z wyjątkiem pierwszej, zawierającej daty (`TIME_PERIOD`). Kolumny te zawierają wartości dla poszczególnych kategorii COICOP. Jako indeks czasu dla obiektu `xts` przyjęto pierwszą kolumnę `data.8[, 1]`, która stanowi oś czasu.

```
data.9 <- xts(data.8[, -1], order.by = data.8[, 1])
```

Następnie powstał obiekt `data.10` jako przekształcona wersja obiektu `data.9`, będącego szeregiem czasowym typu `xts`. Utworzono logarytmy naturalne wszystkich wartości w obiekcie `data.9` za pomocą funkcji `log`. Następnie utworzono pierwsze różnice zlogarytmowanych wartości przy użyciu funkcji `diff`, co odpowiada względnej zmianie (przyrostowi logarytmicznemu), będącej aproksymacją stopy wzrostu. W wyniku tych operacji powstał obiekt `data.10`, który zawiera miesięczne zmiany procentowe dla każdej kategorii COICOP, uporządkowane w strukturze szeregu czasowego `xts`.

```
data.10 <- data.9 %>%
  log() %>%
  diff(1) %>%
  {. * 100}
```

Utworzono obiekt (ramkę danych) `data.11`, konwertując obiekt `data.10` do standardowej struktury `data.frame` za pomocą funkcji `as.data.frame`. Utworzono nową kolumnę `date`, dodając ją na początku ramki danych przy użyciu funkcji `cbind`. Kolumna ta zawiera wartości pobrane z `rownames(.)`, czyli z nazw wierszy, które pierwotnie pełniły rolę indeksu czasowego w obiekcie `xts`.

```
data.11 <- data.10 %>%
  as.data.frame() %>%
  cbind(date = rownames(.), .)
```

Użyto funkcji `as.Date`, która przekształca wartości tekstowe (ciągi znaków) w obiekty klasy `Date`, co umożliwia ich dalsze wykorzystanie w analizach czasowych i wizualizacjach.

```
data.11$date <- as.Date(data.11$date)
```

Utworzono zaktualizowany zestaw nazw kolumn, który jest zgodny z konwencjami języka R dotyczącymi identyfikatorów (np. do użycia w modelowaniu, w funkcjach `ggplot`, `select` itp.). W rezultacie powstały jednolite, bezpieczne nazwy kolumn, co minimalizuje ryzyko błędów składniowych w dalszej analizie.

```
names(data.11) <- gsub("-", "_", names(data.11))
```

Następnie powstał obiekt `na.columns` jako wektor wartości logicznych, informujący, które kolumny w ramce danych `data.11` zawierają wyłącznie brakujące wartości `NA`. Wykorzystano funkcję `sapply`. W wyniku tych operacji utworzono wektor `na.columns`, w którym:

- nazwy elementów odpowiadają nazwom kolumn,
- wartości `TRUE` oznaczają kolumny zawierające wyłącznie brakujące dane,
- wartości `FALSE` oznaczają kolumny zawierające przynajmniej jedną niebrakującą obserwację.

```
na.columns <- sapply(data.11, function(x) all(is.na(x)))
```

Utworzono obiekt `data.12` jako nową wersję obiektu (ramki danych) `data.11`. Wykorzystano wcześniej utworzony wektor wartości logicznych `na.columns`, który wskazuje, które kolumny zawierają same wartości `NA`. W rezultacie utworzono obiekt `data.12`, który stanowi oczyszczoną wersję danych, pozbawioną całkowicie pustych (nieinformacyjnych) kolumn.

```
data.12 <- data.11 %>%
  .[, !na.columns]
```

Następnie powstał obiekt `obs`, który zawiera liczbę obserwacji (wierszy) w kolumnie `y` ramki danych `data.12`. Obiekt `s` reprezentuje 30% długości szeregu czasowego

y, zaokrąglone w górę do najbliższej liczby całkowitej (ceiling). Utworzono również obiekt obs_min, który reprezentuje sumę s oraz dodatkowych 30% obserwacji z obiektu obs, również zaokrągloną w górę. Odpowiada to minimalnej liczbie obserwacji wymaganych do rozpoczęcia testowania modelu. Obiekt var_x zawiera wektor nazw wszystkich kolumn w obiekcie data.12, z wyjątkiem kolumny date i zmiennej docelowej y. Na koniec utworzono obiekt h jako wektor liczb całkowitych od 1 do 12, który oznacza horyzont prognozy w miesiącach.

```
obs     <- length(data.12[, y])
s       <- ceiling(.3*obs)
obs.min <- ceiling((.3*obs)+s)
var.x   <- names(data.12)[
  -which(names(data.12) %in% c("date", y))
]
h <- 1:12
```

Funkcja tp służy do identyfikacji punktów zwrotnych w szeregu czasowym – tj. lokalnych minimów i maksimów. W wyniku działania tej funkcji utworzono wektor wartości logicznych (TRUE/FALSE), w którym:

- TRUE oznacza wystąpienie punktu zwrotnego (lokalnego minimum lub maksimum),
- FALSE oznacza brak punktu zwrotnego w danym indeksie.

```
tp <- function(z) {
  signs <- sign(diff(z))
  c(FALSE, (diff(signs) != 0), FALSE)
}
```

Poniższy kod tworzy listę modeli (oraz ich ocen) dla różnych horyzontów prognozy h = 1:12 i różnych zmiennych objaśniających (var_x) względem jednej zmiennej zależnej y, zawartej w ramce danych data.12. Kod jest złożony i wymaga szczegółowego omówienia.

1. Występują pętle zagnieżdżone lapply(h, ...) i lapply(var.x, ...):

- dla każdego horyzontu prognozy h (od 1 do 12),
- dla każdej zmiennej objaśniającej x z listy var.x,
- w celu oszacowania parametrów osobnych modeli.

2. Dla każdej kombinacji (h, x) powstaje tymczasowy zbiór danych df.0 zawierający kolumny: date, y, x.

3. Tworzy się ramkę `na_rows` z brakującymi obserwacjami NA, którą dołącza się na końcu `df.0`, by umożliwić prognozowanie do przodu o `h` kroków.

4. Powstaje kolumna `lagged.x`, zawierająca wartości zmiennej `x` opóźnione o `h` okresów.

5. Oblicza się liczbę kompletnych wierszy (po `na.omit`) jako `n`.

6. Jeśli liczba kompletnych wierszy jest większa lub równa `obs.min`, to:

- tworzy się wersję `df.1`, w której `y` zostaje ustawione na NA dla ostatnich `s+h` obserwacji (*out-of-sample*),
- dopasowuje się model liniowy `y ~ lagged.x`,
- oblicza się wskaźniki dopasowania i diagnostyki modelu:
  - BIC, testy: Jarque'a–Bery (normalność), Durbina–Watsona (autokorelacja), Breuscha–Godfrey'a (autokorelacja), White'a (heteroskedastyczność), R2 skorygowany,
- oblicza się błędy równania SEE (dla próbek `dev` i `oos`),
- wyznacza się prognozy:
  - `dev` – dopasowanie *in-sample*,
  - `oos` – prognoza *out-of-sample*,
  - `oot` – prognoza poza zakresem oryginalnych danych.

7. Tworzy się ramkę `df.4` zawierającą:

- wartości rzeczywiste (`y`),
- prognozy `dev`, `oos`, `oot` w odpowiednich miejscach (inne pola są ustawiane na NA),
- zaokrąglone wartości numeryczne do 6 miejsc po przecinku.

8. Wyznacza się ocenę trafności przewidywania punktów zwrotnych poprzez:

- wywołanie funkcji `tp` dla rzeczywistego i prognozowanego przebiegu (`dev` oraz `oos`),
- obliczenie odsetka zgodnych punktów zwrotnych: `tp.dev` i `tp.oos`.

9. Tworzy się listę z wynikami dla danego modelu zawierającą:

- model, formułę, zmienne `y` i `x`, statystyki dopasowania, błąd SEE, zgodność punktów zwrotnych, ramkę danych z prognozami (`df`), oraz horyzont prognozy `h`.

10. W przypadku błędu, ostrzeżenia lub komunikatu model nie jest zwracany (zastosowano funkcję `tryCatch` z `NULL` jako wynikiem domyślnym).

Jako końcowy rezultat powstał obiekt `model`, będący listą wielowymiarową. Zewnętrzna lista zawiera 12 elementów (dla każdego horyzontu prognozy `h`). Każdy element to lista modeli dopasowanych dla każdej zmiennej `x` z `var.x`. Każdy model zawiera pełen zestaw wyników analizy.

```
model <- lapply(h, function(h) {
  lapply(var.x, function(x) {

    tryCatch(
      expr = {
        df.0 <- data.12[, c("date", y, x)]

        na.rows <- as.data.frame(
          matrix(NA, nrow = h, ncol = ncol(df.0))
        )

        colnames(na.rows) <- colnames(df.0)

        df.0 <- df.0 %>%
          rbind(na.rows) %>%
          set_rownames(1:nrow(.))

        df.0$date <- df.0$date[1] + months(0:(nrow(df.0)-1))

        df.0$lagged.x <- lag(df.0[, x], h)

        n <- df.0 %>%
          na.omit %>%
          nrow

        if (n >= obs.min) {

          df.1 <- df.0

          df.1[(nrow(df.1)-h-s):nrow(df.1), y] <- NA

          df.1 <- df.1 %>%
            na.omit

          formula <- as.formula(paste0(y, " ~ lagged.x"))
          model <- lm(formula, data = df.1)
          bic <- BIC(model)
          jb  <- jarque.bera.test(model$residuals)$p.value
          dw  <- dwtest(model)$p.value
          bg  <- bgtest(model, order = 1)$p.value
```

```
          wh  <- bptest(model, ~ I(lagged.x^2), data = df.1)$p.value
          r2  <- 100*(summary(model)$adj.r.squared)

          actual <- df.1[, y] %>%
            as.vector

          dev <- model$fitted.values %>%
            as.vector() %>%
            cbind(df.1, dev = .) %>%
            select(date, dev)

          oos <- predict(model, newdata = df.0) %>%
            cbind(df.0, oos = .) %>%
            select(date, oos) %>%
            na.omit

          oot <- predict(model, newdata = df.0) %>%
            cbind(df.0, oot =  .) %>%
            select(date, oot) %>%
            na.omit

          df.2 <- list(df.0, dev, oos, oot) %>%
            reduce(
              left_join,
              by = "date"
            )

          df.3 <- df.2 %>%
            .[, c("date", y, "dev", "oos", "oot")]

          df.4 <- df.3 %>%
            mutate(
              oos = ifelse(!is.na(dev), NA, oos)
            ) %>%
            mutate(
              oot = ifelse(!is.na(.data[[y]]), NA, oot)
            ) %>%
            mutate(
              oos = ifelse(!is.na(oot), NA, oos)
            ) %>%
            mutate_if(
              is.numeric, round, 6
            )

          see.dev <- df.4 %>%
            {.[, y]-.[, "dev"]} %>%
            as.vector %>%
            na.omit %>%
            {sum(.^2)/(length(.)-1)} %>%
```

```
    sqrt

  see.oos <- df.4 %>%
    {.[, y]-.[, "oos"]} %>%
    as.vector %>%
    na.omit %>%
    {sum(.^2)/(length(.)-1)} %>%
    sqrt

  df.dev <- df.4 %>%
    select(date, .data[[y]], dev) %>%
    na.omit

  actual.tp.dev <- tp(df.dev[, y])
  predic.tp.dev <- tp(df.dev[, "dev"])

  match.tp.dev <- sum(
    actual.tp.dev == predic.tp.dev,
    na.rm = TRUE
  )

  tp.stat.dev <- 100*(match.tp.dev / length(actual.tp.dev))

  df.oos <- df.4 %>%
    select(date, .data[[y]], oos) %>%
    na.omit

  actual.tp.oos <- tp(df.oos[, y])
  predic.tp.oos <- tp(df.oos[, "oos"])

  match.tp.oos <- sum(
    actual.tp.oos == predic.tp.oos,
    na.rm = TRUE
  )

  tp.stat.oos <- 100*(match.tp.oos / length(actual.tp.oos))

  return(list(
    model = model,
    formula = formula,
    var.y = y,
    var.x = x,
    bic = bic,
    jb = jb,
    dw = dw,
    bg = bg,
    wh = wh,
    r2 = r2,
    see.dev = see.dev,
```

```
            see.oos = see.oos,
            tp.dev = tp.stat.dev,
            tp.oos = tp.stat.oos,
            df = df.4,
            h = h
          ))
        } else return(NULL)
      },
      error = function(e) {
        NULL
      },
      warning=function(w) {
        NULL
      },
      message = function(m) {
        NULL
      }
    )
  })
})
```

Następnie utworzono obiekt `valid.models`, który zawiera jedynie poprawnie dopasowane modele z listy `model`.

```
valid.models <- Filter(Negate(is.null), unlist(model, recursive =
 ↪  FALSE))
```

W kolejnym kroku utworzono obiekt `output.0` jako ramkę danych (`data.frame`), zawierającą podsumowanie statystyk dla wszystkich poprawnie dopasowanych modeli (`valid.models`). Zastosowano funkcję `sapply` do każdego elementu listy `valid.models`, aby wyodrębnić konkretne atrybuty modelu:

- `x` – nazwa zmiennej objaśniającej (predyktora),
- `bic` – wartość kryterium informacyjnego BIC,
- `jb` – *p*-wartość testu Jarque'a–Bery,
- `dw` – *p*-wartość testu Durbina–Watsona,
- `bg` – *p*-wartość testu Breuscha–Godfrey'a,
- `wh` – *p*-wartość testu White'a,
- `r2` – skorygowany współczynnik determinacji,
- `see.dev` – błąd SEE dla prognoz *in-sample*,
- `see.oos` – błąd SEE dla prognoz *out-of-sample*,
- `tp.dev` – zgodność punktów zwrotnych *in-sample* (w %),
- `tp.oos` – zgodność punktów zwrotnych *out-of-sample* (w %),
- `h` – horyzont prognozy.

Wszystkie kolumny poza x (zmienna tekstowa) i h (liczba całkowita) zostały zaokrąglone do dwóch miejsc po przecinku przy użyciu funkcji round. W efekcie utworzono ramkę danych output.0 będącą czytelnym zestawieniem dotyczącym jakości modeli.

```
output.0 <- data.frame(
  x = sapply(valid.models, function(x) x$var.x),
  bic = sapply(valid.models, function(x) x$bic),
  jb = sapply(valid.models, function(x) x$jb),
  dw = sapply(valid.models, function(x) x$dw),
  bg = sapply(valid.models, function(x) x$bg),
  wh = sapply(valid.models, function(x) x$wh),
  r2  = sapply(valid.models, function(x) x$r2),
  see.dev = sapply(valid.models, function(x) x$see.dev),
  see.oos = sapply(valid.models, function(x) x$see.oos),
  tp.dev = sapply(valid.models, function(x) x$tp.dev),
  tp.oos = sapply(valid.models, function(x) x$tp.oos),
  h = sapply(valid.models, function(x) x$h)
) %>%
  mutate(across(-c(x, h), function(x) round(x, digits = 2)))
```

Utworzono obiekt output.1 jako przefiltrowaną i uporządkowaną wersję ramki danych output.0, zawierającą wybrane modele spełniające określone kryteria jakości. Zastosowano funkcję filter, aby wybrać tylko te wiersze (modele), które spełniają warunek tp.oos >= tp.dev, tj. trafność przewidywania punktów zwrotnych poza próbą (tp.oos) jest co najmniej tak wysoka, jak trafność w próbie (tp.dev). Komentarz w kodzie (# e.g. jb > 0.05 &) sugeruje możliwość dalszego doprecyzowania warunków filtrowania, np. o testy diagnostyczne (normalność reszt, autokorelację itp.). Utworzono uporządkowany wynik końcowy, sortując dane według kolumny h, czyli horyzontu prognozy (rosnąco) przy użyciu funkcji arrange.

```
output.1 <- output.0 %>%
  filter(
    # filtrowanie wyników
    # e.g.
    # jb > 0.05 &
    (tp.oos >= tp.dev)
  ) %>%
  arrange(h)
```

Utworzono obiekt output.2 jako zbiór najlepszych modeli wybranych z ramki output.1, osobno dla każdego horyzontu prognozy h. Zastosowano funkcję group_by, która utworzyła grupy w danych według wartości horyzontu prognozy h (np. $h = 1, 2, \ldots, 12$). W każdej grupie utworzono wybór modelu o najwyższym skorygowanym współczynniku determinacji r2 za pomocą funkcji slice_max, tj. wybrano dokładnie jeden model (n = 1) z najwyższym r2 w każdej grupie.

Następnie utworzono posortowaną wersję zbioru modeli według r2 malejąco (arrange(desc(r2))), czyli od najlepszego dopasowania w dół.

```
output.2 <- output.1 %>%
  group_by(h) %>%
  slice_max(r2, n = 1) %>%
  arrange(desc(r2))
```

Utworzono obiekt id jako nową ramkę danych, zawierającą tylko wybrane kolumny x i h z ramki output.2. Kolumna x reprezentuje nazwy zmiennych objaśniających (predyktorów), które zostały wybrane jako najlepsze modele w danym horyzoncie prognozy. Kolumna h zawiera odpowiadające tym modelom horyzonty prognozy. W efekcie utworzono podzbiór id, który umożliwia szybkie wyszukiwanie modeli w zbiorze valid.models, np. do dalszej analizy, selekcji lub wizualizacji wyników.

```
id <- output.2[, c("x", "h")]
```

Utworzono obiekt final.model.0 jako wektor pozycji (indeksów) modeli na liście valid.models, które odpowiadają najlepszym kombinacjom zmiennej objaśniającej x i horyzontu prognozy h zapisanym w ramce id. Dla każdego wiersza x w ramce id (czyli dla każdej najlepszej pary (x, h)):

- utworzono wewnętrzną funkcję sapply, która sprawdza wszystkie modele w obiekcie valid.models,
- dla każdego modelu:
  - porównano y$var_x == id[x, "x"] oraz y$h == id[x, "h"],
  - zwrócono indeksy modeli spełniających oba te warunki (funkcja which()).

Zastosowano funkcję lapply do wygenerowania listy pozycji, a następnie funkcję unlist, aby przekształcić wynik w pojedynczy wektor.

```
final.model.0 <- lapply(
  1:nrow(id),
  function(x) {
    which(
      sapply(
        valid.models,
        function(y) y$var.x == id[x, "x"] & y$h == id[x, "h"]
      )
    )
  }
) %>%
  unlist
```

Utworzono obiekt `final.model.1` jako listę wykresów typu `ggplot`, po jednym dla każdego najlepszego modelu z listy `final.model.0`. Dla każdej pozycji `x` w `final.model.0` utworzono zbiór danych `df` pobrany z listy `valid.models[[x]]$df`. Zawiera on daty (`date`) oraz wartości:

- rzeczywiste obserwacje (`CP00`, czyli `y`),
- dopasowania *in-sample* (`dev`),
- prognozy *out-of-sample* (`oos`),
- prognozy *ex-ante* (`oot`).

Następnie utworzono przekształcenie danych do formatu stosu (*long format*) przy użyciu funkcji `melt` z biblioteki `reshape2`. Utworzono wykres liniowy `ggplot`, na którym:

- oś X to: `date`,
- oś Y to: `value`,
- kolor reprezentuje zmienną (`CP00`, `dev`, `oos`, `oot`).

Utworzono tytuł wykresu dynamicznie za pomocą funkcji `paste`. Zastosowano własną paletę kolorów dla poszczególnych zmiennych. Ustawiono formatowanie osi czasu:

- etykiety co 12 miesięcy (`date_breaks = "12 months"`),
- format daty: rok-miesiąc-dzień (`%Y-%m-%d`).

```
final.model.1 <- lapply(
  final.model.0,
  function(x) valid.models[[x]]$df %>%
    melt(
      id.vars = "date",
      variable.name = "variable",
      value.name = "value"
    ) %>%
    ggplot(aes(x = date, y = value, color = variable, linetype =
     ↪  variable)) +
    geom_line() +
    labs(
      title = paste("Najlepszy model (", valid.models[[x]]$var.x, ")",
       ↪  sep = ""),
```

```
      y = "CP00",
      x = "Time"
    ) +
    scale_color_manual(
      values = c(
        "CP00" = "black",
        "dev" = "grey80",
        "oos" = "grey60",
        "oot" = "grey40"
      )
    ) +
    scale_linetype_manual(
      values = c(
        "CP00" = "solid",
        "dev" = "dashed",
        "oos" = "dashed",
        "oot" = "dashed"
      )
    ) +
    scale_x_date(
      date_breaks = "12 months",
      date_labels = "%Y-%m-%d"
    ) +
    theme_minimal() +
    theme(
      legend.position = "top",
      axis.text.x = element_text(angle = 90, hjust = 1)
    )
)
```

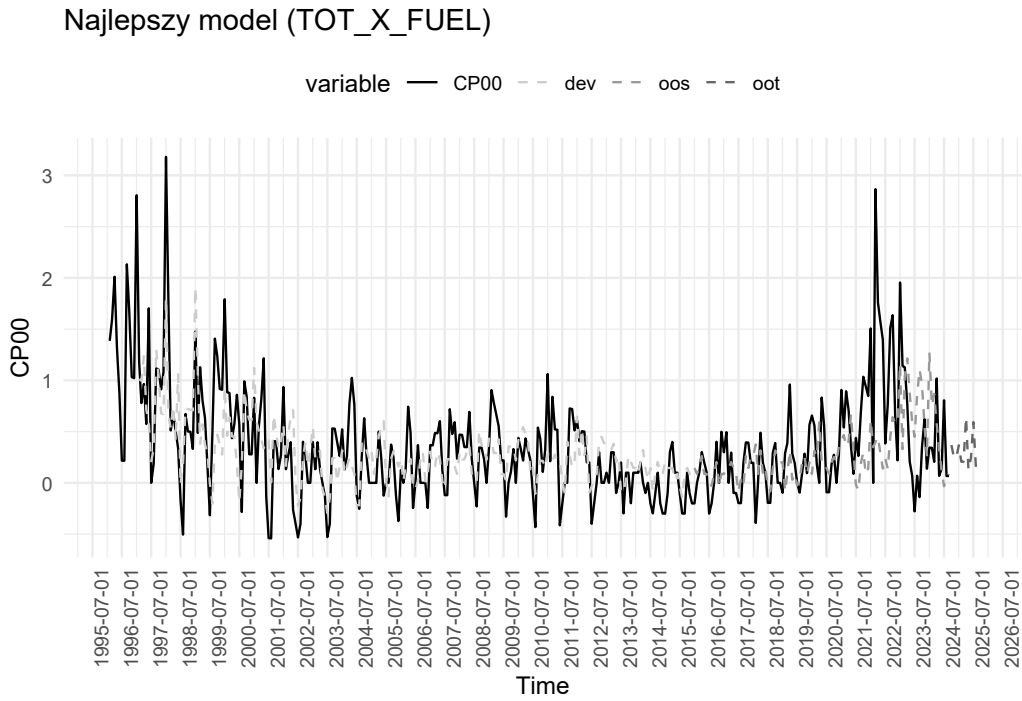
Utworzono obiekt `rows`, który zawiera liczbę wykresów zgromadzonych na liście `final.model.1`. Odpowiada to liczbie najlepszych modeli, dla których utworzono wykresy. Utworzono macierz `layout`, która definiuje układ wykresów na stronie a następnie wielostronicowy obiekt graficzny przy użyciu funkcji `marrangeGrob` z biblioteki `gridExtra`[9].

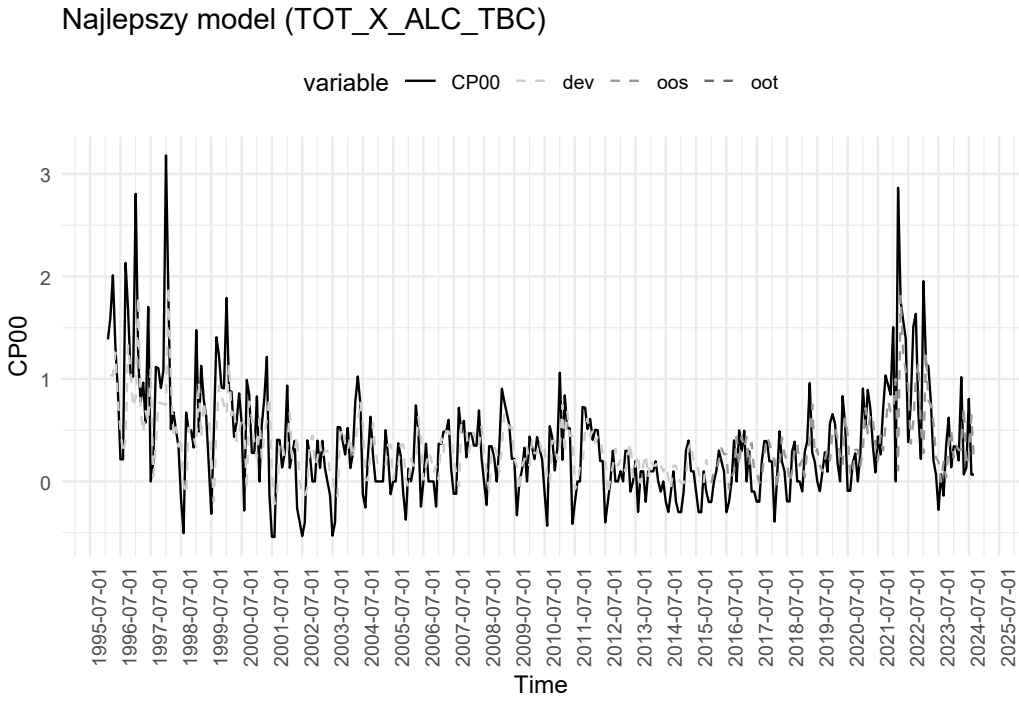
```
rows <- length(final.model.1)
layout <- matrix(
  rep(1:rows, each = 2), ncol = 1
)
final.model.1 %>%
  marrangeGrob(
    ncol = 1,
    nrow = length(.),
    layout_matrix = layout, top = NULL
  )
```

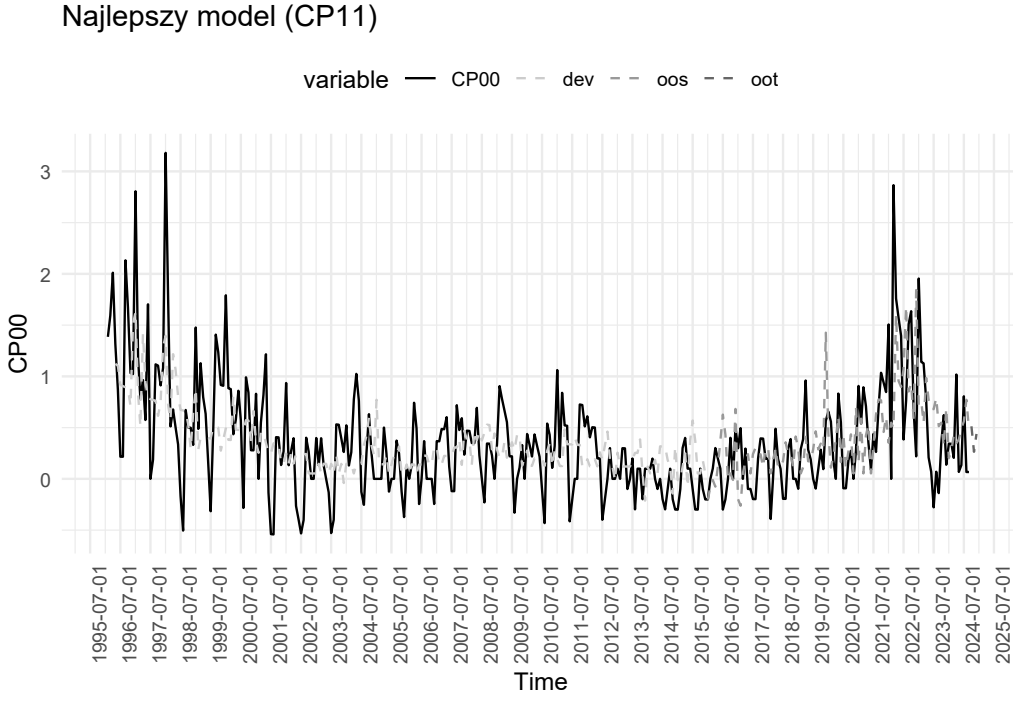
[9] Ze względu na rozmiar rysunku nie został on uwzględniony w wydruku.

Alternatywnie utworzono pętlę `for`, która iteruje po wszystkich elementach listy `final.model.1`. Funkcja `print` skutkuje wygenerowaniem i wyświetleniem kolejno wszystkich wykresów pt. `Najlepszy model` dla poszczególnych kategorii cen. To podejście jest przydatne w środowiskach interaktywnych (np. `RStudio`), gdzie każdy wykres zostanie automatycznie wygenerowany jeden po drugim. Wykresy pokazane poniżej przedstawiają dopasowanie wartości teoretycznych w próbkach `dev`, `oos` i `oot` do wartości empirycznych zmiennej objaśnianej `CP00`.

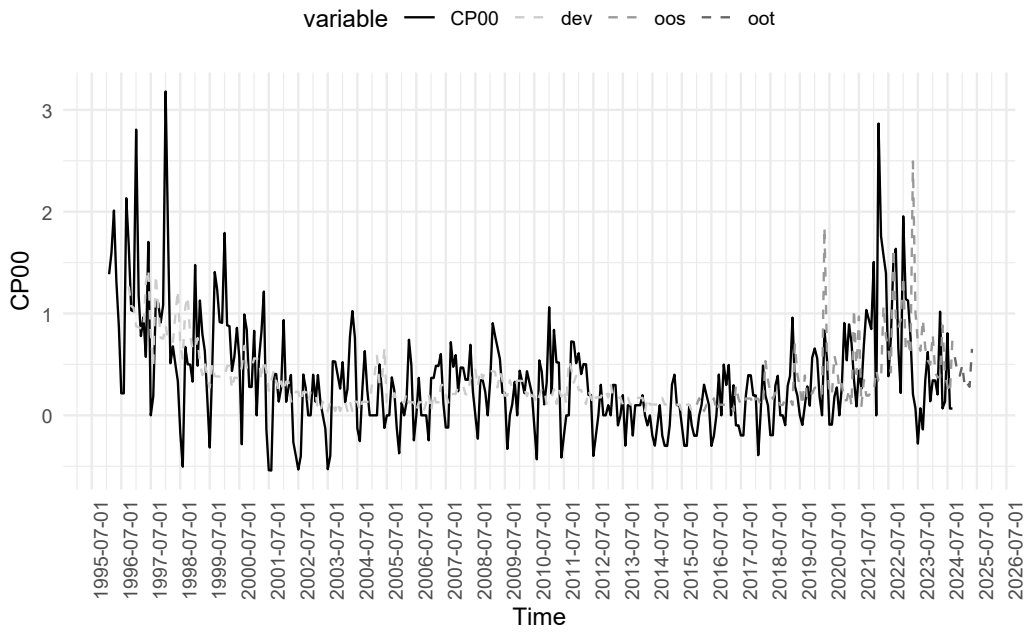
```
for (plot in final.model.1) {
  print(plot)
  knitr::opts_chunk$set(fig.cap="Najlepszy model")
}
```



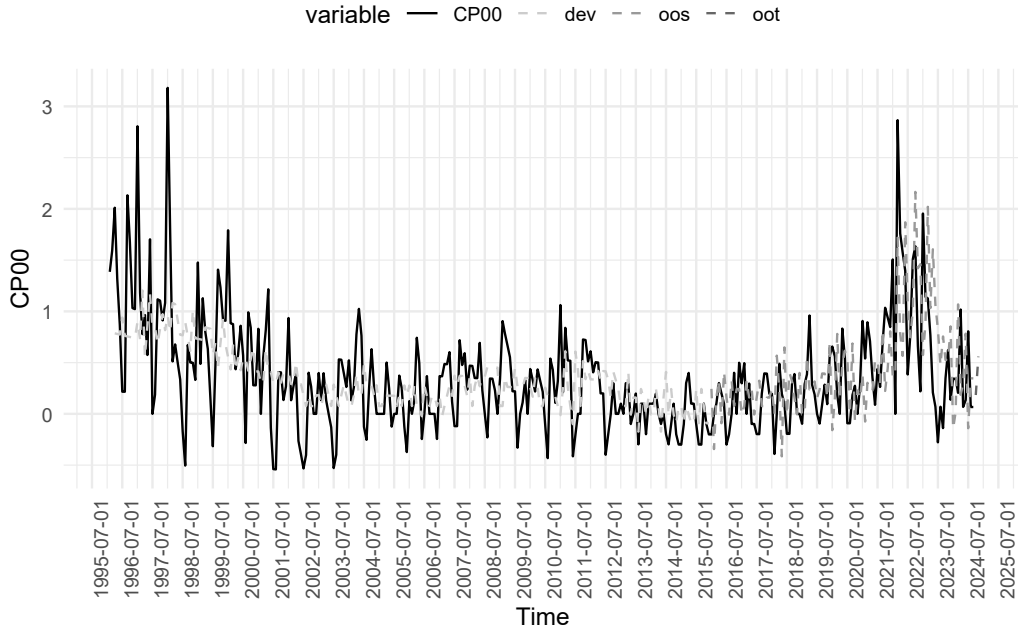

Najlepszy model (TOT_X_ALC_TBC)
variable
CP00
dev
oos
oot
CP00
Time

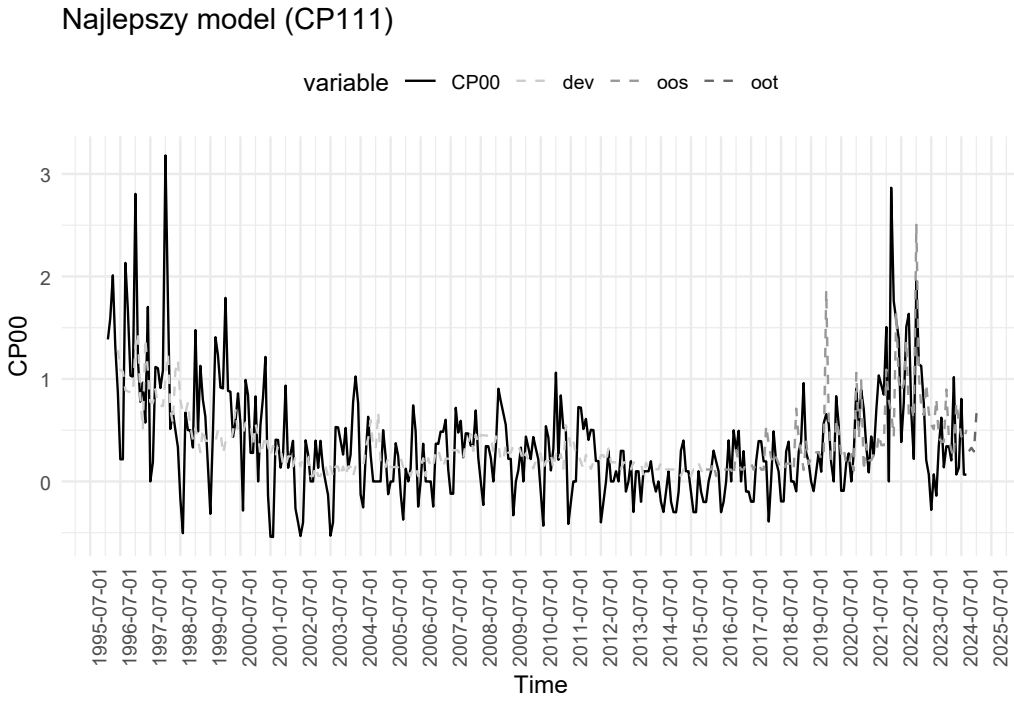

Najlepszy model (CP11)
variable
CP00
dev
oos
oot
CP00
Time

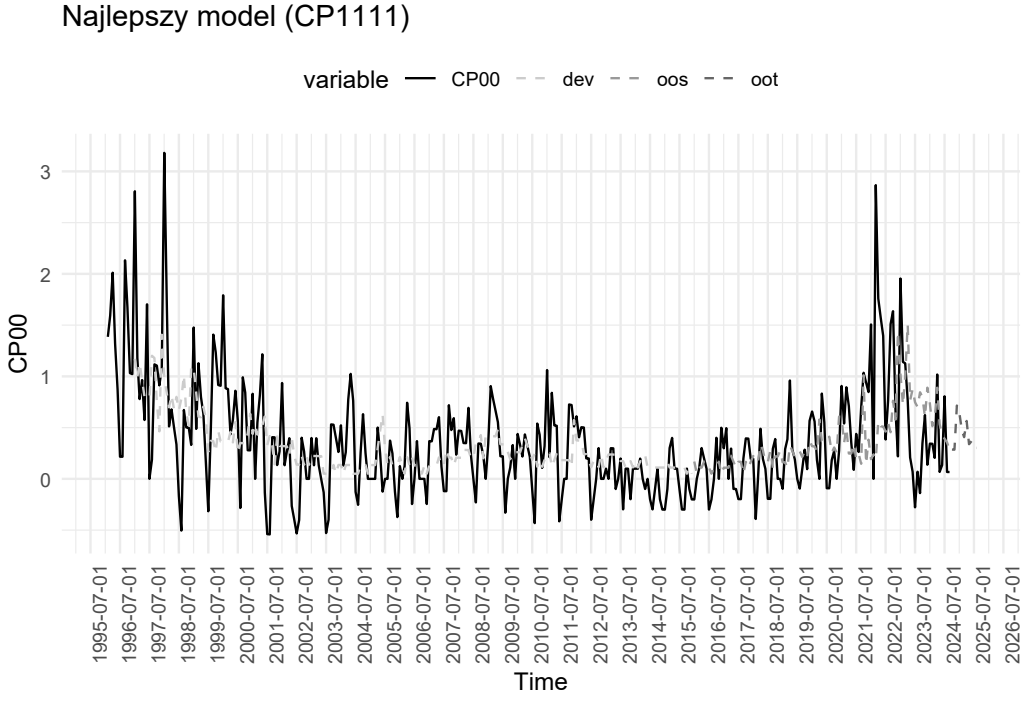
Najlepszy model (CP111)

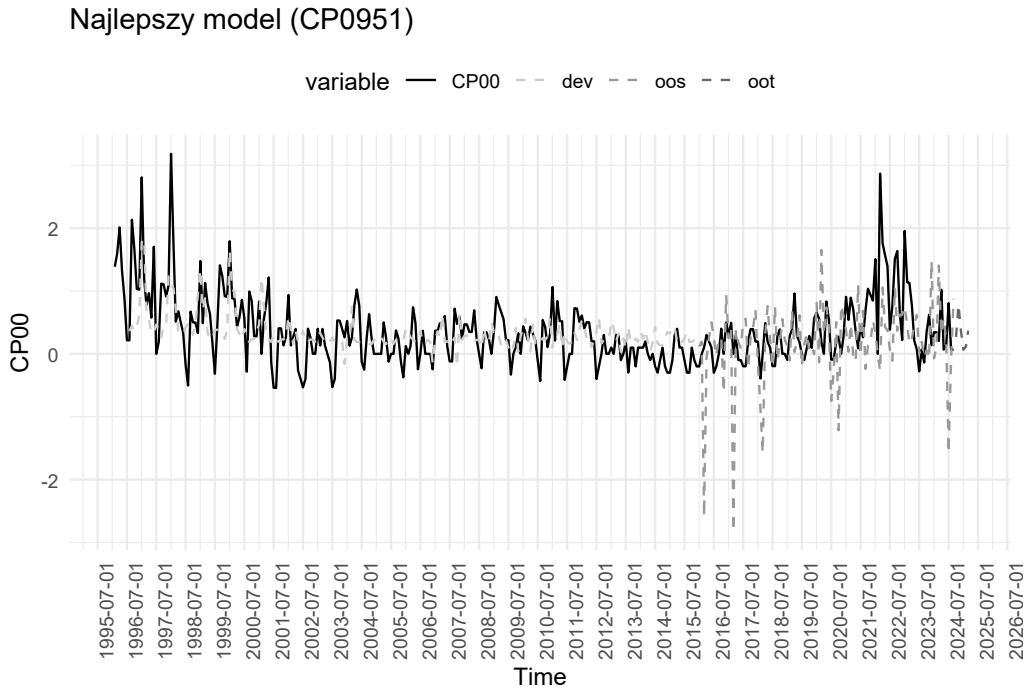


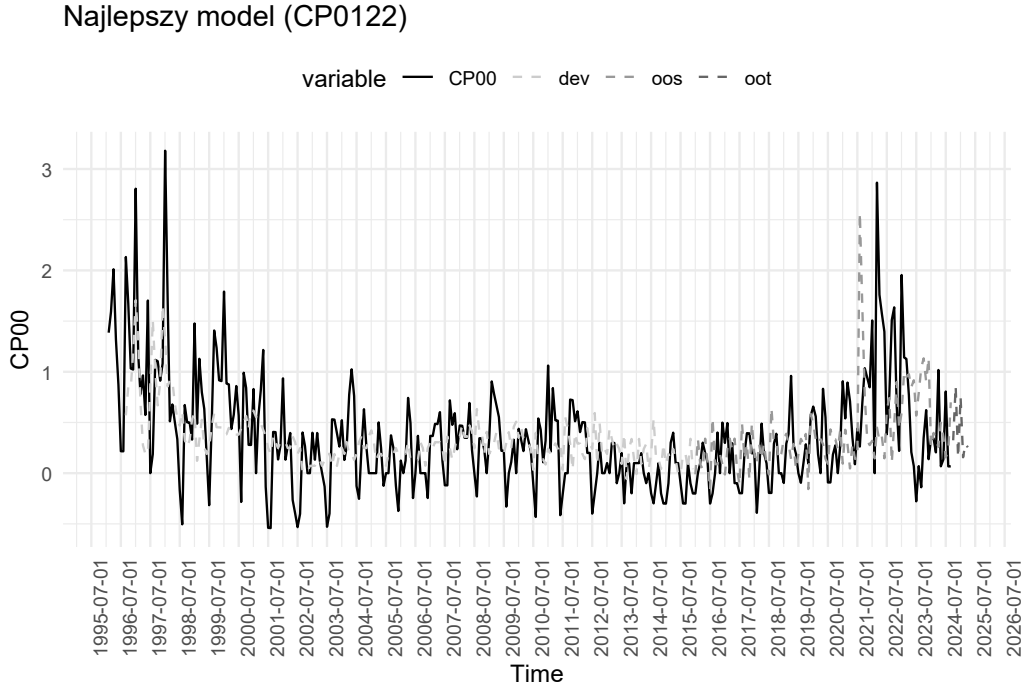
Najlepszy model (CP056)

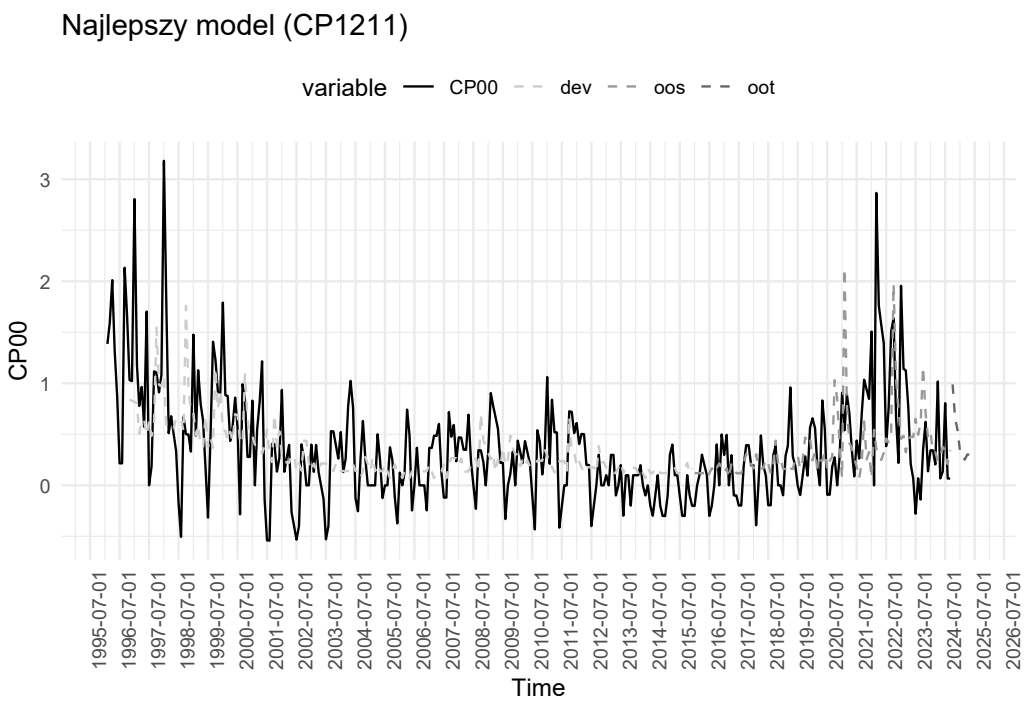


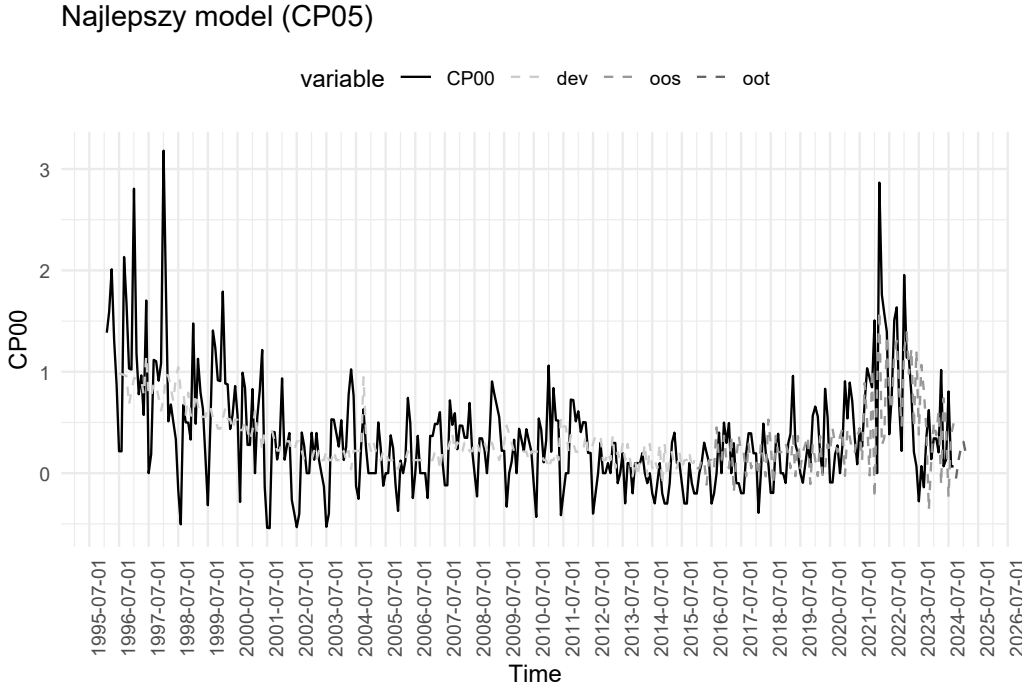

Najlepszy model (CP111)
variable
CP00
dev
oos
oot
CP00
Time

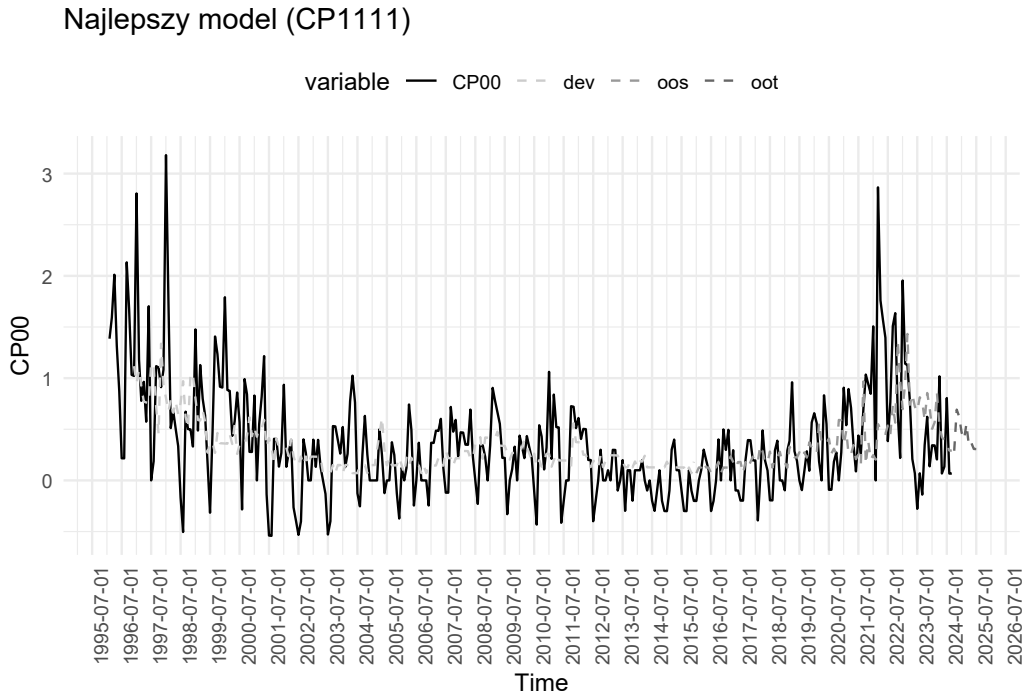

Najlepszy model (CP1111)
variable
CP00
dev
oos
oot
CP00
Time


Najlepszy model (CP0951)
variable
CP00
dev
oos
oot
CP00
2
0
-2
1995-07-01
1996-07-01
1997-07-01
1998-07-01
1999-07-01
2000-07-01
2001-07-01
2002-07-01
2003-07-01
2004-07-01
2005-07-01
2006-07-01
2007-07-01
2008-07-01
2009-07-01
2010-07-01
2011-07-01
2012-07-01
2013-07-01
2014-07-01
2015-07-01
2016-07-01
2017-07-01
2018-07-01
2019-07-01
2020-07-01
2021-07-01
2022-07-01
2023-07-01
2024-07-01
2025-07-01
2026-07-01
Time


Najlepszy model (CP0122)
variable
CP00
dev
oos
oot
CP00
3
2
1
0
1995-07-01
1996-07-01
1997-07-01
1998-07-01
1999-07-01
2000-07-01
2001-07-01
2002-07-01
2003-07-01
2004-07-01
2005-07-01
2006-07-01
2007-07-01
2008-07-01
2009-07-01
2010-07-01
2011-07-01
2012-07-01
2013-07-01
2014-07-01
2015-07-01
2016-07-01
2017-07-01
2018-07-01
2019-07-01
2020-07-01
2021-07-01
2022-07-01
2023-07-01
2024-07-01
2025-07-01
2026-07-01
Time


Najlepszy model (CP1211)
variable
CP00
dev
oos
oot
CP00
Time


Najlepszy model (CP05)
variable
CP00
dev
oos
oot
CP00
Time



Na koniec utworzono obiekt `output.3` jako rozszerzoną wersję ramki danych `output.2`, wzbogaconą o opisy kategorii COICOP. Tak utworzona ramka `output.3` umożliwia czytelniejszą prezentację wyników – z nazwami zmiennych zamiast samych kodów COICOP.

```
output.3 <- output.2 %>%
  left_join(coicop.label, by = c("x" = "code_name"))
```

Cały kod stanowi kompletny, zautomatyzowany przykład analityczny, służący do budowy, oceny i wizualizacji modeli prognostycznych dla szeregów czasowych inflacji w Polsce, modelowanej jako funkcja zdezagregowanych zmiennych inflacyjnych opartych na danych Eurostatu `COICOP`.

Przykład obejmuje pełną ścieżkę analityczną – od pobrania danych, poprzez ich przygotowanie i estymację modeli, aż po selekcję oraz wizualizację najlepszych wyników prognostycznych.

W kolejnym podrozdziale zaprezentowany zostanie model cen samochodów elektrycznych. Przedstawione będą etapy przygotowania i przetwarzania danych, a następnie konstrukcja oraz estymacja modelu ekonometrycznego umożliwiającego identyfikację kluczowych czynników wpływających na poziom cen pojazdów elektrycznych dostępnych na rynku europejskim. Ujęcie to pozwoli przejść od ogólnej charakterystyki źródeł informacji do praktycznej analizy ilościowej.

3.3.3.5. Model cen samochodów elektrycznych

W tym podrozdziale zostanie przedstawiony model cen samochodów elektrycznych na podstawie danych Komisji Europejskiej przedstawionych w bazie *Europejskiego Obserwatorium Paliw Alternatywnych* (*European Alternative Fuels Observatory*, EAFO):

- https://alternative-fuels-observatory.ec.europa.eu/consumer-portal/available-electric-vehicle-models.

EAFO jest inicjatywą, której celem jest dostarczanie informacji na temat paliw alternatywnych oraz infrastruktury ładowania i tankowania w Europie. EAFO wspiera rozwój zrównoważonego transportu poprzez dostarczanie informacji i analiz umożliwiających decydentom, przemysłowi i obywatelom podejmowanie świadomych decyzji dotyczących zrównoważonej mobilności. Więcej informacji można znaleźć na stronie internetowej EAFO. Baza danych EAFO nie oferuje interfejsu API, dlatego dane statystyczne zostały pobrane bezpośrednio ze strony internetowej[10]. Na początku zostały wczytane odpowiednie biblioteki.

```
library(car)
library(dplyr)
library(GGally)
library(ggplot2)
library(gridExtra)
library(httr)
library(jsonlite)
library(magrittr)
library(netstat)
library(plotly)
library(purrr)
library(readxl)
library(reshape2)
library(rgl)
library(RSelenium)
library(rvest)
library(stargazer)
library(stringr)
library(stringi)
library(tableHTML)
library(tidyr)
library(wdman)
library(writexl)
```

[10] Treści należące do UE na stronie internetowej EAFO są licencjonowane na podstawie licencji Creative Commons 4.0 (https://creativecommons.org/licenses/by/4.0/). Oznacza to, że ponowne wykorzystanie jest dozwolone, pod warunkiem odpowiedniego uznania autorstwa i wskazania wprowadzonych zmian.

Wprowadzenie do zastosowań automatyzacji zadań przeglądarki internetowej zostało podane w rozdziale 2.3.2. Na początku należy uruchomić serwer `Selenium` z przeglądarką *Firefox*[11].

```
server <- rsDriver(
  port = free_port(),
  browser = "firefox",
  extraCapabilities = list(
    "moz:firefoxOptions" = list(
      args = list('')
    )
  ),
  chromever = NULL,
  phantomver = NULL
)
```

Funkcja `free_port` z biblioteki `netstat` służy do znajdowania dostępnego portu TCP. Jest przydatna podczas konfigurowania usług, takich jak serwery internetowe lub połączenia z bazami danych, które komunikują się poprzez port sieciowy. Następnie należy przekazać sterowanie do przeglądarki.

```
browser <- server$client
```

Adres strony internetowej bazy danych EAFO został zapisany w zmiennej `url`.

```
url <- paste0(
  "https://alternative-fuels-observatory.ec.europa.eu",
  "/consumer-portal/available-electric-vehicle-models?page="
)
```

Identyfikator znaczników klasy CSS z danymi statystycznymi dla każdego samochodu przypisano do zmiennej `class.name`.

```
class.name <- "ecl-col-12 ecl-col-m-6 ecl-col-l-4 ecl-u-pb-l"
```

Strona internetowa bazy danych EAFO zawiera wiele podstron.

Poniższy kod uruchamia stronę internetową, oczekuje 3 sekundy na pełne załadowanie strony i symuluje naciśnięcie przycisku akceptacji warunków przeglądania strony (*Accept all cookies*) na podstawie adresu przycisku w postaci ścieżki `XPath`.

[11] Por. https://www.mozilla.org/pl/firefox/.

```
browser$navigate(url)
Sys.sleep(3)
link <- browser$findElement(
  'xpath',
  '/html/body/div[2]/div/div/div[2]/a[1]'
)
link$clickElement()
```

Po załadowaniu treści strony i zaakceptowaniu warunków jej użytkowania można przystąpić do pobrania danych. W celu zgromadzenia danych statystycznych wykorzystano listę `cars.0`. Na każdej współrzędnej znajduje się informacja z jednej podstrony (`subpage`), przy czym numeracja podstron rozpoczyna się od zera. Najpierw algorytm na podstawie ścieżki `XPath` odnajduje i wybiera przycisk `Cards`, co powoduje załadowanie strony z informacją uporządkowaną w ramkach ze zdjęciami samochodów. Następnie z każdej ramki pobierana jest informacja tekstowa zapisana w klasie `class.name`.

```
cars.0 <- lapply(
  1:subpage,
  function(x) {
    browser$navigate(paste0(url, x-1))
    Sys.sleep(3)
    link <- browser$findElement(
      'xpath',
      paste0(
        '/html/body/div[2]/main/div/div/div/div/div[2]',
        '/article/div/div[3]/button[2]/span/span'
      )
    )
    link$clickElement()
    Sys.sleep(3)
    webelem <- browser$findElements(
      "xpath",
      paste0(
        '//*[contains(@class, "', class.name, '")]'
      )
    )
    lapply(
      1:length(webelem),
      function(y) {
        webelem[[y]]$getElementText() %>%
          as.character %>%
          strsplit("\n", fixed = TRUE)
      }
    )
  }
)
```

Lista `cars.0` zawiera 8 współrzędnych i jest obiektem wielowymiarowym. Każda współrzędna wyższego poziomu zawiera listy z informacją o indywidualnych samochodach. Tę strukturę uproszczono do listy `cars.1`.

```
cars.1 <- do.call(c, cars.0) %>%
  do.call(c, .)
```

Lista `cars.1` jest listą jednowymiarową, której każda współrzędna zawiera informację o jednym samochodzie. Należy zwrócić uwagę, że informacja o niektórych samochodach jest niepełna i należy ją uzupełnić. Zatem do listy `check` przepisano wszystkie współrzędne listy `cars.1`, których wektor informacji ma nieprawidłową długość.

```
check <- lapply(
  cars.1,
  function(x) {
    if(length(x) != 14) x else NULL
  }
) %>%
  compact()
check
```

Łatwo zauważyć, że brakujący wpis dotyczy informacji `Fastcharge speed`, tj. przyrostu zasięgu w kilometrach na każdą godzinę ładowania prądem stałym (DC) na szybkiej ładowarce. Tę informację uzupełnia się wartością `NA` w odpowiednim miejscu wektora i zapisuje na liście `impute`.

```
impute <- lapply(
  check,
  function(x) {
    x %>%
      append(c("Fastcharge speed", NA), after = 10)
  }
)
```

Lista `impute` zawiera wektory o długości 14 współrzędnych. Następnie do listy `cars.2` dodaje się listę z uzupełnioną informacją, tak aby wszystkie współrzędne nowej listy `cars.2` miały poprawną długość.

```
cars.2 <- lapply(
    cars.1,
    function(x) {
      if(length(x) != 14) NULL else x
    }
```

```
  ) %>%
  compact() %>%
  append(impute)
```

Informacja zawarta na liście `cars.2` jest następnie porządkowana i zapisywana do ramki danych `cars.3`.

```
cars.3 <- lapply(
  1:length(cars.2),
  function(x) {
    a <- cars.2[[x]]
    b <- data.frame(
      a[c(1, 3, 5, 7,  9, 11, 13)],
      a[c(2, 4, 6, 8, 10, 12, 14)]
    ) %>%
      t() %>%
      set_colnames(.[1, ]) %>%
      as.data.frame() %>%
      .[-1, ]
  }
) %>%
  do.call(rbind, .) %>%
  set_rownames(1:nrow(.))
```

Informacja zawarta w ramce `cars.3` podlega dalszemu uporządkowaniu w ramce `cars.4`. Jednym z celów porządkowania jest obliczenie ceny samochodów w PLN w miejsce EUR. Konieczne jest zatem określenie aktualnego kursu walutowego EUR/PLN. W tym celu zapisano funkcję `eurpln`, która pobiera kurs ze strony internetowej Narodowego Banku Polskiego.

```
eurpln <- function() {
  library(httr)
  library(jsonlite)
  url <- paste0(
    "https://api.nbp.pl/api/exchangerates/rates",
    "/a/eur?format=json"
  )
  response <- GET(url)
  if (status_code(response) == 200) {
    eurpln <- fromJSON(content(response, "text")) %>%
      .$rates %>%
      .$mid
  } else {
    cat("Kod błędu:", status_code(response), "\n")
  }
  return(eurpln)
}
```

Aktualny kurs EUR/PLN na dzień 21.12.2025 ma wartość 4.2094. Dalsze uporządkowanie polega na pobraniu z kolumn:

- `Range`,
- `Battery size`,
- `Efficiency`,
- `Fastcharge speed`,

tylko informacji numerycznej oraz obliczeniu, po uporządkowaniu, ceny `Price (approx.)` w PLN w miejsce EUR. Następnie angielskie nazwy kolumn zostały zamienione na nazwy polskie.

```
cars.4 <- cars.3 %>%
  mutate(
    across(c(
      Range,
      `Battery size`,
      Efficiency,
      `Fastcharge speed`
    ),
    ~ str_extract(., "\\d+") %>%
      as.numeric()
    ),
    `Price (approx.)` = `Price (approx.)` %>%
      str_replace(",", "") %>%
      str_extract(., "\\d+") %>%
      as.numeric() %>%
      {.*eurpln()} %>%
      round(0)
  ) %>%
  set_colnames(c(
    "Model",
    "Dostępny",
    "Zasięg",
    "Bateria",
    "Zużycie",
    "Doładowanie",
    "Cena"
  ))
```

W celu ułatwienia dalszej pracy zbiór danych `cars.4` został zapisany do pliku Excel `cars.xlsx` w ścieżce `path` za pomocą funkcji `write_xlsx` z biblioteki `writexl`.

```
path <- "F:\\docs\\R\\b2\\ch3\\"
file <- path %>%
  paste0("cars.xlsx")
```

```
cars.4 %>%
  write_xlsx(file)
```

Zbiór można odczytać za pomocą funkcji `read_excel` z biblioteki `readxl`.

```
cars.4 <- read_excel(file)
```

Poniżej podano kompletną procedurę pobrania danych.

```
if (!file.exists(file)) {
  server <- rsDriver(
    port = free_port(),
    browser = "firefox",
    extraCapabilities = list(
      "moz:firefoxOptions" = list(
        args = list('')
      )
    ),
    chromever = NULL
  )
  browser <- server$client
  url <- paste0(
    "https://alternative-fuels-observatory.ec.europa.eu",
    "/consumer-portal/available-electric-vehicle-models?page="
  )
  class.name <- "ecl-col-12 ecl-col-m-6 ecl-col-l-4 ecl-u-pb-l"
  subpage <- 8
  browser$navigate(url)
  Sys.sleep(3)
  link <- browser$findElement(
    'xpath',
    '/html/body/div[2]/div/div/div[2]/a[1]'
  )
  link$clickElement()
  cars.0 <- lapply(
    1:subpage,
    function(x) {
      browser$navigate(paste0(url, x-1))
      Sys.sleep(3)
      link <- browser$findElement(
        'xpath',
        paste0(
          '/html/body/div[2]/main/div/div/div/div/div[2]',
          '/article/div/div[3]/button[2]/span/span'
        )
      )
      link$clickElement()
```

```
    Sys.sleep(3)
    webelem <- browser$findElements(
      "xpath",
      paste0(
        '//*[contains(@class, "', class.name, '")]'
      )
    )
    lapply(
      1:length(webelem),
      function(y) {
        webelem[[y]]$getElementText() %>%
          as.character %>%
          strsplit("\n", fixed = TRUE)
      }
    )
  }
)
cars.1 <- do.call(c, cars.0) %>%
  do.call(c, .)
check <- lapply(
  cars.1,
  function(x) {
    if(length(x) != 14) x else NULL
  }
) %>%
  compact()
check
impute <- lapply(
  check,
  function(x) {
    x %>%
      append(c("Fastcharge speed", NA), after = 10)
  }
)
cars.2 <- lapply(
    cars.1,
    function(x) {
      if(length(x) != 14) NULL else x
    }
  ) %>%
  compact() %>%
  append(impute)
cars.3 <- lapply(
  1:length(cars.2),
  function(x) {
    a <- cars.2[[x]]
    b <- data.frame(
      a[c(1, 3, 5, 7,  9, 11, 13)],
      a[c(2, 4, 6, 8, 10, 12, 14)]
```

```
      ) %>%
        t() %>%
        set_colnames(.[1, ]) %>%
        as.data.frame() %>%
        .[-1, ]
    }
  ) %>%
    do.call(rbind, .) %>%
    set_rownames(1:nrow(.))
  cars.4 <- cars.3 %>%
    mutate(
      across(c(
        Range,
        `Battery size`,
        Efficiency,
        `Fastcharge speed`
      ),
      ~ str_extract(., "\\d+") %>%
        as.numeric()
      ),
      `Price (approx.)` = `Price (approx.)` %>%
        str_replace(",", "") %>%
        str_extract(., "\\d+") %>%
        as.numeric() %>%
        {.*eurpln()} %>%
        round(0)
    ) %>%
    set_colnames(c(
      "Model",
      "Dostępny",
      "Zasięg",
      "Bateria",
      "Zużycie",
      "Doładowanie",
      "Cena"
    ))
  cars.4 %>%
    write_xlsx(file)
  browser$close()
  server$server$stop()
} else {
  cars.4 <- read_excel(file)
}
```

Ramka danych `cars.4` zawiera uporządkowaną informację.

```
head(cars.4)
```

```
## # A tibble: 6 x 7
##   Model                        Dostępny Zasięg Bateria Zużycie
 ↪  Doładowanie   Cena
##   <chr>                        <chr>     <dbl>   <dbl>   <dbl>
 ↪  <dbl>  <dbl>
## 1 Abarth 500e Convertible      05-2023     225      37      16
 ↪  370 170672
## 2 Abarth 500e Hatchback        05-2023     225      37      16
 ↪  370 158725
## 3 Aiways U5                    04-2022     315      60      19
 ↪  380 176219
## 4 Aiways U6                    12-2022     350      60      17
 ↪  430 206086
## 5 Alfa Romeo Junior Elettric~ 04-2024     320      50      15
 ↪  510 167685
## 6 Alfa Romeo Junior Elettric~ 06-2024     310      50      16
 ↪  500 206086
```

Można zauważyć, że kolumna `Model` zawiera pełną nazwę samochodu.

```
cars.4$Model[1]
```

```
## [1] "Abarth 500e Convertible"
```

Warto podzielić tę kolumnę na kolumny zawierające: markę, model i dalsze oznaczenia.

```
max.space <- cars.4 %>%
  {max(str_count(.$Model, " ")) + 1}
```

Ponieważ poszczególne informacje są oddzielone spacjami, do obiektu `max.space` wpisano liczbę spacji, czyli liczbę potencjalnych składników informacji (8). Następnie do ramki danych `cars.5` za pomocą funkcji `separate` z biblioteki `tidyr`, zapisano kolumny `Model_1`, `Model_2` itd., które zawierają podzieloną informację z kolumny `Model`. Całość uzupełniono symbolem myślnika `-`, jeśli informacja nie występowała.

```
cars.5 <- cars.4 %>%
  separate(
    "Model",
    into = c(paste0("Model_", 1:max.space)),
    sep = " "
```

```
  ) %>%
  mutate(
    across(
      starts_with("Model_"),
      ~ifelse(is.na(.), "-", .)
    )
  )
```

Ramka danych `cars.5` została posortowana według kolumny `Model_1`.

```
cars.6 <- cars.5 %>%
  arrange(Model_1)
```

Następnie dane zawarte w kolumnach `Model_1` do `Model_8` zostały uporządkowane w taki sposób, że kolumna `Model_1` zawiera nazwę marki, kolumna `Model_2` – nazwę modelu, a kolumna `Model_3` – połączoną informację z kolumn `Model_3` do `Model_8`.

```
cars.7 <- cars.6 %>%
  unite("Model_3", Model_3:Model_8, sep = " ") %>%
  mutate_at(
    vars(Model_3),
    ~gsub("-", "", .)
  ) %>%
  mutate_at(
    vars(Model_3),
    ~trimws(.)
  ) %>%
  mutate_at(
    vars(Model_3),
    ~gsub("^$", "-", .)
  )
```

Ramka danych `cars.7` zawiera końcowy zbiór danych, który zostanie poddany analizie.

```
head(cars.7)
```

```
## # A tibble: 6 x 9
##   Model_1 Model_2 Model_3     Dostępny Zasięg Bateria Zużycie
 ↪  Doładowanie   Cena
##   <chr>   <chr>   <chr>       <chr>     <dbl>   <dbl>   <dbl>
 ↪  <dbl>  <dbl>
## 1 Abarth  500e    Convertible 05-2023     225      37      16
 ↪  370 170672
```

```
## 2 Abarth  500e    Hatchback   05-2023     225      37      16
 ↪  370 158725
## 3 Aiways  U5      -           04-2022     315      60      19
 ↪  380 176219
## 4 Aiways  U6      -           12-2022     350      60      17
 ↪  430 206086
## 5 Alfa    Romeo   Junior Ele~ 04-2024     320      50      15
 ↪  510 167685
## 6 Alfa    Romeo   Junior Ele~ 06-2024     310      50      16
 ↪  500 206086
```

Niech nowy zbiór nosi nazwę `data` (liczba obserwacji statystycznych: 400).

```
data <- cars.7
```

Najpierw przeprowadzono analizę graficzną i statystyczną zbioru `data`. W tym celu zdefiniowano funkcję `fun.1`, która odpowiada za wyświetlenie wykresu przedstawiającego rozkłady wartości poszczególnych zmiennych zbioru `data` w podziale na producentów samochodów. Zatem w funkcji `fun.1` następuje grupowanie informacji według kolumny `Model_1` (producent). Następnie wyznaczono liczbę modeli dla danego producenta (`n`) i średnią wartość cechy z kolumny `col` (argument funkcji), przy czym uwzględniono tylko tych producentów, dla których liczba modeli jest większa od wartości `n_min` (argument funkcji). Wykresy na rysunku są sortowane według rosnącej wartości średniej danej cechy. Następnie wyświetlono wykres dla przekształconego zbioru `data` za pomocą funkcji `ggplot` z odpowiednimi parametrami. Można zauważyć, że jednym z tych parametrów jest `fill = arg[[1]]`, czyli pierwszy z dodatkowych argumentów przekazanych do funkcji `fun.1` za pomocą operatora `...` (por. podrozdział 2.2.4) i pobranych do listy `arg`.

```
fun.1 <- function(data, col, n_min, ...) {
  arg <- list(...)
  data %>%
    group_by(Model_1) %>%
    mutate(
      n = n(),
      mean = mean(!!sym(col), na.rm = TRUE)
    ) %>%
    subset(n >= n_min) %>%
    ungroup() %>%
    arrange(mean) %>%
    mutate(
      Model_1 = factor(
        Model_1,
        levels = unique(Model_1)
      )
    ) %>%
```

```
    ggplot(aes(x = !!sym(col))) +
    geom_histogram(
      fill = arg[[1]],
      color = "black",
      alpha = 0.7,
      bins = 30
    ) +
    geom_vline(
      aes(xintercept = mean),
      color = "grey40",
      linetype = "dashed",
      linewidth = 1.5
    ) +
    facet_grid(
      Model_1 ~ .,
      scales = "free_y"
    ) +
    labs(
      title = paste(
        "Histogram",
        col,
        "według Model_1 z średnią"
      ),
      x = col,
      y = "Częstość"
    ) +
    theme_minimal() +
    theme(
      text = element_text(family = "Arial"),
      strip.text = element_text(
        size = 9,
        face = "bold"
      ),
      strip.text.y = element_text(
        angle = 0,
        hjust = 0.5
      ),
      strip.placement = "outside",
      panel.spacing = unit(0.5, "lines"),
      panel.grid.major = element_line(
        color = "grey90"
      ),
      panel.grid.minor = element_blank()
    )
}
```

Funkcję `fun.1` można wykorzystać do przedstawienia rozkładu cech z ramki `data`.

```
colnames(data)[5:9]
```

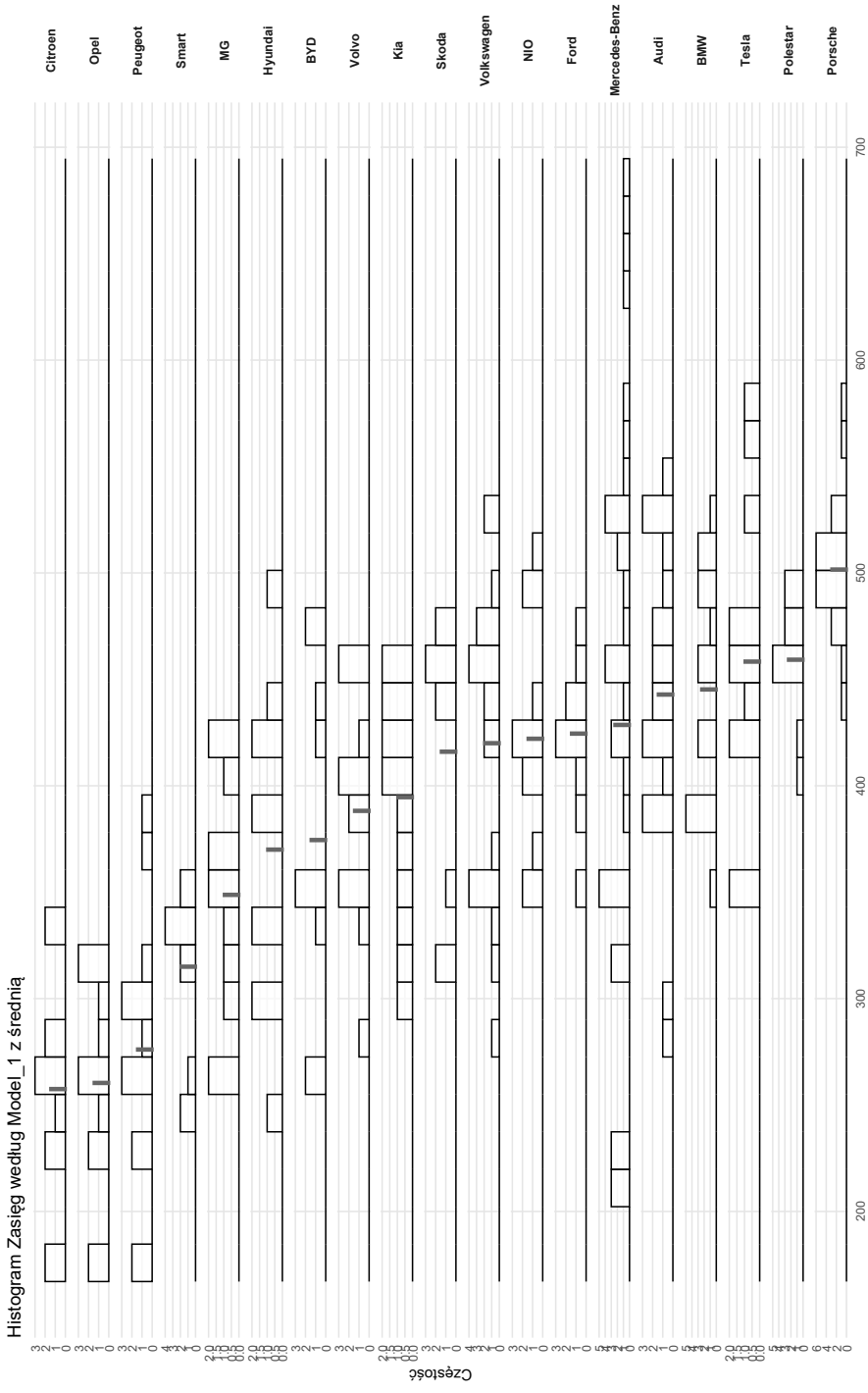
```
## [1] "Zasięg"      "Bateria"     "Zużycie"     "Doładowanie" "Cena"
```

Wykresy – w postaci *landscape* – można umieścić w pętli `for` w sposób podany poniżej.

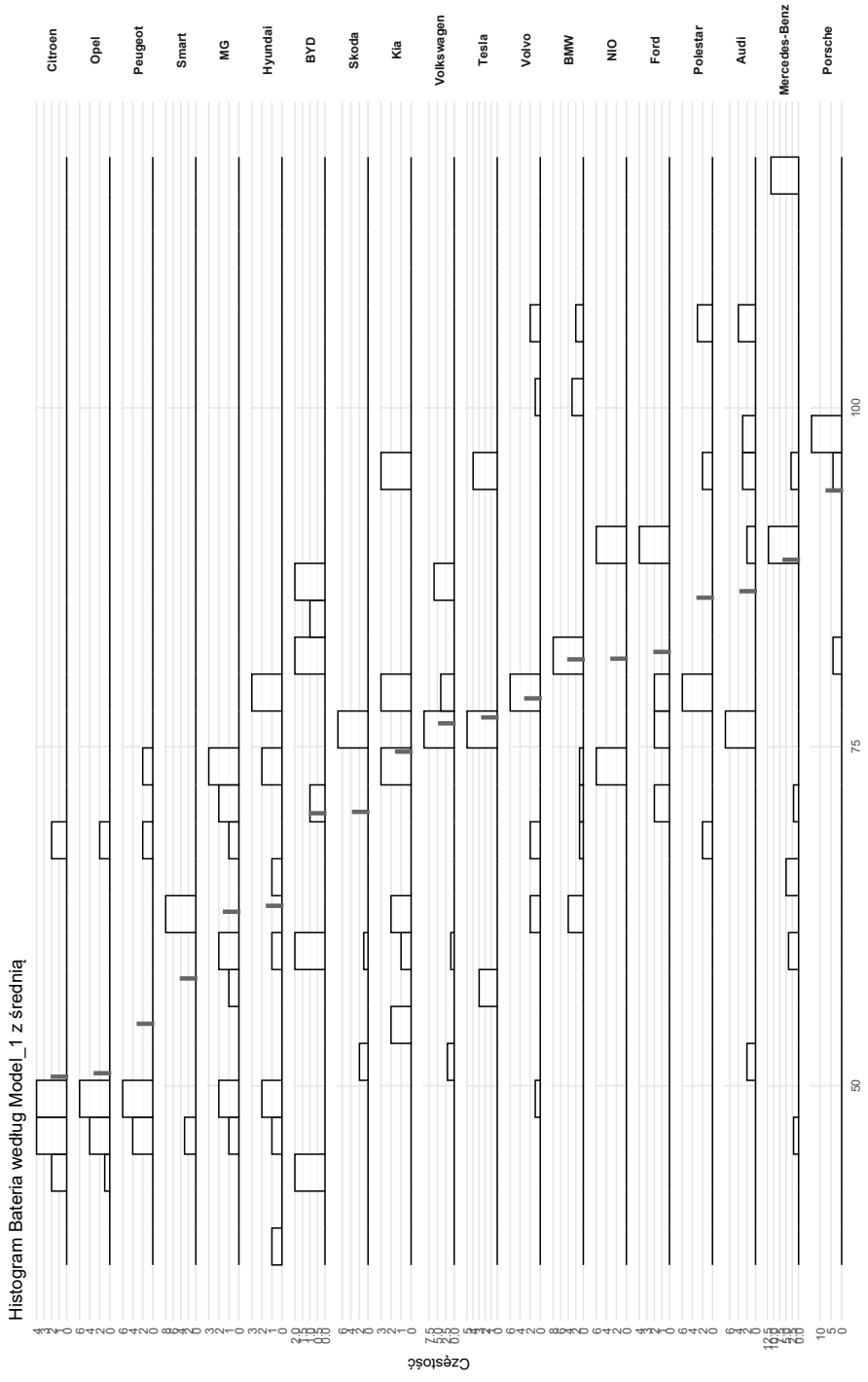
```
i.all <- seq_along(colnames(data)[5:9])
rnd.str <- stri_rand_strings(
  n = length(i.all),
  length = 6,
  pattern = "[a-z]"
)
captions <- lapply(
  i.all,
  function(x) {
    var.name <- colnames(data)[5:9][x]
    paste0("Model\\_1 z średnią: ", var.name)
  }
)
```

```
pl.temp <- list()
for (i in i.all) {
  pl.temp[[i]] <- data %>% fun.1(colnames(data)[5:9][i], 10, "white")
  ggplot2::ggsave(
    paste0("output/plots/plot_", rnd.str[i], ".pdf"),
    plot = pl.temp[[i]],
    width = 14,
    height = 9,
    units = "in",
    device = grDevices::cairo_pdf
  )
}
```

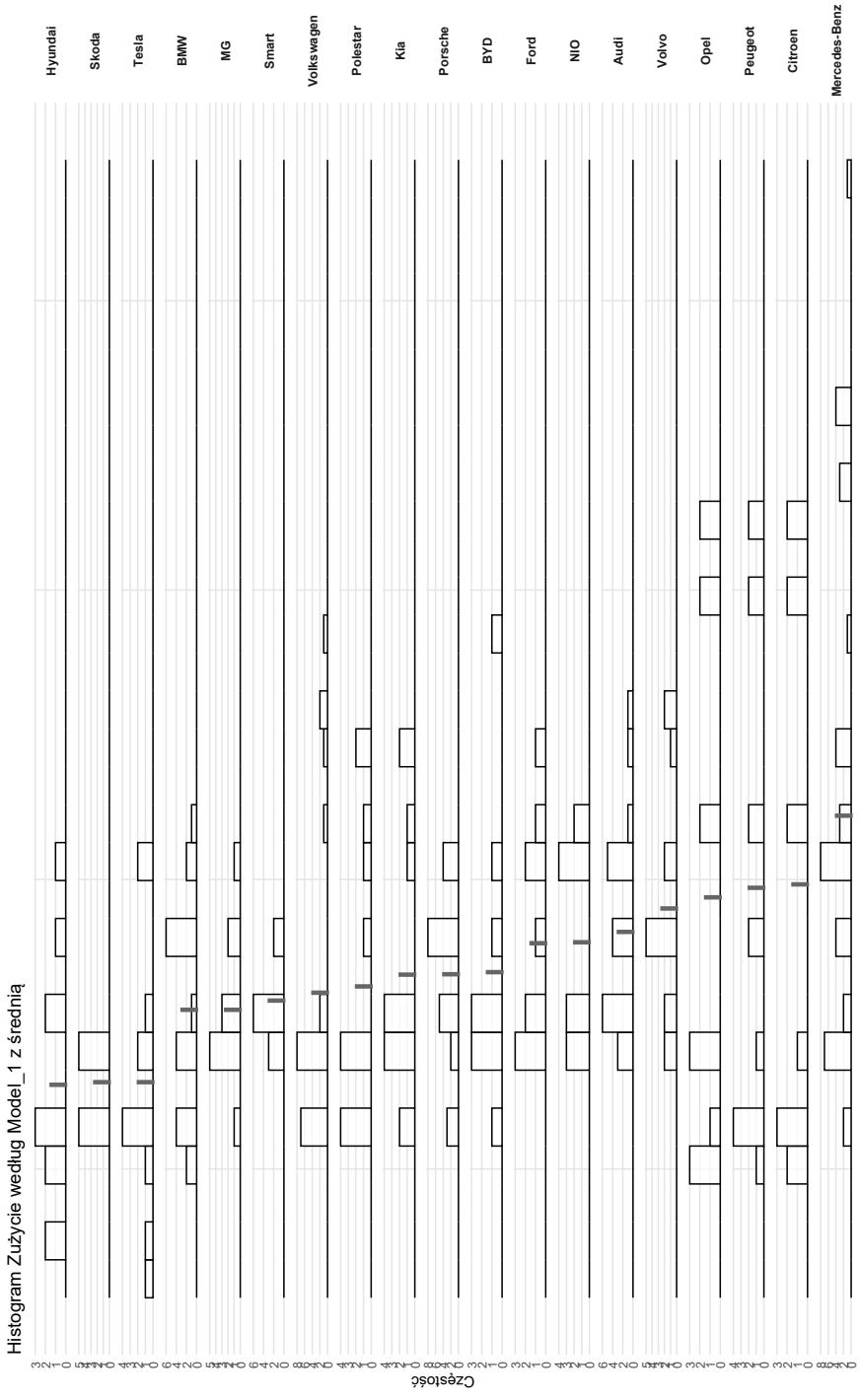
```
for (i in i.all) {
  cat("\\begin{landscape}\n")
  cat("\\begin{figure}[htbp]\n")
  cat("  \\centering\n")
  cat("  \\includegraphics[angle=0,
   ↪  width=\\linewidth]{output/plots/plot_", rnd.str[i], ".pdf}\n",
   ↪  sep = "")
  cat("  \\caption{", captions[[i]], "}\n", sep = "")
  cat("  \\label{fig:", rnd.str[i], "}\n", sep = "")
  cat("\\end{figure}\n")
  cat("\\end{landscape}\n\n")
}
```



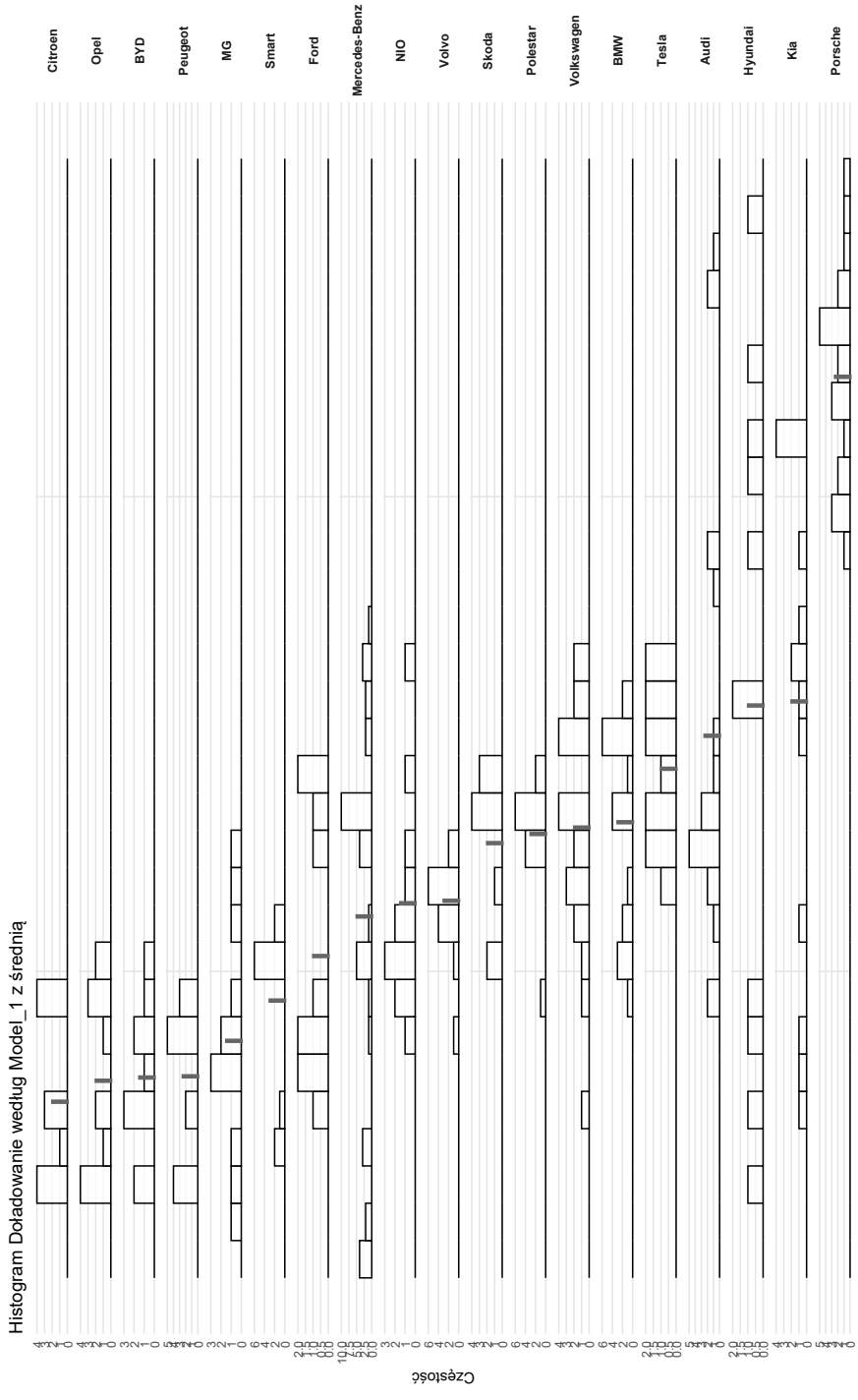
Rysunek 3.30. Model_1 z średnią: Zasięg



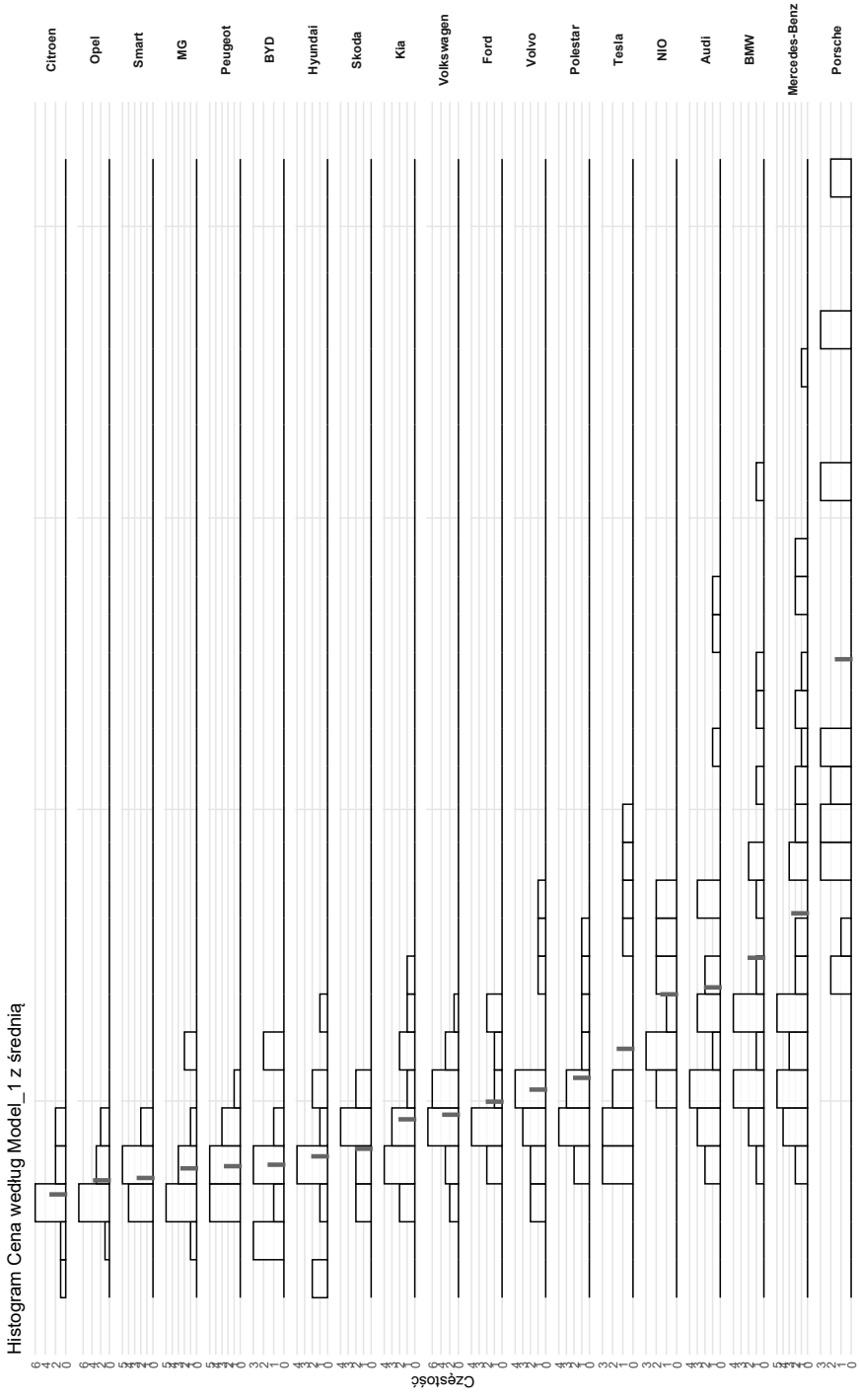
Rysunek 3.31. Model_1 z średnią: Bateria



Rysunek 3.32. Model_1 z średnią: Zużycie



Rysunek 3.33. Model_1 z średnią: Doładowanie



Rysunek 3.34. Model_1 z średnią: Cena

W podobny sposób można utworzyć funkcję `fun.2`, w której grupowanie odbywa się dla kolumn `Model_1` i `Model_2`, tj. według marki i modelu.

```
fun.2 <- function(data, col, n_min, ...) {
  arg <- list(...)
  data %>%
    group_by(Model_1, Model_2) %>%
    mutate(
      n = n(),
      mean = mean(!!sym(col), na.rm = TRUE)
    ) %>%
    subset(n >= n_min) %>%
    ungroup() %>%
    arrange(mean) %>%
    mutate(
      Model_1 = factor(
        Model_1,
        levels = unique(Model_1)
      ),
      Model_2 = factor(
        Model_2,
        levels = unique(Model_2)
      )
    ) %>%
    ggplot(aes(x = !!sym(col))) +
    geom_histogram(
      fill = arg[[1]],
      color = "black",
      alpha = 0.7,
      bins = 30
    ) +
    geom_vline(
      aes(xintercept = mean),
      color = "grey40",
      linetype = "dashed",
      linewidth = 1.5
    ) +
    facet_grid(
      Model_2 + Model_1 ~ .,
      scales = "free_y"
    ) +
    labs(
      title = paste(
        "Histogram",
        col,
        "według Model_1 + Model_2 z średnią"
      ),
      x = col,
      y = "Częstość"
```

```
    ) +
    theme_minimal() +
    theme(
      text = element_text(family = "Arial"),
      strip.text = element_text(
        size = 9,
        face = "bold"
      ),
      strip.text.y = element_text(
        angle = 0,
        hjust = 0.5
      ),
      strip.placement = "outside",
      panel.spacing = unit(0.5, "lines"),
      panel.grid.major = element_line(
        color = "grey90"),
      panel.grid.minor = element_blank()
    )
}
```

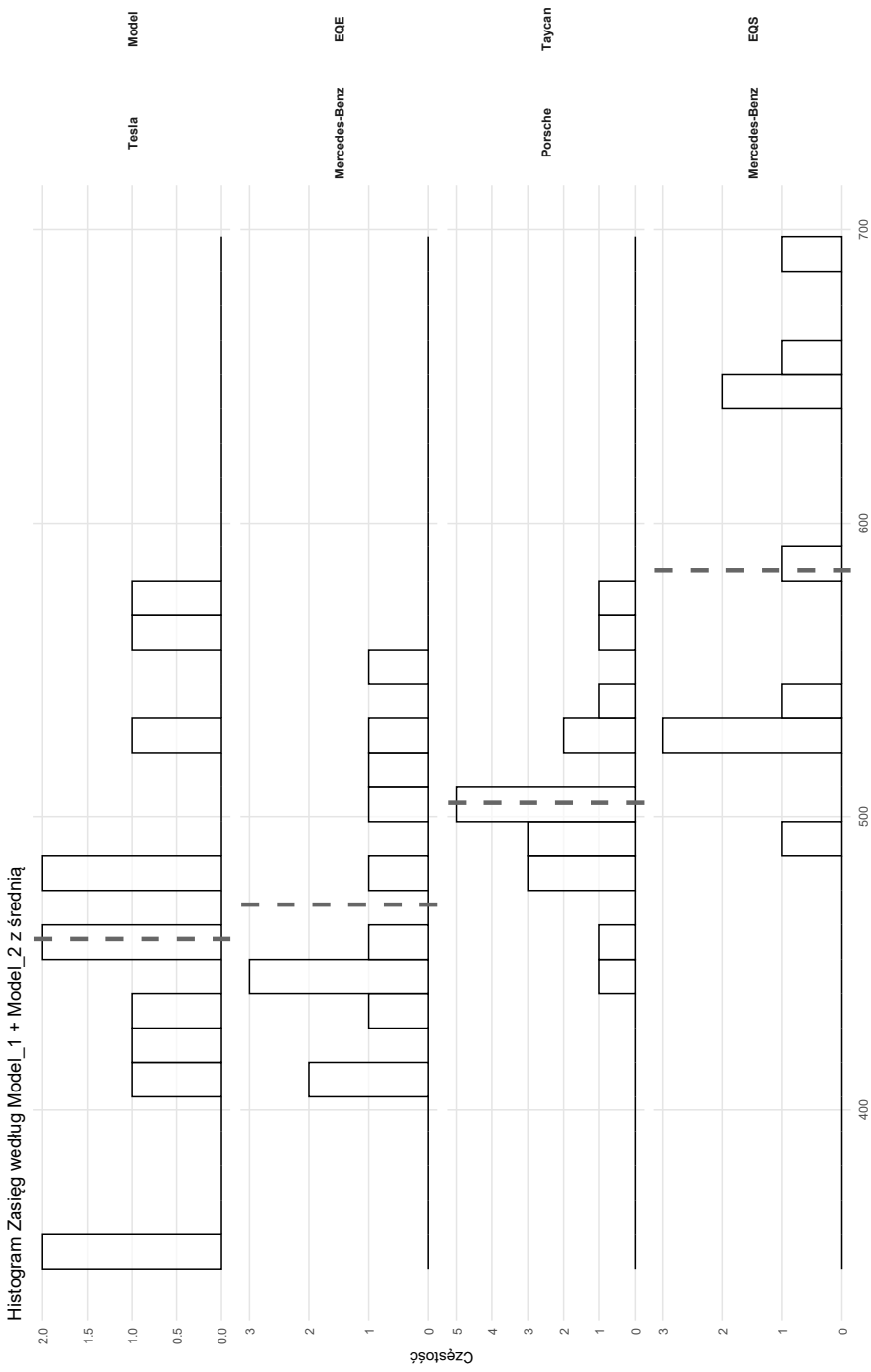
Funkcję `fun.2`, podobnie jak funkcję `fun.1`, można wykonać w pętli.

```
i.all <- seq_along(colnames(data)[5:9])
rnd.str <- stri_rand_strings(
  n = length(i.all),
  length = 6,
  pattern = "[a-z]"
)
captions <- lapply(
  i.all,
  function(x) {
    var.name <- colnames(data)[5:9][x]
    paste0("Model\\_1 + Model\\_2 z średnią: ", var.name)
  }
)
```

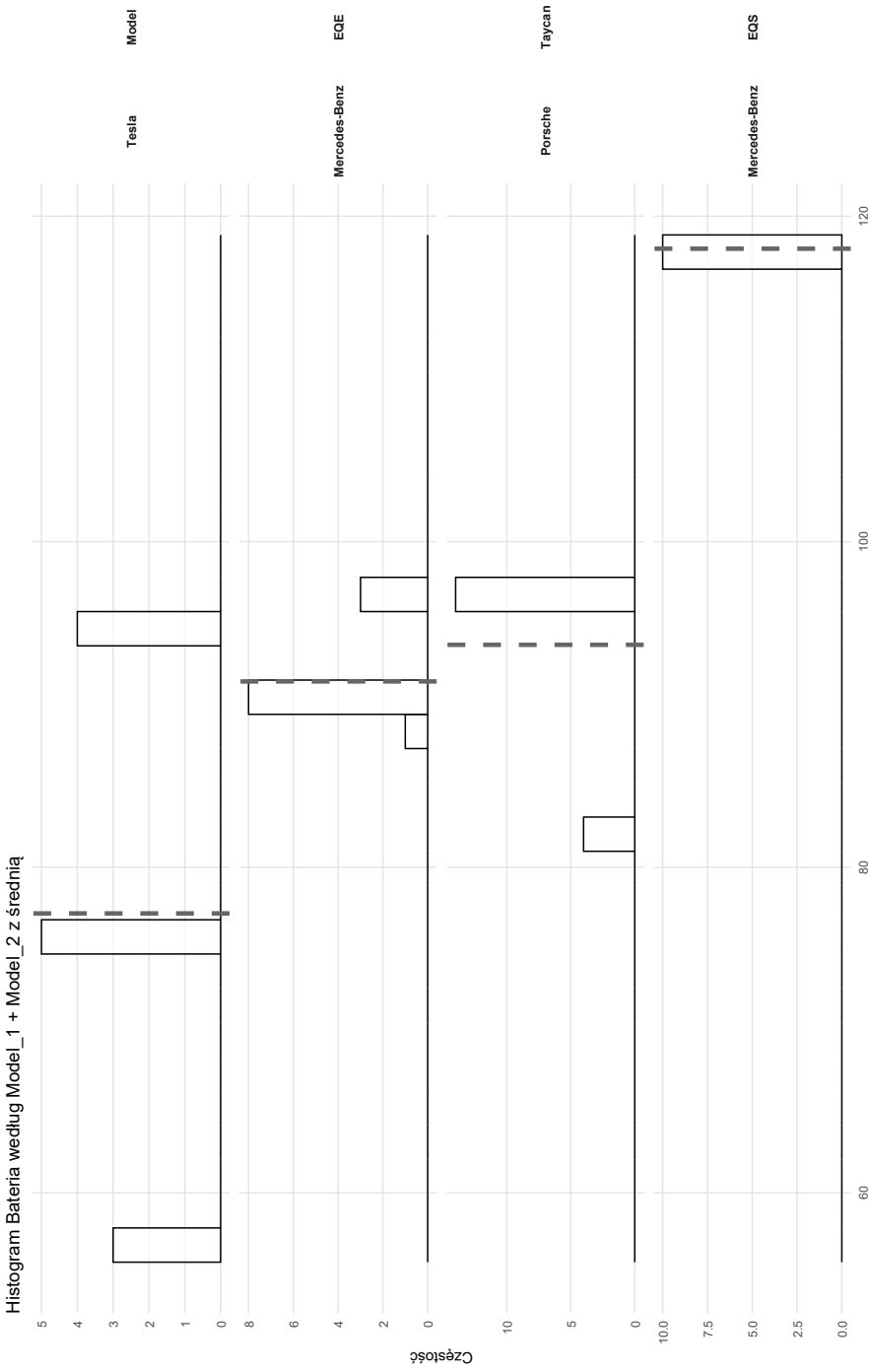
```
pl.temp <- list()
for (i in i.all) {
  pl.temp[[i]] <- data %>% fun.2(colnames(data)[5:9][i], 10, "white")
  ggplot2::ggsave(
    paste0("output/plots/plot_", rnd.str[i], ".pdf"),
    plot = pl.temp[[i]],
    width = 14,
    height = 9,
    units = "in",
    device = grDevices::cairo_pdf
  )
```

```
}
```

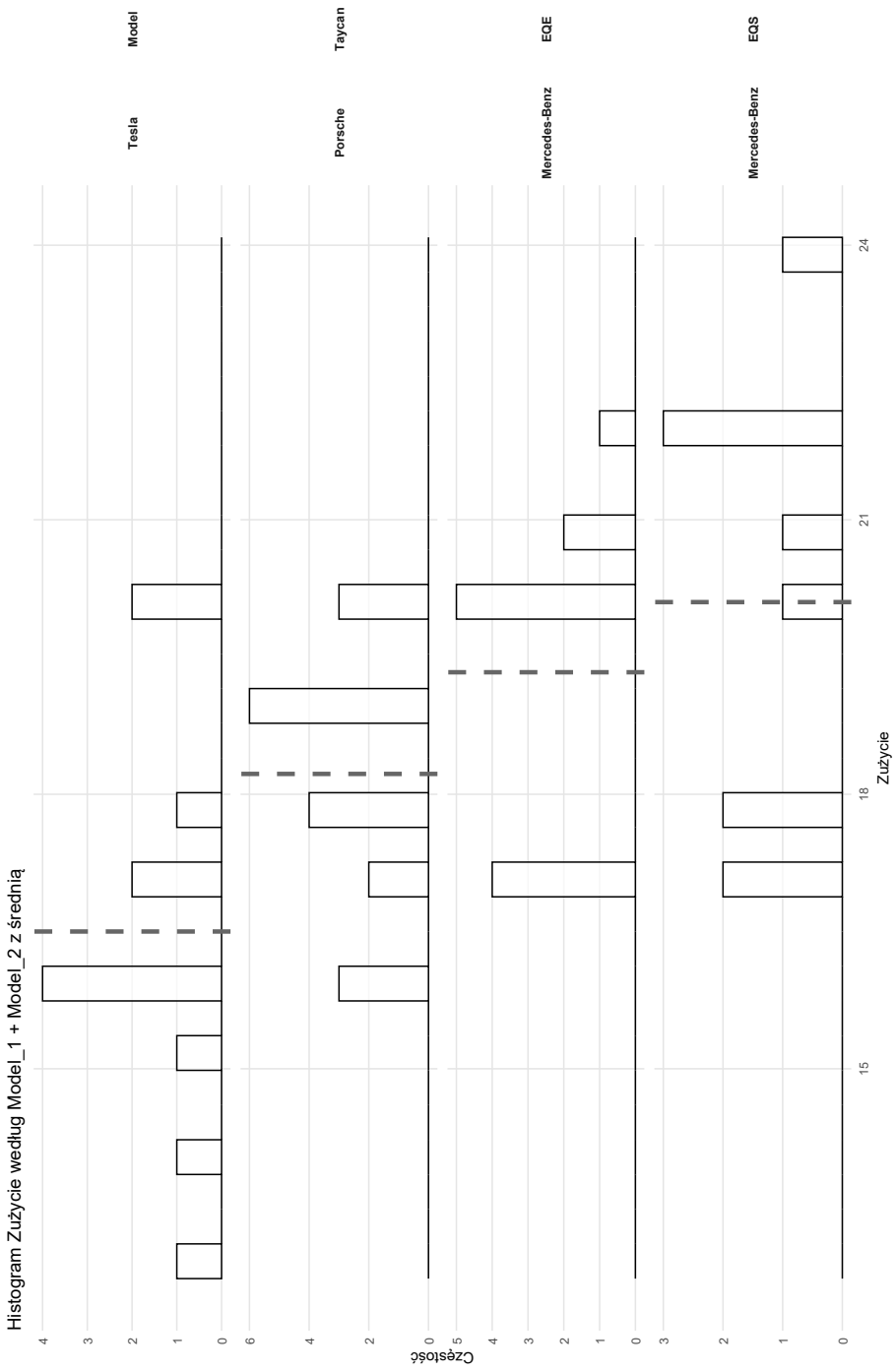
```
for (i in i.all) {
  cat("\\begin{landscape}\n")
  cat("\\begin{figure}[htbp]\n")
  cat("  \\centering\n")
  cat("  \\includegraphics[angle=0,
   ↪  width=\\linewidth]{output/plots/plot_", rnd.str[i], ".pdf}\n",
   ↪  sep = "")
  cat("  \\caption{", captions[[i]], "}\n", sep = "")
  cat("  \\label{fig:", rnd.str[i], "}\n", sep = "")
  cat("\\end{figure}\n")
  cat("\\end{landscape}\n\n")
}
```



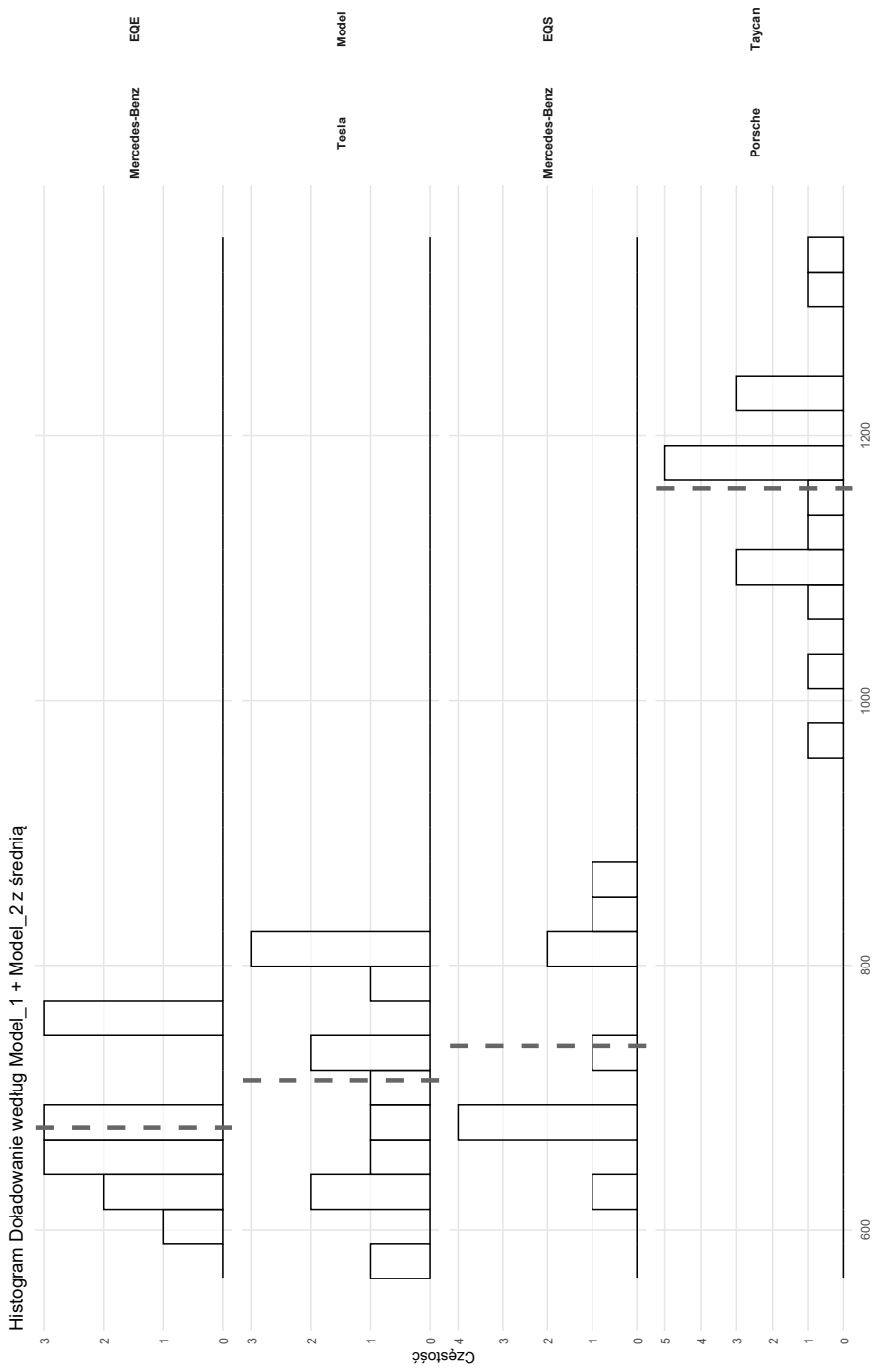
Rysunek 3.35. Model_1 + Model_2 z średnią: Zasięg



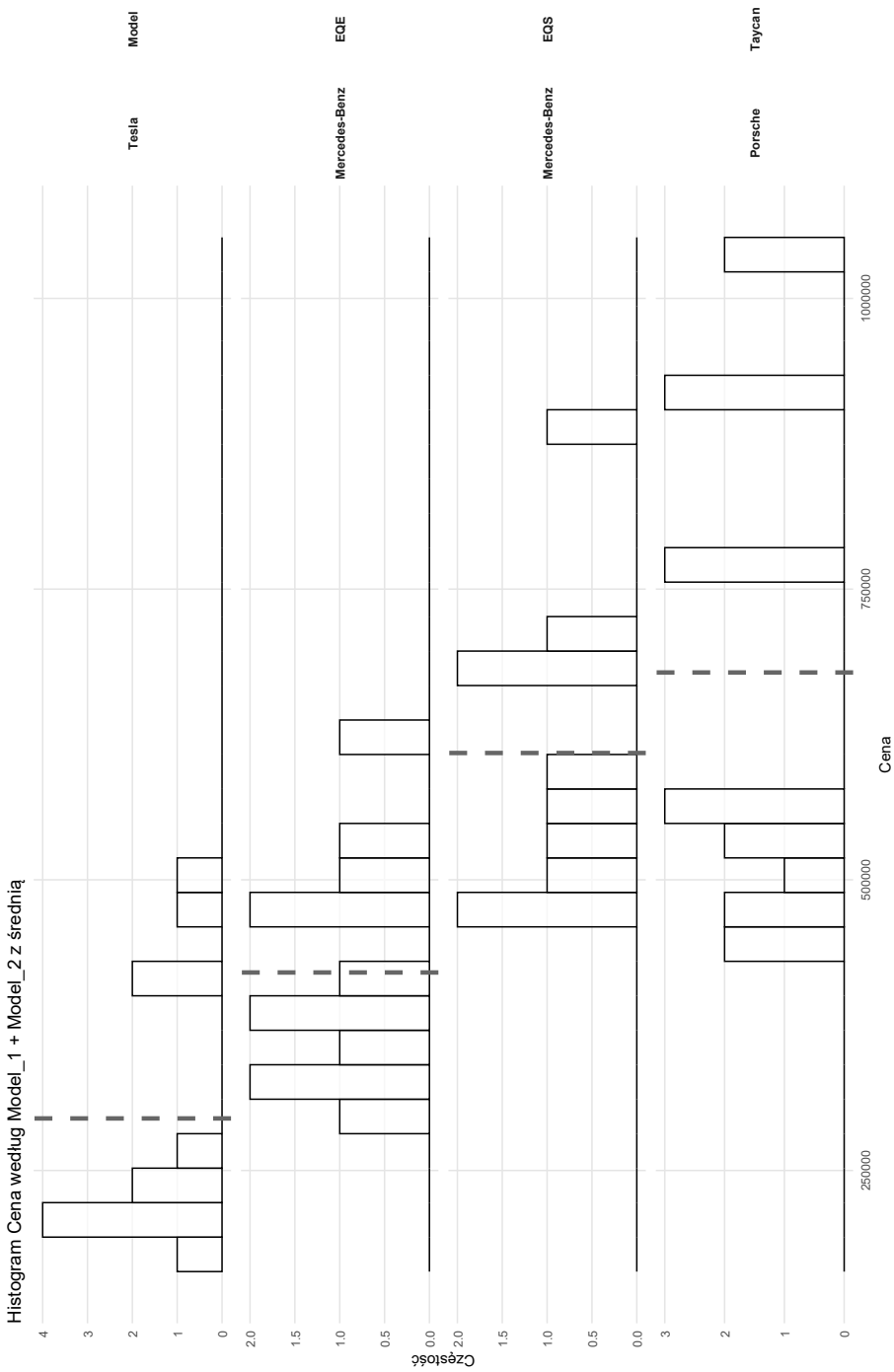
Rysunek 3.36. Model_1 + Model_2 z średnią: Bateria



Rysunek 3.37. Model_1 + Model_2 z średnią: Zużycie



Rysunek 3.38. Model_1 + Model_2 z średnią: Doładowanie



Rysunek 3.39. Model_1 + Model_2 z średnią: Cena

Funkcja `fun.3` służy do tworzenia wykresów rozrzutu według marki (`Model_1`) dla dwóch cech `x` i `y` wraz z linią regresji i przedziałem ufności.

```
fun.3 <- function(data, x_col, y_col, n_min, ...) {
  arg <- list(...)
  data %>%
    group_by(Model_1) %>%
    mutate(
      n = n()
    ) %>%
    subset(n >= n_min) %>%
    ungroup() %>%
    arrange(desc(n)) %>%
    mutate(Model_1 = factor(
      Model_1,
      levels = unique(Model_1))
    ) %>%
    ggplot(aes(x = !!sym(x_col), y = !!sym(y_col))) +
    geom_point(alpha = 0.7) +
    geom_smooth(
      method = "lm",
      se = TRUE,
      color = "grey40",
      fill = arg[[1]],
      alpha = 0.3
    ) +
    facet_grid(
      Model_1 ~ .,
      scales = "free"
    ) +
    scale_y_continuous(labels = scales::comma) +
    labs(
      title = paste(
        "Wykres rozrzutu",
        y_col,
        "vs.",
        x_col,
        "według Model_1"
      ),
      x = x_col,
      y = y_col
    ) +
    theme_minimal() +
    theme(
      text = element_text(family = "Arial"),
      strip.text = element_text(
        size = 9,
        face = "bold"
      ),
```

```
      strip.text.y = element_text(
        angle = 0,
        hjust = 0.5
      ),
      panel.spacing = unit(0.5, "lines"),
      panel.grid.major = element_line(
        color = "grey80"
      ),
      panel.grid.minor = element_blank()
    )
}
```

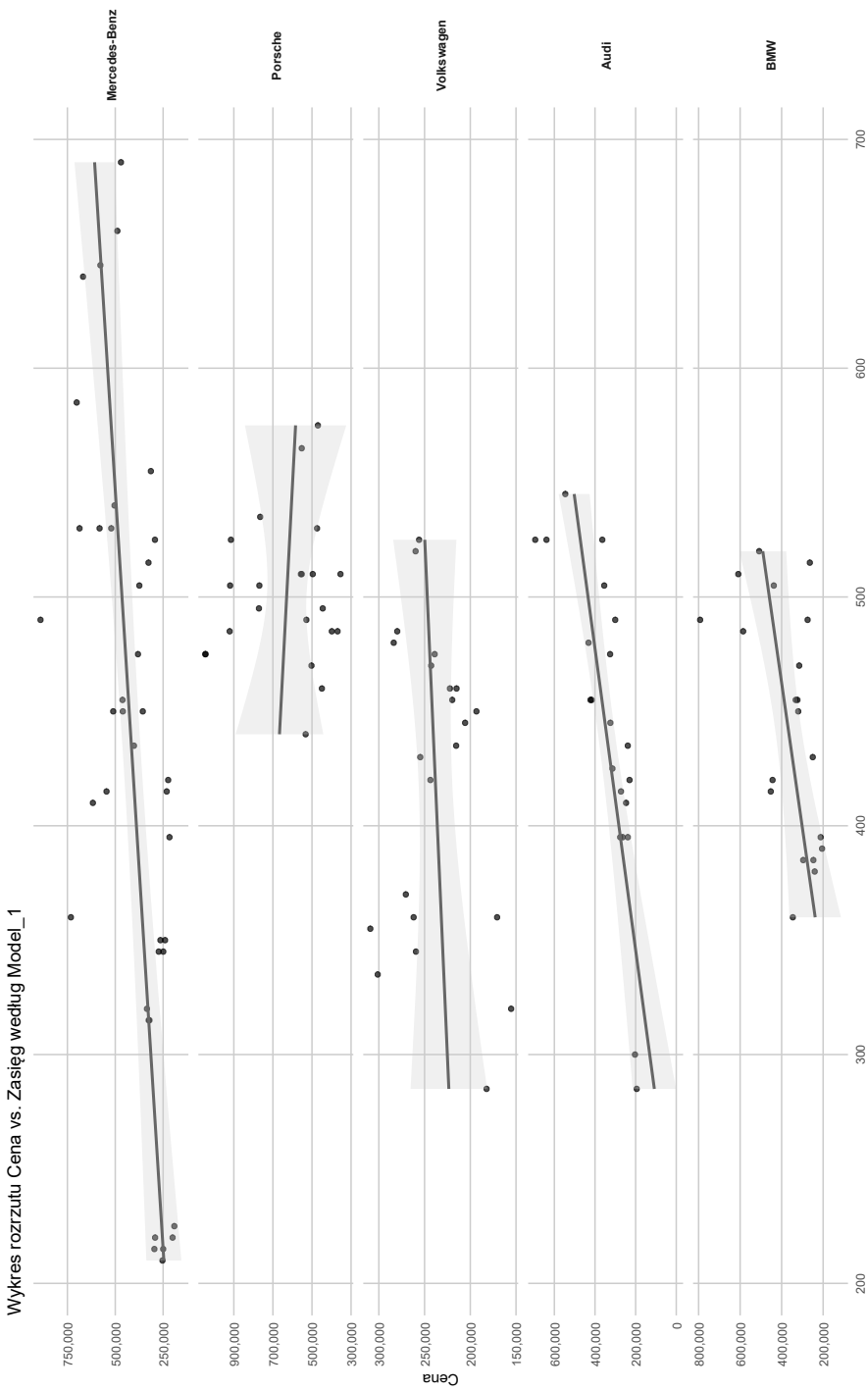
Przekazanie odpowiednich argumentów do funkcji `fun.3` powoduje wyświetlenie wykresów rozrzutu dla różnych cech.

```
i.all <- seq_along(c("Zasięg", "Bateria", "Zużycie", "Doładowanie"))
rnd.str <- stri_rand_strings(
  n = length(i.all),
  length = 6,
  pattern = "[a-z]"
)
captions <- lapply(
  i.all,
  function(x) {
    var.name <- c("Zasięg", "Bateria", "Zużycie", "Doładowanie")[x]
    paste0("Wykres rozrzutu według Model\\_1: ", var.name)
  }
)
```

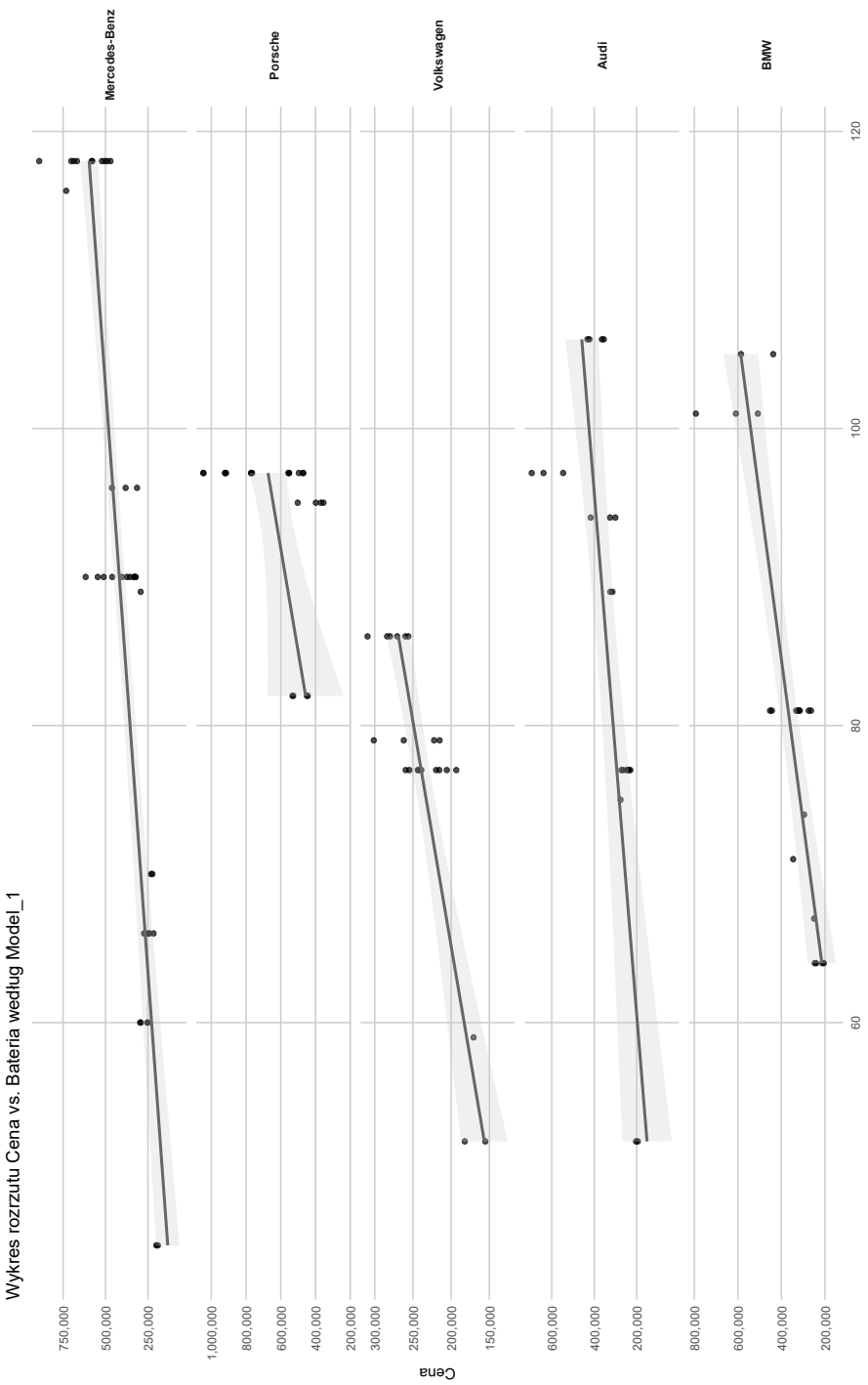
```
pl.temp <- list()
for (i in i.all) {
  pl.temp[[i]] <- data %>% fun.3(c("Zasięg", "Bateria", "Zużycie",
 ↪  "Doładowanie")[i], "Cena", 15, "grey80")
  ggplot2::ggsave(
    paste0("output/plots/plot_", rnd.str[i], ".pdf"),
    plot = pl.temp[[i]],
    width = 14,
    height = 9,
    units = "in",
    device = grDevices::cairo_pdf
  )
}
```

```
for (i in i.all) {
  cat("\\begin{landscape}\n")
  cat("\\begin{figure}[htbp]\n")
```

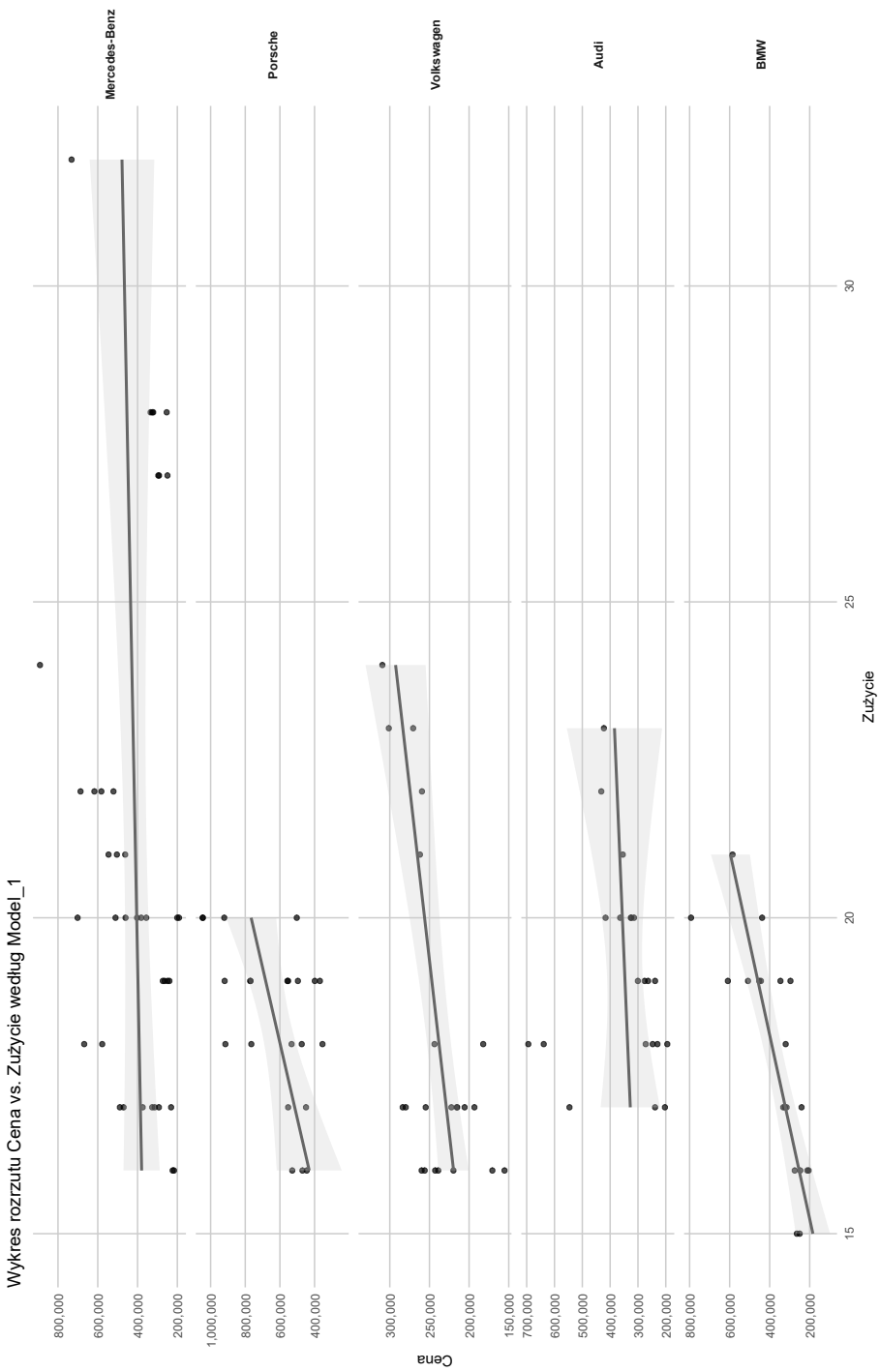
```
  cat("  \\centering\n")
  cat("  \\includegraphics[angle=0,
   ↪  width=\\linewidth]{output/plots/plot_", rnd.str[i], ".pdf}\n",
   ↪  sep = "")
  cat("  \\caption{", captions[[i]], "}\n", sep = "")
  cat("  \\label{fig:", rnd.str[i], "}\n", sep = "")
  cat("\\end{figure}\n")
  cat("\\end{landscape}\n\n")
}
```



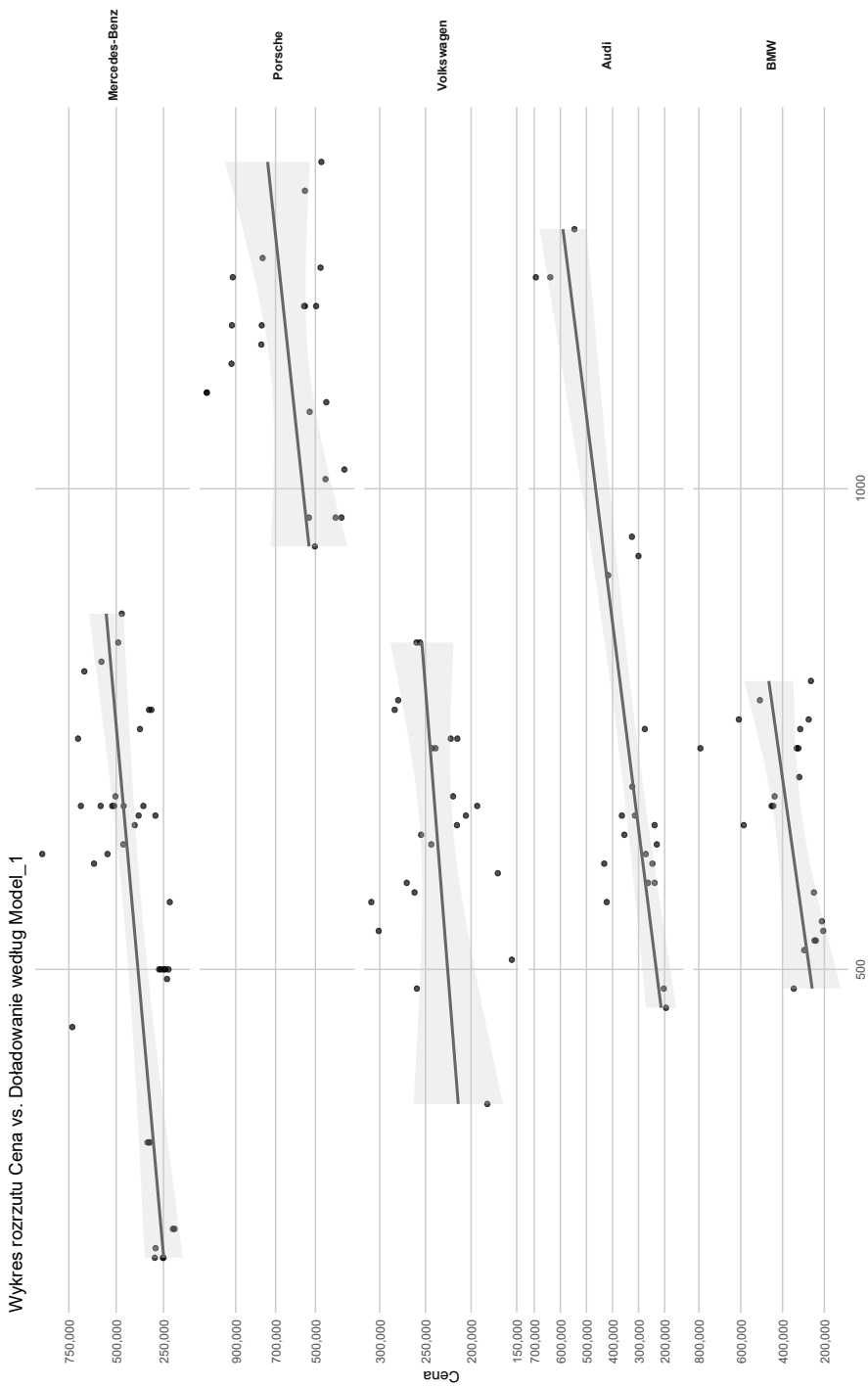
Rysunek 3.40. Wykres rozrzutu według Model_1: Zasięg



Rysunek 3.41. Wykres rozrzutu według Model_1: Bateria



Rysunek 3.42. Wykres rozrzutu według Model_1: Zużycie



Rysunek 3.43. Wykres rozrzutu według Model_1: Doładowanie

Analogicznie do funkcji `fun.2`, można wykonać grupowanie według modelu oraz marki, korzystając z funkcji `fun.4`.

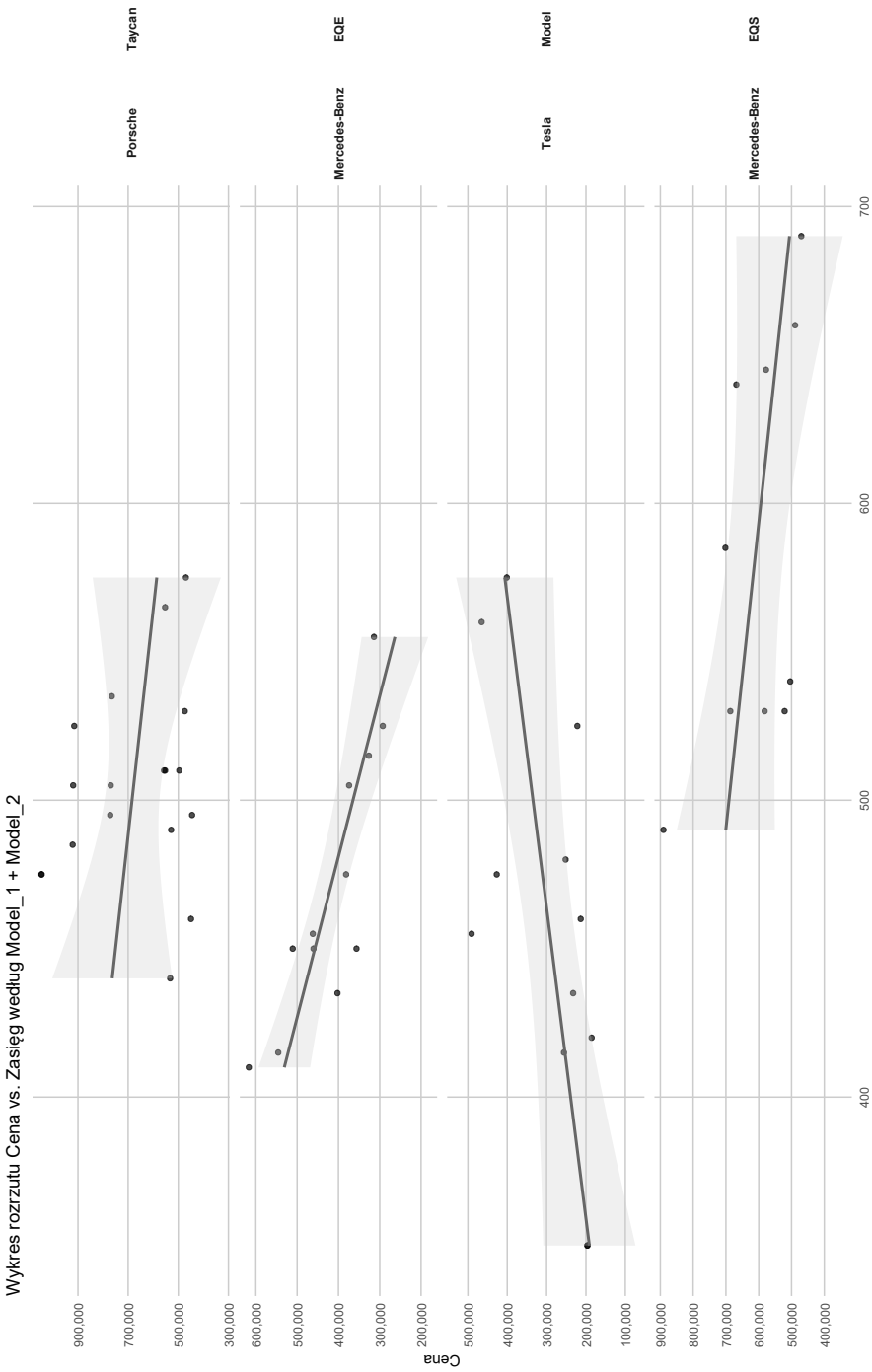
```
fun.4 <- function(data, x_col, y_col, n_min, ...) {
  arg <- list(...)
  data %>%
    group_by(Model_1, Model_2) %>%
    mutate(
      n = n()
    ) %>%
    subset(n >= n_min) %>%
    ungroup() %>%
    arrange(desc(n)) %>%
    mutate(
      Model_1 = factor(
        Model_1,
        levels = unique(Model_1)
      ),
      Model_2 = factor(
        Model_2,
        levels = unique(Model_2)
      )
    ) %>%
    ggplot(aes(x = !!sym(x_col), y = !!sym(y_col))) +
    geom_point(alpha = 0.7) +
    geom_smooth(
      method = "lm",
      se = TRUE,
      color = "grey40",
      fill = arg[[1]],
      alpha = 0.3
    ) +
    facet_grid(
      Model_2 + Model_1 ~ .,
      scales = "free"
    ) +
    scale_y_continuous(labels = scales::comma) +
    labs(
      title = paste(
        "Wykres rozrzutu",
        y_col,
        "vs.",
        x_col,
        "według Model_1 + Model_2"
      ),
      x = x_col,
      y = y_col
    ) +
    theme_minimal() +
```

```
    theme(
      text = element_text(family = "Arial"),
      strip.text = element_text(
        size = 9,
        face = "bold"
      ),
      strip.text.y = element_text(
        angle = 0,
        hjust = 0.5
      ),
      panel.spacing = unit(0.5, "lines"),
      panel.grid.major = element_line(
        color = "grey80"
      ),
      panel.grid.minor = element_blank()
    )
}
```

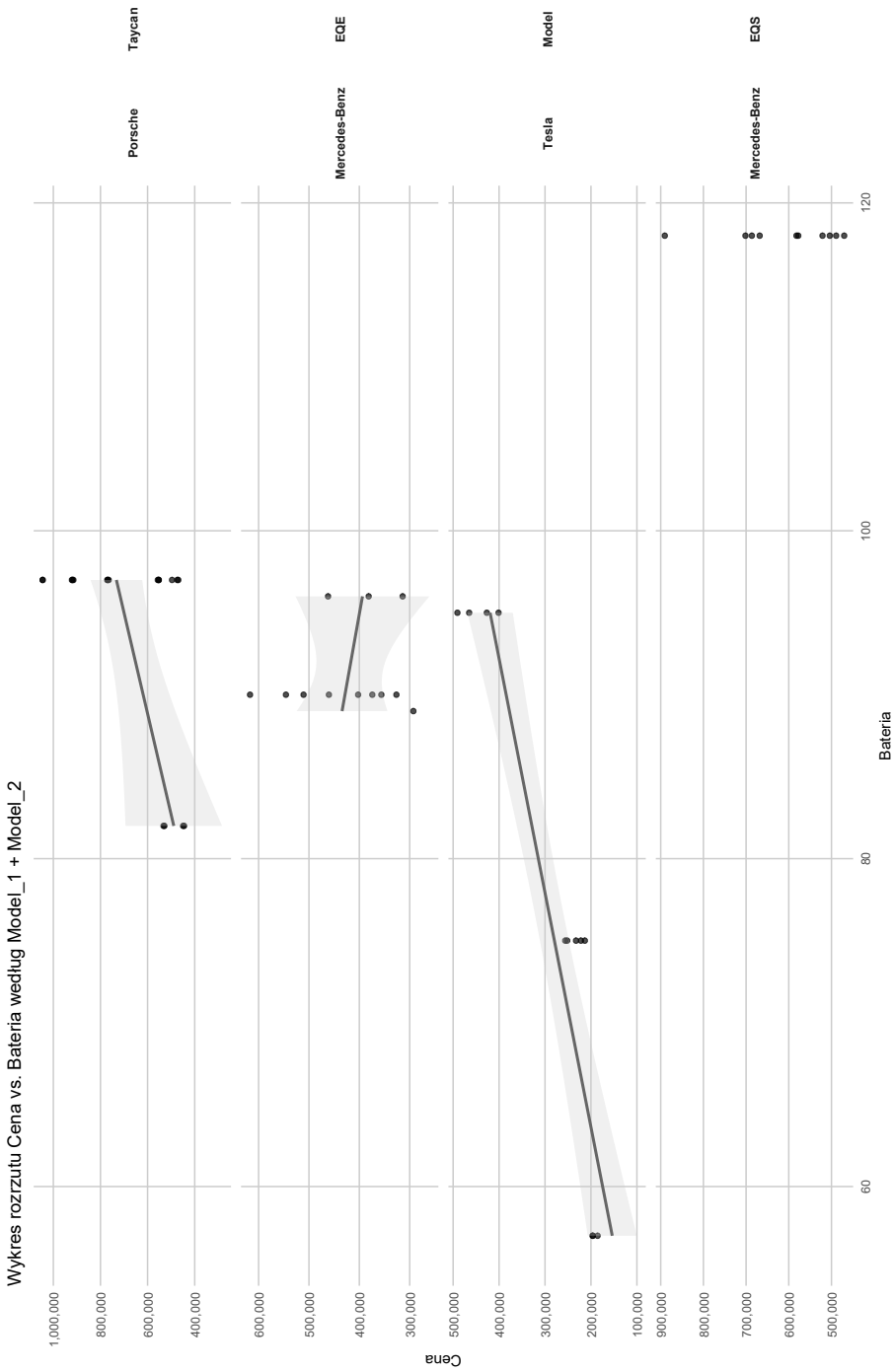
```
i.all <- seq_along(c("Zasięg", "Bateria", "Zużycie", "Doładowanie"))
rnd.str <- stri_rand_strings(
  n = length(i.all),
  length = 6,
  pattern = "[a-z]"
)
captions <- lapply(
  i.all,
  function(x) {
    var.name <- c("Zasięg", "Bateria", "Zużycie", "Doładowanie")[x]
    paste0("Wykres rozrzutu według Model\\_1 + Model\\_2: ", var.name)
  }
)
```

```
n <- 10
pl.temp <- list()
for (i in i.all) {
  pl.temp[[i]] <- data %>% fun.4(c("Zasięg", "Bateria", "Zużycie",
 ↪  "Doładowanie")[i], "Cena", n, "grey80")
  ggplot2::ggsave(
    paste0("output/plots/plot_", rnd.str[i], ".pdf"),
    plot = pl.temp[[i]],
    width = 14,
    height = 9,
    units = "in",
    device = grDevices::cairo_pdf
  )
}
```

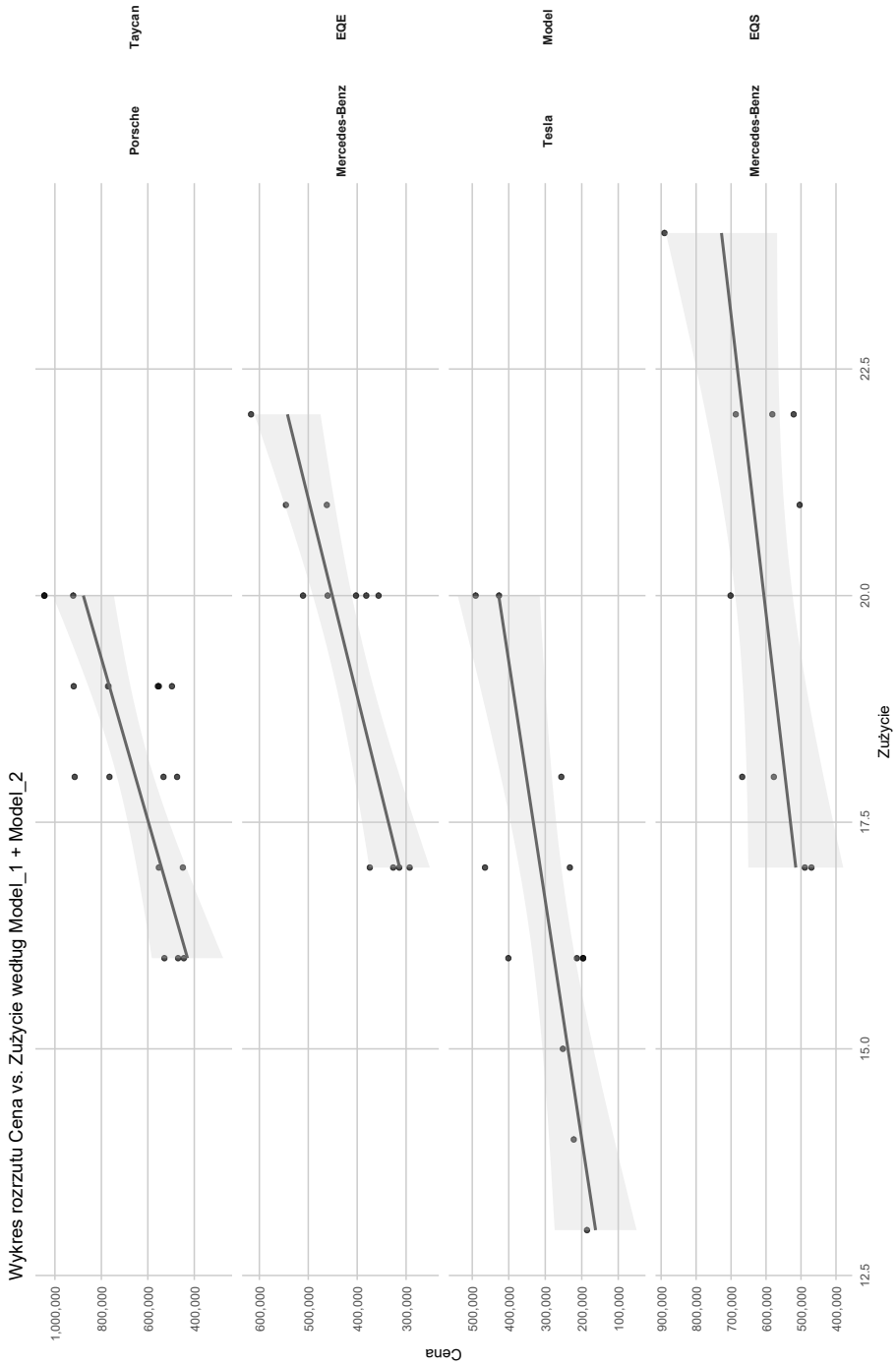
```
for (i in i.all) {
  cat("\\begin{landscape}\n")
  cat("\\begin{figure}[htbp]\n")
  cat("  \\centering\n")
  cat("  \\includegraphics[angle=0,
   ↪  width=\\linewidth]{output/plots/plot_", rnd.str[i], ".pdf}\n",
   ↪  sep = "")
  cat("  \\caption{", captions[[i]], "}\n", sep = "")
  cat("  \\label{fig:", rnd.str[i], "}\n", sep = "")
  cat("\\end{figure}\n")
  cat("\\end{landscape}\n\n")
}
```



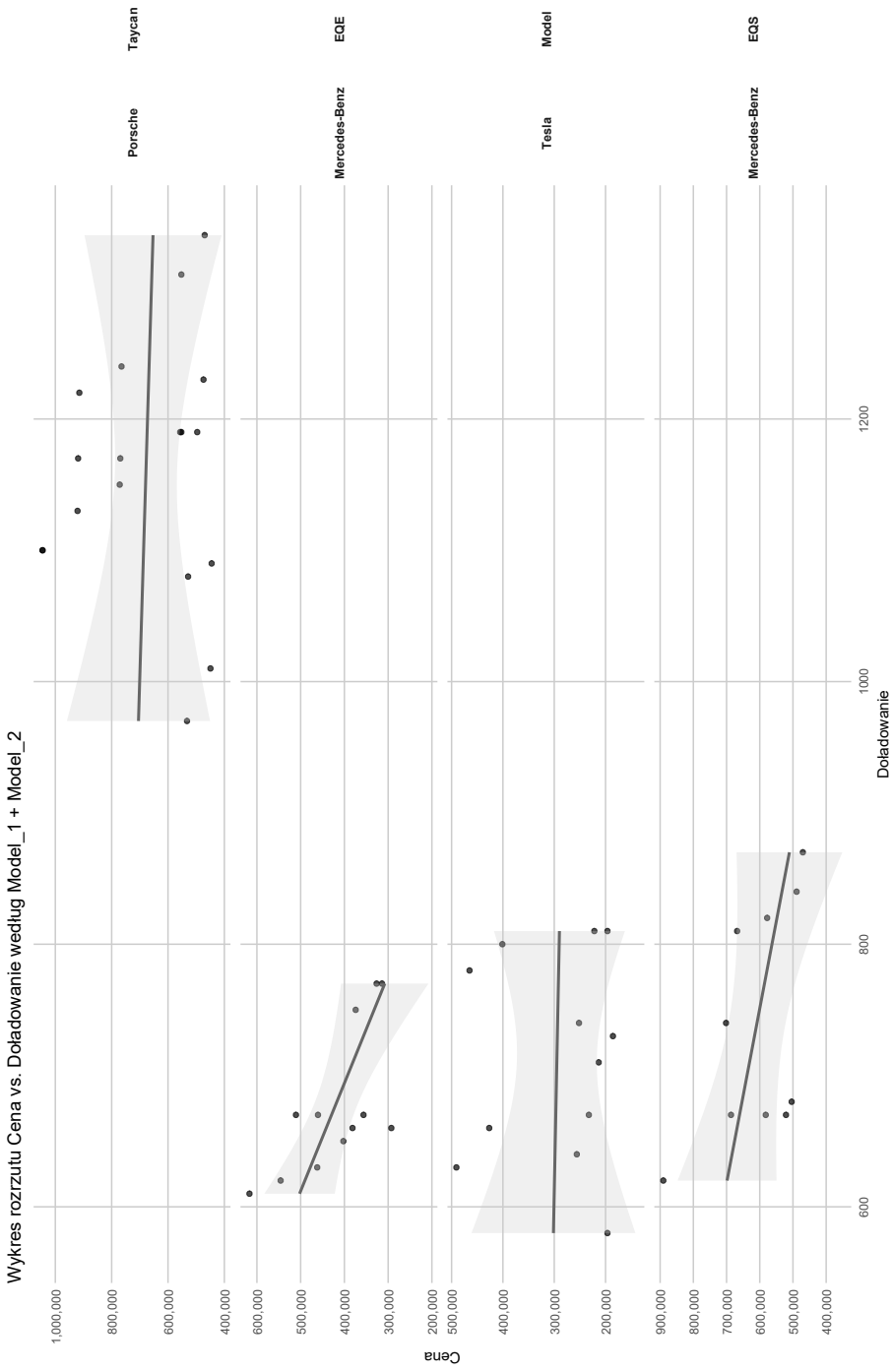
Rysunek 3.44. Wykres rozrzutu według Model_1 + Model_2: Zasięg



Rysunek 3.45. Wykres rozrzutu według Model_1 + Model_2: Bateria



Rysunek 3.46. Wykres rozrzutu według Model_1 + Model_2: Zużycie



Rysunek 3.47. Wykres rozrzutu według Model_1 + Model_2: Doładowanie

W zbiorze `data` każdy model samochodu elektrycznego ma pięć atrybutów ilościowych: `Zasięg`, `Bateria`, `Zużycie`, `Doładowanie` i `Cena`. Można wyznaczyć kombinację liniową tych cech i w ten sposób nadać samochodom ranking. Takie zadanie można sformułować w postaci problemu optymalizacji ważonej kombinacji cech (tzw. portfela cech) z ograniczeniami, który stanowi szczególny przypadek bardziej ogólnych problemów programowania kwadratowego. Zastosowana zostanie funkcja `solve.QP` z biblioteki `quadprog` (Turlach, 2019) w celu numerycznego rozwiązania problemu.

Zagadnienie programowania kwadratowego ma postać (Boyd i Vandenberghe, 2004; Nocedal i Wright, 2006):

$$\min_x \left(\frac{1}{2} x' D x - d' x \right) \tag{3.92}$$

przy warunkach:

$$A_{eq} x = b_{eq} \tag{3.93}$$

dla l ograniczeń równościowych oraz:

$$A_{ineq} x \geq b_{ineq} \tag{3.94}$$

dla m ograniczeń nierównościowych, gdzie:

D – macierz $N \times N$,
x i d – wektory $N \times 1$,
A_{eq} – macierz $l \times N$,
b_{eq} – wektor $l \times 1$,
A_{ineq} – macierz $m \times N$,
b_{ineq} – wektor $m \times 1$,
N – wymiar zmiennej decyzyjnej x.

Funkcja `solve.QP` otrzymuje jako argumenty macierze i wektory: D, d, A_{eq}, b_{eq}, A_{ineq} i b_{ineq}.

```
solve.QP(
  Dmat,
  dvec,
  Amat,
  bvec,
  meq = l,
  factorized = FALSE
)
```

Argumenty `Dmat` i `dvec` odpowiadają macierzy D i wektorowi d. Funkcja zakłada, że macierze A_{eq}, A_{ineq} oraz wektory b_{eq}, b_{ineq} są łączone w jedną macierz A, $(l+m) \times N$ oraz jeden wektor b, $(l+m) \times 1$, w postaci:

$$A = \begin{bmatrix} A_{eq} \\ A_{ineq} \end{bmatrix}, \; b = \begin{bmatrix} b_{eq} \\ b_{ineq} \end{bmatrix} \tag{3.95}$$

W funkcji `solve.QP`, `Amat` reprezentuje macierz A', natomiast `bvec` – wektor b. Argument `meq` określa liczbę ograniczeń równościowych.

Przedstawiony problem optymalizacyjny sprowadza się do znalezienia optymalnego *portfela cech* samochodów elektrycznych. Jest on w pełni analogiczny do klasycznego zadania budowy *portfela akcji* o minimalnej wariancji, w którym ryzyko opisuje się za pomocą wyrażenia $m'\Sigma m$. Tę właśnie miarę ryzyka – oznaczoną $\sigma^2_{p,m} = m'\Sigma m$ – otrzymuje się z ogólnej funkcji celu zadania programowania kwadratowego po podstawieniu: $x = m$, $D = 2\Sigma$ oraz $d = (0, \dots, 0)$. Wówczas:

$$\frac{1}{2} x' D x - d' x = m' \Sigma m = \sigma^2_{p,m}. \tag{3.96}$$

Występuje jedno ograniczenie równościowe $m'1 = 1$ oraz N ograniczeń nierównościowych $m_i \geq 0$ dla $i = 1, \dots, N$, które można wyrazić jako $m \geq 0$. Ograniczenia te można wyrazić w postaci:

$$\underset{(1\times N)}{A_{eq}} = 1', \; \underset{(1\times 1)}{b_{eq}} = 1, \tag{3.97}$$

$$\underset{(N\times N)}{A_{ineq}} = I_N, \underset{(N\times 1)}{b_{ineq}} = (0, \dots, 0), \tag{3.98}$$

w taki sposób, że:

$$A = \begin{bmatrix} 1' \\ I_N \end{bmatrix}, \; b = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{3.99}$$

W tym przypadku:

$$A_{eq} m = 1' m = 1 = b_{eq}, \tag{3.100}$$

$$A_{ineq} m = I_N m = m \geq 0 = b_{ineq}. \tag{3.101}$$

Poniżej podano rozwiązanie problemu, który sprowadza się do minimalizacji formy kwadratowej przy liniowych ograniczeniach. W pierwszym kroku wczytano odpowiednie biblioteki niezbędne do rozwiązania zadania.

```
library(kableExtra)
library(knitr)
library(pracma)
library(quadprog)
```

Następnie wykonano normalizację cech na przedziale $\langle 0, 1 \rangle$ na podstawie funkcji `std`.

```
std <- function(x) {
  (x - min(x)) / (max(x) - min(x))
}
```

Ramka danych `data.std` zawiera znormalizowane wartości cech.

```
data.std <- data %>%
  na.omit %>%
  mutate_if(
    is.numeric,
    function(x) std(x)
  )
head(data.std)
```

```
## # A tibble: 6 x 9
##   Model_1 Model_2 Model_3     Dostępny Zasięg Bateria Zużycie
 ↪  Doładowanie   Cena
##   <chr>   <chr>   <chr>       <chr>     <dbl>   <dbl>   <dbl>
 ↪  <dbl>  <dbl>
## 1 Abarth  500e    Convertible 05-2023   0.162   0.165   0.158
 ↪  0.171 0.0559
## 2 Abarth  500e    Hatchback   05-2023   0.162   0.165   0.158
 ↪  0.171 0.0483
## 3 Aiways  U5      -           04-2022   0.324   0.402   0.316
 ↪  0.179 0.0595
## 4 Aiways  U6      -           12-2022   0.387   0.402   0.211
 ↪  0.222 0.0785
## 5 Alfa    Romeo   Junior Ele~ 04-2024   0.333   0.299   0.105
 ↪  0.291 0.0540
## 6 Alfa    Romeo   Junior Ele~ 06-2024   0.315   0.299   0.158
 ↪  0.282 0.0785
```

Celem jest wyznaczenie rankingu modeli samochodów na podstawie kombinacji liniowej cech przy założeniu, że ranking jest:

- rosnącą funkcją zmiennych `Zasięg` i `Doładowanie`,
- malejącą funkcją zmiennych `Bateria`, `Zużycie` i `Cena`.

W związku z tym przyjęto poniższe założenia.

```
data.std.1 <- data.std %>%
  mutate(
    Bateria = 1 - Bateria,
    Zużycie = 1 - Zużycie,
    Cena = 1 - Cena
  )
head(data.std.1)
```

```
## # A tibble: 6 x 9
##   Model_1 Model_2 Model_3      Dostępny Zasięg Bateria Zużycie
 ↪  Doładowanie  Cena
##   <chr>   <chr>   <chr>        <chr>     <dbl>   <dbl>   <dbl>
 ↪  <dbl> <dbl>
## 1 Abarth  500e    Convertible  05-2023   0.162   0.835   0.842
 ↪  0.171 0.944
## 2 Abarth  500e    Hatchback    05-2023   0.162   0.835   0.842
 ↪  0.171 0.952
## 3 Aiways  U5      -            04-2022   0.324   0.598   0.684
 ↪  0.179 0.941
## 4 Aiways  U6      -            12-2022   0.387   0.598   0.789
 ↪  0.222 0.922
## 5 Alfa    Romeo   Junior Elet~ 04-2024   0.333   0.701   0.895
 ↪  0.291 0.946
## 6 Alfa    Romeo   Junior Elet~ 06-2024   0.315   0.701   0.842
 ↪  0.282 0.922
```

Założenia oznaczają, że:

$$\text{rank} = f(\overset{+}{\text{X1}}, \overset{-}{\text{X2}}, \overset{-}{\text{X3}}, \overset{+}{\text{X4}}, \overset{-}{\text{X5}})$$

gdzie:

$rank$ – ranking (w punktach),
$X1$ – zasięg (w km),
$X2$ – pojemność baterii (w kWh),
$X3$ – zużycie energii elektrycznej (w kWh/100 km),
$X4$ – wartość doładowania energii elektrycznej (w km/h),
$X5$ – cena (w złotych).

Najpierw należy wyznaczyć macierz wariancji-kowariancji V.

```
V <- data.std.1 %>%
  .[, 5:9] %>%
  cov()
```

Następnie macierz D, wektor d, macierz A i wektor b.

```
options(scipen = 999)
Dmat <- 2 * V
dvec <- rep(0, ncol(V))
Amat <- cbind(rep(1, ncol(V)), diag(ncol(V)))
bvec <- c(1, rep(0, ncol(V)))
Dmat
```

```
##                   Zasięg     Bateria     Zużycie Doładowanie
 ↪  Cena
## Zasięg       0.064321149 -0.06528984 0.007304121  0.05758975
 ↪  -0.02190688
## Bateria     -0.065289836  0.08611359 0.020853188 -0.05514907
 ↪  0.03148197
## Zużycie      0.007304121  0.02085319 0.045502333  0.01103001
 ↪  0.00989425
## Doładowanie  0.057589746 -0.05514907 0.011030006  0.09191760
 ↪  -0.02536168
## Cena        -0.021906884  0.03148197 0.009894250 -0.02536168
 ↪  0.02381737
```

```
dvec
```

```
## [1] 0 0 0 0 0
```

```
Amat
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    1    0    0    0    0
## [2,]    1    0    1    0    0    0
## [3,]    1    0    0    1    0    0
## [4,]    1    0    0    0    1    0
## [5,]    1    0    0    0    0    1
```

```
bvec
```

```
## [1] 1 0 0 0 0 0
```

Zastosowanie funkcji `solve.QP` prowadzi do uzyskania następującego rozwiązania optymalnego:

```
weight <- solve.QP(Dmat, dvec, Amat, bvec, meq = 1) %>%
  .$solution
weight %>% round(2)
```

```
## [1] 0.49 0.37 0.00 0.00 0.14
```

Łatwo sprawdzić, że suma wag jest równa 1.

```
sum(weight)
```

```
## [1] 1
```

Współrzędne wektora `weight` oznaczają wartości współczynników kombinacji liniowej. Można zauważyć, że w rankingu znaczenie mają zmienne: `Zasięg`, `Bateria` oraz `Cena`. Do wyznaczenia rankingu `rank` służy macierz wartości cech `variables` oraz wektor wag `weight`.

```
variables <- data.std.1 %>%
  .[, 5:9] %>%
  as.matrix()
```

```
rank <- variables %*% weight %>%
  {.*100} %>%
  round(2)
head(rank)
```

```
##       [,1]
## [1,] 52.18
## [2,] 52.29
## [3,] 51.32
## [4,] 54.12
## [5,] 55.64
## [6,] 54.41
```

Wartości statystyk opisowych dla zmiennej `rank` są następujące:

```
summary(rank)
```

```
##        V1
##  Min.   :28.97
##  1st Qu.:48.95
##  Median :52.54
##  Mean   :51.64
##  3rd Qu.:54.77
##  Max.   :63.71
```

Ranking został dołączony w postaci kolumny `Rank` do ramki danych `data`.

```
data.rank <- data %>%
  na.omit() %>%
  cbind(Rank = rank) %>%
  arrange(desc(Rank))
head(data.rank[, c(1:3, 10)])
```

```
##         Model_1 Model_2                  Model_3  Rank
## 1         Tesla   Model 3 Long Range Dual Motor 63.71
## 2         Tesla   Model                       3 61.63
## 3       Hyundai   IONIQ        6 Long Range 2WD 61.49
## 4         Lucid     Air                 Touring 60.22
## 5           BMW      i4                eDrive40 60.17
## 6 Mercedes-Benz     EQS                    450+ 59.64
```

Zbiór danych `data.rank` można zmodyfikować w taki sposób, aby ranking podzielić na klasy. W tym celu można wykorzystać funkcję `quantile`, aby obliczyć kwantyle zmiennej `Rank`.

```
q <- data.rank %>%
  .$Rank %>%
  quantile()
q
```

```
##     0%    25%    50%    75%   100%
## 28.970 48.955 52.540 54.770 63.710
```

Do przydziału klas wykorzystano funkcję `cut`, która służy do podziału zmiennych ciągłych na przedziały (`bins`), co pozwala na ich grupowanie i analizę. Argumenty takie jak `breaks` umożliwiają określenie liczby przedziałów lub ich dokładnych granic, a `labels` na przypisanie nazwy do każdego przedziału. Funkcja obsługuje argumenty takie jak `include.lowest`, tj. czy dolna granica ma być włączona do pierwszego przedziału i `right` (czy przedziały są prawostronnie domknięte), co daje dużą elastyczność w definiowaniu zakresów. Wynik funkcji `cut` zapisano do ramki `data.rank.1`.

```
data.rank.1 <- data.rank %>%
  mutate(
    Class = cut(
      Rank,
      breaks = q,
      labels = 4:1,
      include.lowest = TRUE
    )
  )
head(data.rank.1[, c(1:3, 11)])
```

```
##         Model_1 Model_2                 Model_3 Class
## 1         Tesla   Model 3 Long Range Dual Motor     1
## 2         Tesla   Model                       3     1
## 3       Hyundai   IONIQ        6 Long Range 2WD     1
## 4         Lucid     Air                 Touring     1
## 5           BMW      i4                eDrive40     1
## 6 Mercedes-Benz     EQS                    450+     1
```

Można teraz przystąpić do analizy zbioru `data.rank.1`. Na początku utworzona została funkcja `fun.5`, której zadaniem jest pokazanie wykresu histogramów zmiennej według klasy `Class`.

```
fun.5 <- function(data, col, ...) {
  arg <- list(...)
  data %>%
  mutate(
    Class = factor(
      Class,
      levels = sort(unique(Class), decreasing = TRUE)
    )
  ) %>%
    group_by(Class) %>%
    mutate(
      mean = mean(!!sym(col)),
      median = median(!!sym(col)),
      n = n()
    ) %>%
    ungroup() %>%
    ggplot(aes(x = !!sym(col))) +
    geom_histogram(
      fill = arg[[1]],
      color = "black",
      alpha = 0.7,
      bins = 30
    ) +
    geom_vline(
```

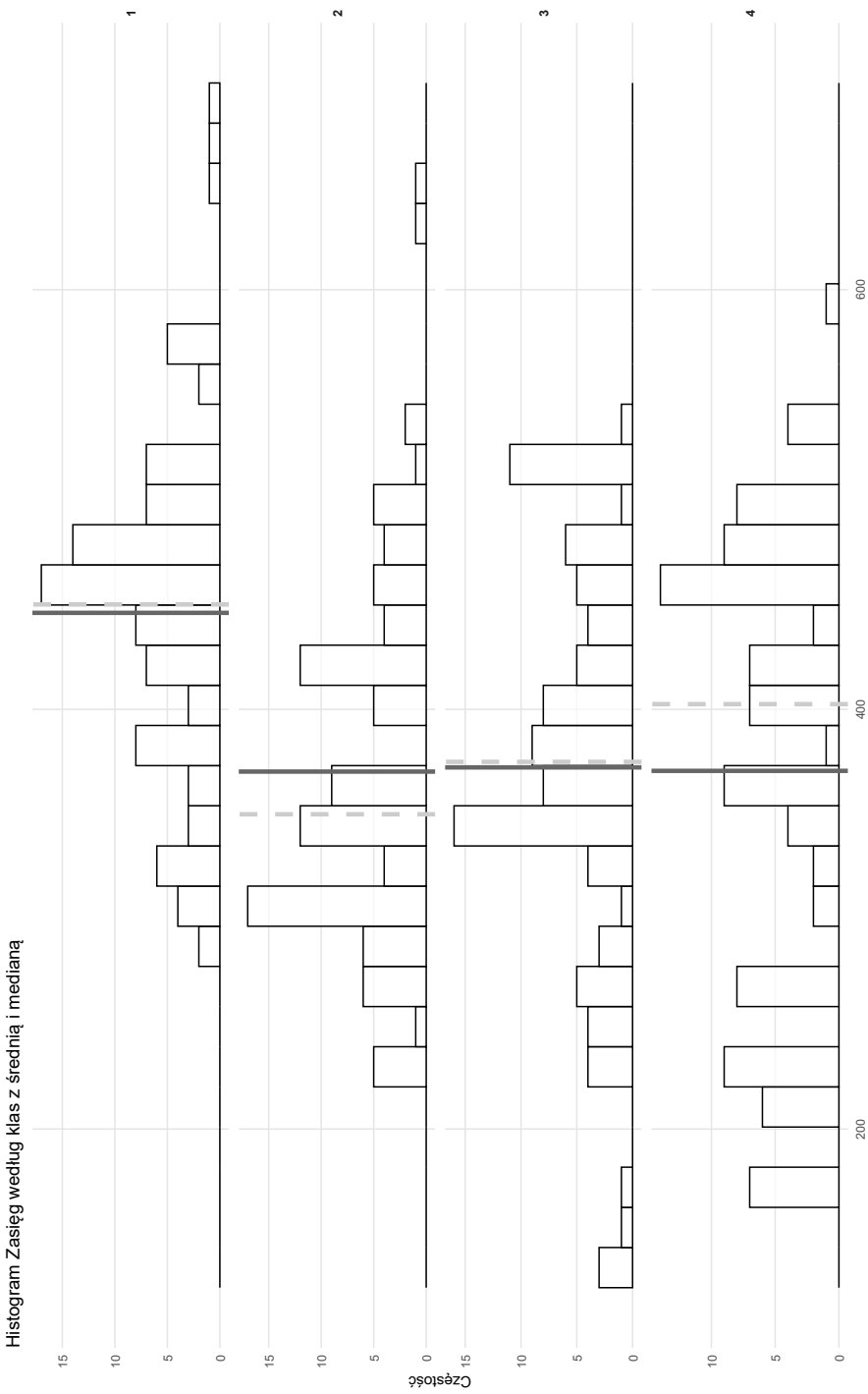
```
      aes(xintercept = mean),
      color = "grey40",
      linetype = "solid",
      linewidth = 1.5
    ) +
    geom_vline(
      aes(xintercept = median),
      color = "grey80",
      linetype = "dashed",
      linewidth = 1.5
    ) +
    facet_grid(
      Class ~ .,
      scales = "free_y"
    ) +
    labs(
      title = paste(
        "Histogram",
        col,
        "według klas z średnią i medianą"
      ),
      x = col,
      y = "Częstość"
    ) +
    theme_minimal() +
    theme(
      text = element_text(family = "Arial"),
      strip.text = element_text(
        size = 9,
        face = "bold"
      ),
      strip.text.y = element_text(
        angle = 0,
        hjust = 0.5
      ),
      strip.placement = "outside",
      panel.spacing = unit(0.5, "lines"),
      panel.grid.major = element_line(
        color = "grey90"
      ),
      panel.grid.minor = element_blank()
    )
}
```

Funkcję `fun.5` można wywołać w pętli w poniższy sposób.

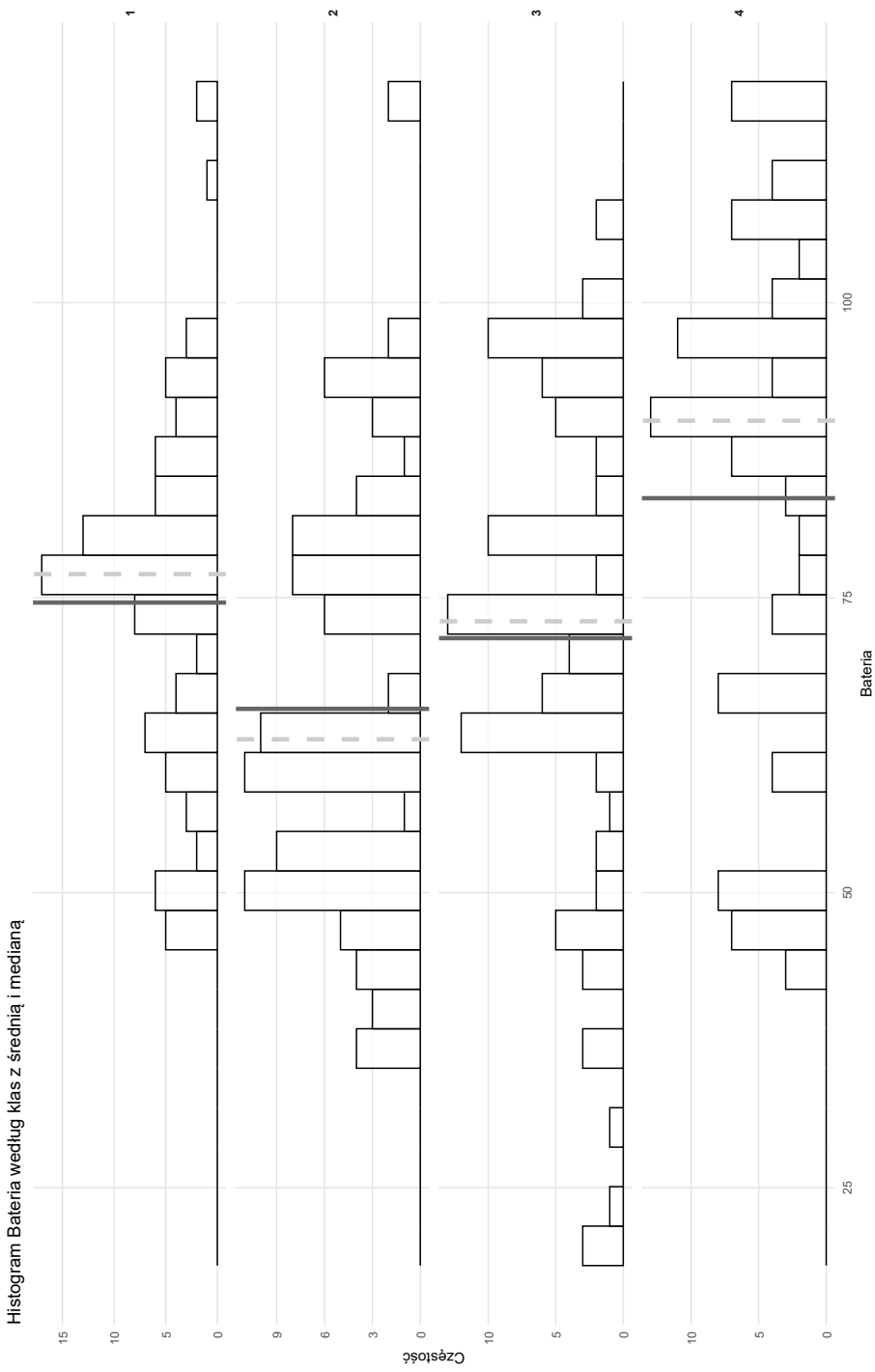
```
i.all <- seq_along(c("Zasięg", "Bateria", "Zużycie", "Doładowanie",
 ↪  "Cena"))
rnd.str <- stri_rand_strings(
  n = length(i.all),
  length = 6,
  pattern = "[a-z]"
)
captions <- lapply(
  i.all,
  function(x) {
    var.name <- c("Zasięg", "Bateria", "Zużycie", "Doładowanie",
 ↪  "Cena")[x]
    paste0("Histogram według klas z średnią i medianą: ", var.name)
  }
)
```

```
pl.temp <- list()
for (i in i.all) {
  pl.temp[[i]] <- data.rank.1 %>% fun.5(c("Zasięg", "Bateria",
 ↪  "Zużycie", "Doładowanie", "Cena")[i], "white")
  ggplot2::ggsave(
    paste0("output/plots/plot_", rnd.str[i], ".pdf"),
    plot = pl.temp[[i]],
    width = 14,
    height = 9,
    units = "in",
    device = grDevices::cairo_pdf
  )
}
```

```
for (i in i.all) {
  cat("\\begin{landscape}\n")
  cat("\\begin{figure}[htbp]\n")
  cat("  \\centering\n")
  cat("  \\includegraphics[angle=0,
   ↪  width=\\linewidth]{output/plots/plot_", rnd.str[i], ".pdf}\n",
   ↪  sep = "")
  cat("  \\caption{", captions[[i]], "}\n", sep = "")
  cat("  \\label{fig:", rnd.str[i], "}\n", sep = "")
  cat("\\end{figure}\n")
  cat("\\end{landscape}\n\n")
}
```

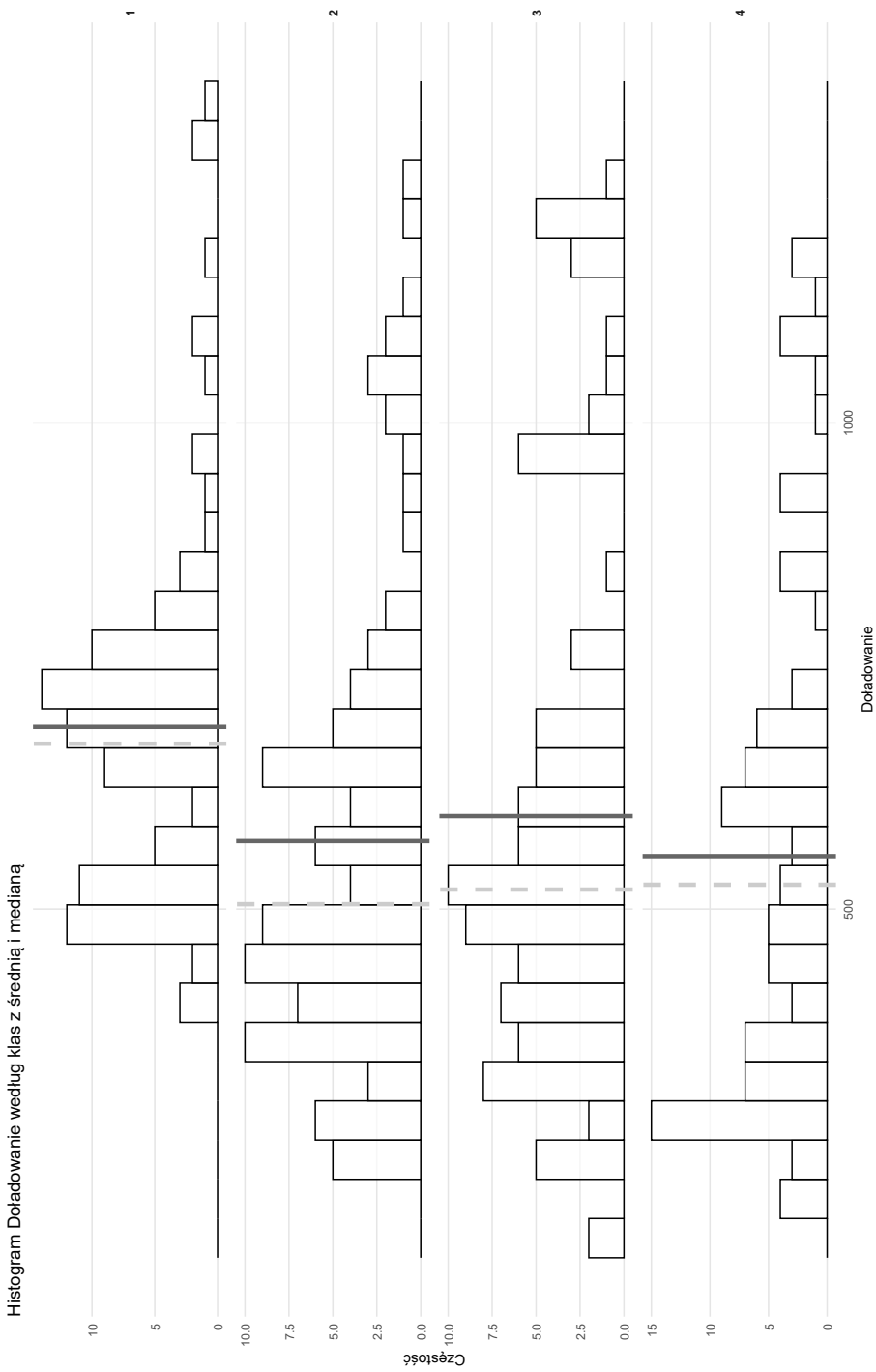


Rysunek 3.48. Histogram według klas z średnią i medianą: Zasięg

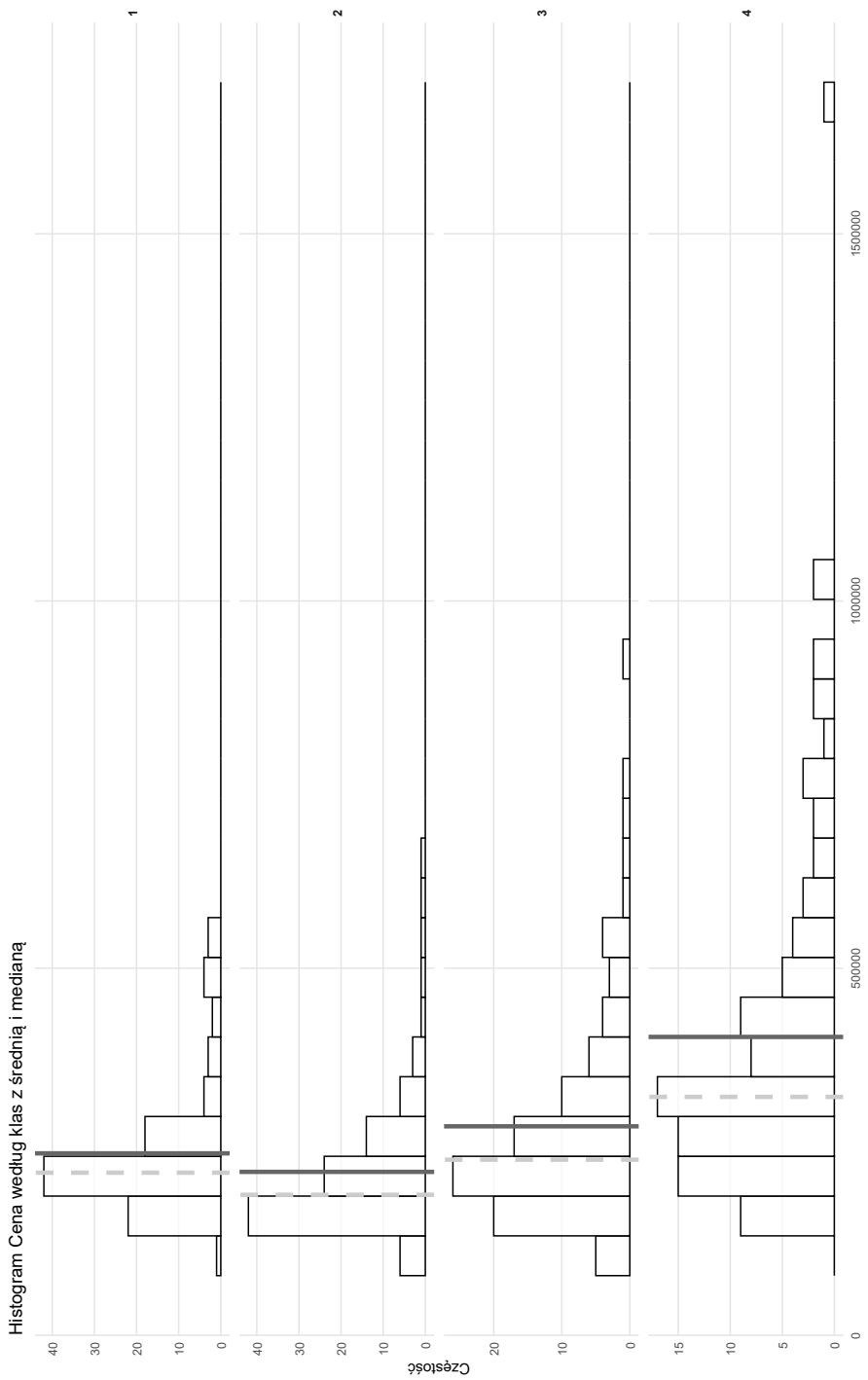


Rysunek 3.49. Histogram według klas z średnią i medianą: Bateria

Rysunek 3.50. Histogram według klas z średnią i medianą: Zużycie



Rysunek 3.51. Histogram według klas z średnią i medianą: Doładowanie



Rysunek 3.52. Histogram według klas z średnią i medianą: Cena

Łatwo zauważyć, że zasięg jest zdecydowanie największy w klasie 1. Wielkość baterii w klasie 1 nie ustępuje wielkościom w klasach aut drogich (3 i 4), a zużycie energii jest wyraźnie mniejsze niż w klasie aut drogich. Szybkość doładowania baterii również dominuje w klasie 1, a cena samochodów w tej klasie jest wysoce konkurencyjna i tylko nieznacznie, przeciętnie wyższa od samochodów w klasie 2. Jeśli przyjmie się założenia utworzonego rankingu, pierwszych dziesięć miejsc zajmują następujące samochody:

```
data.rank.1 %>%
  .[1:10, c(1:4, 10)]
```

```
##          Model_1 Model_2                 Model_3 Dostępny  Rank
## 1          Tesla   Model 3 Long Range Dual Motor  09-2023 63.71
## 2          Tesla   Model                       3  09-2023 61.63
## 3        Hyundai   IONIQ       6 Long Range 2WD  12-2022 61.49
## 4          Lucid     Air                 Touring  05-2023 60.22
## 5            BMW      i4                eDrive40  06-2024 60.17
## 6  Mercedes-Benz     EQS                    450+  04-2024 59.64
## 7          Tesla   Model          3 Performance  04-2024 59.47
## 8          Lucid     Air                Pure RWD  02-2024 59.45
## 9     Volkswagen    ID.7                   Pro S  06-2024 59.23
## 10         Skoda   Enyaq                Coupe 85  09-2023 59.00
```

Rozkład zmiennej Rank podano na rys. 3.53.

```
## [[1]]
```

Można również zauważyć, że samochody w klasach 2, 3 i 4 mają znacząco niższy ranking niż samochody w klasie 1. W klasie 1 występują samochody o nietypowo wysokim rankingu, natomiast w klasie 4 auta o nietypowo niskim rankingu. Pierwsze miejsce w rankingu zajmuje:

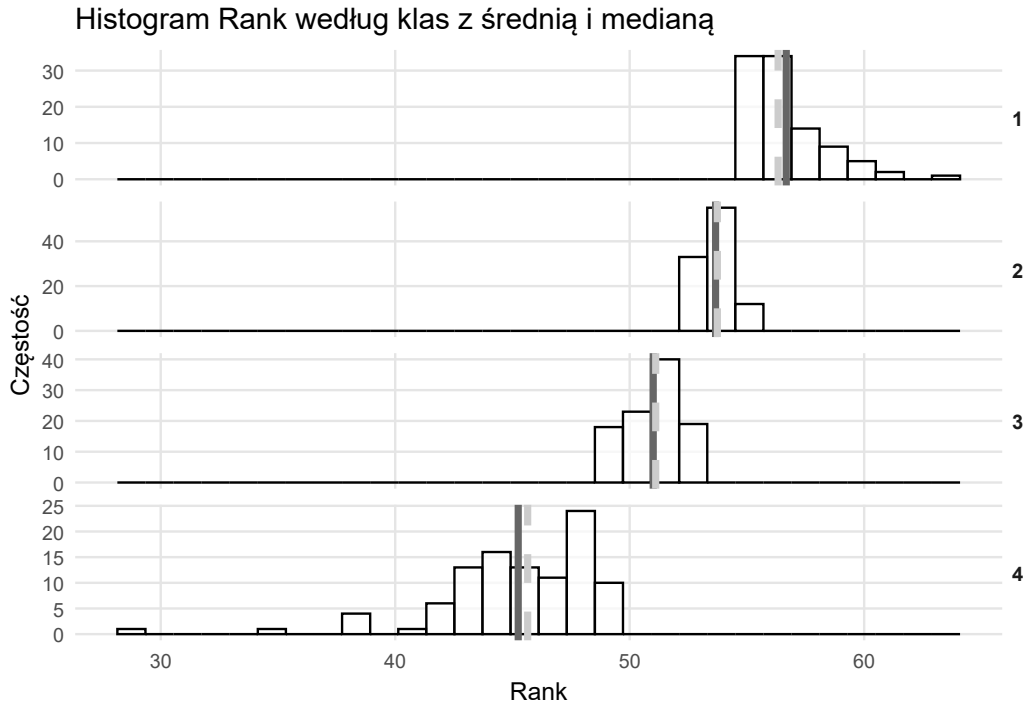
- Tesla Model 3 Long Range Dual Motor

natomiast miejsce ostatnie:

- Mercedes-Benz G 580.

Cena pierwszego auta w rankingu stanowi 30.32% ceny auta ostatniego, natomiast ranking stanowi odpowiednio 219.92%.

Tablica 3.24 przedstawia 10 producentów o najwyższym rankingu (klasa 1) z największą liczbą modeli samochodów.



Rysunek 3.53. Histogram według klas z średnią i medianą: Ranking

```
data.rank.1 %>%
  subset(Class == 1) %>%
  group_by(Model_1) %>%
  summarize(
    n = n(),
    .groups = 'drop'
  ) %>%
  arrange(desc(n)) %>%
  head(10) %>%
  kable(
    caption = "Producenci w klasie 1 z największą liczbą modeli
     ↪  samochodów\\label{tab:hchoky}",
    col.names = c("Marka", "N"),
    format = "latex"
  ) %>%
  kableExtra::column_spec(1, width = "4cm") %>%
  kableExtra::column_spec(2, width = "2cm")
```

Tablica 3.25 przedstawia 10 producentów o najniższym rankingu (klasa 4) z największą liczbą modeli samochodów.

Tablica 3.24. Producenci w klasie 1 z największą liczbą modeli samochodów

Marka	N
Volkswagen	14
Mercedes-Benz	9
Tesla	9
BMW	8
Polestar	8
Skoda	6
Hyundai	5
CUPRA	4
Porsche	4
Renault	4

```
data.rank.1 %>%
  subset(Class == 4) %>%
  group_by(Model_1) %>%
  summarize(
    n = n(),
    .groups = 'drop'
  ) %>%
  arrange(desc(n)) %>%
  head(10) %>%
  kable(
    caption = "Producenci w klasie 4 z największą liczbą modeli
     ↪  samochodów\\label{tab:salyae}",
    col.names = c("Marka", "N"),
    format = "latex"
  ) %>%
  kableExtra::column_spec(1, width = "4cm") %>%
  kableExtra::column_spec(2, width = "2cm")
```

Tablica 3.26 – w formacie *landscape* – przedstawia modele o najwyższym rankingu (klasa 1), w cenie poniżej 200 tys. zł, uszeregowane rosnąco według zużycia energii.

```
data.rank.1 %>%
  subset(Class == 1 & Cena <= 200000) %>%
  select(-c(Class, Rank, Dostępny)) %>%
  arrange(Zużycie) %>%
  kable(
    caption = "Modele o najwyższym rankingu (klasa 1) -- posortowane
     ↪  rosnąco według zużycia energii -- z ceną poniżej 200 tys.
     ↪  zł\\label{tab:zyimhe}",
    format = "latex"
  ) %>%
```

Tablica 3.25. Producenci w klasie 4 z największą liczbą modeli samochodów

Marka	N
Mercedes-Benz	21
Porsche	7
Citroen	6
Opel	6
Peugeot	6
Toyota	5
Volkswagen	5
Audi	4
Kia	4
Lotus	4

```
kable_styling(
  latex_options = c("scale_down"),
  font_size = 8,
  position = "center"
) %>%
landscape()
```

Tablica 3.26. Modele o najwyższym rankingu (klasa 1) – posortowane rosnąco według zużycia energii – z ceną poniżej 200 tys. zł

Model_1	Model_2	Model_3	Zasięg	Bateria	Zużycie	Doładowanie	Cena
Tesla	Model	3	420	57	13	730	185606
Hyundai	IONIQ	6 Standard Range 2WD	335	50	14	780	191579
Renault	Megane	ETech EV60 220hp	380	60	15	530	183472
Renault	Megane	ETech EV60 130hp	380	60	15	530	184326
Citroen	e-C4	54 kWh	330	50	15	470	160858
Citroen	e-C4	X 54 kWh	330	50	15	470	163418
Opel	Corsa	Electric 51 kWh	315	48	15	490	162138
Alfa	Romeo	Junior Elettrica 54 kWh	320	50	15	510	167685
Lancia	Ypsilon	-	310	48	15	480	160858
Opel	Astra	Electric	320	50	15	510	174939
Peugeot	e-208	51 kWh	310	48	15	480	170672
Hyundai	INSTER	Long Range	295	46	15	420	113070
Opel	Corsa	Electric 50 kWh	295	46	15	470	148485
CUPRA	Born	170 kW 77 kWh	460	77	16	710	196699
Polestar	2	Standard Range Single Motor	405	67	16	480	194566
Kia	Niro	EV	385	64	16	390	182619
Hyundai	Kona	Electric 65 kWh	390	65	16	480	194566
CUPRA	Born	170 kW 59 kWh	360	59	16	630	168539
Volkswagen	ID.3	Pro	360	59	16	600	170672
Tesla	Model	Y	350	57	16	810	196273
Tesla	Model	Y	350	57	16	580	196273
Renault	5	ETech 52kWh 150hp	320	52	16	400	140378
Volkswagen	ID.3	Pure	320	52	16	510	155312
Fiat	600e	-	310	50	16	500	154458
BYD	SEAL	82.5 kWh RWD Design	480	82	17	530	194139
Kia	EV3	Long Range	455	78	17	570	176646
Skoda	Elroq	85	450	77	17	670	186886
Volkswagen	ID.3	Pro S	450	77	17	670	193286
MG	MG4	Electric 77 kWh	425	74	17	590	186032
Mini	Countryman	E	380	64	17	530	185606
MG	MG4	Electric 64 kWh	360	61	17	630	161712

Przedstawiona wcześniej analiza statystyczna ukazuje szerokie możliwości analityczne języka R. Warto przeprowadzić również wielowymiarową analizę ekonometryczną, tj. oszacować parametry przekrojowego modelu liniowego:

$$y_i = \alpha_0 + \alpha_1 X_{1i} + \alpha_2 X_{2i} + \alpha_3 X_{3i} + \alpha_4 X_{4i} + \epsilon_i \tag{3.102}$$

gdzie:

y – cena (w złotych),
X_1 – zasięg (w km),
X_2 – pojemność baterii (w kWh),
X_3 – zużycie energii elektrycznej (w kWh/100 km),
X_4 – szybkość uzupełniania zasięgu (w km/h),
ϵ – składnik losowy,
i – obiekt.

Wynik analizy regresji zapisano do obiektu (listy) `reg.1`.

```
reg.1 <- data.rank.1 %>%
  lm(Cena ~ Zasięg+Bateria+Zużycie+Doładowanie, data = .)
summary(reg.1)
```

```
##
## Call:
## lm(formula = Cena ~ Zasięg + Bateria + Zużycie + Doładowanie,
##     data = .)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -225654  -59777   -6902   36892 1198505
##
## Coefficients:
##              Estimate Std. Error t value         Pr(>|t|)
## (Intercept) 204773.32  124016.98   1.651           0.0995 .
## Zasięg       -1763.71     374.51  -4.709 0.00000344816432 ***
## Bateria      12402.55    1899.21   6.530 0.00000000020262 ***
## Zużycie     -15616.64    6453.54  -2.420           0.0160 *
## Doładowanie    247.31      34.84   7.098 0.00000000000594 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 114700 on 394 degrees of freedom
## Multiple R-squared:  0.5568, Adjusted R-squared:  0.5523
## F-statistic: 123.7 on 4 and 394 DF,  p-value: < 0.00000000000000022
```

Listę `reg.1` można przedstawić również za pomocą funkcji `stargazer`.

```
reg.1 %>%
  stargazer(
    type = "latex",
    header = FALSE,
    title = "Wyniki analizy modelu \\texttt{reg.1}",
    out = NULL
  )
```

Tablica 3.27. Wyniki analizy modelu `reg.1`

	Dependent variable:
	Cena
Zasięg	−1,763.706***
	(374.507)
Bateria	12,402.550***
	(1,899.205)
Zużycie	−15,616.640**
	(6,453.541)
Doładowanie	247.309***
	(34.843)
Constant	204,773.300*
	(124,017.000)
Observations	399
R^2	0.557
Adjusted R^2	0.552
Residual Std. Error	114,716.300 (df = 394)
F Statistic	123.733*** (df = 4; 394)
Note:	*p<0.1; **p<0.05; ***p<0.01

Na podstawie wyników oszacowań regresji `reg.1` można sformułować następujące wnioski:

1. Wszystkie oceny parametrów, z wyjątkiem wyrazu wolnego, są statystycznie istotnie różne od zera na poziomie istotności 0,05.
2. Wzrost zasięgu o 1 km, *ceteris paribus*, powoduje przeciętnie w próbie wzrost ceny o -1763.7059 zł.
3. Wzrost pojemności baterii o 1 kWh, *ceteris paribus*, powoduje przeciętnie wzrost ceny o 12402.5474 zł.

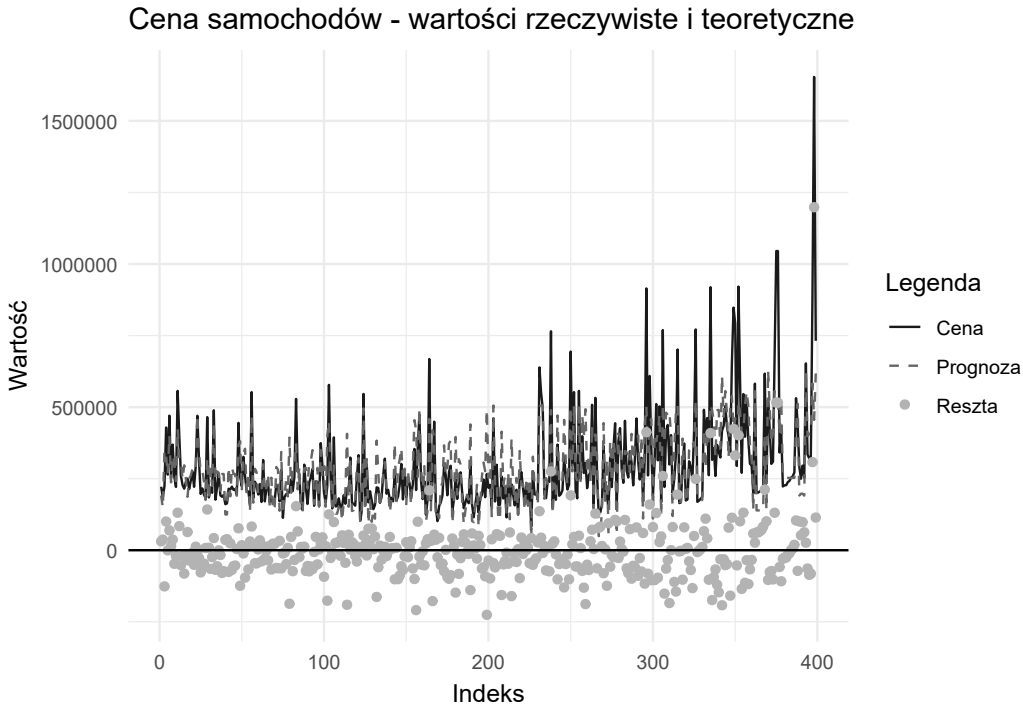
4. Wzrost zużycia energii o 1 kWh/100 km, *ceteris paribus*, powoduje przeciętnie wzrost ceny o -15616.6425 zł.
5. Wzrost szybkości doładowania o 100 km/h, *ceteris paribus*, powoduje przeciętnie wzrost ceny o 24730.9223 zł.
6. Skorygowany współczynnik determinacji wynosi 55.23%. Oznacza to, że 55.23% zróżnicowania zmiennej zależnej jest wyjaśniane przez zmienne objaśniające ujęte w modelu, po uwzględnieniu kary za liczbę szacowanych parametrów.

Dopasowanie wartości teoretycznych do wartości empirycznych zmiennej objaśnianej przedstawiono na rys. 3.54.

```
data.rank.2 <- data.rank.1 %>%
  mutate(
    Cena_f = fitted(reg.1),
    Reszta = resid(reg.1)
  )
```

```
data.rank.2 %>%
  ggplot(aes(x = seq_along(Cena))) +
    geom_line(aes(y = Cena, color = "Cena")) +
    geom_line(
      aes(y = Cena_f, color = "Prognoza"),
      linetype = "dashed"
    ) +
    geom_point(aes(y = Reszta, color = "Reszta")) +
    geom_hline(
      yintercept = 0,
      linetype = "solid",
      color = "black"
    ) +
    labs(
      title = "Cena samochodów - wartości rzeczywiste i teoretyczne",
      x = "Indeks",
      y = "Wartość"
    ) +
    scale_color_manual(
      name = "Legenda",
      values = c(
        "Cena" = "grey10",
        "Prognoza" = "grey40",
        "Reszta" = "grey70"
      )
    ) +
    theme_minimal()
```

Łatwo zauważyć, że dla niektórych modeli samochodów prognoza ceny jest wyższa od ceny (reszta $y - \hat{y}$ jest ujemna). Oznacza to, że auta są w rzeczywistości tańsze niż



Rysunek 3.54. Cena samochodów – wartości rzeczywiste i teoretyczne

wynikałoby to z ich wyceny na podstawie cech. W klasie 1 są to następujące modele (pierwszych 10 posortowanych rosnąco według reszty):

```
data.rank.2 %>%
  subset(Reszta < 0 & Class == 1) %>%
  arrange(Reszta) %>%
  select(c(1:3, 9:12)) %>%
  select(-Class) %>%
  head(10)
```

```
##       Model_1 Model_2           Model_3   Cena  Rank   Cena_f
## 1       XPENG      G6     RWD Long Range 206513 55.37 393533.4
## 2     Hyundai   IONIQ   6 Long Range 2WD 223580 61.49 349923.3
## 3     Hyundai   IONIQ   6 Long Range AWD 256861 56.34 381070.6
## 4       Zeekr     001     Long Range RWD 258995 56.16 354773.8
## 5       XPENG      P7     RWD Long Range 215047 58.48 296728.9
## 6       Skoda   Elroq                 85 186886 56.67 266316.1
## 7       CUPRA    Born     170 kW  77 kWh 196699 57.46 274188.0
## 8  Volkswagen    ID.3               GTX 215047 56.54 290795.7
## 9        Ford   Capri Extended Range AWD 235954 55.03 309832.1
## 10    Renault  Scenic   ETech EV87 220hp 206513 55.31 279771.2
```

Można również oszacować parametry modelu w podziale na klasy `Class`. Wynik zapisano do listy `reg.2`, której każda współrzędna zawiera listy a i b. Lista a zawiera wyniki oszacowań regresji, natomiast lista b wyniki podsumowania (`summary`) dla tej regresji.

```
reg.2 <- lapply(
  1:4,
  function(x) {
    a <- data.rank.1 %>%
      subset(Class == x) %>%
      lm(
        Cena ~ Zasięg+Bateria+Zużycie+Doładowanie,
        data = .
      )
    b <- summary(a)
    return(list(a = a, b = b))
  }
)
names(reg.2) <- paste("Klasa", 1:4)
```

Można wykorzystać obiekty a z listy `reg.2` pobierając je następnie do listy `models`.

```
models <- lapply(
  reg.2,
  function(x) x$a
)
```

Ponownie można wykorzystać funkcję `stargazer` do przedstawienia wyników analizy regresji dla klas `Class` (tabl. 3.28).

```
models %>%
  stargazer(
    type = "latex",
    header = FALSE,
    title = "Wyniki analizy regresji według klas",
    label = "tab:jcbsvl",
    out = "output/tables/stargazer_chunk-TvmKre.tex",
    model.names = TRUE,
    column.labels = names(models),
    dep.var.labels = "Cena",
    covariate.labels = c(
      "Zasięg",
      "Bateria",
      "Zużycie",
      "Doładowanie"
    )
  )
```

Tablica 3.28. Wyniki analizy regresji według klas

	Dependent variable:			
	Cena			
	OLS			
	Klasa 1	Klasa 2	Klasa 3	Klasa 4
	(1)	(2)	(3)	(4)
Zasięg	−789.957	2,448.264***	1,317.136	2,921.544
	(654.933)	(809.867)	(986.500)	(1,971.693)
Bateria	8,831.581**	−6,941.582	−914.750	−8,688.859
	(3,873.829)	(4,361.508)	(4,920.574)	(8,439.154)
Zużycie	−30,512.530*	−17,899.300	−35,964.670**	39,915.870
	(15,705.710)	(12,828.680)	(14,650.820)	(26,594.840)
Doładowanie	131.214***	37.422	175.438***	317.885***
	(39.392)	(25.421)	(48.978)	(120.489)
Constant	346,824.000	56,188.480	425,883.600*	−1,025,708.000*
	(254,584.100)	(225,186.600)	(255,929.200)	(610,636.400)
Observations	99	100	100	100
R^2	0.697	0.815	0.767	0.478
Adjusted R^2	0.684	0.807	0.757	0.456
Residual Std. Error	51,800.110 (df = 94)	41,965.740 (df = 95)	72,824.640 (df = 95)	178,642.600 (df = 95)
F Statistic	54.020*** (df = 4; 94)	104.354*** (df = 4; 95)	78.289*** (df = 4; 95)	21.778*** (df = 4; 95)
Note:				*p<0.1; **p<0.05; ***p<0.01

W przypadku regresji w podziale na klasy samochodów według utworzonego wcześniej rankingu, można zauważyć, że istotność statystyczna ocen parametrów różni się pomiędzy oszacowanymi modelami. I tak:

- W przypadku ceny samochodów w klasie 1 istotny statystycznie, na poziomie istotności 0,05, wpływ mają zmienne: `Bateria` oraz `Doładowanie`.
- W klasie 2 jest to jedynie `Zasięg`.
- W klasie 3 istotne są `Zużycie` oraz `Doładowanie`.
- Natomiast w klasie 4 istotne statystycznie jest jedynie `Doładowanie`.

Skorygowany współczynnik determinacji osiąga najwyższą wartość dla modelu dotyczącego ceny samochodów w klasie 2. Wynosi on 80.68%, co oznacza, że w takiej części zmienność ceny samochodów w tej klasie została wyjaśniona za pomocą wariancji zmiennych objaśniających.

Interpretacja istotnych statystycznie ocen parametrów przedstawia się następująco:

Klasa 1

1. Wzrost pojemności baterii samochodu o 1 kWh powoduje, *ceteris paribus*, przeciętnie wzrost ceny samochodu o 8831.58 zł.
2. Wzrost prędkości ładowania o 1 km/h powoduje, *ceteris paribus*, przeciętnie wzrost ceny samochodu o 131.21 zł.

Klasa 2

1. Wzrost zasięgu samochodu o 1 km powoduje, *ceteris paribus*, przeciętnie wzrost ceny samochodu o 2448.26 zł.

Klasa 3

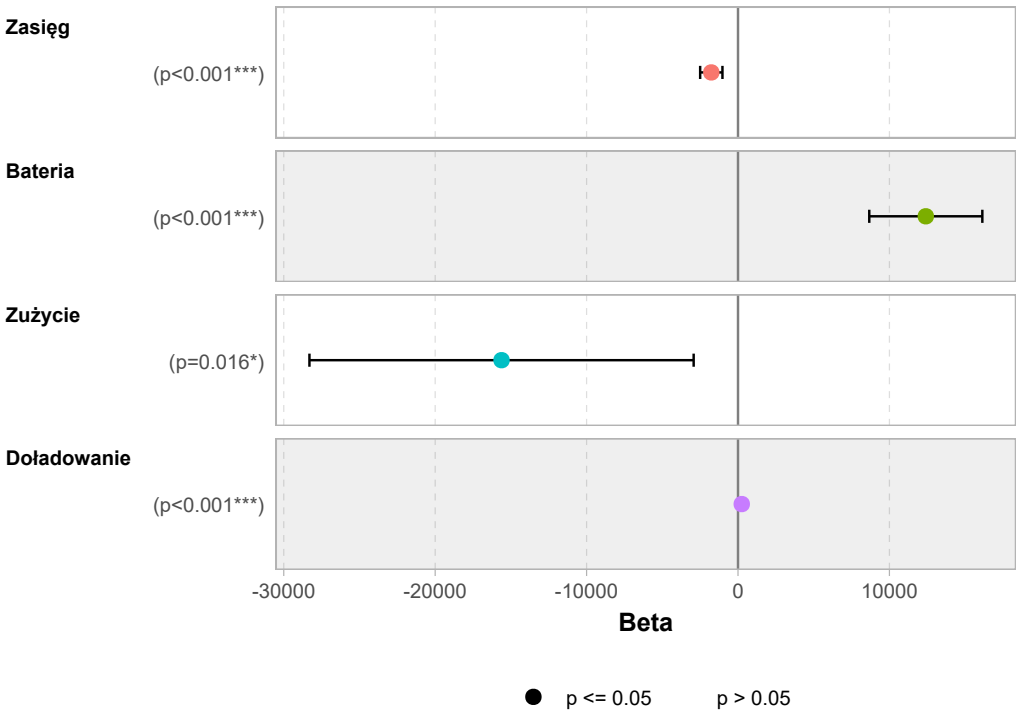
1. Wzrost zużycia energii o 1 kWh/100 km powoduje, *ceteris paribus*, przeciętnie wzrost ceny samochodu o -35964.67 zł.
2. Wzrost prędkości ładowania o 1 km/h powoduje, *ceteris paribus*, przeciętnie wzrost ceny samochodu o 175.44 zł.

Klasa 4

1. Wzrost prędkości ładowania o 1 km/h powoduje, *ceteris paribus*, przeciętnie wzrost ceny samochodu o 317.89 zł.

Za pomocą biblioteki `GGally` (Schloerke i in., 2024) można przedstawić oceny parametrów wraz z przedziałem ufności (rys. 3.55 dla regresji `reg . 1`).

```
reg.1 %>%
  ggcoef_model()
```



Rysunek 3.55. Wyniki analizy regresji `reg.1` – oceny parametrów

Podobny rysunek można wykonać dla listy regresji `models`.

```
marrangeGrob(
  lapply(models, function(x) ggcoef_model(x)),
    nrow = 4,
    ncol = 1,
    top = NULL
)
```

Można również pokazać wyłącznie oceny parametrów z regresji `reg.2` w jednej tablicy.

```
results <- lapply(
  reg.2,
  function(x) {
```

```
  df.coef <- as.data.frame(x$b$coefficients)
  df.coef$zmienna <- rownames(df.coef)
  df.coef <- df.coef %>%
    select(
      zmienna,
      ocena = Estimate,
      `p-val` = `Pr(>|t|)`
    )
  return(df.coef)
})
```

```
results.tab <- Reduce(
  function(x, y) full_join(
    x, y, by = "zmienna"
  ),
  results
) %>%
  set_colnames(c(
    "Zmienna",
    paste(
      rep(names(reg.2), each = 2),
      c("Ocena", "p-val"))
    )) %>%
  mutate_if(is.numeric, function(x) round(x, 2))
```

```
results.tab <- results.tab %>%
  mutate(
    across(contains("Ocena"),
    ~ ifelse(
        get(sub("Ocena", "p-val", cur_column())) < 0.05,
        paste0(round(.x, 2)),
        round(.x, 2))
    )
  )
```

Ostateczny rezultat podano w tabl. 3.29.

```
results.tab %>%
  select(
    Zmienna,
    contains("Ocena")
  ) %>%
  kable(
    format = "latex",
    caption = "Oceny parametrów według klas",
    booktabs = TRUE,
```

```
    escape = FALSE,
    label = "vjrjgb"
  ) %>%
  kable_styling(
    latex_options = c("hold_position", "striped")
  )
```

Tablica 3.29. Oceny parametrów według klas

Zmienna	Klasa 1 Ocena	Klasa 2 Ocena	Klasa 3 Ocena	Klasa 4 Ocena
(Intercept)	346824.05	56188.48	425883.64	-1025708.39
Zasięg	-789.96	2448.26	1317.14	2921.54
Bateria	8831.58	-6941.58	-914.75	-8688.86
Zużycie	-30512.53	-17899.3	-35964.67	39915.87
Doładowanie	131.21	37.42	175.44	317.89

Podobną analizę można przeprowadzić dla funkcji potęgowej, tj. oszacować parametry modelu liniowego:

$$\log(y_i) = \alpha_0 + \alpha_1 \log(X_{1i}) + \alpha_2 \log(X_{2i}) + \alpha_3 \log(X_{3i}) + \alpha_4 \log(X_{4i}) + \epsilon_i \tag{3.103}$$

gdzie:

$\log(y)$ – cena w złotych (logarytm naturalny),
$\log(X_1)$ – zasięg w km (logarytm naturalny),
$\log(X_2)$ – pojemność baterii w kWh (logarytm naturalny),
$\log(X_3)$ – zużycie energii elektrycznej w kWh/100 km (logarytm naturalny),
$\log(X_4)$ – szybkość uzupełniania zasięgu (w km/h) (logarytm naturalny).

Najpierw należy utworzyć logarytmy zmiennych.

```
data.rank.1.log <- data.rank.1 %>%
  mutate_at(
    vars(
      Zasięg,
      Bateria,
      Zużycie,
      Doładowanie,
      Cena
    ),
    log
  )
```

Następnie należy oszacować parametry modelu. Czytelnik może przeprowadzić taką analizę samodzielnie, przyjmując za podstawę wyniki analizy regresji zawarte

w obiekcie `reg.1.log`. Należy pamiętać, że parametry funkcji potęgowej interpretuje się jako elastyczności, które są stałe w czasie.

```
reg.1.log <- data.rank.1.log %>%
  lm(Cena ~ Zasięg+Bateria+Zużycie+Doładowanie, data = .)
summary(reg.1.log)
```

```
##
## Call:
## lm(formula = Cena ~ Zasięg + Bateria + Zużycie + Doładowanie,
##     data = .)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.53244 -0.16805 -0.04799  0.11529  1.44862
##
## Coefficients:
##             Estimate Std. Error t value          Pr(>|t|)
## (Intercept) 11.99720    3.62197   3.312           0.00101 **
## Zasięg      -1.72140    0.81098  -2.123           0.03441 *
## Bateria      2.38017    0.81055   2.936           0.00351 **
## Zużycie     -0.65691    0.78352  -0.838           0.40231
## Doładowanie  0.38742    0.05072   7.639 0.000000000000168 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2526 on 394 degrees of freedom
## Multiple R-squared:  0.6934, Adjusted R-squared:  0.6902
## F-statistic: 222.7 on 4 and 394 DF,  p-value: < 0.00000000000000022
```

Wyniki zawarte w `reg.1.log` można przedstawić za pomocą funkcji `stargazer` (tabl. 3.30).

```
reg.1.log %>%
  stargazer(
    type = "latex",
    header = FALSE,
    title = "Wyniki analizy modelu \\texttt{reg.1.log}",
    out = NULL,
    label = "tab:wanksb"
  )
```

Wyniki przeprowadzonej analizy mają istotne i bezpośrednie znaczenie praktyczne, ponieważ umożliwiają bardziej świadome, porównywalne i oparte na danych podjęcie decyzji o zakupie samochodu elektrycznego. Oszacowane parametry pokazują, w jakim stopniu zmiany kluczowych cech pojazdu – takich jak zasięg, pojemność baterii, efektywność energetyczna czy szybkość ładowania – wpływają na kształtowanie

Tablica 3.30. Wyniki analizy modelu `reg.1.log`

	Dependent variable:
	Cena
Zasięg	−1.721**
	(0.811)
Bateria	2.380***
	(0.811)
Zużycie	−0.657
	(0.784)
Doładowanie	0.387***
	(0.051)
Constant	11.997***
	(3.622)
Observations	399
R^2	0.693
Adjusted R^2	0.690
Residual Std. Error	0.253 (df = 394)
F Statistic	222.722*** (df = 4; 394)
Note:	*p<0.1; **p<0.05; ***p<0.01

się ceny ofertowej producenta. Dzięki temu potencjalny nabywca może precyzyjnie ocenić, które cechy techniczne najbardziej podnoszą wartość pojazdu, a które mają znaczenie drugorzędne, co pozwala na bardziej racjonalne i dopasowane do potrzeb podjęcie decyzji zakupowej. Tego typu analiza stanowi więc konkretne, empirycznie uzasadnione narzędzie, umożliwiające porównanie modeli nie tylko na podstawie katalogowych parametrów, ale także ich rzeczywistego wpływu na poziom cen.

W następnym podrozdziale podano zastosowania metody programowania obiektowego.

3.4. Zastosowania programowania obiektowego

W podrozdziale 2.4 podano wprowadzenie do techniki programowania obiektowego w R, która umożliwia tworzenie struktur danych i metod zorganizowanych w klasy i obiekty. Taka technika programowania pozwala na uporządkowane podejście do tworzenia kodu wielokrotnego użytku, co sprzyja budowie bardziej złożonych projektów.

Poniżej podano przykład zastosowania klasy S4, który wykorzystuje dziedziczenie i walidację do analizy szeregów czasowych. Utworzona zostanie hierarchia z klasą bazową `TimeSeries` oraz dwiema klasami dziedziczącymi: `ARIMA` i `ExpSmooth`, które reprezentują różne metody analizy szeregów czasowych. Każda klasa będzie posiadała mechanizm walidacji sprawdzający poprawność danych wejściowych, a także metody analizy modeli i prognozowania.

W klasie S4 należy rozpocząć od zdefiniowania klasy bazowej `TimeSeries` z mechanizmem walidacji za pomocą funkcji `setClass`.

```
setClass(
  "TimeSeries",
  slots = list(data = "numeric"),
  validity = function(object) {
    if (length(object@data) == 0) {
      return("Dane szeregów czasowych nie mogą być puste.")
    }
    if (!is.numeric(object@data)) {
      return("Dane szeregów czasowych muszą być liczbowe.")
    }
    TRUE
  }
)
```

Następnie można zdefiniować metodę za pomocą funkcji `setMethod`.

```
setMethod(
  "plot",
  signature(x = "TimeSeries", y = "missing"),
  function(x, y, ...) {
    plot(
      x@data,
      type = "l",
      col = "grey",
      main = "Szereg czasowy",
      xlab = "Czas",
      ylab = "Wartość",
      ...
    )
  }
)
```

W systemie klas S4 pojęcie `signature` służy do definiowania typów argumentów, dla których metoda ma być wywoływana. `Signature` nie jest funkcją, lecz argumentem lub konstruktorem używanym przy definiowaniu metod. W przypadku:

```
signature(x = "TimeSeries", y = "missing")
```

oznacza to, że:

1. Argument `x` ma być obiektem klasy `TimeSeries`, tj. metoda zostanie wywołana tylko wtedy, gdy argument `x` jest obiektem tej klasy.
2. Argument `y` nie jest przekazywany, czyli w wywołaniu funkcji `plot` argument `y` musi być pominięty. Jeśli użytkownik spróbuje wywołać funkcję `plot(x, y)`, metoda nie zostanie zastosowana, ponieważ wymaga się, aby `y` było pominięte.

Warunki punktu 2. są potrzebne, ponieważ funkcja `plot` w R jest funkcją generyczną, którą można stosować dla obiektów różnych typów[12]. Domyślnie przyjmuje ona dwa argumenty `x` i `y`. Jeśli zatem definiuje się metodę `plot` dla własnej klasy S4, należy określić sposób jej działania wobec zastosowania różnych argumentów tej funkcji. Przypadek `y = "missing"` oznacza, że metoda dotyczy tylko jednego argumentu (`x`), a wartość `y` jest ignorowana. Dzięki temu można uniknąć konfliktów z innymi metodami `plot`, które mogą wymagać dwóch argumentów (`x` i `y`).

W następnej kolejności utworzono klasę `ARIMA`, która dziedziczy po klasie `TimeSeries`. Zdefiniowano dwa sloty:

- `order` – wektor numeryczny zawierający trzy wartości (p, d, q), które określają parametry modelu `ARIMA`.

[12] W R pojęcie *funkcja generyczna* oznacza funkcję, która nie wykonuje od razu konkretnego kodu, lecz wybiera odpowiednią metodę w zależności od typu (klasy) obiektu.

- `model` – dopasowany model `ARIMA`.

Dalej sprawdza się poprawność obiektu (`validity`), tj. `order` musi zawierać dokładnie trzy wartości i wszystkie z nich muszą być liczbami.

```
setClass(
  "ARIMA",
  slots = list(order = "numeric", model = "ANY"),
  contains = "TimeSeries",
  validity = function(object) {
    if (length(object@order) != 3) {
      return("Funkcja ARIMA musi zawierać 3 wartości (p, d, q).")
    }
    if (!all(is.numeric(object@order))) {
      return("Argumenty funkcji ARIMA muszą być liczbami.")
    }
    TRUE
  }
)
```

Dla utworzonej klasy `ARIMA`:

1. Definiuje się metodę `initialize`.
2. Wywołuje się domyślną metodę inicjalizacji (`callNextMethod`).

W podanej wersji metoda nie uwzględnia dodatkowej logiki, ale jest gotowa na jej rozszerzenie – na przykład o automatyczne dopasowanie modelu na podstawie danych lub weryfikację jakości danych podczas tworzenia obiektu.

```
setMethod(
  "initialize",
  "ARIMA",
  function(.Object, ...) {
    .Object <- callNextMethod(.Object, ...)
    .Object
  }
)
```

Tworzy się generyczną funkcję `fit.model`, którą można nadpisywać dla różnych klas.

```
setGeneric(
  "fit.model",
  function(object) standardGeneric("fit.model")
)
```

```
## [1] "fit.model"
```

W kolejnym kroku:

1. Dopasowuje się model ARIMA do danych w obiekcie.
2. Sprawdza się, czy `object@data` zawiera co najmniej 100 obserwacji (w przeciwnym razie zgłaszany jest błąd).
3. Dopasowuje się model ARIMA przy użyciu funkcji `arima`.
4. Wynik przechowuje się w `object@model`.
5. Generuje się podsumowanie szacunków modelu (`summary`).
6. Zwraca się obiekt z dopasowanym modelem.

```
setMethod(
  "fit.model",
  "ARIMA",
  function(object) {
    if (length(object@data) < 100) {
      stop("Zbyt mało danych do dopasowania modelu ARIMA.")
    }
    object@model <- arima(
      object@data,
      order = object@order
    )
    print(summary(object@model))
    object
  }
)
```

Następnie tworzy się generyczną funkcję `forecast`, która oczekuje dwóch argumentów:

- `object` – np. obiekt klasy ARIMA,
- `steps` – horyzont prognozy.

Funkcja pozwala na przewidywanie wartości zmiennej na podstawie modelu.

```
setGeneric(
  "forecast",
  function(object, steps) standardGeneric("forecast")
)
```

```
## Tworzenie nowej ogólnej funkcji dla 'forecast' w pakiecie the
 ↪  global environment
```

```
## [1] "forecast"
```

W kolejnym kroku:

1. Generuje się prognozę na `steps` kroków naprzód.
2. Sprawdza się, czy model został dopasowany (`object@model` nie ma wartości `NULL`).
3. Używa się funkcji `predict` do wyznaczenia prognozy.
4. Zwraca się wartości prognoz (`forecast.values$pred`).

```
setMethod(
  "forecast",
  "ARIMA",
  function(object, steps) {
    if (is.null(object@model)) {
      stop("Model ARIMA nie został dopasowany.")
    }
    forecast.values <- predict(
      object@model,
      n.ahead = steps
    )
    return(forecast.values$pred)
  }
)
```

Następnie tworzy się klasę `ExpSmooth` również dziedziczącą po klasie `TimeSeries`. Czytelnik może na podstawie analizy klasy `ARIMA` przeprowadzić samodzielnie interpretację klasy `ExpSmooth`.

```
setClass(
  "ExpSmooth",
  slots = list(
    smoothingLevel = "numeric",
    model = "ANY"
  ),
  contains = "TimeSeries",
  validity = function(object) {
    if (!is.numeric(object@smoothingLevel) ||
        object@smoothingLevel <= 0 ||
        object@smoothingLevel >= 1) {
      return("smoothingLevel musi być liczbą z przedziału (0,1).")
    }
    TRUE
  }
)
```

Utworzono również metody dla klasy `ExpSmooth`, tj. `fit.model` i `forecast`.

```
setMethod(
  "fit.model",
  "ExpSmooth",
  function(object) {
    if (length(object@data) < 100) {
      stop("Zbyt mało danych do dopasowania modelu wygładzania.")
    }
    object@model <- HoltWinters(
      object@data,
      alpha = object@smoothingLevel,
      beta = FALSE,
      gamma = FALSE
    )
    print(summary(object@model))
    object
  }
)
```

```
setMethod(
  "forecast",
  "ExpSmooth",
  function(object, steps) {
    if (is.null(object@model)) {
      stop("Model wygładzania wykładniczego nie został dopasowany.")
    }
    forecast.values <- predict(
      object@model,
      n.ahead = steps
    )
    return(forecast.values)
  }
)
```

Po zdefiniowaniu klas `TimeSeries`, `ARIMA` i `ExpSmooth` można podać przykład ich użycia. Ustalono *ziarno* (warunki początkowe) generatora liczb pseudolosowych za pomocą własnej funkcji `set_seed` (zob. podrozdział 2.2.4). Następnie utworzono losowy szereg czasowy `data` obejmujący 200 obserwacji, wygenerowanych z rozkładu normalnego o średniej 0 i odchyleniu standardowym 1. W dalszym kroku proces ten został przekształcony w proces błądzenia losowego (*random walk*).

```
set_seed()
data <- cumsum(rnorm(200))
head(data)
```

```
## [1] -1.2070657 -0.9296365  0.1548047 -2.1908930 -1.7617683
 ↪  -1.2557125
```

Następnie utworzono obiekt (*instancję*) klasy `TimeSeries` przy użyciu systemu obiektowego S4, do której w slocie `data` przypisano wcześniej wygenerowany szereg `data`. Ponieważ klasa `TimeSeries` ma zdefiniowaną funkcję `validity`, kod automatycznie sprawdza:

- czy `data` nie jest puste,
- czy jest typu numerycznego.

```
ts <- new("TimeSeries", data = data)
```

Łatwo sprawdzić, że obiekt `ts` zostaje poprawnie utworzony. Na rys. 3.56 umieszczono jego wykres. Wywołanie funkcji `plot` uruchamia metodę `plot` zdefiniowaną wcześniej specjalnie dla klasy `TimeSeries`.

```
plot(ts)
```

Rysunek 3.56. Proces błądzenia losowego dla klasy `TimeSeries`

Poniższy kod tworzy nowy obiekt klasy `ARIMA`:

- `new("ARIMA", ...)` – wywołuje konstruktor klasy `ARIMA`,

- `data = data` – przekazuje zmienną `data`,
- `order = c(1, 1, 1)` – określa parametry modelu ARIMA:
  - `p = 1` – liczba opóźnień w części autoregresyjnej (AR),
  - `d = 1` – stopień różnicowania,
  - `q = 1` – liczba opóźnień w części średniej ruchomej (MA).

Warto zauważyć, że *w tle* działa metoda `initialize`, która obecnie tylko wywołuje wersję domyślną, oraz funkcja `validity`, która sprawdza:

- czy `order` przyjmuje dokładnie trzy wartości,
- czy każda z nich jest numeryczna.

Obiekt `arima.model` został poprawnie utworzony, ale nie zawiera jeszcze dopasowanego modelu, gdyż slot `model` jest pusty.

```
arima.model <- new(
  "ARIMA",
  data = data,
  order = c(1, 1, 1)
)
```

W wyniku zastosowania poniższego kodu:

- model `ARIMA(1,1,1)` zostaje dopasowany do danych `data`,
- wynik zostaje przekazany do slotu `model`,
- zostaje wyświetlone podsumowanie modelu (z metody).

Obiekt `arima.model.fit` zawiera wszystko, co potrzebne do dalszych analiz.

```
arima.model.fit <- arima.model %>%
  fit.model()
```

```
##           Length Class  Mode
## coef        2    -none- numeric
## sigma2      1    -none- numeric
## var.coef    4    -none- numeric
## mask        2    -none- logical
## loglik      1    -none- numeric
## aic         1    -none- numeric
## arma        7    -none- numeric
## residuals 200    ts     numeric
## call        3    -none- call
## series      1    -none- character
```

```
## code        1    -none- numeric
## n.cond      1    -none- numeric
## nobs        1    -none- numeric
## model      10    -none- list
```

Poniższy kod działa zgodnie z oczekiwaniami:

- Sprawdza, czy model ARIMA został wcześniej dopasowany, tj. czy `object@model` nie jest `NULL`. Tak jest, gdyż wcześniej użyto `fit.model`.
- Wyznacza prognozę na 10 okresów do przodu, tj. używa funkcji `predict` z argumentem `n.ahead = 10`.
- Zwraca tylko wektor wartości prognozowanych `forecast.values$pred`.
- Zapisuje prognozę do zmiennej `arima.f`.

```
arima.f <- forecast(
  arima.model.fit,
  steps = 10
)
```

Następny kod pokazuje (lub drukuje w dokumencie) prognozowane wartości razem z tytułem.

```
cat("\nPrognoza ARIMA:\n", arima.f, "\n")
```

```
##
## Prognoza ARIMA:
##  -11.81366 -11.86632 -11.87691 -11.87904 -11.87947 -11.87955
 ↪  -11.87957 -11.87957 -11.87957 -11.87957
```

Szereg `arima.f` zawiera prognozy uzyskane z modelu ARIMA, które można przedstawić na wykresie po utworzeniu obiektu klasy `TimeSeries` (rys. 3.57).

```
arima.f %>%
  as.numeric %>%
  new("TimeSeries", data = .) %>%
  plot()
```

W następnej kolejności podano zastosowanie klasy `ExpSmooth`. Poniższy kod tworzy nowy obiekt klasy `ExpSmooth`, przekazuje ten sam szereg czasowy `data`, co wcześniej, oraz ustawia poziom wygładzania $\alpha = 0.5$. Dzięki metodzie `validity`, sprawdzany jest warunek, czy `smoothingLevel` jest liczbą z zakresu $(0, 1)$.

Rysunek 3.57. Prognozy z modelu ARIMA dla klasy `TimeSeries`

```
exp.smooth.model <- new(
  "ExpSmooth",
  data = data,
  smoothingLevel = 0.5
)
```

Obiekt `exp.smooth.model` został poprawnie utworzony, ale nie zawiera jeszcze dopasowanego modelu (`model = NULL`). Zadanie to realizuje natomiast poniższy kod, który szacuje model wygładzania wykładniczego. W tle używana jest zdefiniowana wcześniej metoda `fit.model` dla klasy `ExpSmooth`. Na koniec zwracany jest obiekt z dopasowanym modelem i zapisywany w zmiennej `exp.smooth.model.fit`.

```
exp.smooth.model.fit <- exp.smooth.model %>%
  fit.model()
```

```
##              Length Class  Mode
## fitted       398    mts    numeric
## x            200    ts     numeric
## alpha          1    -none- numeric
## beta           1    -none- logical
```

```
## gamma          1    -none- logical
## coefficients   1    -none- numeric
## seasonal       1    -none- character
## SSE            1    -none- numeric
## call           5    -none- call
```

Na tym etapie można przystąpić do prognozowania. Poniższy kod działa następująco:

- Sprawdza, czy model został dopasowany, tj. czy `object@model` nie jest `NULL`.
- Wyznacza prognozę korzystając z funkcji `predict` z `n.ahead = 10`, więc generuje 10 prognoz *ex ante*.
- Zwraca wynik, tj. wektor wartości prognozowanych.

```
exp.smooth.f <- forecast(
  exp.smooth.model.fit,
  steps = 10
)
```

Wynik można pokazać w następujący sposób:

```
cat(
  "\nPrognoza z modelu wygładzania wykładniczego:\n",
  exp.smooth.f,
  "\n"
)
```

```
##
## Prognoza z modelu wygładzania wykładniczego:
##  -10.51574 -10.51574 -10.51574 -10.51574 -10.51574 -10.51574
 ↪  -10.51574 -10.51574 -10.51574 -10.51574
```

Poniższy kod realizuje tę samą procedurę, co w przypadku modelu `ARIMA`, lecz odnosi się do prognozy uzyskanej z modelu wygładzania wykładniczego.

```
exp.smooth.f %>%
  as.numeric() %>%
  new("TimeSeries", data = .) %>%
  plot()
```

Na rys. 3.58 można zauważyć, że prognoza z modelu `ExpSmooth` dąży do poziomu ustalonego przez ostatnie obserwacje w danych. Jest to typowe dla tego modelu, szczególnie przy braku trendu i sezonowości. Teraz zostanie przedstawiona klasa `LinRegr` również dziedzicząca po klasie `TimeSeries`.

Rysunek 3.58. Prognozy z modelu wygładzania wykładniczego

```
setClass(
  "LinRegr",
  slots = list(
    x = "numeric",
    y = "numeric",
    model = "ANY"
  ),
  contains = "TimeSeries",
  validity = function(object) {
    if (length(object@x) != length(object@y)) {
      return("Liczba obserwacji dla zmiennych x i y musi być taka
      ↪  sama.")
    }
    if (!is.numeric(object@x) || !is.numeric(object@y)) {
      return("Zmienne x i y muszą być liczbami.")
    }
    if (length(object@data) == 0) {
      return("Dane szeregu czasowego nie mogą być puste.")
    }
    if (!is.numeric(object@data)) {
      return("Dane szeregu czasowego muszą być liczbowe.")
    }
```

```
    TRUE
  }
)
```

Powyższy kod definiuje nową klasę S4 o nazwie `LinRegr`, która reprezentuje model osadzony w kontekście analizy szeregów czasowych. Klasa `LinRegr` dziedziczy wszystko po klasie `TimeSeries`, w tym slot `data` i związane z nią metody. Zawiera sloty specyficzne dla regresji oraz funkcję walidacji (`validity`), która określa warunki poprawności obiektu. Jeśli wszystkie warunki są spełnione, to zwracana jest wartość `TRUE`, czyli obiekt jest poprawny.

Następny kod ponownie wywołuje definicję funkcji generycznej S4 o nazwie `fit.model`. W praktyce nie jest to konieczne, o ile funkcja generyczna `fit.model` została już wcześniej zdefiniowana (np. dla klas `ARIMA` czy `ExpSmooth`). W takim przypadku wystarczyłoby dodać nową metodę za pomocą `setMethod`. W przykładowym kodzie funkcja generyczna jest jednak ponownie zapisywana dla zachowania kompletności wywodu.

```
setGeneric(
  "fit.model",
  function(object) standardGeneric("fit.model"))
```

```
## [1] "fit.model"
```

Następnie zapisano metodę `fit.model` dla klasy `LinRegr`. Jej założenia są następujące:

1. Sprawdzenie liczby obserwacji.
2. Dopasowanie modelu liniowego `lm(y ~ x)` na podstawie danych zapisanych w slotach `x` i `y` obiektu. Model jest zapisywany do slotu `model`.
3. Podsumowanie modelu.
4. Zwracanie obiektu, przy czym `invisible(object)` pozwala przypisać wynik bez wypisywania go w konsoli.

```
setMethod(
  "fit.model",
  "LinRegr",
  function(object) {
    if (length(object@x) < 100) {
      stop("Zbyt mało danych do dopasowania modelu.")
    }
    object@model <- lm(
      y ~ x,
      data = data.frame(
```

```
        x = object@x,
        y = object@y
      )
    )
    print(summary(object@model))
    invisible(object)
  }
)
```

Poniższy kod ponownie definiuje funkcję generyczną `forecast`, która przyjmuje dwa argumenty:

- `object` – obiekt klasy `LinRegr`,
- `x.new` – nowe wartości dla zmiennej niezależnej, na podstawie których mają być wygenerowane prognozy.

Funkcja `standardGeneric("forecast")` pozwala systemowi metod S4 wybrać odpowiednią wersję `forecast` w zależności od klasy `object`. Podobnie jak w przypadku `fit.model`, w praktycznych zastosowaniach wystarcza jednokrotna definicja funkcji generycznej, a następnie dodawanie kolejnych metod dla różnych klas.

```
setGeneric(
  "forecast",
  function(object, x.new) standardGeneric("forecast"))
```

```
## [1] "forecast"
```

Etapy metody `forecast` dla klasy `LinRegr` przebiegają w następujący sposób:

1. Sprawdzenie, czy model został dopasowany. Jeśli slot `model` jest pusty (`NULL`), procedura przerywa działanie.
2. Generowanie prognozy za pomocą funkcji `predict` dla wcześniej dopasowanego modelu (`lm`) na podstawie nowego zbioru danych (`data.frame(x = x.new)`).
3. Zwrócenie wyniku. Zmienna `prediction` zawiera wektor prognozowanych wartości odpowiadających `x.new`.

```
setMethod(
  "forecast",
  "LinRegr",
  function(object, x.new) {
    if (is.null(object@model)) {
      stop("Model nie został dopasowany.")
```

```
    }
    prediction <- predict(
      object@model,
      newdata = data.frame(x = x.new)
    )
    return(prediction)
  }
)
```

Poniżej przedstawiono przykład zastosowania klasy `LinRegr` na podstawie zdefiniowanych zmiennych `x` i `y`.

```
set_seed()
n <- 200
x <- rnorm(n) %>%
  cumsum()
y <- 2 + 3 * x + rnorm(n)
```

Poniższy kod tworzy nowy obiekt klasy `LinRegr`, gdzie:

`x` – zmienna niezależna,
`y` – zmienna zależna,
`data` – dla spójności z klasą `TimeSeries` przypisano `y`.

W tle działa funkcja `validity`, która sprawdza:

- czy wektory `x` i `y` mają tę samą długość (liczbę obserwacji),
- czy są typu `numeric`,
- czy `data` nie jest puste i zawiera wartości liczbowe.

Obiekt `lm.model` został poprawnie utworzony.

```
lm.model <- new(
  "LinRegr",
  x = x,
  y = y,
  data = y
)
```

Poniższy krok uruchamia metodę `fit.model` dla obiektu klasy `LinRegr`. Wynik działania metody zapisywany jest w slocie `model`, czyli pod adresem `object@model`, jako dopasowany model regresji (`lm`).

```
lm.model.fit <- fit.model(lm.model)
```

```
##
## Call:
## lm(formula = y ~ x, data = data.frame(x = object@x, y = object@y))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.5043 -0.6526  0.0437  0.6411  2.7613
##
## Coefficients:
##             Estimate Std. Error t value            Pr(>|t|)
## (Intercept)  2.23013    0.19378   11.51 <0.0000000000000002 ***
## x            3.01045    0.01201  250.73 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.008 on 198 degrees of freedom
## Multiple R-squared:  0.9969, Adjusted R-squared:  0.9968
## F-statistic: 6.286e+04 on 1 and 198 DF,  p-value: <
 ↪  0.00000000000000022
```

Można teraz zastosować metodę `forecast`. W tym celu utworzono nowy wektor wartości (`x.new`), dla których mają być wyznaczone prognozy z dopasowanego modelu.

```
x.new <- c(-10, 0, 10)
```

Zadanie prognostyczne dla wektora `x.new` realizuje poniższy kod.

```
lm.forecast <- lm.model.fit %>%
  forecast(x.new)
```

Wartości prognoz można pokazać w następujący sposób:

```
cat("\nPrognoza z modelu:\n", lm.forecast, "\n")
```

```
##
## Prognoza z modelu:
##  -27.8744 2.230133 32.33466
```

Na koniec pozostało zadanie wizualizacji prognoz (rys. 3.59). Poniższy kod:

1. Rysuje wykres punktowy.

2. Dodaje linię regresji dopasowaną za pomocą `lm.model.fit@model`, w ciemnoszarym kolorze i o zwiększonej grubości (parametr `lwd = 2`).
3. Dodaje legendę w lewym górnym rogu wykresu (`"topleft"`).

```
plot(
  lm.model@x,
  lm.model@y,
  main = "Regresja liniowa",
  xlab = "x",
  ylab = "y",
  col = "grey",
  pch = 16
)
abline(
  lm.model.fit@model,
  col = "grey30",
  lwd = 2
)
legend(
  "topleft",
  legend = c("Dane", "Linia regresji"),
  col = c("grey", "grey30"),
  pch = c(16, NA),
  lty = c(NA, 1)
)
```

Na podstawie doświadczeń zdobytych przy budowie klas S4 utworzono klasę `RegExpSystem`, stanowiącą prosty system ekspercki do analizy regresji. W ocenie jakości modelu wykorzystano zagadnienia wprowadzone w podrozdziale 3.3. W systemie będą wykorzystywane poniższe biblioteki.

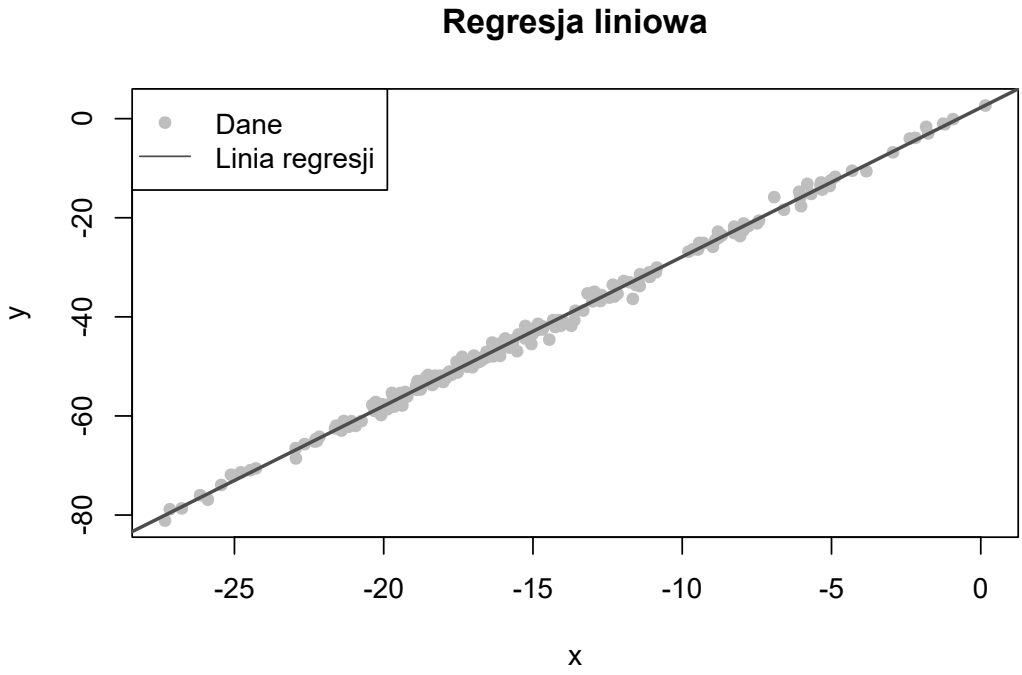
```
library(car)
library(ggplot2)
library(lmtest)
library(magrittr)
library(stargazer)
library(tseries)
```

Poniżej zdefiniowano klasę i pozostałe składowe systemu `RegExpSystem`:

- `setClass` – tworzy nową klasę S4,
- `slots` – definiuje pola (sloty) obiektów tej klasy.

Niech będą dane trzy sloty:

- `x` – wektor liczbowy (`numeric`),



Rysunek 3.59. Regresja liniowa

- y – wektor liczbowy (`numeric`),
- `model` – dowolny obiekt (`ANY`).

Funkcja `validity` sprawdza poprawność obiektu podczas jego tworzenia.

```
setClass(
  "RegExpSystem",
  slots = list(
    x = "numeric",
    y = "numeric",
    model = "ANY"
  ),
  validity = function(object) {
    if (length(object@x) != length(object@y)) {
      return("Zmienne x i y muszą mieć tę samą długość.")
    }
    if (any(!is.numeric(object@x)) || any(!is.numeric(object@y))) {
      return("Zmienne x i y muszą być liczbami.")
    }
    TRUE
  }
)
```

Poniższy kod tworzy generyczną funkcję `fit.model`.

```
setGeneric(
  "fit.model",
  function(object) standardGeneric("fit.model"))
```

```
## [1] "fit.model"
```

Metoda `fit.model` dopasowuje model do danych `x` i `y` w obiekcie klasy `RegExpSystem`, pod warunkiem, że liczba obserwacji jest wystarczająca.

```
setMethod(
  "fit.model",
  "RegExpSystem",
  function(object) {
    if (length(object@x) < 100) {
      stop("Zbyt mało danych do oszacowania modelu.")
    }
    object@model <- lm(
      y ~ x,
      data = data.frame(
        x = object@x,
        y = object@y
      )
    )
    cat("Model został oszacowany\n")
    return(object)
  }
)
```

Poniższy kod tworzy kolejną funkcję S4 o nazwie:

`evaluate.statistical.significance`.

```
setGeneric(
  "evaluate.statistical.significance",
  function(object)
   ↪  standardGeneric("evaluate.statistical.significance"))
```

```
## [1] "evaluate.statistical.significance"
```

Celem metody `evaluate.statistical.significance` dla obiektów klasy `RegExpSystem` jest kompleksowa ocena jakości dopasowanego modelu. Metoda działa w następujący sposób:

1. Sprawdzenie, czy model istnieje:

   - jeśli slot `model` jest pusty (`NULL`), zgłaszany jest błąd – model musi być wcześniej oszacowany.

2. Wykonanie obliczeń statystycznych:

   - `summary(object@model)` – generuje pełne podsumowanie szacunków modelu:
     - `residuals` – obliczanie reszt regresji,
     - `p.value` – pobranie *p*-wartości dla współczynnika `x`,
     - `r.squared` – pobranie wartości współczynnika determinacji R^2,
     - `see` – obliczenie błędu standardowego,
     - zmienna `k` stanowi liczbę poprawnych wyników testów jakości modelu.

3. Wykonanie testów diagnostycznych:

   - Jarque–Bera (`jarque.bera.test`) – test normalności składników losowych,
   - Durbin–Watson (`dwtest`) – test autokorelacji 1. rzędu składników losowych,
   - Breusch–Godfrey (`bgtest`) – test autokorelacji składników losowych,
   - White (`bptest`) – test heteroskedastyczności składników losowych,
   - dla każdego testu drukowane są wyniki, jeśli $p - value < 0.05$, wskazuje się problem (np. brak normalności, autokorelacja, heteroskedastyczność), jeśli obliczona wartość $p - value \geq 0.05$, `k` jest zwiększane (wyniki testu są pozytywne).

4. Wykonanie testu *t*:

   - sprawdzana jest istotność współczynnika kierunkowego przy zmiennej `x`.

5. Podsumowanie wyników:

   - czy estymatory są statystycznie istotne (na podstawie *p*-wartości),
   - czy R^2 świadczy o dobrej jakości dopasowania (> 0.7),
   - wypisywany jest błąd `SEE`,
   - wartość `k` podsumowuje liczbę *spełnionych* kryteriów jakości.

6. Zwracanie wyników:

   - zwracana jest lista z wynikami statystycznymi i diagnostycznymi.

```
setMethod(
  "evaluate.statistical.significance",
  "RegExpSystem",
  function(object) {
    if (is.null(object@model)) {
      stop("Model nie został oszacowany")
    }
    k <- 0
    summary <- summary(object@model)
    print(summary)
    resid <- residuals(object@model)
    p.value <- summary$coefficients[2, 4]
    r.squared <- summary$r.squared
    see <- sqrt(sum(resid^2) / (length(resid)-1))
    jb <- jarque.bera.test(resid)
    jb.p.value <- jb$p.value
    cat("Test Jarque'a-Bery:\n")
    print(jb)
    if(jb.p.value < 0.05) {
      cat("Składnik losowy nie ma rozkładu normalnego.\n")
    } else k <- k+1
    dw <- dwtest(object@model)
    dw.p.value <- dw$p.value
    cat("Test Durbina-Watsona:\n")
    print(dw)
    if(dw.p.value < 0.05) {
      cat("Występuje statystycznie istotna autokorelacja pierwszego
       ↪ rzędu.\n")
    } else k <- k+1
    bg <- bgtest(object@model)
    bg.p.value <- bg$p.value
    cat("Test Breuscha-Godfreya:\n")
    print(bg)
    if(bg.p.value < 0.05) {
      cat("Występuje statystycznie istotna autokorelacja pierwszego
       ↪ rzędu.\n")
    } else k <- k+1
    wh <- bptest(object@model)
    wh.p.value <- wh$p.value
    cat("Test White'a:\n")
    print(wh)
    if(wh.p.value < 0.05) {
      cat("Występuje statystycznie istotna heteroskedastyczność
       ↪ składnika losowego.\n")
    } else k <- k+1
    t.test <- summary$coefficients[2, 3]
    cat("Test t-Studenta dla współczynnika kierunkowego:\n")
    cat("t-value:", t.test, "\n")
    cat("\nPodsumowanie wyników:\n")
```

```
    if (p.value < 0.05) {
      cat("Model jest statystycznie istotny (p-value < 0.05)\n")
      k <- k+1
    } else {
      cat("Model nie jest statystycznie istotny (p-value >= 0.05)\n")
    }
    cat("R2:", r.squared, "\n")
    if (r.squared > 0.7) {
      cat("Model ma dobrą jakość dopasowania (R2 > 0.7)\n")
      k <- k+1
    } else {
      cat("Model ma niską jakość dopasowania (R2 <= 0.7)\n")
    }
    cat("SEE:", see, "\n")
    return(list(
      summary = summary,
      pval    = p.value,
      r2      = r.squared,
      see     = see,
      jb      = jb.p.value,
      dw      = dw.p.value,
      bg      = bg.p.value,
      wh      = wh.p.value,
      k       = k
    ))
  }
)
```

Poniższy kod tworzy funkcję S4 o nazwie:

`evaluate.economic.significance`.

```
setGeneric(
  "evaluate.economic.significance",
  function(object) standardGeneric("evaluate.economic.significance"))
```

```
## [1] "evaluate.economic.significance"
```

Celem metody `evaluate.economic.significance` dla obiektów klasy `RegExpSystem` jest ocena *istotności ekonomicznej* modelu, czyli tego, czy związek między zmiennymi ma praktyczne znaczenie. Etapy metody są następujące:

1. Sprawdzenie modelu. Jeśli model nie został wcześniej oszacowany (`object@model` jest `NULL`), to funkcja przerywa działanie z komunikatem o błędzie.

2. Pobranie współczynnika kierunkowego `coef` przy zmiennej `x` z oszacowanego modelu.
3. Ocena znaku współczynnika. Informacja ta jest zapisywana w zmiennej `c.x`.
4. Ocena wielkości efektu.

- Jeśli `abs(coef)<0.01`, współczynnik jest zbyt mały, by mieć praktyczne znaczenie – relacja ekonomiczna jest słaba lub nieistotna.
- W przeciwnym razie współczynnik jest na tyle duży, że może mieć sens ekonomiczny. Zmienna `s` jest zwiększana o 1.

Wynik zapisywany jest w zmiennej `rel`.

5. Zwracanie wyników w postaci listy z trzema elementami:

- `coef.x`: znak i interpretacja współczynnika,
- `rel`: ocena znaczenia ekonomicznego,
- `s`: liczba punktów za *ekonomiczną siłę* relacji (0 lub 1).

```
setMethod(
  "evaluate.economic.significance",
  "RegExpSystem",
  function(object) {
    if (is.null(object@model)) {
      stop("Model nie został oszacowany")
    }
    s <- 0
    summary <- summary(object@model)
    coef <- summary$coefficients[2, 1]
    if (coef > 0) {
      cat("Współczynnik kierunkowy jest dodatni, co oznacza dodatnią
       ↪  zależność między x i y.\n")
      c.x <- "Coef(x) > 0"
    } else if (coef < 0) {
      cat("Współczynnik kierunkowy jest ujemny, co oznacza ujemną
       ↪  zależność między x i y.\n")
      c.x <- "Coef(x) < 0"
    } else {
      cat("Współczynnik kierunkowy jest równy zero, co wskazuje na
       ↪  brak zależności między x i y.\n")
      c.x <- "Coef(x) = 0"
    }
    if (abs(coef) < 0.01) {
      cat("Współczynnik kierunkowy jest niski, co może wskazywać na
       ↪  brak praktycznych walorów relacji  ekonomicznej.\n")
```

```
      rel <- "Brak praktycznych walorów relacji  ekonomicznej"
    } else {
      cat("Współczynnik kierunkowy ma sens ekonomiczny.\n")
      rel <- "Współczynnik kierunkowy ma sens ekonomiczny"
      s <- s+1
    }
    return(list(
      coef.x = c.x,
      rel    = rel,
      s      = s
    ))
  }
)
```

Kod poniżej przygotowuje miejsce na funkcję `make.decision`, której zadaniem będzie podjęcie decyzji o jakości modelu na podstawie wcześniej przeprowadzonych analiz.

```
setGeneric(
  "make.decision",
  function(object) standardGeneric("make.decision"))
```

```
## [1] "make.decision"
```

Zadaniem metody `make.decision` jest podsumowanie jakości modelu i wydanie decyzji, czy jest on użyteczny zarówno statystycznie, jak i ekonomicznie. Algorytm metody przebiega następująco:

1. Funkcja wypisuje w konsoli informację o rozpoczęciu etapu decyzyjnego.
2. Wywoływana jest metoda:

`evaluate.statistical.significance`.

Z jej wyników pobierana jest wartość `k`, czyli liczba *pozytywnych* rezultatów dla testów statystycznych. Następnie wywoływana jest metoda:

`evaluate.economic.significance`.

Pobierana jest wartość `s`, czyli ocena jakości ekonomicznej (0 lub 1).

3. Jeśli model spełnia więcej niż 1 warunek statystyczny i jest on ekonomicznie uzasadniony (`s>0`), funkcja wypisuje pozytywną decyzję. W przeciwnym razie decyzja jest negatywna.

4. Kod zwraca listę z dwoma elementami: `stat` (`k`) i `econ` (`s`), które mogą być użyte do dalszej analizy.

```
setMethod(
  "make.decision",
  "RegExpSystem",
  function(object) {
    cat("\n*** Podjęcie decyzji na podstawie analizy ***\n\n")
    k <- evaluate.statistical.significance(object)$k
    s <- evaluate.economic.significance(object)$s
    if(k > 1 & s > 0) {
      cat("\n\n*** Model posiada walory statystyczne i ekonomiczne
       ↪  ***\n\n")
    } else {
      cat("\n\n*** Model nie posiada walorów statystycznych
       ↪  i ekonomicznych ***\n\n")
    }
    return(list(
      stat = k,
      econ = s
    ))
  }
)
```

Kolejny kod tworzy funkcję generyczną S4 o nazwie `plot.diagnostics`.

```
setGeneric(
  "plot.diagnostics",
  function(object) standardGeneric("plot.diagnostics"))
```

```
## [1] "plot.diagnostics"
```

Celem metody jest przeprowadzenie diagnostyki *obrazowej* oszacowań modelu – głównie za pomocą wykresów dopasowania i reszt. Działanie funkcji obejmuje następujące kroki:

1. Sprawdzenie, czy model istnieje. Jeśli model (`object@model`) nie został oszacowany, to funkcja przerywa działanie z komunikatem o błędzie.
2. Wykonanie obliczeń pomocniczych:

- `fitted` – wartości dopasowane przez model,
- `resid` – reszty, czyli różnice między wartościami rzeczywistymi y a dopasowanymi.

3. Utworzenie ramki danych `df.plot` zawierającej `x`, `y`, wartości dopasowane i reszty, służącej do utworzenia wykresów.
4. Utworzenie wykresu dopasowania (`p1`):

- punkty rzeczywiste (`x`, `y`),
- linia dopasowania (wartości przewidywane przez model),
- tytuł.

5. Wykonanie wykresu reszt (`p2`):

- rozrzut reszt względem `x`, który pomaga zidentyfikować nieliniowość, heteroskedastyczność lub inne problemy.

6. Wyświetlenie wykresów w konsoli graficznej.

```
setMethod(
  "plot.diagnostics",
  "RegExpSystem",
  function(object) {
    if (is.null(object@model)) {
      stop("Model nie został oszacowany")
    }
    fitted <- fitted(object@model)
    resid <- residuals(object@model)
    df.plot <- data.frame(
      x = object@x,
      y = object@y,
      fitted = fitted,
      residuals = resid
    )
    p1 <- df.plot %>%
      ggplot(aes(x = x)) +
      geom_point(
        aes(y = y),
        color = "grey",
        size = 2
      ) +
      geom_line(
        aes(y = fitted),
        color = "grey20",
        size = 1
      ) +
      ggtitle("Wykres dopasowania modelu") +
      xlab("X") +
      ylab("Y")
```

```
    p2 <- df.plot %>%
      ggplot(aes(x = x, y = residuals)) +
      geom_point(
        color = "grey",
        size = 2
      ) +
      ggtitle("Wykres reszt") +
      xlab("X") +
      ylab("Reszty")
    # print(p1)
    # print(p2)
    return(list(p1, p2))
  }
)
```

Przedstawiony powyżej kod definiuje system analizy regresji liniowej oparty na klasie S4. System zawiera następujące komponenty:

1. Klasę `RegExpSystem`, która przechowuje dane wejściowe i model:

   - `slot`:
     - `x` – wektor liczbowy (zmienna niezależna),
     - `y` – wektor liczbowy (zmienna zależna),
     - `model` – dowolny obiekt (`ANY`), który przechowuje wyniki oszacowanego modelu.
   - `validity`:
     - sprawdzenie, czy `x` i `y` mają tę samą długość,
     - weryfikację, czy obie zmienne są liczbowe.

2. Metodę `fit.model`, która szacuje parametry modelu dla danych `x` i `y`:

   - utworzenie modelu za pomocą funkcji `lm`,
   - zgłoszenie błędu (przerwanie działania), jeśli liczba obserwacji jest zbyt mała,
   - zwracanie obiektu z oszacowanym modelem.

3. Metodę `evaluate.statistical.significance`, która ocenia statystyczną jakość modelu. Metoda informuje, czy oceny parametrów modelu są statystycznie istotne i jaka jest jakość dopasowania modelu do danych empirycznych:

   - statystyki:

  - p-wartość dla współczynnika kierunkowego,
  - R^2 – wartość współczynnika determinacji jako miara dopasowania,
  - standardowy błąd równania (SEE),
  - testy diagnostyczne:
    * Jarque–Bera (normalność rozkładu składnika losowego),
    * Durbin–Watson (autokorelacja rzędu pierwszego składnika losowego),
    * Breusch–Godfrey (autokorelacja wyższych rzędów składnika losowego),
    * White (heteroskedastyczność składnika losowego).

4. Metodę `evaluate.economic.significance`, która ocenia walory ekonomiczne modelu:

- analiza współczynnika kierunkowego:
  - czy wskazuje na zależność dodatnią lub ujemną,
  - czy wartość współczynnika kierunkowego ma praktyczne znaczenie.

5. Metodę `make.decision`, która podsumowuje wyniki analizy w celu podjęcia decyzji o jakości modelu.

6. Metodę `plot.diagnostics`, która tworzy wykresy diagnostyczne:

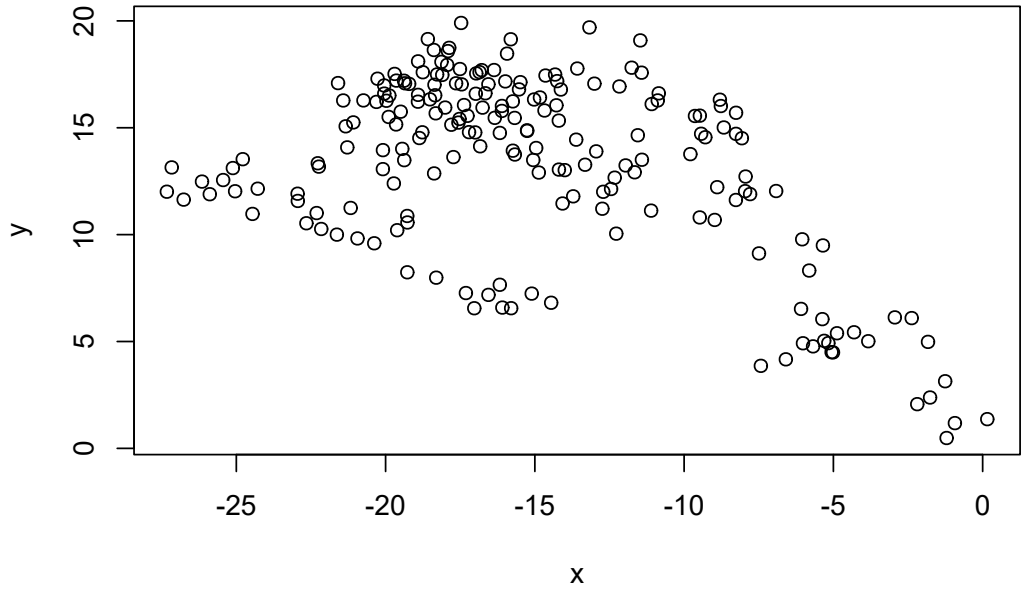
- wykres dopasowania modelu (`x` względem `y` i linia regresji),
- wykres reszt (`x` względem reszt).

Klasa `RegExpSystem` ma spełniać funkcję praktyczną. Ma umożliwiać kompleksową analizę regresji liniowej, obejmując zarówno walidację danych, dopasowanie modelu do danych, ocenę i wizualizację wyników. Poniżej podano praktyczne zastosowanie systemu, dla dwóch zmiennych `x` i `y`, które zdefiniowano jako procesy niestacjonarne, obejmujące 200 obserwacji.

```
set_seed()
x <- rnorm(200) %>%
  cumsum()
y <- rnorm(200) %>%
  cumsum()
```

Utworzono wykres punktowy zmiennych `x` i `y` (rys. 3.60).

```
plot(x, y)
```



Rysunek 3.60. Wykres punktowy

Następnie utworzono obiekt klasy `RegExpSystem` z wcześniej wygenerowanymi zmiennymi `x` i `y` (*instancję* klasy S4 z przypisanymi danymi). Slot `model` pozostaje pusty (`NULL`), dopóki nie zostanie wywołana funkcja `fit.model`. Jeśli walidacja jest pozytywna, to obiekt `model` klasy `RegExpSystem` zostanie poprawnie utworzony i będzie gotowy do dalszej analizy.

```
model <- new(
  "RegExpSystem",
  x = x,
  y = y
)
```

Poniższy kod wywołuje metodę `fit.model` dla obiektu `model`. Powoduje to dopasowanie modelu regresji liniowej.

```
reg <- fit.model(model)
```

```
## Model został oszacowany
```

Następne polecenia uruchamiają metody:

- `evaluate.statistical.significance`,

- `evaluate.economic.significance`,
- `make.decision`,
- `plot.diagnostics` (rys. 3.61 i 3.62),

dla obiektu `reg`, czyli wcześniej oszacowanego modelu.

```
stat <- evaluate.statistical.significance(reg)
```

```
##
## Call:
## lm(formula = y ~ x, data = data.frame(x = object@x, y = object@y))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.6816 -3.4834  0.9071  2.8960  7.1597
##
## Coefficients:
##             Estimate Std. Error t value             Pr(>|t|)
## (Intercept)  7.72448    0.71744  10.767 < 0.0000000000000002 ***
## x           -0.36648    0.04445  -8.244   0.0000000000000227 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.731 on 198 degrees of freedom
## Multiple R-squared:  0.2555, Adjusted R-squared:  0.2518
## F-statistic: 67.96 on 1 and 198 DF,  p-value: 0.00000000000002267
##
## Test Jarque'a-Bery:
##
##  Jarque Bera Test
##
## data:  resid
## X-squared = 13.541, df = 2, p-value = 0.001147
##
## Składnik losowy nie ma rozkładu normalnego.
## Test Durbina-Watsona:
##
##  Durbin-Watson test
##
## data:  object@model
## DW = 0.089302, p-value < 0.00000000000000022
## alternative hypothesis: true autocorrelation is greater than 0
##
## Występuje statystycznie istotna autokorelacja pierwszego rzędu.
## Test Breuscha-Godfreya:
##
##  Breusch-Godfrey test for serial correlation of order up to 1
##
```

```
## data:  object@model
## LM test = 178.46, df = 1, p-value < 0.00000000000000022
##
## Występuje statystycznie istotna autokorelacja pierwszego rzędu.
## Test White'a:
##
##  studentized Breusch-Pagan test
##
## data:  object@model
## BP = 3.2747, df = 1, p-value = 0.07035
##
## Test t-Studenta dla współczynnika kierunkowego:
## t-value: -8.244055
##
## Podsumowanie wyników:
## Model jest statystycznie istotny (p-value < 0.05)
## R2: 0.2555396
## Model ma niską jakość dopasowania (R2 <= 0.7)
## SEE: 3.721203
```

```
econ <- evaluate.economic.significance(reg)
```

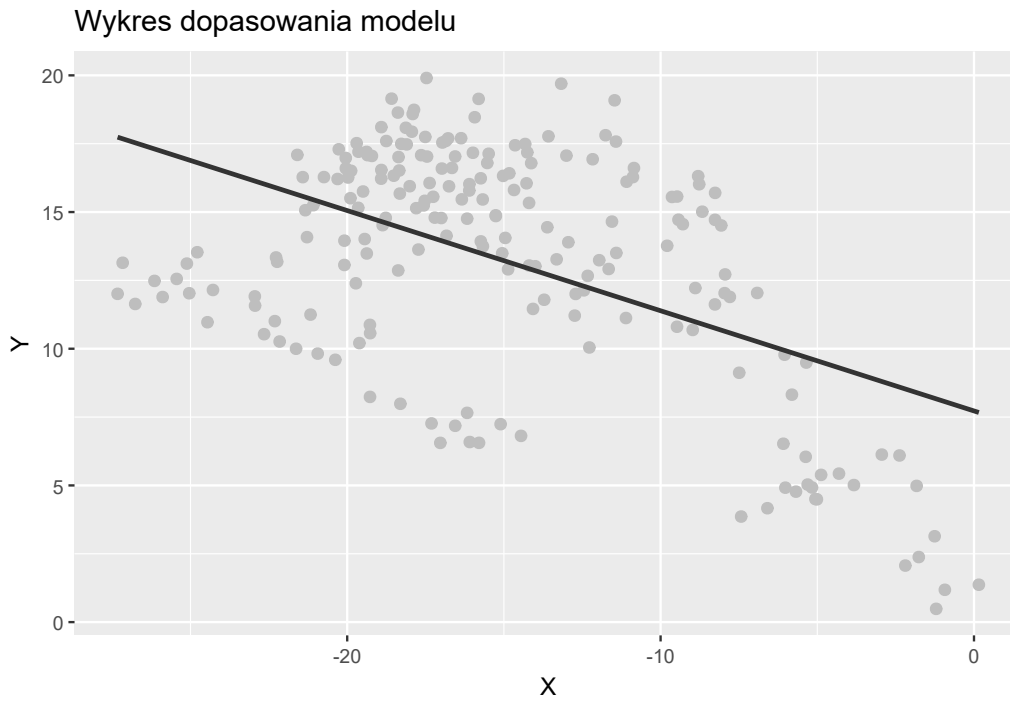
```
## Współczynnik kierunkowy jest ujemny, co oznacza ujemną zależność
 ↪  między x i y.
## Współczynnik kierunkowy ma sens ekonomiczny.
```

```
dec <- make.decision(reg)
```

```
##
## *** Podjęcie decyzji na podstawie analizy ***
##
##
## Call:
## lm(formula = y ~ x, data = data.frame(x = object@x, y = object@y))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.6816 -3.4834  0.9071  2.8960  7.1597
##
## Coefficients:
##             Estimate Std. Error t value             Pr(>|t|)
## (Intercept)  7.72448    0.71744  10.767 < 0.0000000000000002 ***
## x           -0.36648    0.04445  -8.244   0.0000000000000227 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.731 on 198 degrees of freedom
```

```
## Multiple R-squared:  0.2555, Adjusted R-squared:  0.2518
## F-statistic: 67.96 on 1 and 198 DF,  p-value: 0.00000000000002267
##
## Test Jarque'a-Bery:
##
##  Jarque Bera Test
##
## data:  resid
## X-squared = 13.541, df = 2, p-value = 0.001147
##
## Składnik losowy nie ma rozkładu normalnego.
## Test Durbina-Watsona:
##
##  Durbin-Watson test
##
## data:  object@model
## DW = 0.089302, p-value < 0.00000000000000022
## alternative hypothesis: true autocorrelation is greater than 0
##
## Występuje statystycznie istotna autokorelacja pierwszego rzędu.
## Test Breuscha-Godfreya:
##
##  Breusch-Godfrey test for serial correlation of order up to 1
##
## data:  object@model
## LM test = 178.46, df = 1, p-value < 0.00000000000000022
##
## Występuje statystycznie istotna autokorelacja pierwszego rzędu.
## Test White'a:
##
##  studentized Breusch-Pagan test
##
## data:  object@model
## BP = 3.2747, df = 1, p-value = 0.07035
##
## Test t-Studenta dla współczynnika kierunkowego:
## t-value: -8.244055
##
## Podsumowanie wyników:
## Model jest statystycznie istotny (p-value < 0.05)
## R2: 0.2555396
## Model ma niską jakość dopasowania (R2 <= 0.7)
## SEE: 3.721203
## Współczynnik kierunkowy jest ujemny, co oznacza ujemną zależność
 ↪  między x i y.
## Współczynnik kierunkowy ma sens ekonomiczny.
##
##
## *** Model posiada walory statystyczne i ekonomiczne ***
```

```
plot.diagnostics(reg)[[1]]
```

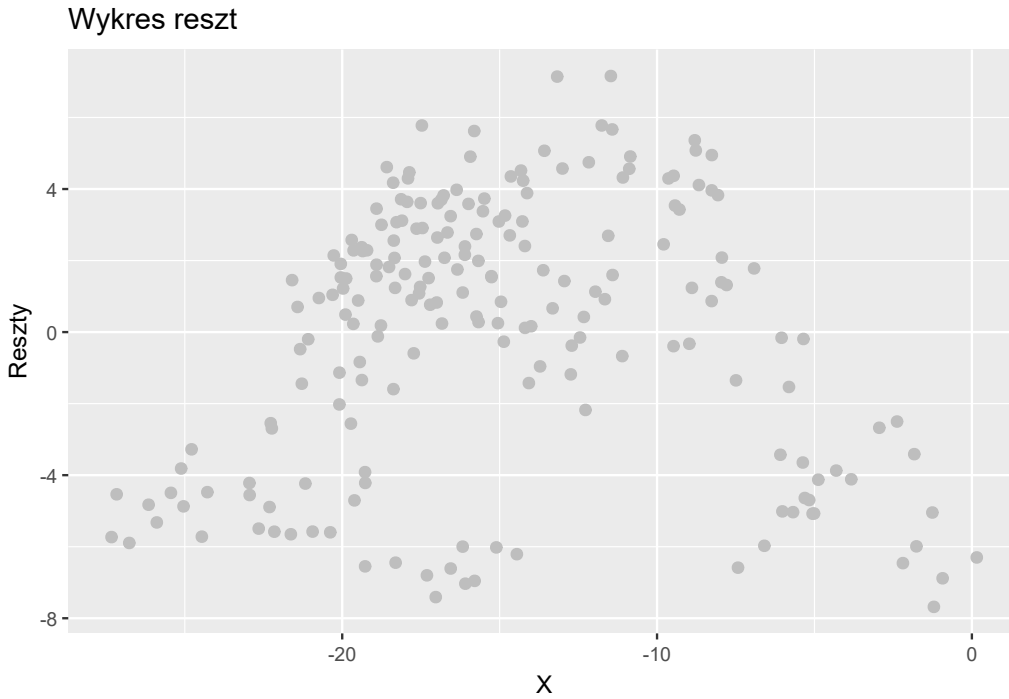


Rysunek 3.61. Dopasowanie w modelu – klasa RegExpSystem – przypadek 1

```
plot.diagnostics(reg)[[2]]
```

Dla porównania podano skrócone wyniki analizy regresji dla funkcji `lm` i obiektu `reg` (tabl. 3.31 i 3.32). Wyniki są identyczne.

```
lm(y ~ x)$coefficients %>%
  stargazer(
    type = "latex",
    title = "Wyniki regresji dla funkcji \\texttt{lm} i obiektu
     ↪  \\texttt{reg} -- \\texttt{stargazer}",
    label = "tab:jgkxne",
    header = FALSE
  )
```



Rysunek 3.62. Dopasowanie w modelu – klasa `RegExpSystem` – przypadek 2

Tablica 3.31. Wyniki regresji dla funkcji `lm` i obiektu `reg` – `stargazer`

(Intercept)	x
7.724	-0.366

```
reg@model
```

Call: lm(formula = y ~ x, data = data.frame(x = object@x, y = object@y))

Coefficients: (Intercept) x
7.7245 -0.3665

```
slot(reg, "model") %>%
  stargazer(
    type = "latex",
    title = "Wyniki regresji dla funkcji \\texttt{lm} i obiektu
     ↪  \\texttt{reg} -- \\texttt{stargazer} i \\texttt{slot}",
    label = "tab:bzoxhn", header = FALSE
  )
```

Tablica 3.32. Wyniki regresji dla funkcji `lm` i obiektu `reg` – `stargazer` i `slot`

	Dependent variable:
	y
x	−0.366***
	(0.044)
Constant	7.724***
	(0.717)
Observations	200
R^2	0.256
Adjusted R^2	0.252
Residual Std. Error	3.731 (df = 198)
F Statistic	67.964*** (df = 1; 198)
Note:	*p<0.1; **p<0.05; ***p<0.01

Należy przypomnieć, że obiekt `reg` należy do klasy S4 o nazwie `RegExpSystem`, w której występują sloty: `x`, `y` i `model`. Metoda `fit.model` przechowuje wyniki estymacji modelu w slocie `model`. Odwołanie się do tego slotu w obiekcie `reg` jest możliwe za pomocą operatora @ lub funkcji `slot`, jak wyżej.

W następnym podrozdziale przedstawiono zagadnienie tworzenia interaktywnych stron internetowych z wykorzystaniem aplikacji `R/Shiny`.

Rozdział 4

Aplikacje R/Shiny

4.1. Wprowadzenie

W tym podrozdziale zostanie omówione zagadnienie tworzenia interaktywnych stron internetowych. Jest to możliwe dzięki opracowanemu przez organizację RStudio serwerowi kodu HTML o nazwie Shiny Server (dalej: R/Shiny) i bibliotece shiny. Systemy R/Shiny stanowią narzędzia pozwalające na tworzenie interaktywnych aplikacji bez konieczności znajomości języków HTML, CSS czy JavaScript. Uruchomienie kodu R z elementami shiny jest możliwe na dwa sposoby, tj. poprzez zainstalowanie:

1. Na lokalnym komputerze programu R, biblioteki shiny i – dla wygody obsługi – programu RStudio.
2. Na zdalnym komputerze pod kontrolą systemu Linux programów: R, Shiny Server i biblioteki shiny.

Serwer R/Shiny można obsługiwać również za pomocą RStudio Connect oraz kontenerów Docker. R/Shiny umożliwia rozszerzanie funkcjonalności poprzez API oraz integrację z bazami danych, co sprawia, że rozwiązania oparte na R/Shiny są szeroko używane w analizie danych i raportowaniu.

Szczególnie przydatna jest lokalna instalacja systemu (rozwiązanie pierwsze powyżej), gdyż serwer jest uruchamiany *w locie* i można natychmiast obserwować działanie kodu. Takie środowisko pracy ułatwia tworzenie, testowanie i modyfikowanie aplikacji.

4.2. Podstawy budowy aplikacji

Budowę kodu aplikacji R/Shiny rozpoczyna się od utworzenia katalogu o nazwie aplikacji i pliku app.R w programie RStudio (*File|New File|Shiny Web App*), postępując zgodnie z instrukcjami kreatora. Plik app.R stanowi jądro aplikacji (interaktywnej strony internetowej). Uzyskanie kodu HTML strony internetowej polega na przesłaniu kodu R/Shiny na serwer, gdzie następuje jego wykonanie i zwrócenie wyniku do przeglądarki internetowej. W ten sposób otrzymuje się obraz strony internetowej. Najistotniejsze jest to, że na takiej stronie można umieścić elementy formularzy HTML, za pomocą których przekazuje się parametry do kodu R. Obliczenia są wykonywane na serwerze i wizualizowane na stronie internetowej poprzez przeglądarkę.

Kod aplikacji można przygotować na różne sposoby:

- w programie RStudio,
- w innym edytorze tekstowym.

W poniższym opracowaniu zastosowano pierwsze rozwiązanie. Jak wspomniano wcześniej, kod aplikacji jest zapisany w pliku app.R. Do uruchomienia kodu będą potrzebne co najmniej dwie biblioteki R – shiny i shinydashboard (Chang i Borges Ribeiro, 2021) – oraz inne specjalistyczne biblioteki. Aplikacja składa się z trzech podstawowych elementów:

- nagłówka (*header*),
- panelu bocznego (*sidebar*),
- panelu głównego (*body*).

Szkic kodu programu podano poniżej.

```
library(shiny)
library(shinydashboard)

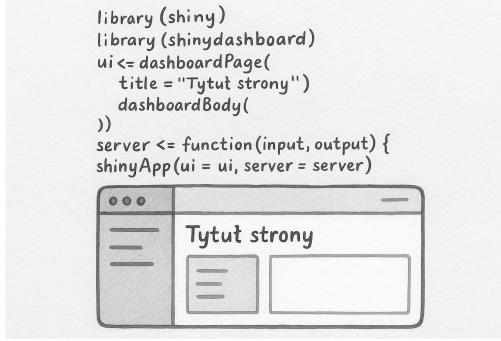
ui <- dashboardPage(
    dashboardHeader(
        title = "Tytuł strony"
    ),
    dashboardSidebar(
    ),
    dashboardBody(
    )
)

server <- function(input, output) {
}

shinyApp(ui = ui, server = server)
```

Zanim przejdzie się do dalszych rozważań, można – czysto poglądowo i z przymrużeniem oka – sprawdzić, jaką artystyczną interpretację powyższego kodu potrafi wygenerować narzędzie takie jak Sora. Rysunek 4.1 stanowi swobodną wizualizację interfejsu użytkownika, utworzoną przez model AI.

Warto przy tym zaznaczyć, że Sora nie jest dużym modelem językowym (LLM) w tradycyjnym sensie, takim jak GPT-4[1]. Jest to model typu *tekst-do-wideo*, opracowany przez firmę OpenAI. Jego zadaniem jest generowanie realistycznych obrazów lub sekwencji wideo na podstawie opisu tekstowego.



Rysunek 4.1. Obraz wygenerowany przez model AI Sora
https://openai.com/sora

Naciśnięcie przycisku Run/Run App w RStudio spowoduje wygenerowanie kodu HTML strony internetowej i wyświetlenie wyniku w domyślnej przeglądarce. Jak widać, biblioteki shiny i shinydashboard zawierają swoiste słowa kluczowe, lecz w samym kodzie strony można osadzać kod obliczeniowy w języku R. Ta własność bibliotek oznacza, że na stronie internetowej można wydawać polecenia i przekazywać parametry do kodu R, a wyniki obliczeń są aktualizowane na bieżąco. W praktyce daje to ogromne możliwości przetwarzania danych. Dzięki metodom *reaktywnym*, język R/Shiny umożliwia dynamiczne aktualizowanie interfejsu użytkownika w odpowiedzi na zmiany w danych wejściowych. Integracja z ekosystemem języka R umożliwia wykorzystanie bibliotek statystycznych i wizualizacyjnych, takich jak ggplot2, dplyr czy plotly.

Z powyższego wynika, że w kodzie strony mają zastosowanie wszelkie zasady programowania w języku R. Podany w następnym podrozdziale kod strony internetowej będzie przedstawiony tak, aby pokazać możliwości wykonywania interaktywnych obliczeń i ich wizualizacji.

[1] Duży model językowy (*Large Language Model*, LLM) to rodzaj modelu sztucznej inteligencji, który uczy się rozpoznawać, interpretować i generować język naturalny na podstawie ogromnych zbiorów tekstów. Działa poprzez przewidywanie kolejnego słowa na podstawie wcześniejszego kontekstu, wykorzystując zaawansowane architektury sieci neuronowych. Dzięki temu LLM potrafi w praktyce *rozumieć* i *tworzyć* tekst, ponieważ podczas uczenia przyswaja złożone wzorce i zależności językowe występujące w danych treningowych.

4.3. System prognostyczny

Poniżej podano zarys systemu prognostycznego dla kursu walutowego. Aplikacja wykonuje obliczenia na podstawie trzech modeli prognostycznych:

1. Trendu liniowego (Montgomery i in., 2015).
2. Holta (Holt, 1957).
3. ARIMA (Box i Jenkins, 1970).

Model trendu liniowego opisuje zmienność y w czasie jako funkcję liniową w postaci:

$$y_t = \beta_0 + \beta_1 t + \epsilon_t \tag{4.1}$$

gdzie:

y_t – zmienna objaśniana (prognozowana),
t – zmienna czasowa, $t = 1, \dots, T$,
β_0 – wyraz wolny,
β_1 – współczynnik trendu,
ϵ_t – składnik losowy.

Model ten stosuje się do prognozowania tendencji długookresowej, tj. gdy zmienne wykazują stałą tendencję wzrostową lub spadkową, bez uwzględniania efektów sezonowych lub cyklicznych. Szacowanie parametrów modelu odbywa się za pomocą metody MNK, co pozwala na estymację parametrów β_0 i β_1. Model trendu liniowego jest szeroko stosowany w analizie ekonomicznej, finansowej oraz w prognozowaniu szeregów czasowych.

Model Holta jest metodą prognozowania szeregów czasowych opartą na wygładzaniu wykładniczym, która uwzględnia dwie składowe szeregu: poziom oraz trend (przyrost). W modelu Holta występują dwie funkcje: F_t i S_t. Pierwsza z nich jest odpowiedzialna za składową *poziomu* szeregu czasowego, natomiast druga za składową *przyrostu* tego szeregu. W modelu występują dwa parametry: α i β. Pierwszy z nich jest stałą wygładzania dla składowej poziomu zmiennej, natomiast drugi jest odpowiedzialny za korektę przyrostu zmiennej. Model Holta ma postać:

$$\begin{aligned} F_t &= \alpha y_t + (1-\alpha)(F_{t-1} + S_{t-1}) \\ S_t &= \beta(F_t - F_{t-1}) + (1-\beta)S_{t-1} \end{aligned} \tag{4.2}$$

gdzie α i β są stałymi wygładzania dla poziomu i trendu, $\alpha, \beta \in (0, 1)$.

Wartości początkowe dla funkcji F_t i S_t można przyjąć w następujący sposób:

$$\begin{aligned} F_1 &= y_1 \\ S_1 &= 0 \end{aligned} \tag{4.3}$$

Model `ARIMA(p, d, q)` jest szeroko stosowanym modelem szeregów czasowych, który łączy w sobie składowe autoregresyjne (AR), różnicowanie (I) oraz średnią ruchomą (MA). Model pozwala na analizę zarówno procesów stacjonarnych, jak i niestacjonarnych (por. podrozdział 3.2.1).

Parametr p określa liczbę opóźnień analizowanego szeregu czasowego w składowej autoregresyjnej, d oznacza rząd różnicowań potrzebnych do uzyskania stacjonarności, a q wskazuje liczbę opóźnień składnika losowego (innowacji) uwzględnionych w składowej średniej ruchomej. Równanie modelu `ARIMA(p, d, q)` można zapisać w postaci:

$$\phi(B)(1-B)^d Y_t = \theta(B)\epsilon_t \tag{4.4}$$

gdzie:

B – operator przesunięcia w czasie,
$\phi(B)$ i $\theta(B)$ – wielomiany stopnia autoregresji i średniej ruchomej,
ϵ_t – składnik losowy.

Estymacja parametrów modelu ARIMA odbywa się najczęściej za pomocą metody największej wiarygodności, a wybór optymalnych wartości p, d i q można przeprowadzić za pomocą kryteriów informacyjnych, takich jak AIC czy BIC (Box i Jenkins, 1970; Akaike, 1974; Schwarz, 1978).

Model ARIMA znajduje szerokie zastosowanie w analizie ekonomicznej, finansowej i prognozowaniu, zwłaszcza w przypadkach, gdy zmienne wykazują zależności w czasie, lecz nie posiadają wyraźnej struktury sezonowej.

Poniżej podano kod aplikacji `R/Shiny`. Wystarczy skopiować poniższy kod i uruchomić w `RStudio`.

```
# System prognostyczny R/Shiny
# Piotr Wdowiński

library(bdscale)
library(dplyr)
library(forecast)
library(ggplot2)
library(magrittr)
library(plotly)
library(quantmod)
library(reshape2)
library(shiny)
library(shinydashboard)

getSymbols(
  "EURPLN=X",
  src = "yahoo",
```

```
  from = "2020-01-01",
  to   = "2024-12-31"
)

data <- Cl(`EURPLN=X`) %>%
  .[!weekdays(index(.)) %in% c("sobota", "niedziela"), ]

df <- data.frame(
  date  = as.Date(index(data)),
  value = as.numeric(coredata(data))
)

holt <- function(y, alpha, beta, h) {
  n <- length(y)
  F <- numeric(n+h)
  S <- numeric(n+h)

  F[1] <- y[1]
  S[1] <- y[2]-y[1]

  for(t in 2:n){
    F[t] <- alpha*y[t]+(1-alpha)*(F[t-1]+S[t-1])
    S[t] <- beta*(F[t]-F[t-1])+(1-beta)*S[t-1]
  }

  fitted <- F[1:n]

  for(T in (n+1):(n+h)){
    F[T] <- F[n] + (T-n)*S[n]
  }

  forecast <- F[(n+1):(n+h)]
  return(list(
    fitted   = fitted,
    forecast = forecast
  ))
}

url <- paste0(
  "https://fonts.googleapis.com/",
  "css2?family=Noto+Sans&display=swap"
)

ui <- dashboardPage(
  dashboardHeader(title = "Prognoza kursu EUR/PLN"),
  dashboardSidebar(
    selectInput(
      "model",
      "Model",
```

```
      choices  = c("Liniowy", "Holt", "ARIMA"),
      selected = "Liniowy"
    ),
    conditionalPanel(
      condition = "input.model=='Holt'",
      sliderInput(
        "alfa",
        "Wybierz alfa",
        min   = 0,
        max   = 1,
        value = 0.5,
        step  = 0.01
      ),
      sliderInput(
        "beta",
        "Wybierz beta",
        min   = 0,
        max   = 1,
        value = 0.5,
        step  = 0.01
      )
    ),
    conditionalPanel(
      condition = "input.model=='ARIMA'",
      numericInput(
        "p",
        "Parametr p",
        value = 1,
        min   = 0,
        max   = 5,
        step  = 1
      ),
      numericInput(
        "q",
        "Parametr q",
        value = 1,
        min   = 0,
        max   = 5,
        step  = 1
      )
    ),
    selectInput(
      "n",
      "Okres danych",
      choices = list(
        "Ostatni miesiąc" = 22,
        "Ostatni rok"     = 252,
        "Cała próba"      = nrow(df)
      ),
```

```
      selected = 22
    ),
    sliderInput(
      "h",
      "Horyzont prognozy (dni)",
      min   = 1,
      max   = 10,
      value = 5,
      step  = 1
    ),
    selectInput(
      "palette",
      "Schemat kolorów wykresu",
      choices  = c("Set1", "Set2", "Set3"),
      selected = "Set1"
    ),
    sliderInput(
      "size",
      "Rozmiar linii",
      min   = 0.5,
      max   = 3,
      value = 1,
      step  = 0.1
    ),
    actionButton("predict", "Prognozuj")
  ),
  dashboardBody(
    tags$head(
      tags$link(
        href = url,
        rel  = "stylesheet"
      ),
      tags$style(HTML("
        body{font-family:'NotoSans', sans-serif;}")
      )
    ),
    fluidRow(
      box(
        title = "Źródło danych",
        width = 12,
        tableOutput("source")
      )
    ),
    fluidRow(
      box(
        title = "Wykres prognozy",
        width = 12,
        plotlyOutput("plot")
      )
```

```
    ),
    fluidRow(
      box(
        title = "Błędy prognoz",
        width = 12,
        tableOutput("error")
      )
    ),
    fluidRow(
      box(
        title = "Oszacowane parametry modelu",
        width = 12,
        tableOutput("params")
      )
    ),
    fluidRow(
      box(
        title = "Komunikaty błędów",
        width = 12,
        textOutput("message")
      )
    )
  )
)

server <- function(input, output, session) {

  forecast.data <- reactiveVal(NULL)
  error.message <- reactiveVal("")

  output$source <- renderTable({
    "Źródło danych: Yahoo Finance"
  })

  observeEvent(input$predict, {
    tryCatch({

      n <- as.numeric(input$n)

      df.subset <- df %>%
        tail(as.numeric(n)) %>%
        transform(index = seq_along(value))

      h <- as.numeric(input$h)
      params <- NULL

      if(input$model == "Liniowy") {

        model <- lm(value ~ index, data = df.subset)
```

```
    h.index <- (nrow(df.subset)+1):(nrow(df.subset)+h)
    fitted   <- predict(model, newdata = df.subset)
    forecast <- predict(
      model, newdata = data.frame(index = h.index)
    )
    params <- coef(model)

  } else if(input$model == "Holt") {

    model <- holt(
      df.subset$value,
      alpha = input$alfa,
      beta  = input$beta,
      h     = h
    )
    fitted   <- model$fitted
    forecast <- model$forecast
    params <- c(input$alfa, input$beta)
    names(params) <- c("alfa", "beta")

  } else {

    model <- Arima(
      df.subset$value,
      order = c(input$p, 1, input$q)
    )
    fitted   <- fitted(model)
    forecast <- forecast(model, h = h)$mean
    params <- coef(model)

  }

  actual <- df.subset$value
  me     <- mean(abs(actual - fitted), na.rm = TRUE)
  mape   <- mean(
    abs((actual - fitted) / actual) * 100,
    na.rm = TRUE
  )

  df.error <- data.frame(
    "Średni błąd bezwzględny" = round(me, 4),
    "Średni błąd procentowy (%)" = round(mape, 4),
    check.names = FALSE
  )

  output$error <- renderTable(
    {df.error},
    digits = 4
  )
```

```
    if(!is.null(params)) {
      output$params <- renderTable({
        data.frame(
          `Parametry` = names(params),
          `Wartości`  = round(params, 4)
        )
      }, digits = 4)
    } else {
      output$params <- renderTable({
        data.frame(
          Parametry = "Brak parametrów",
          Wartości  = NA
        )
      })
    }

    f.dates <- seq(
      from       = max(df.subset$date),
      by         = "days",
      length.out = 2*(h+1)
    )[-1]

    f.dates <- f.dates[
      !weekdays(f.dates) %in% c("sobota", "niedziela")
    ][1:h]

    plot.data <- data.frame(
      date = c(df.subset$date, f.dates),
      actual = c(actual, rep(NA, h)),
      fitted = c(fitted, rep(NA, h)),
      forecast = c(rep(NA, n), forecast)
    ) %>%
      melt(id.vars="date") %>%
      mutate(
        type = case_when(
          variable == "actual" ~ "Rzeczywiste",
          variable == "fitted" ~ "Dopasowane",
          variable == "forecast" ~ "Prognozowane"
        )
      )

    forecast.data(plot.data)
    error.message("")

  }, error = function(e) {
    error.message(
      paste(
        "Błąd: Nie można dopasować modelu - ",
```

```
        e$message
      )
    )
    forecast.data(NULL)
  })
})

output$plot <- renderPlotly({
  req(forecast.data())
  p <- ggplot(
    forecast.data(),
    aes(
      x = date,
      y = value,
      color = type,
      group = type,
      text = paste0(
        "Data: ", date,
        "<br>Wartość: ", round(value, 4),
        "<br>Typ: ", type
      )
    )) +
    geom_line(linewidth = input$size) +
    scale_colour_brewer(palette = input$palette) +
    scale_x_bd(
      business.dates = forecast.data()$date,
      max.major.breaks = 25,
      labels = scales::date_format("%d %b %Y"),
      guide = guide_axis(angle = 90)
    ) +
  theme_minimal() +
    theme(
      axis.text.x = element_text(
        angle = 90,
        vjust = 0.5,
        hjust = 1
      )
    ) +
    labs(
      title = "Prognoza kursu EUR/PLN",
      x     = "Data",
      y     = "Kurs EUR/PLN"
    )

  ggplotly(p, tooltip = "text")
})

output$message <- renderText({
  error.message()
```

```
  })
}

shinyApp(ui = ui, server = server)
```

Poniżej podano opis najważniejszych obszarów aplikacji.

1. **Importowanie bibliotek**:

- Załadowanie bibliotek potrzebnych do budowy aplikacji `Shiny`, analizy danych i wizualizacji:
  - `shiny`, `shinydashboard` – budowanie interfejsu aplikacji internetowej,
  - `quantmod` – pobieranie danych finansowych,
  - `forecast` – modelowanie szeregów czasowych,
  - `ggplot2`, `plotly` – tworzenie interaktywnych wykresów,
  - `magrittr`, `dplyr`, `reshape2` – przetwarzanie danych,
  - `bdscale` – obsługa skali czasu z uwzględnieniem dni roboczych.

2. **Pobranie danych finansowych**:

- Funkcja `getSymbols` pobiera kurs EUR/PLN z serwisu *Yahoo Finance* w okresie od 1 stycznia 2020 do 31 grudnia 2024.
- Następnie wyodrębniana jest kolumna kursu zamknięcia (`Cl()`), a weekendy są usuwane z danych.

3. **Przygotowanie ramki danych** (`df`):

- Tworzona jest ramka danych z kolumnami: `date` (data) i `value` (kurs EUR/PLN).

4. **Implementacja metody Holta**:

- Definiowana jest funkcja `holt()`, która implementuje model wygładzania wykładniczego Holta z parametrami `alpha` i `beta`.

5. **Tworzenie interfejsu użytkownika** (UI):

- `dashboardPage` definiuje strukturę aplikacji.
- `dashboardSidebar` zawiera:
  - wybór modelu prognostycznego (`Liniowy`, `Holt`, `ARIMA`),
  - parametry dla modeli (`alpha`, `beta`, p, q),

  - wybór okresu danych (`miesiąc`, `rok`, `cały zbiór`),
  - wybór horyzontu prognozy (`h` dni),
  - opcje wizualizacji (schemat kolorów, rozmiar linii),
  - przycisk *Prognozuj*.
- `dashboardBody` zawiera:
  - źródło danych (`Yahoo Finance`),
  - wykres prognozy (`plotly`),
  - tablicę błędów prognoz,
  - tablicę parametrów modelu,
  - komunikaty błędów.

6. **Tworzenie serwera aplikacji** (`server`):

- `reactiveVal` przechowuje prognozę oraz ewentualne komunikaty błędów.
- `renderTable` wyświetla źródło danych.
- `observeEvent(input$predict, {...})`:
  - pobiera liczbę dni do analizy,
  - tworzy podzbiór danych i transformuje indeksy dat,
  - w zależności od wybranego modelu:
    * `Liniowy`: szacuje parametry regresji liniowej (`lm`),
    * `Holt`: prognozuje przy użyciu funkcji `holt`,
    * `ARIMA`: szacuje parametry modelu `ARIMA` (`Arima`).
  - oblicza błędy prognoz (średni błąd absolutny `MAE`, średni absolutny błąd procentowy `MAPE`),
  - tworzy dane do wizualizacji (wartości: rzeczywiste, dopasowane i prognozowane),
  - `renderPlotly` generuje interaktywny wykres prognoz,
  - `renderText` wyświetla komunikaty błędów.

7. **Uruchomienie aplikacji** (`shinyApp`):

- `shinyApp(ui, server)` startuje aplikację Shiny.

Dzięki przedstawionemu kodowi i jego opisowi, Czytelnik może zaimplementować, uruchomić oraz dostosować aplikację do własnych potrzeb w środowisku `RStudio`.

Zakończenie

Książkę przygotowano w celu szerokiego ujęcia wybranych zagadnień z zakresu modelowania ekonometrycznego oraz zaawansowanego wykorzystania języka R w zastosowaniach praktycznych. Szczególny nacisk został położony na przedstawienie zarówno podstaw teoretycznych, jak i aspektów praktycznych, co odpowiada rosnącym wymaganiom stawianym współczesnym analitykom danych, badaczom oraz praktykom w dziedzinach takich jak ekonomia, finanse, statystyka i ekonometria.

Przedstawiono najważniejsze zagadnienia modelowania ekonometrycznego, obejmujące procesy stochastyczne, liniowe modele regresji, metody testowania założeń klasycznych modeli liniowych, zagadnienia związane z weryfikacją statystyczną oraz oceną jakości dopasowania modeli do danych empirycznych. Szczególną uwagę zwrócono na problematykę testowania składników losowych: normalności rozkładu, heteroskedastyczności, autokorelacji oraz prawidłowości postaci funkcyjnej, które stanowią fundament analizy ekonometrycznej. Podkreślono znaczenie nie tylko budowy modeli, lecz również konieczności ich wszechstronnej weryfikacji, co warunkuje wiarygodność uzyskiwanych wyników oraz decyzji podejmowanych na ich podstawie.

Zaletą książki jest ścisłe powiązanie zagadnień teoretycznych z praktycznymi przykładami implementacji w języku R. Zaprezentowano w niej modelowanie zmiennych makroekonomicznych, takich jak stopa procentowa czy inflacja, konstrukcję modeli prognostycznych dla indeksów giełdowych oraz analizę rynku samochodów elektrycznych, obejmującą modelowanie cen z uwzględnieniem technicznych cech pojazdów. Wykorzystanie języka R w podanym kontekście pozwala w pełni wykorzystać jego potencjał obliczeniowy, graficzny oraz możliwości automatyzacji procesów obliczeniowych, co ma istotne znaczenie w pracy współczesnych analityków.

Język R, będący centralnym narzędziem omawianym w książce, wyróżnia się wysoką elastycznością, szerokim zakresem funkcjonalności oraz możliwością integracji z innymi technologiami. Jego ekosystem bibliotek umożliwia realizację złożonych zadań analitycznych, począwszy od podstawowych obliczeń statystycznych, poprzez analizę szeregów czasowych i modelowanie wielowymiarowe, aż po implementację metod

uczenia maszynowego. Szczególnie istotna pozostaje możliwość pracy z dużymi zbiorami danych oraz tworzenia dynamicznych dokumentów i aplikacji internetowych, co sprawia, że język R pełni rolę narzędzia o charakterze interdyscyplinarnym.

Współczesne środowisko pracy analityków danych wymaga umiejętności nie tylko w zakresie konstruowania modeli ekonometrycznych, lecz również w obszarze prezentacji wyników oraz wniosków w sposób przejrzysty i zrozumiały. Z tego względu w książce uwzględniono zagadnienia związane z tworzeniem dynamicznych dokumentów w języku R/Markdown oraz budową aplikacji R/Shiny. Wdrożenie tych rozwiązań umożliwia prezentację wyników analiz w formie interaktywnej, co znacząco zwiększa ich wartość użytkową, pozwalając odbiorcom na eksplorację danych statystycznych oraz samodzielne przeprowadzanie symulacji czy analiz. Integracja wymienionych narzędzi z językiem R zapewnia spójny system, który umożliwia nie tylko realizację zaawansowanych analiz, lecz także efektywną prezentację ich wyników w różnych formatach, takich jak HTML, PDF czy DOCX.

Rozwój metod analitycznych, postępująca cyfryzacja oraz dynamicznie zmieniające się otoczenie gospodarcze sprawiają, że przed współczesną ekonometrią stawiane są coraz bardziej złożone wyzwania. Oprócz tradycyjnych modeli statystycznych, coraz większe znaczenie zyskują metody eksploracji danych, algorytmy uczenia maszynowego oraz zaawansowane metody prognozowania. Wspomniane metody umożliwiają identyfikację skomplikowanych wzorców, relacji nieliniowych oraz zjawisk o charakterze ukrytym, które często pozostają poza zasięgiem klasycznych narzędzi analizy ekonometrycznej. W tym kontekście język R, dzięki swojej elastyczności, pozwala na wdrażanie nowoczesnych metod analitycznych, co znacząco poszerza zakres jego możliwych zastosowań w analizie danych ekonomicznych i finansowych.

Podkreślono również, że efektywne wykorzystanie języka R oraz metod ekonometrycznych wymaga nie tylko znajomości zasad statystyki i ekonometrii, lecz także kompetencji w zakresie programowania, znajomości struktur danych oraz umiejętności logicznego i krytycznego myślenia. Współczesne analizy danych coraz częściej realizowane są w warunkach znacznej niepewności, przy ograniczonych zasobach czasowych oraz konieczności przetwarzania dużych zbiorów danych pochodzących z wielu źródeł. Z tego względu niezbędne stają się narzędzia umożliwiające automatyzację procesów analitycznych, integrację zbiorów danych oraz dynamiczną aktualizację wyników analiz, co również znalazło odzwierciedlenie w treści książki.

Liczne przykłady implementacji kodu w języku R zamieszczone w książce mają na celu ułatwienie zrozumienia mechanizmów działania omawianych metod oraz pokazanie sposobów przenoszenia rozwiązań teoretycznych do zastosowań praktycznych. Uwzględniono zagadnienia związane z pracą z obiektami języka R, przetwarzaniem potokowym, tworzeniem funkcji użytkownika, organizacją zbiorów danych statystycznych oraz integracją języka R z innymi narzędziami i środowiskami. Przedstawione zagadnienia stanowią istotny element warsztatu współczesnego ekonometryka i analityka danych.

Książka, oprócz przekazywania wiedzy merytorycznej, wskazuje również kierunki dalszego rozwoju w zakresie modelowania ekonometrycznego i analizy danych. Współczesne trendy w ekonometrii obejmują nie tylko rozwój nowych metod statystycznych, lecz także integrację ekonometrii z technologiami `big data` oraz zastosowania w obszarach takich jak ekonomia behawioralna, analiza sieci społecznych czy modelowanie wpływu czynników środowiskowych na procesy gospodarcze. Język R, dzięki otwartości kodu źródłowego oraz aktywnej społeczności użytkowników i twórców bibliotek, stanowi środowisko sprzyjające wdrażaniu takich innowacji oraz prowadzeniu eksperymentów statystycznych.

Przekazana w książce wiedza może okazać się przydatna zarówno dla osób rozpoczynających pracę z modelowaniem ekonometrycznym, jak i dla bardziej doświadczonych użytkowników języka R, poszukujących bardziej zaawansowanych rozwiązań analizy danych. Przedstawione zagadnienia mogą stanowić podstawę do dalszego pogłębiania wiedzy oraz inspirować do podejmowania własnych projektów badawczych i praktycznych, z wykorzystaniem języka R.

Podsumowując, książka wpisuje się w nurt nowoczesnego podejścia do analizy danych, w którym solidne podstawy teoretyczne łączone są z praktycznym zastosowaniem narzędzi programistycznych, takich jak język R. Takie podejście odpowiada współczesnym wyzwaniom, związanym z ilością i różnorodnością danych, potrzebą szybkiego podejmowania decyzji oraz koniecznością tworzenia analiz o wysokim stopniu precyzji i elastyczności. Zgromadzona w książce wiedza oraz liczne przykłady praktyczne stanowią wartościowe wsparcie dla wszystkich zainteresowanych prowadzeniem pogłębionych analiz ekonometrycznych oraz rozwijaniem własnych kompetencji w tej dynamicznie rozwijającej się dziedzinie.

Summary

This book has been prepared with the aim of providing a comprehensive and coherent treatment of selected topics in econometric modelling, as well as an advanced application of the R programming language in practical contexts. Particular emphasis has been placed on presenting both theoretical foundations and practical aspects, reflecting the growing demands placed on contemporary data analysts, researchers, and practitioners in fields such as economics, finance, statistics, and econometrics.

The book discusses the most important issues in econometric modelling, including stochastic processes, linear regression models, methods for testing the assumptions of classical linear models, statistical inference and verification, and the assessment of model fit to empirical data. Special attention is devoted to testing error-term normality, heteroscedasticity, autocorrelation, and functional form specification, all of which form the foundation of rigorous econometric analysis. Their importance is emphasised not only in model construction, but also in conducting comprehensive model diagnostics, which determines the reliability of the conclusions drawn and the decisions based on them.

A key advantage of the book lies in its explicit integration of theoretical concepts with practical examples of implementation in the R language. It presents the modelling of macroeconomic variables, such as interest rates and inflation, the development of forecasting models for stock market indices, and the analysis of the electric vehicle market, including price modelling based on the technical characteristics of the vehicles. The use of R in these contexts enables the full exploitation of its computational, graphical, and process automation capabilities, which is of substantial importance in the work of modern analysts.

The R language, which serves as the central tool addressed in this book, is distinguished by its high degree of flexibility, extensive functionality, and capacity for integration with other technologies. Its ecosystem of libraries facilitates the execution of complex analytical tasks, ranging from basic statistical calculations, through time series analysis and multivariate modelling, to the implementation of machine learning methods. Of particular importance is the ability to work with large datasets and to create dynamic documents and web applications, which positions R as a tool with an inherently interdisciplinary character.

The contemporary work environment of data analysts requires proficiency not only in constructing econometric models, but also in the clear and comprehensible presentation of results and the communication of findings. For this reason, the book also addresses topics related to the creation of dynamic documents using R/Markdown and the development of web applications with the R/Shiny framework. The adoption of these solutions makes it possible to present analytical results in an interactive format, which significantly enhances their practical value by enabling users to explore data and conduct simulations or analyses independently. The integration of these tools with R ensures a coherent system that allows both the execution of advanced analyses and the effective presentation of results in various formats, such as HTML, PDF, and DOCX.

The development of analytical methods, ongoing digitalisation, and the rapidly evolving economic environment mean that contemporary econometrics faces increasingly complex challenges. In addition to traditional statistical models, growing importance is being attached to data mining techniques, machine learning algorithms, and advanced forecasting methods. These approaches enable the identification of complex patterns, non-linear relationships, and latent phenomena that frequently remain beyond the reach of classical econometric tools. In this context, the flexibility of R allows for the implementation of modern analytical methods, thereby substantially broadening the range of its potential applications in economic and financial data analysis.

It has also been emphasised that the effective use of R and econometric methods requires not only knowledge of statistical and econometric principles but also competencies in programming, familiarity with data structures, and the ability to think logically and critically. Contemporary data analyses are increasingly conducted under conditions of considerable uncertainty, limited time resources, and the necessity of processing large datasets derived from multiple sources. Consequently, tools that enable the automation of analytical processes, the integration of data repositories, and the dynamic updating of analytical results have become indispensable, and these aspects have also been reflected in the book's content.

Numerous examples of R code implementation included in the book are intended to facilitate understanding of the mechanisms underlying the methods discussed and to illustrate how theoretical concepts can be translated into practical applications. Topics addressed include working with R objects, pipeline processing, user-defined function creation, data management, and the integration of R with other tools and environments. These subjects represent an essential component of the skill set of the modern econometrician and data analyst.

In addition to conveying substantive knowledge, the book points to further directions for development in the areas of econometric modelling and data analysis. Contemporary trends in econometrics involve not only the advancement of new statistical methods but also the integration of econometrics with Big Data technologies and applications in areas such as behavioural economics, social network analysis, and modelling the impact of environmental factors on economic processes. Owing to its

open-source nature and the presence of an active community of users and library developers, R provides a conducive environment for the implementation of such innovations and for conducting methodological experimentation.

The knowledge imparted in this book may prove valuable both for individuals commencing work in econometric modelling and for advanced users of the R language seeking more sophisticated analytical solutions. The topics presented may serve as a basis for further, in-depth study and may inspire the pursuit of independent research and analytical projects using R.

In summary, the book is situated within the paradigm of modern data analysis, in which robust theoretical foundations are combined with practical applications of programming tools such as R. This approach responds to contemporary challenges associated with the volume and diversity of data, the need for rapid decision-making, and the requirement for analyses characterised by a high degree of precision and flexibility. The knowledge compiled in the book, together with numerous practical examples, constitutes a valuable and comprehensive resource for all those interested in conducting thorough econometric analyses and in developing their own competencies within this dynamically evolving field.

Dodatek A

Lista książek poświęconych językowi R i jego zastosowaniom

Kod realizujący zadanie wygenerowania tablicy z podsumowaniem dla książek (tabl. A.1).

```
library(dplyr)
library(kableExtra)
library(knitr)
library(magrittr)
library(readxl)
library(stringr)

df <- read_excel("input/R_books.xlsx") %>%
  mutate(Rok = as.numeric(Rok)) %>%
  arrange(is.na(Rok), Rok)

df %>%
  kbl(
    format = "latex",
    booktabs = TRUE,
    longtable = TRUE,
    caption = "Lista książek poświęconych językowi \\texttt{R} i jego
     ↪  zastosowaniom",
    label = "zmxdsr"
  ) %>%
  kable_styling(
    latex_options = c("striped", "repeat_header"),
    font_size = 8,
    position = "center"
  ) %>%
  column_spec(1, width = "0.80cm") %>%
  column_spec(2, width = "3.50cm") %>%
  column_spec(3, width = "2.80cm") %>%
```

```
  column_spec(4, width = "2.00cm") %>%
  column_spec(5, width = "1.50cm") %>%
  column_spec(6, width = "1.90cm") %>%
  column_spec(7, width = "3.50cm") %>%
  landscape()
```

Tablica A.1. Lista książek poświęconych językowi R i jego zastosowaniom

Rok	Tytuł	Autorzy	Wydawnictwo	ISBN	Główny temat	Krótki opis
2008	Applied Econometrics with R (en)	Christian Kleiber; Achim Zeileis	Springer	978-0387773162	Ekonometria	Praktyczne procedury ekonometryczne z serii „Use R!”
2008	Applied Spatial Data Analysis with R (en)	Roger S. Bivand; Edzer Pebesma; Virgilio Gómez-Rubio	Springer	978-0387781709	Analiza danych przestrzennych	Kompendium dotyczące przetwarzania, wizualizacji i analizy danych przestrzennych
2008	Bioconductor Case Studies (en)	Florian Hahne; Wolfgang Huber; Robert Gentleman; Seth Falcon	Springer	978-0387772394	Bioinformatyka	Przewodnik oparty na studiach przypadków do analizy danych genomowych
2008	Introductory Statistics with R (en)	Peter Dalgaard	Springer	978-0387790534	Statystyka	Wprowadzenie do statystyki z przykładami i zadaniami
2009	ggplot2: Elegant Graphics for Data Analysis (en)	Hadley Wickham	Springer	978-0387981406	Grafika	Wprowadzenie do grafiki i pakietu ggplot2 do tworzenia wizualizacji
2011	The Art of R Programming (en)	Norman Matloff	No Starch Press	978-1593273842	Programowanie	Wszechstronny przegląd na potrzeby efektywnego tworzenia oprogramowania
2012	Beginning R: The Statistical Programming Language (en)	Mark Gardener	Wrox (Wiley)	978-1118164303	Programowanie i statystyka dla początkujących	Krok po kroku: wprowadzenie do programowania i podstawowych metod statystycznych
2012	Modelowanie polskiej gospodarki z pakietem R (pl)	Michał Rubaszek	Szkoła Główna Handlowa. Oficyna Wydawnicza	978-8373787421	Modelowanie makroekonomiczne	Modelowanie ekonometryczne gospodarki Polski

Tablica A.1. Lista książek poświęconych językowi R i jego zastosowaniom *(continued)*

Rok	Tytuł	Autorzy	Wydawnictwo	ISBN	Główny temat	Krótki opis
2012	Receptury w R: Podręcznik dla ekonomistów (pl)	Bogumił Kamiński; Mateusz Zawisza	Szkoła Główna Handlowa. Oficyna Wydawnicza	978-83-7378-762-9	Ekonomia	Podręcznik w formie „przepisów" dla studentów i praktyków ekonomii
2012	The R Book (2nd ed.) (en)	Michael J. Crawley	Wiley	978-0470973929	Kompleksowe opracowanie	Szerokie kompendium obejmujące korzystanie z R, statystykę i programowanie
2013	An Introduction to Statistical Learning: with Applications in R (en)	Gareth James; Daniela Witten; Trevor Hastie; Robert Tibshirani	Springer	978-1461471370	Uczenie statystyczne	Praktyczne wprowadzenie do kluczowych metod uczenia maszynowego/statystycznego
2013	Analiza danych z programem R. Modele liniowe z efektami stałymi, losowymi i mieszanymi (pl)	Przemysław Biecek	Wydawnictwo Naukowe PWN	978-8301174538	Modele liniowe/mieszane	Przewodnik po modelach liniowych
2013	Applied Predictive Modeling (en)	Max Kuhn; Kjell Johnson	Springer	978-1461468486	Prognozowanie	Modelowanie predykcyjne z wykorzystaniem ekosystemu caret
2014	Hands-On Programming with R (en)	Garrett Grolemund	O'Reilly	978-1449359010	Podstawy programowania	Nauka programowania poprzez budowę małych projektów: funkcje i struktury danych
2015	R Packages (en)	Hadley Wickham	O'Reilly	978-1491910597	Tworzenie pakietów	Jak budować, dokumentować, testować i udostępniać pakiety
2015	R in Action (2nd ed.) (en)	Robert I. Kabacoff	Manning	978-1617291388	Statystyka stosowana i grafika	Szeroki, oparty na przykładach, przegląd R do analizy danych, grafiki i modelowania
2016	Programowanie w języku R (pl)	Marek Gągolewski	Wydawnictwo Naukowe PWN	978-83-01-18939-6	Programowanie i analiza danych	Podręcznik programowania, analiz i symulacji

Tablica A.1. Lista książek poświęconych językowi R i jego zastosowaniom *(continued)*

Rok	Tytuł	Autorzy	Wydawnictwo	ISBN	Główny temat	Krótki opis
2017	Przewodnik po pakiecie R (pl)	Przemysław Biecek	Oficyna Wydawnicza GiS	978-83-62780-44-0	Wprowadzenie	Przewodnik po języku R i jego ekosystemie dla początkujących i praktyków
2017	Data Mining with R: Learning with Case Studies (2nd ed.) (en)	Luís Torgo	CRC Press	978-1482234893	Eksploracja danych	Wprowadzenie do technik eksploracji danych
2017	Efficient R Programming (en)	Colin Gillespie; Robin Lovelace	O'Reilly	978-1491950784	Wydajność/produktywność	Wskazówki i techniki pisania szybszego, bardziej wydajnego kodu
2017	R for Data Science (en)	Hadley Wickham; Garrett Grolemund	O'Reilly	978-1491910399	Tidyverse i data science	Gramatyka oparta na tidyverse do importu, transformacji, wizualizacji i modelowania
2017	Text Mining with R: A Tidy Approach (en)	Julia Silge; David Robinson	O'Reilly	978-1491981658	Eksploracja tekstu	Narzędzia do analizy tekstu
2018	R Graphics Cookbook (en)	Winston Chang	O'Reilly	978-1491978603	Grafika	Rozwiązania w formie „przepisów" do tworzenia typowych wykresów w base R i ggplot2
2018	Data Science and Predictive Analytics: Biomedical and Health Applications using R (en)	Ivo D. Dinov	Springer	978-3-319-72346-4	Data science w ochronie zdrowia	Metody data science i SI zastosowane do studiów przypadków biomedycznych
2018	R Markdown: The Definitive Guide (en)	Yihui Xie; J.J. Allaire; Garrett Grolemund	CRC Press	978-1138359420	Raportowanie	Tworzenie raportów, dokumentów i książek za pomocą R Markdown
2019	Practical Data Science with R (2nd ed.) (en)	Nina Zumel; John Mount	Manning	978-1617295874	Data science	Praktyczny proces data science: przygotowanie danych, modelowanie, ocena, wdrożenie

Tablica A.1. Lista książek poświęconych językowi R i jego zastosowaniom *(continued)*

Rok	Tytuł	Autorzy	Wydawnictwo	ISBN	Główny temat	Krótki opis
2019	Machine Learning with R (3rd ed.) (en)	Brett Lantz	Packt	978-1788295864	Uczenie maszynowe	Praktyczne techniki i przykłady uczenia maszynowego
2019	Advanced R (2nd ed.) (en)	Hadley Wickham	CRC Press	978-0815384571	Zaawansowane programowanie	Omówienie R, programowania i meta-programowania
2020	Wstęp do programowania i analizy danych w języku R (pl)	Piotr Wdowiński	Uniwersytet Łódzki	978-83-8220-278-6	Programowanie i analiza danych	Wprowadzenie do programowania i analizy danych
2022	Metody ilościowe w R: aplikacje ekonomiczne i finansowe (pl)	Katarzyna Kopczewska; Tomasz Kopczewski; Piotr Wójcik	CeDeWu	978-83-8102-549-2	Metody ilościowe	Zastosowania metod ilościowych w ekonomii i finansach
2024	R w naukach społecznych. Zastosowania naukowe i edukacja (pl)	Arkadiusz Kołodziej	Wydawnictwo Naukowe Scholar	978-83-68091-05-2	Nauki społeczne	Zastosowania R w badaniach i dydaktyce nauk społecznych

Dodatek B

Test Shapiro–Wilka

Poniższa interpretacja pochodzi z kodu źródłowego testu Shapiro–Wilka, który jest dostępny w repozytorium R na platformie GitHub pod adresem:

- https://github.com/wch/r-source/blob/trunk/src/library/stats/R/shapiro.test.R

Występuje tam pełna implementacja funkcji `shapiro.test` wraz z komentarzami, które mogą pomóc w lepszym zrozumieniu działania testu.

```
shapiro.test <- function(x)
{
    DNAME <- deparse1(substitute(x))
    stopifnot(is.numeric(x))
    x <- sort(x[complete.cases(x)])
    n <- length(x)
    if(is.na(n) || n < 3L || n > 5000L)
    stop("sample size must be between 3 and 5000")
    rng <- x[n] - x[1L]
    if(rng == 0) stop("all 'x' values are identical")
    if(rng < 1e-10) x <- x/rng # rescale to avoid ifault=6 with single
     ↪  version.
    res <- .Call(C_SWilk, x)
    RVAL <- list(statistic = c(W = res[1]), p.value = res[2],
         method = "Shapiro-Wilk normality test", data.name = DNAME)
    class(RVAL) <- "htest"
    return(RVAL)
}
```

Powyższy kod implementuje test Shapiro–Wilka, który sprawdza, czy dana próbka pochodzi z rozkładu normalnego. Funkcja `shapiro.test` przyjmuje jako argument wektor `x` i zwraca listę zawierającą wyniki według poniższego schematu:

1. **Przypisanie nazwy danych**

Funkcja `deparse1(substitute(x))` przypisuje zmiennej `DNAME` nazwę zmiennej wejściowej `x` jako ciąg znaków. Dzięki temu, wynik testu zawiera informację o nazwie testowanych danych.

2. **Sprawdzenie, czy** `x` **jest liczbowe**

Funkcja `stopifnot` sprawdza, czy `x` jest wektorem liczbowym. Jeśli nie, zwróci błąd.

3. **Przetworzenie danych**

Usuwane są brakujące wartości (NA) z `x`, a następnie `x` jest sortowane. Zmienna `n` przechowuje liczbę elementów w `x`.

4. **Weryfikacja rozmiaru próbki**

Funkcja sprawdza, czy rozmiar próbki `n` mieści się w zakresie od 3 do 5000. Jeśli nie, zwraca błąd, ponieważ test Shapiro–Wilka jest przeznaczony do badania próbek o takiej liczebności.

5. **Sprawdzenie zakresu wartości w** `x`

Zmienna `rng` reprezentuje zakres wartości w `x` (różnicę między największą i najmniejszą wartością). Jeśli wszystkie wartości w `x` są identyczne (`rng == 0`), funkcja zgłasza błąd. Jeśli `rng` jest bardzo mały (bliski zera), wartości w `x` są skalowane, aby uniknąć problemów z dokładnością.

6. **Przeprowadzenie testu**

Funkcja `.Call` wywołuje wewnętrzną funkcję C (`C_SWilk`), która wykonuje faktyczny test Shapiro–Wilka na danych `x`. Wynik testu jest przechowywany w `res`.

7. **Utworzenie wyniku**

Wynik testu jest przypisany do listy `RVAL`, która zawiera:

- `statistic`: wartość statystyki testowej `W`,
- `p.value`: *p*-wartość testu,
- `method`: nazwę metody,
- `data.name`: nazwę danych (`DNAME`).

Wynik jest klasyfikowany jako obiekt `htest` (typowy dla testów statystycznych w `R`), co umożliwia formatowanie i interpretację wyników w standardowy sposób.

Test Jarque'a–Bery

Poniższa funkcja `jarque.bera.test` pochodzi z biblioteki `tseries` (Trapletti i Hornik, 2024) (por. https://github.com/cran/tseries/blob/master/R/test.R).

```
jarque.bera.test <-
function(x)
{
    if((NCOL(x) > 1) || is.data.frame(x))
        stop("x is not a vector or univariate time series")
    if(any(is.na(x)))
        stop("NAs in x")
    DNAME <- deparse(substitute(x))
    n <- length(x)
    m1 <- sum(x)/n
    m2 <- sum((x-m1)^2)/n
    m3 <- sum((x-m1)^3)/n
    m4 <- sum((x-m1)^4)/n
    b1 <- (m3/m2^(3/2))^2
    b2 <- (m4/m2^2)
    STATISTIC <- n*b1/6+n*(b2-3)^2/24
    PVAL <- 1 - pchisq(STATISTIC,df = 2)
    PARAMETER <- 2
    METHOD <- "Jarque Bera Test"
    names(STATISTIC) <- "X-squared"
    names(PARAMETER) <- "df"
    structure(list(statistic = STATISTIC,
       parameter = PARAMETER,
       p.value = PVAL,
       method = METHOD,
       data.name = DNAME),
       class = "htest")
}
```

Powyższy kod implementuje test Jarque'a–Bery, który sprawdza, czy rozkład danych `x` jest normalny. Funkcja `jarque.bera.test` przyjmuje jako argument wektor `x` i zwraca obiekt z wynikami według poniższego schematu:

1. **Sprawdzenie, czy** `x` **jest wektorem**

Funkcja sprawdza, czy `x` jest jednowymiarowym wektorem lub szeregiem czasowym. Jeśli `x` jest wielowymiarowy lub jest ramką danych, wyświetlany jest komunikat o błędzie.

2. **Sprawdzenie braku wartości** `NA` **w** `x`

Jeśli x zawiera brakujące wartości (NA), funkcja zwróci błąd.

3. **Przypisanie nazwy danych**

Funkcja `deparse(substitute(x))` przypisuje zmiennej DNAME nazwę zmiennej wejściowej x jako ciąg znaków. Jest to wykorzystywane do wyświetlania w wyniku nazwy testowanych danych.

4. **Wyznaczenie liczby obserwacji** `n` **oraz momentów** `m1`**,** `m2`**,** `m3`**,** `m4`

Zmienna n przechowuje liczbę elementów w x. m1 to średnia z x, m2 to wariancja, m3 to trzeci moment centralny (skośność), a m4 to czwarty moment centralny (kurtoza).

5. **Obliczenie wartości statystyk skośności** `b1` **i kurtozy** `b2`

b1 reprezentuje statystykę skośności, a b2 kurtozy. Wartości te są wykorzystywane w statystyce testowej.

6. **Obliczenie wartości statystyki testowej** `STATISTIC`

Wartość statystyki testowej (STATISTIC) jest wyznaczana na podstawie liczby obserwacji n, statystyki skośności (b1) i kurtozy (b2).

7. **Obliczenie wartości p** (PVAL)

Wartość *p* (*p*-wartość) (PVAL) jest wyznaczana na podstawie rozkładu χ^2 z dwoma stopniami swobody. Im mniejsza wartość PVAL, tym bardziej dane odbiegają od rozkładu normalnego.

8. **Sformatowanie wyniku**

PARAMETER określa liczbę stopni swobody rozkładu $\chi^2(2)$, a METHOD to nazwa testu. Nazwy statystyki i parametru są przypisane dla czytelności wyniku.

9. **Utworzenie wyniku**

Wynik jest zwracany jako lista zawierająca:

- `statistic`: wartość statystyki testowej,
- `parameter`: liczba stopni swobody (2),
- `p.value`: *p*-wartość testu,
- `method`: nazwę testu,
- `data.name`: nazwę danych (DNAME).

Wynik jest klasyfikowany jako obiekt `htest`, co umożliwia formatowanie i interpretację wyników w standardowy sposób w R.

Test Goldfelda–Quandta

Test Goldfelda–Quandta jest zaimplementowany w bibliotece `lmtest` (`gqtest`: `Goldfeld-Quandt Test`) (por. https://www.rdocumentation.org/packages/lmtest/versions/0.9-40/topics/gqtest).

Postać funkcji podano poniżej (por. https://github.com/cran/lmtest/blob/master/R/gqtest.R).

```
gqtest <- function(formula, point = 0.5, fraction = 0,
  alternative = c("greater", "two.sided", "less"), order.by = NULL,
   ↪  data = list())
{
  dname <- paste(deparse(substitute(formula)))
  alternative <- match.arg(alternative)

  if(!inherits(formula, "formula")) {
    X <- if(is.matrix(formula$x))
           formula$x
         else model.matrix(terms(formula), model.frame(formula))
    y <- if(is.vector(formula$y))
           formula$y
         else model.response(model.frame(formula))
  } else {
    mf <- model.frame(formula, data = data)
    y <- model.response(mf)
    X <- model.matrix(formula, data = data)
  }

  k <- ncol(X)
  n <- nrow(X)
  if(point > 1) {
    if(fraction < 1) fraction <- floor(fraction * n)
    point1 <- point - ceiling(fraction/2)
    point2 <- point + ceiling(fraction/2 + 0.01)
  } else {
    if(fraction >= 1) fraction <- fraction/n
    point1 <- floor((point-fraction/2) * n)
    point2 <- ceiling((point+fraction/2) * n + 0.01)
  }
  if (point2 > n-k+1 | point1 < k) stop("inadmissable breakpoint/too
   ↪  many central observations omitted")

  if(!is.null(order.by))
  {
    if(inherits(order.by, "formula")) {
      z <- model.matrix(order.by, data = data)
      z <- as.vector(z[,ncol(z)])
```

```
    } else {
      z <- order.by
    }
    X <- as.matrix(X[order(z),])
    y <- y[order(z)]
  }

  rss1 <- sum(lm.fit(as.matrix(X[1:point1,]),y[1:point1])$residuals^2)
  rss2 <- sum(lm.fit(as.matrix(X[point2:n,]),y[point2:n])$residuals^2)
  mss <- c(rss1/(point1-k), rss2/(n-point2+1-k))

  gq <- mss[2]/mss[1]
  df <- c(n-point2+1-k, point1-k)
  names(df) <- c("df1", "df2")

  PVAL <- switch(alternative,
    "two.sided" = (2*min(pf(gq, df[1], df[2]), pf(gq, df[1], df[2],
     ↪  lower.tail = FALSE))),
    "less" = pf(gq, df[1], df[2]),
    "greater" = pf(gq, df[1], df[2], lower.tail = FALSE))

  alternative <- switch(alternative,
    "two.sided" = "variance changes from segment 1 to 2",
    "less" = "variance decreases from segment 1 to 2",
    "greater" = "variance increases from segment 1 to 2")

  method <- "Goldfeld-Quandt test"
  names(gq) <- "GQ"
  RVAL <- list(statistic = gq,
      parameter = df,
      method = method,
      alternative = alternative,
      p.value= PVAL,
      data.name=dname)

  class(RVAL) <- "htest"
  return(RVAL)
}
```

Powyższy kod implementuje test Goldfelda–Quandta (`gqtest`), który sprawdza stabilność wariancji składnika losowego na podstawie reszt modelu w różnych segmentach danych. Test ten zakłada, że wariancje reszt mogą różnić się między pierwszą a drugą częścią danych, co może być przydatne np. przy podejrzeniu heteroskedastyczności. Funkcja `gqtest` jest sformułowana według następującego schematu:

1. **Oznaczenie parametrów wejściowych**

- `formula`: formuła modelu.

- `point`: punkt podziału, który określa, gdzie dane będą dzielone na dwie grupy.
- `fraction`: określa liczbę obserwacji wyłączonych z centralnej części danych (do omijania centralnych obserwacji).
- `alternative`: rodzaj testu jednostronnego (“greater” lub “less”) lub dwustronnego (“two.sided”).
- `order.by`: opcjonalna zmienna używana do sortowania danych przed podziałem.
- `data`: opcjonalna ramka danych, z której pobierane są zmienne modelu.

2. **Przygotowanie argumentów i danych**

Zmienna `dname` zapisuje nazwę danych do wyświetlenia w wynikach. `match.arg` sprawdza, czy wartość `alternative` jest jedną z dozwolonych opcji (“greater”, “two.sided”, “less”).

3. **Pobranie zmiennych z formuły**

Jeśli `formula` nie jest obiektem klasy “formula”, kod pobiera macierz `X` i wektor `y` z modelu `formula`. W przeciwnym razie, jeśli `formula` jest poprawnym wzorem, tworzy ramkę modelu i przypisuje `X` i `y` do odpowiednich wartości z danych.

4. **Obliczenie punktu podziału**

- Jeśli `point` jest większy niż 1, traktowany jest jako liczba całkowita, czyli indeks w danych.
- Jeśli `point` jest liczbą z przedziału $(0, 1)$, traktowany jest jako proporcja długości danych.
- `point1` i `point2` wyznaczają indeksy, które wskazują, gdzie dane zostaną podzielone na dwie części.
- Sprawdzenie, czy podział jest wykonalny (czy obie grupy mają wystarczającą liczbę obserwacji), w przeciwnym razie funkcja zwraca błąd.

5. **Sortowanie danych**

Jeśli `order.by` jest ustawione, dane `X` i `y` są sortowane według tej zmiennej.

6. **Obliczenie reszt i średnich kwadratów reszt**

Kod oblicza sumę kwadratów reszt (`rss1` i `rss2`) dla dwóch podzbiorów danych, korzystając z funkcji `lm.fit`. Następnie dzieli każdą z nich przez liczbę stopni swobody, tworząc średnie kwadraty reszt (`mss`), czyli szacunki wariancji.

7. **Obliczenie wartości statystyki testowej** `gq` **i stopni swobody** `df`

Statystyka testowa `gq` jest stosunkiem wariancji drugiego segmentu do wariancji pierwszego segmentu. `df` to wektor zawierający liczbę stopni swobody dla obu grup.

8. **Obliczenie wartości p** (PVAL)

Wartość *p* (PVAL) jest obliczana na podstawie rozkładu F, w zależności od wybranej hipotezy alternatywnej.

9. **Określenie hipotezy alternatywnej**

- `"two.sided"` = "wariancja zmienia się między segmentem 1 a 2",
- `"less"` = "wariancja maleje między segmentem 1 a 2",
- `"greater"` = "wariancja rośnie między segmentem 1 a 2".

10. **Utworzenie wyniku**

Wynik testu jest zwracany jako lista zawierająca:

- `statistic`: wartość statystyki Goldfelda–Quandta,
- `parameter`: liczbę stopni swobody dla obu segmentów,
- `method`: nazwę testu,
- `alternative`: wybraną hipotezę alternatywną,
- `p.value`: *p*-wartość testu,
- `data.name`: nazwę danych (`dname`).

11. **Zwrócenie wyniku jako obiektu klasy** `htest`

Wynik jest klasyfikowany jako obiekt `htest`, co ułatwia wyświetlanie i interpretację w języku R.

Test Breuscha–Pagana

Test Breuscha–Pagana jest zaimplementowany w bibliotece `lmtest` (Hothorn i in., 2022) (por. https://github.com/cran/lmtest/blob/master/R/bptest.R).

```
bptest <- function(formula, varformula = NULL, studentize = TRUE,
  data = list(), weights = NULL)
{
  dname <- paste(deparse(substitute(formula)))

  if(!inherits(formula, "formula")) {
    X <- if(is.matrix(formula$x))
       formula$x
     else model.matrix(terms(formula), model.frame(formula))
    y <- if(is.vector(formula$y))
       formula$y
     else model.response(model.frame(formula))
    Z <- if(is.null(varformula)) X
          else model.matrix(varformula, data = data)
    wts <- weights(formula)
  } else {
    mf <- if(is.null(weights)) {
      model.frame(formula, data = data)
    } else {
      model.frame(formula, weights = weights, data = data)
    }
    y <- model.response(mf)
    X <- model.matrix(formula, data = data)
    Z <- if(is.null(varformula)) X
          else model.matrix(varformula, data = data)
    wts <- model.weights(mf)
  }
  if(is.null(wts)) wts <- rep.int(1, NROW(X))

  ## only use complete cases that are in both models
  if(!(all(c(row.names(X) %in% row.names(Z), row.names(Z) %in%
   ↪  row.names(X))))) {
    allnames <- row.names(X)[row.names(X) %in% row.names(Z)]
    X <- X[allnames,]
    Z <- Z[allnames,]
    y <- y[allnames]
    wts <- wts[row.names(X) %in% row.names(Z)]
  }

  ## need at least one intercept plus one regressor
  if(ncol(Z) < 2L) stop("the auxiliary variance regression requires at
   ↪  least an intercept and a regressor")

  k <- ncol(X)
  n <- sum(wts > 0) ## nrow(X)

  resi <- lm.wfit(X, y, wts)$residuals
  sigma2 <- sum(wts * resi^2)/n
```

```
  if(studentize) {
    w <- resi^2 - sigma2
    aux <- lm.wfit(Z, w, wts)
    bp <- n * sum(wts * aux$fitted.values^2)/sum(w^2)
    method <- "studentized Breusch-Pagan test"
  } else {
    f <- resi^2/sigma2 -1
    aux <- lm.wfit(Z, f, wts)
    bp <- 0.5 * sum(wts * aux$fitted.values^2)
    method <- "Breusch-Pagan test"
  }

  names(bp) <- "BP"
  df <- c("df" = aux$rank - 1)
  RVAL <- list(statistic = bp,
      parameter = df,
      method = method,
      p.value= pchisq(bp, df, lower.tail = FALSE),
      data.name = dname)

  class(RVAL) <- "htest"
  return(RVAL)
}
```

Powyższy kod implementuje test Breuscha–Pagana (`bptest`), który sprawdza obecność heteroskedastyczności składnika losowego na podstawie reszt modelu. Funkcja zwraca wynik testu, który sprawdza, czy wariancja reszt zależy od wybranych zmiennych regresji według schematu:

1. **Oznaczenie argumentów funkcji**

- `formula`: formuła modelu.
- `varformula`: opcjonalna formuła dla zmiennych do oszacowania wariancji. Jeśli nie podano, używane są zmienne z `formula`.
- `studentize`: określa, czy należy zastosować test studentyzowany (domyślnie `TRUE`).
- `data`: opcjonalne dane dla zmiennych.
- `weights`: opcjonalne wagi dla obserwacji.

2. **Przygotowanie danych**

- `dname` zapisuje nazwę danych, aby wyświetlić ją w wynikach testu.
- Jeśli `formula` nie jest obiektem typu "formula", kod pobiera odpowiednie macierze `X` (zmienne objaśniające), `y` (zmienna zależna), oraz `Z` (zmienne do oszacowania wariancji) z `formula`. Jeśli `varformula` nie jest podane, przyjmuje `Z = X`. Pobierane są także wagi obserwacji (`wts`).

- Jeśli `formula` jest obiektem typu "formula", dane są pobierane bezpośrednio z argumentów `data` i `weights`.

3. **Sprawdzenie przypadków**

Sprawdzenie, czy używane są tylko te obserwacje, które są obecne zarówno w macierzy `X`, jak i macierzy `Z`.

4. **Sprawdzenie minimalnej liczby zmiennych**

Sprawdzenie, czy macierz `Z` ma przynajmniej dwie kolumny (przynajmniej jeden regresor i wyraz wolny).

5. **Wyliczenie reszt i wariancji reszt** (`sigma2`)

Obliczenie reszt (`resi`) dla modelu `X` względem `y` oraz ich wariancję ważoną (`sigma2`).

6. **Ustalenie warunków dla testu Breuscha–Pagana**

Jeśli `studentize` jest `TRUE`, różnice kwadratów reszt (`resi^2 - sigma2`) są wykorzystywane do regresji względem `Z`. Statystyka testowa `bp` jest obliczana jako wartość przeskalowana przez `n`, a metoda jest opisana jako *studentized Breusch–Pagan test*.

W przeciwnym razie test jest klasyczny. Używa stosunku kwadratów reszt (`resi^2/sigma2 - 1`) w regresji pomocniczej względem `Z`. Wartość statystyki testowej `bp` jest obliczana jako połowa sumy ważonych wartości dopasowanych.

7. **Zwrócenie wyniku testu**

Wynik jest zwracany jako lista zawierająca:

- `statistic`: wartość statystyki Breuscha–Pagana,
- `parameter`: liczbę stopni swobody regresji pomocniczej (`df`),
- `method`: wybraną wersję testu (studentyzowany lub klasyczny),
- `p.value`: *p*-wartość obliczoną na podstawie rozkładu χ^2,
- `data.name`: nazwę danych.

8. **Ustawienie klasy** `htest`

Wynik jest klasyfikowany jako obiekt `htest`, co ułatwia wyświetlanie i interpretację w języku R.

Literatura

Abedin, J., 2017. Modern R Programming Cookbook. Packt Publishing, Birmingham.

Acerbi, C., 2002. Spectral Measures of Risk: A Coherent Representation of Subjective Risk Aversion. Journal of Banking & Finance 26(7), 1505–1518. https://doi.org/10.1016/S0378-4266(02)00207-7.

Acerbi, C., Tasche, D., 2002. On the Coherence of Expected Shortfall. Journal of Banking & Finance 26(7), 1487–1503. https://doi.org/10.1016/S0378-4266(02)00206-5.

Akaike, H., 1974. A New Look at the Statistical Model Identification. IEEE Transactions on Automatic Control 19(6), 716–723. https://doi.org/10.1109/TAC.1974.1100705.

Akaike, H., 1973. Information Theory and an Extension of the Maximum Likelihood Principle, w: Petrov, B.N., Csáki, F. (red.), Second International Symposium on Information Theory. Akadémiai Kiadó, Budapest, s. 267–281.

Allaire, J., Xie, Y., Dervieux, C., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., Cheng, J., Chang, W., Iannone, R., 2025. rmarkdown: Dynamic Documents for R, R package version 2.30, https://github.com/rstudio/rmarkdown.

Anderson, E., 1935. The Irises of the Gaspé Peninsula. Bulletin of the American Iris Society 59, 2–5.

Angeloni, I., Kashyap, A.K., Mojon, B. (Red.), 2003. Monetary Policy Transmission in the Euro Area: A Study by the Eurosystem Monetary Transmission Network. Cambridge University Press, Cambridge, UK. https://doi.org/10.1017/CBO9780511492372.

Araci, D., 2019. FinBERT: Financial Sentiment Analysis with Pre-Trained Language Models, arXiv preprint arXiv:1908.10063, https://arxiv.org/abs/1908.10063. University of Amsterdam.

Arnold, J.B., 2024. ggthemes: Extra Themes, Scales and Geoms for ggplot2, R package version 5.1.0, https://jrnold.github.io/ggthemes/.

Arratia, A., 2014. Computational Finance: An Introductory Course with R, Atlantis Studies in Computational Finance and Financial Engineering. Atlantis Press, Paris.

Auguie, B., 2017. gridExtra: Miscellaneous Functions for ”Grid” Graphics, R package version 2.3, https://CRAN.R-project.org/package=gridExtra.

Baba, Y.J., Engle, R.F., Kraft, D.F., Kroner, K.F., 1990. Multivariate Simultaneous Generalized ARCH. Journal of Econometrics 47(1-2), 27–60. https://doi.org/10.1016/0304-4076(90)90092-X.

Bache, S.M., Wickham, H., 2022. magrittr: A Forward-Pipe Operator for R, R package version 2.0.3, https://magrittr.tidyverse.org.

Baranowski, P., 2008. Optymalna stopa inflacji w modelowaniu wzrostu gospodarczego. Ekonomista (4), 447–466.

Baranowski, P., Doryń, W., Łyziak, T., Stanisławska, E., 2021. Words and Deeds in Managing Expectations: Empirical Evidence from an Inflation Targeting Economy. Economic Modelling 95, 49–67. https://doi.org/10.1016/j.econmod.2020.12.003.

Baranowski, P., Górajski, M., Malaczewski, M., Szafrański, G., 2016. Inflation in Poland under State-Dependent Pricing. Ekonomický časopis 64(10), 937–957.

Barrett, T., Dowle, M., Srinivasan, A., Gorecki, J., Chirico, M., Hocking, T., 2024. data.table: Extension of 'data.frame', R package version 1.15.4, https://r-datatable.com.

Becker, R.A., Chambers, J.M., Wilks, A.R., 1988. The New S Language: A Programming Environment for Data Analysis and Graphics. Wadsworth & Brooks/Cole Advanced Books & Software, Pacific Grove, CA.

Becker, R., Osborn, D.R., Yildirim, D., 2012. A Threshold Cointegration Analysis of Interest Rate Pass-Through to UK Mortgage Rates. Economic Modelling 29(6), 2504–2513. https://doi.org/10.1016/j.econmod.2012.08.004.

Bera, A.K., Jarque, C.M., 1981. Efficient Tests for Normality, Homoscedasticity and Serial Independence of Regression Residuals: Monte Carlo Evidence. Economics Letters 7(4), 313–318. https://doi.org/10.1016/0165-1765(81)90035-5.

Berlinger, E., Illés, F., Somogyi, E., 2015. Mastering R for Quantitative Finance. Packt Publishing, Birmingham.

Białek, J., 2022. Scanner data processing in a newest version of the PriceIndices package. Statistical Journal of the IAOS 38(4), 1369–1397. https://doi.org/10.3233/SJI-220963.

Biecek, P., 2013. Analiza danych z programem R: modele liniowe z efektami stałymi, losowymi i mieszanymi. PWN, Warszawa.

Blanchard, O., Johnson, D.R., 2012. Macroeconomics, 6th ed. Pearson, Upper Saddle River, NJ.

Blangiewicz, M., Miłobędzki, P., 2009. The Rational Expectations Hypothesis of the Term Structure at the Polish Interbank Market. Przegląd Statystyczny 56(1), 23–39. https://doi.org/10.59139/ps.2009.01.2.

Blangiewicz, M., Miłobędzki, P., 2008. Is there a Nonlinear Mean Reversion in the Term Structure of Interest Rates at the Polish Interbank Market?, w: Soares, J.O., Pina, J., Catalão-Lopes, M. (red.), New Developments in Financial Modelling. Cambridge Scholars Publishing, Newcastle upon Tyne, s. 77–94.

Bollerslev, T., 1986. Generalized Autoregressive Conditional Heteroskedasticity. Journal of Econometrics 31(3), 307–327. https://doi.org/10.1016/0304-4076(86)90063-1.

Boole, G., 1854. An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities. Walton & Maberly, London.

Box, G.E.P., Jenkins, G.M., 1970. Time Series Analysis: Forecasting and Control. Holden-Day, San Francisco.
Box, G.E.P., Jenkins, G.M., Reinsel, G.C., Ljung, G.M., 2015. Time Series Analysis: Forecasting and Control, 5th ed. Wiley, Hoboken, NJ.
Box, G.E.P., Pierce, D.A., 1970. Distribution of Residual Autocorrelations in Autoregressive-Integrated Moving Average Time Series Models. Journal of the American Statistical Association 65(332), 1509–1526. https://doi.org/10.1080/01621459.1970.10481180.
Boyd, S., Vandenberghe, L., 2004. Convex Optimization. Cambridge University Press, Cambridge.
Boysel, S., Vaughan, D., 2021. fredr: An R Client for the FRED API, R package version 2.1.0, https://github.com/sboysel/fredr.
Brandt, P., 2016. MSBVAR: Markov-Switching, Bayesian, Vector Autoregression Models, R package version 0.9-3, https://CRAN.R-project.org/package=MSBVAR.
Breiman, L., 2001. Random Forests. Machine Learning 45(1), 5–32. https://doi.org/10.1023/A:1010933404324.
Breusch, T.S., 1978. Testing for Autocorrelation in Dynamic Linear Models. Australian Economic Papers 17(31), 334–355. https://doi.org/10.1111/j.1467-8454.1978.tb00635.x.
Breusch, T.S., Pagan, A.R., 1979. A Simple Test for Heteroscedasticity and Random Coefficient Variation. Econometrica 47(5), 1287–1294. https://doi.org/10.2307/1911963.
Brooks, C., 2019. Introductory Econometrics for Finance, 4th ed. Cambridge University Press, Cambridge. https://doi.org/10.1017/9781108524872.
Brzeszczyński, J., 2018. International Volatility Spillovers: WIG20, DJIA and FTSE100. Acta Universitatis Lodziensis. Folia Oeconomica 6(339), 125–146.
Brzoza-Brzezina, M., 2002. The Relationship between Real Interest Rates and Inflation (NBP Working Paper No. 23), https://static.nbp.pl/publikacje/materialy-i-studia/23_en.pdf. Narodowy Bank Polski.
Brzoza-Brzezina, M., Kłos, B., Kot, A., Łyziak, T., 2002. Hipoteza neutralności pieniądza (Raport). Narodowy Bank Polski, Departament Analiz i Badań.
Brzoza-Brzezina, M., Kolasa, M., Koloch, G., Makarski, K., Rubaszek, M., 2013. Monetary Policy in a Non-Representative Agent Economy: A Survey. Journal of Economic Surveys 27(4), 641–669. https://doi.org/10.1111/j.1467-6419.2011.00710.x.
Bukowski, S.I., 2025. The Determinants of Economic Growth in Poland in 2018–2023. European Research Studies Journal 28(1), 622–639. https://doi.org/10.35808/ersj/3926.
Bukowski, S.I., Gowers, R., 2016. The Degree of Integration of Equity Markets in Central Europe with the US and UK Equity Markets. Central European Review of Economics & Finance 11(1), 5–26.
Burns, P., 2012. The R Inferno. Lulu.com.
Cairns, A.J.G., 2004. Interest Rate Models: An Introduction. Princeton University Press, Princeton, NJ.

Chambers, J.M., 2008. Software for Data Analysis: Programming with R. Springer, New York, NY. https://doi.org/10.1007/978-0-387-75936-4.
Chambers, J.M., 1998. Programming with Data: A Guide to the S Language. Springer, New York, NY.
Chambers, J.M., Hastie, T.J. (Red.), 1992. Statistical Models in S, 1st ed. Chapman & Hall/CRC, London. https://doi.org/10.1201/9780203738535.
Chang, W., Borges Ribeiro, B., 2021. shinydashboard: Create Dashboards with Shiny, R package version 0.7.2, http://rstudio.github.io/shinydashboard/.
Chang, W., Cheng, J., Allaire, J., Sievert, C., Schloerke, B., Xie, Y., Allen, J., McPherson, J., Dipert, A., Borges, B., 2024. shiny: Web Application Framework for R, R package version 1.9.1, https://shiny.posit.co/.
Charemza, W.W., Deadman, D.F., 1997. New Directions in Econometric Practice: General to Specific Modelling, Cointegration, and Vector Autoregression, 2nd ed, https://ideas.repec.org/b/elg/eebook/1139.html. Edward Elgar Publishing, Cheltenham.
Chen, T., Guestrin, C., 2016. XGBoost: A Scalable Tree Boosting System, w: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, New York, NY, s. 785–794. https://doi.org/10.1145/2939672.2939785.
Chirico, M., Oehlschlägel, J., 2025. bit: Classes and Methods for Fast Memory-Efficient Boolean Selections, R package version 4.6.0, https://github.com/r-lib/bit.
Cohen, Y., Cohen, J.Y., 2008. Statistics and Data with R: An Applied Approach Through Examples. Wiley, Hoboken, NJ.
Condylios, S., Rudis, B., Kiener, P., 2022. netstat: Retrieve Network Statistics Including Available TCP Ports, R package version 0.1.2, https://github.com/stevecondylios/netstat.
Crawley, M.J., 2012. The R Book, 2nd ed. Wiley, Chichester.
Croissant, Y., Millo, G., 2018. Panel Data Econometrics with R. Wiley, Chichester.
Cryer, J.D., Chan, K.-S., 2008. Time Series Analysis: With Applications in R. Springer, New York, NY. https://doi.org/10.1007/978-0-387-75959-3.
Dahl, D.B., Scott, D., Roosen, C., Magnusson, A., Swinton, J., 2019. xtable: Export Tables to LaTeX or HTML, R package version 1.8-4, http://xtable.r-forge.r-project.org/.
Dahl, O.-J., Dijkstra, E.W., Hoare, C.A.R. (Red.), 1972. Structured Programming. Academic Press, London.
Dickey, D.A., Fuller, W.A., 1981. Likelihood Ratio Statistics for Autoregressive Time Series with a Unit Root. Econometrica 49(4), 1057–1072.
Dickey, D.A., Fuller, W.A., 1979. Distribution of the Estimators for Autoregressive Time Series with a Unit Root. Journal of the American Statistical Association 74(366a), 427–431. https://doi.org/10.1080/01621459.1979.10482531.
Diebold, F.X., Li, C., 2006. Forecasting the Term Structure of Government Bond Yields. Journal of Econometrics 130(2), 337–364. https://doi.org/10.1016/j.jeconom.2005.03.005.

Doman, M., 2004. Prognozowanie zmienności polskich indeksów giełdowych za pomocą modeli GARCH przy użyciu danych wysokiej częstotliwości. Acta Universitatis Lodziensis. Folia Oeconomica 177, 291–309.

Doman, M., Doman, R., 2004. Ekonometryczne modelowanie dynamiki polskiego rynku finansowego, Prace Habilitacyjne. Akademia Ekonomiczna w Poznaniu, Poznań.

Domański, C., Wagner, W., 1992. Uwagi o mocy testu wielowymiarowej normalności Shapiro–Wilka. Acta Universitatis Lodziensis. Folia Oeconomica 117, 33–45.

Doornik, J.A., Hansen, H., 2008. An Omnibus Test for Univariate and Multivariate Normality. Oxford Bulletin of Economics and Statistics 70(s1), 927–939. https://doi.org/10.1111/j.1468-0084.2008.00537.x.

Drucker, H., Burges, C.J.C., Kaufman, L., Smola, A., Vapnik, V., 1997. Support Vector Regression Machines, w: Mozer, M.C., Jordan, M.I., Petsche, T. (red.), Advances in Neural Information Processing Systems 9, NIPS. MIT Press, Cambridge, MA, s. 155–161.

Durbin, J., Watson, G.S., 1950. Testing for Serial Correlation in Least Squares Regression: I. Biometrika 37(3-4), 409–428. https://doi.org/10.1093/biomet/37.3-4.409.

Embrechts, P., Klüppelberg, C., Mikosch, T., 1997. Modelling Extremal Events for Insurance and Finance, Stochastic Modelling and Applied Probability. Springer, Berlin.

Engle, R.F., Ng, V.K., 1993. Measuring and Testing the Impact of News on Volatility. Journal of Finance 48(5), 1749–1778. https://doi.org/10.1111/j.1540-6261.1993.tb05127.x.

Fair, R.C., Shiller, R.J., 1990. Comparing Information in Forecasts from Econometric Models. The American Economic Review 80(3), 375–389.

Fałdziński, M., Osińska, M., Zdanowicz, T., 2012. Detecting Risk Transfer in Financial Markets Using Different Risk Measures. Central European Journal of Economic Modelling and Econometrics 4(1), 45–64.

Fellows, I., 2018. wordcloud: Word Clouds, R package version 2.6, http://blog.fellstat.com/?cat=11.

Fischer, T., Krauss, C., 2018. Deep Learning with Long Short-Term Memory Networks for Financial Market Predictions. European Journal of Operational Research 270(2), 654–669. https://doi.org/10.1016/j.ejor.2017.11.054.

Fisher, R.A., 1936. The Use of Multiple Measurements in Taxonomic Problems. Annals of Eugenics 7(2), 179–188. https://doi.org/10.1111/j.1469-1809.1936.tb02137.x.

Fiszeder, P., 2009. Modele klasy GARCH w empirycznych badaniach finansowych. Wydawnictwo Naukowe Uniwersytetu Mikołaja Kopernika, Toruń.

Fiszeder, P., 2003. Zastosowanie modeli GARCH w analizie procesów obserwowanych na GPW w Warszawie oraz wybranych rynkach akcji na świecie. Metody ilościowe w naukach ekonomicznych, Trzecie Warsztaty Doktorskie z Zakresu Ekonometrii i Statystyki 11–33.

Fiszeder, P., Bruzda, J., 2001. Badanie zależności pomiędzy indeksami giełdowymi na GPW w Warszawie – analiza wielorównaniowych modeli GARCH. Prace Naukowe Akademii Ekonomicznej we Wrocławiu 890(1), 110–122.

Fiszeder, P., Kwiatkowski, J., 2005. Model GARCH-M ze zmiennym parametrem – analiza wybranych spółek i indeksów notowanych na GPW. Przegląd Statystyczny 52(3), 73–88.
Fiszeder, P., Rowiński, S., 2012. Modelowanie zależności pomiędzy wybranymi procesami makroekonomicznymi a warszawskim indeksem giełdowym. Ekonomia i Prawo 10(3), 153–167.
Forbes, K., Kirkham, L., Theodoridis, K., 2021. A Trendy Approach to UK Inflation Dynamics. Manchester School 89(S1), 23–75. https://doi.org/10.1111/manc.12293.
Franta, M., Horvath, R., Rusnak, M., 2014. Evaluating Changes in the Monetary Transmission Mechanism in the Czech Republic. Empirical Economics 46(3), 827–842. https://doi.org/10.1007/s00181-013-0699-0.
Freitas, W., 2024. bizdays: Business Days Calculations and Utilities, R package version 1.0.16, https://github.com/wilsonfreitas/R-bizdays.
Fuller, W.A., 1976. Introduction to Statistical Time Series. John Wiley & Sons, New York, NY.
Gagolewski, M., 2022. stringi: Fast and portable character string processing in R. Journal of Statistical Software 103(2), 1–59. https://doi.org/10.18637/jss.v103.i02.
Gajdka, J., Pietraszewski, P., 2019. Fundamental Determinants of Polish Stock Market Returns: P/E and Price-to-Book Perspective. Economic Computation and Economic Cybernetics Studies and Research.
Gajdka, J., Schabek, T., 2018. Performance of Textile and Apparel Companies Listed on the Warsaw Stock Exchange. Central European Financial Studies. https://doi.org/10.5604/01.3001.0012.5161.
Gambacorta, L., Iannotti, S., 2007. Are There Asymmetries in the Response of Bank Interest Rates to Monetary Shocks? Applied Economics 39(19), 2503–2517. https://doi.org/10.1080/00036840600707241.
Gatnar, E., Walesiak, M. (Red.), 2009. Statystyczna analiza danych z wykorzystaniem programu R. PWN, Warszawa.
Gągolewski, M., 2016. Programowanie w języku R: analiza danych, obliczenia, symulacje. PWN, Warszawa.
Genz, A., Bretz, F., Miwa, T., Mi, X., Hothorn, T., 2025. mvtnorm: Multivariate Normal and t Distributions, R package version 1.3-3, http://mvtnorm.R-forge.R-project.org.
Gerrard, P., Johnson, R.M., 2015. Mastering Scientific Computing with R. Packt Publishing, Birmingham.
Gibson, R., Lhabitant, F.-S., Talay, D., 2010. Modeling the Term Structure of Interest Rates: A Review of the Literature. Foundations and Trends in Finance 5(1-2), 1–156. https://doi.org/10.1561/0500000032.
Gillespie, C., Lovelace, R., 2017. Efficient R Programming: A Practical Guide to Smarter Programming. O'Reilly Media, Sebastopol, CA.
Godfrey, L.G., 1978. Testing Against General Autoregressive and Moving Average Error Models When the Regressors Include Lagged Dependent Variables. Econometrica 46(6), 1293–1301. https://doi.org/10.2307/1913829.

Goldfeld, S.M., Quandt, R.E., 1965. Some Tests for Homoscedasticity. Journal of the American Statistical Association 60(310), 539–547. https://doi.org/10.1080/01621459.1965.10480811.

Gordon, R.J., 1997. The Time-Varying NAIRU and Its Implications for Economic Policy. Journal of Economic Perspectives 11(1), 11–32. https://doi.org/10.1257/jep.11.1.11.

Gostkowska-Drzewicka, M., Majerowska, E., 2022. Capital Structure Formation in Stock Exchange Listed Companies of the Visegrad Group: A Dynamic Approach. Zeszyty Naukowe Politechniki Śląskiej. Organizacja i Zarządzanie (166), 255–273. https://doi.org/10.29119/1641-3466.2022.166.17.

Gostkowska-Drzewicka, M., Majerowska, E., 2018. Przynależność sektorowa a wyniki spółek notowanych na GPW w Warszawie. Prace Naukowe Uniwersytetu Ekonomicznego we Wrocławiu (531), 139–148.

Gostkowska-Drzewicka, M., Majerowska, E., 2017. Sektor finansowy a koniunktura – analiza ryzyka inwestycyjnego. Zarządzanie i Finanse 15, 153–169.

Gradzewicz, M., Jabłonowski, J., Sasiela, M., Żółkiewski, Z., 2024. The Impact of Energy Price Increases on the Polish Economy. Energy Economics 140. https://doi.org/10.1016/j.eneco.2024.107944.

Granjon, D., van Leemput, V., Perrier, V., Rudolf, I., 2024. shinyMobile: Mobile Ready shiny Apps with Standalone Capabilities, R package version 2.0.0, https://github.com/RinteRface/shinyMobile.

Greene, W.H., 2003. Econometric Analysis, 5th ed. Prentice Hall, Upper Saddle River, New Jersey.

Gregor, J., Melecký, A., Melecký, M., 2021. Interest Rate Pass-Through: A Meta-Analysis of the Literature. Journal of Economic Surveys 35(1), 141–191. https://doi.org/10.1111/joes.12393.

Gries, S.T., 2009. Statistics for Linguistics with R: A Practical Introduction, Trends in Linguistics: Studies and Monographs. Mouton de Gruyter, Berlin. https://doi.org/10.1515/9783110307474.

Grolemund, G., 2014. Hands-On Programming with R: Write Your Own Functions and Simulations. O'Reilly Media, Sebastopol, CA.

Hair, Jr., Joseph F., Black, W.C., Babin, B.J., Anderson, R.E., 2010. Multivariate Data Analysis, 7th ed. Pearson, Upper Saddle River, NJ.

Hałka, A., Leszczyńska-Paczesna, A., 2023. Inflation Measurement in Times of Large Consumption Shifts – Evidence of the CPI Bias from Poland. Acta Oeconomica 73(3), 383–401. https://doi.org/10.1556/032.2023.00022.

Hamilton, J.D., 1994. Time Series Analysis. Princeton University Press, Princeton, NJ.

Harrison, J., 2022. RSelenium: R Bindings for Selenium WebDriver, R package version 1.7.9, https://docs.ropensci.org/RSelenium/.

Harrison, J., 2016. seleniumPipes: R Client Implementing the W3C WebDriver Specification, R package version 0.3.7, https://github.com/johndharrison/seleniumPipes.

Harvey, A.C., 1990. The Econometric Analysis of Time Series. MIT Press, Cambridge, Massachusetts.

Harvey, A.C., Collier, P., 1977. Testing for Functional Misspecification in Regression Analysis. Journal of Econometrics 6(1), 103–119. https://doi.org/10.1016/0304-4076(77)90057-4.
Hastings, C., Mosteller, F., Tukey, J.W., Winsor, C.P., 1947. Low Moments for Small Samples: A Comparative Study of Order Statistics. The Annals of Mathematical Statistics 18(3), 413–426. https://doi.org/10.1214/aoms/1177730388.
Hertel, K., Leszczyńska, A., 2013. Uporczywość inflacji i jej komponentów – badanie empiryczne dla Polski. Przegląd Statystyczny 60(2), 187–210. https://doi.org/10.59139/ps.2013.02.2.
Hlavac, M., 2022. stargazer: Well-Formatted Regression and Summary Statistics Tables, R package version 5.2.3, https://CRAN.R-project.org/package=stargazer.
Holt, C.C., 1957. Forecasting Trends and Seasonal by Exponentially Weighted Averages (Research Memorandum No. 52). Carnegie Institute of Technology.
Hong, H., Liu, A., 2011. Causality in Risk. Journal of Financial Econometrics 9(3), 535–575. https://doi.org/10.1093/jjfinec/nsq019.
Hothorn, T., Zeileis, A., Farebrother, R.W., Cummins, C., 2022. lmtest: Testing Linear Regression Models, R package version 0.9-40, https://CRAN.R-project.org/package=lmtest.
International Accounting Standards Board (Red.), 2023. IFRS Accounting Standards – Issued 2023, Part A: Required Standards, Includes IAS, IFRIC and SIC Standards, plus Conceptual Framework as at 1 January 2023. IFRS Foundation, London.
Jacobson, D., Brail, G., Woods, D., 2012. APIs: A Strategy Guide. O'Reilly Media, Sebastopol, CA.
Janecki, J., 2024. Oczekiwania inflacyjne w Polsce – stan wiedzy i propozycje jej poszerzenia. Zeszyty Naukowe Kolegium Zarządzania i Finansów, Szkoła Główna Handlowa w Warszawie 198, 9–24.
Jarque, C.M., Bera, A.K., 1987. A Test for Normality of Observations and Regression Residuals. International Statistical Review 55(2), 163–172. https://doi.org/10.2307/1403192.
Jarque, C.M., Bera, A.K., 1980. Efficient Tests for Normality, Homoscedasticity and Serial Independence of Regression Residuals. Economics Letters 6(3), 255–259. https://doi.org/10.1016/0165-1765(80)90024-5.
Kaas, R., Goovaerts, M., Dhaene, J., Denuit, M., 2008. Modern Actuarial Risk Theory: Using R. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-70998-5.
Kamiński, B., Zawisza, M., 2012. Receptury w R: podręcznik dla ekonomistów. Oficyna Wydawnicza SGH, Warszawa.
Kelm, R., 2010. The Exchange Rate and Two Price Inflations in Poland in the Period 1999–2009: Do Globalization and Balassa–Samuelson Effect Matter? Central European Journal of Economic Modelling and Econometrics 2(4), 315–349. https://doi.org/10.24425/cejeme.2010.119333.
Kelm, R., Majsterek, M., 2005. An I(2) Analysis of Inflationary Processes in Poland. Acta Universitatis Lodziensis. Folia Oeconomica 190, 55–73.
Kelm, R., Sobiech Pellegrini, I., 2023. Imported Inflation and the Wage-Price Nexus in Poland. Gospodarka Narodowa 315(3), 48–70. https://doi.org/10.33119/GN/169436.

Kianifard, F., Swallow, W.H., 1996. A Review of the Development and Application of Recursive Residuals in Linear Models. Journal of the American Statistical Association 91(433), 391–400. https://doi.org/10.2307/2291419.
Kleiber, C., Zeileis, A., 2008. Applied Econometrics with R. Springer, New York, NY. https://doi.org/10.1007/978-0-387-77318-6.
Kliber, A., Kliber, P., Płuciennik, P., Piwnicka, M., 2016. POLONIA Dynamics During the Years 2006–2012 and the Effectiveness of the Monetary Policy of the National Bank of Poland. Empirica 43(1), 37–59. https://doi.org/10.1007/s10663-015-9287-1.
Koenker, R., 1981. A Note on Studentizing a Test for Heteroscedasticity. Journal of Econometrics 17(1), 107–112. https://doi.org/10.1016/0304-4076(81)90062-2.
Kolasa, M., Rubaszek, M., Skrzypczyński, P., 2012. Putting the New Keynesian DSGE Model to the Real-Time Forecasting Test. Journal of Money, Credit and Banking 44(7), 1301–1324. https://doi.org/10.1111/j.1538-4616.2012.00533.x.
Kopczewska, K., Kopczewski, T., Wójcik, P., 2016. Metody ilościowe w R: aplikacje ekonomiczne i finansowe, 2nd ed. CeDeWu, Warszawa.
Kostyra, T.P., Rubaszek, M., 2020. Forecasting the Yield Curve for Poland. Econometric Research in Finance 5(2), 103–117. https://doi.org/10.2478/erfin-2020-0006.
Kotłowski, J., 2008. Forecasting Inflation with Dynamic Factor Model – The Case of Poland (Working Paper No. 24), Published 24 February 2008. Retrieved via SGH Repository (handle 20.500.12182/1359), http://kolegia.sgh.waw.pl/pl/KAE/struktura/IE/struktura/ZES/Documents/Working_Papers/aewp02-08.pdf. Department of Applied Econometrics, Warsaw School of Economics.
Kwiatkowski, D., Phillips, P.C.B., Schmidt, P., Shin, Y., 1992. Testing the Null Hypothesis of Stationarity Against the Alternative of a Unit Root. Journal of Econometrics 54(1–3), 159–178. https://doi.org/10.1016/0304-4076(92)90104-Y.
Kwiatkowski, J., 2019. Modele ze zmiennymi ukrytymi w analizie inflacji w Polsce. Wydawnictwo Naukowe Uniwersytetu Mikołaja Kopernika, Toruń.
Kwiatkowski, Ł., 2010. Markov Switching In-Mean Effect. Bayesian Analysis in Stochastic Volatility Framework. Central European Journal of Economic Modelling and Econometrics 2(1), 59–94. https://doi.org/10.24425/cejeme.2010.119320.
Lahti, L., Huovari, J., Kainu, M., Biecek, P., 2023. eurostat: Tools for Eurostat Open Data, R package version 4.0.0, https://ropengov.github.io/eurostat/.
Le Pennec, E., Slowikowski, K., 2024. ggwordcloud: A Word Cloud Geom for ggplot2, R package version 0.6.2, https://github.com/lepennec/ggwordcloud.
Leroy, A., Lucotte, Y., 2016. Structural and Cyclical Determinants of Bank Interest-Rate Pass-Through in the Eurozone. Comparative Economic Studies 58(2), 196–225. https://doi.org/10.1057/ces.2016.6.
Leszczyńska-Paczesna, A., 2019. Inflacja bazowa w polityce pieniężnej – analiza w świetle modelu DSGE (Rozprawa doktorska). Uniwersytet Łódzki, Wydział Ekonomiczno-Socjologiczny.
Lewis, P.D., 2010. R for Medicine and Biology. Jones & Bartlett Publishers, Sudbury, Massachusetts.

Li, H., Majerowska, E., 2008. Testing Stock Market Linkages for Poland and Hungary: A Multivariate GARCH Approach. Research in International Business and Finance 22(3), 247–266. https://doi.org/10.1016/j.ribaf.2007.04.001.
Lim, G.C., 2001. Bank Interest Rate Adjustments: Are They Asymmetric? Economic Record 77(237), 135–147. https://doi.org/10.1111/1475-4932.00009.
Lintner, J., 1965. The Valuation of Risk Assets and the Selection of Risky Investments in Stock Portfolios and Capital Budgets. The Review of Economics and Statistics 47(1), 13–37. https://doi.org/10.2307/1924119.
Ljung, G.M., Box, G.E.P., 1978. On a Measure of Lack of Fit in Time Series Models. Biometrika 65(2), 297–303. https://doi.org/10.1093/biomet/65.2.297.
Lütkepohl, H., Krätzig, M. (Red.), 2004. Applied Time Series Econometrics. Cambridge University Press, Cambridge. https://doi.org/10.1017/CBO9780511606885.
Łyziak, T., 2019. Do Global Output Gaps Help Forecast Domestic Inflation? Evidence from Phillips Curves for Poland. International Journal of Forecasting 35(3), 1032–1041. https://doi.org/10.1016/j.ijforecast.2019.03.006.
Łyziak, T., 2016a. Do Inflation Expectations Matter in a Stylised New Keynesian Model? The Case of Poland (NBP Working Paper No. 234), https://static.nbp.pl/publikacje/materialy-i-studia/234_en.pdf. Narodowy Bank Polski.
Łyziak, T., 2016b. Survey Measures of Inflation Expectations in Poland: Are They Relevant from the Macroeconomic Perspective? Baltic Journal of Economics 16(1), 33–52. https://doi.org/10.1080/1406099X.2016.1165402.
Maddala, G.S., 2006. Ekonometria. PWN, Warszawa.
Mailund, T., 2017. Advanced Object-Oriented Programming in R: Statistical Programming for Data Science, Analysis and Finance. Apress, New York, NY.
Majerowska, E., Spigarska, E., 2021. Influence of Selected Internal Factors on the Outputs of the Financial-Sector Companies Traded on the Warsaw Stock Exchange, w: Bilgin, M.H., Danis, H., Demir, E., Vale, S. (red.), Eurasian Economic Perspectives, Eurasian Studies w Business i Economics. Springer International Publishing, Cham, s. 193–204. https://doi.org/10.1007/978-3-030-63149-9_14.
Makhabel, B., 2015. Learning Data Mining with R. Packt Publishing, Birmingham.
Marcinkowska, M., Wdowiński, P., Flejterski, S., Bukowski, S., Zygierewicz, M., 2014. Wpływ regulacji sektora bankowego na wzrost gospodarczy – wnioski dla Polski. Materiały i Studia 305, 3–227.
Marin, J.-M., Robert, C.P., 2014. Bayesian Essentials with R. Springer, New York, NY. https://doi.org/10.1007/978-1-4614-8687-9.
Meschiari, S., 2022. latex2exp: Use LaTeX Expressions in Plots, R package version 0.9.6, https://www.stefanom.io/latex2exp/.
Milo, W. (Red.), 2002. Prognozowanie i symulacja. Wydawnictwo Uniwersytetu Łódzkiego, Łódź.
Milo, W., 1990. Szeregi czasowe. PWE, Warszawa.
Milo, W., Cieśluk, U., Zglińska-Pietrzak, A., 2001. Supply and Price Effects of Monetary and Fiscal Policy in Poland under Transition: An Econometric Analysis, w: Papazoglou, C., Pentecost, E.J. (red.), Exchange Rate Policies, Prices and Supply-Side

Response. Palgrave Macmillan, London. https://doi.org/10.1057/9780230554535_11.

Miłobędzki, P., 2020. Price Clustering in Stocks from the WIG 20 Index, w: Jajuga, K., Locarek-Junge, H., Orłowski, L.T., Staehr, K. (red.), Contemporary Trends and Challenges in Finance, Springer Proceedings w Business i Economics. Springer International Publishing, Cham, Switzerland, s. 169–177. https://doi.org/10.1007/978-3-030-43078-8_14.

Miłobędzki, P., 2010. The Term Structure of the Polish Interbank Rates. A Note on the Symmetry of their Reversion to the Mean. Dynamic Econometric Models 10, 83–95. https://doi.org/10.12775/DEM.2010.007.

Miłobędzki, P., 2009. Stock Price and Volume Relation at the Warsaw Stock Exchange, w: Milo, W., Szafrański, G., Wdowiński, P. (red.), Financial Markets: Principles of Modelling Forecasting and Decision-Making. University of Łódź Press, Łódź, s. 11–21.

Miłobędzki, P., 2008. O regresyjnych sposobach weryfikacji hipotezy oczekiwań struktury terminowej stóp procentowych na rynku depozytów międzybankowych w Polsce. Prace i Materiały Wydziału Zarządzania Uniwersytetu Gdańskiego (4), 67–84.

Miłobędzki, P., 2006. Asymmetry in the Adjustment of Main Capital Market Indices in Poland, w: Milo, W., Wdowiński, P. (red.), Financial Markets: Principles of Modelling Forecasting and Decision-Making. Wydawnictwo Uniwersytetu Łódzkiego, Łódź, s. 221–239.

Miłobędzki, P., Nowak, S., 2022. The Components of Bid-Ask Spread on the Warsaw Stock Exchange, w: Nguyen, D.K. (red.), Handbook of Banking and Finance in Emerging Markets. Edward Elgar Publishing, Cheltenham, UK, s. 131–151.

Montgomery, D.C., Jennings, C.L., Kulahci, M., 2015. Introduction to Time Series Analysis and Forecasting, 2nd ed. Wiley, Hoboken, NJ.

Müller, K., Wickham, H., 2023. tibble: Simple Data Frames, R package version 3.2.1, https://tibble.tidyverse.org/.

Munzert, S., Rubba, C., Meißner, P., Nyhuis, D., 2015. Automated Data Collection with R: A Practical Guide to Web Scraping and Text Mining. Wiley, Chichester.

Neuwirth, E., 2022. RColorBrewer: ColorBrewer Palettes, R package version 1.1-3, https://CRAN.R-project.org/package=RColorBrewer.

Nocedal, J., Wright, S.J., 2006. Numerical Optimization, 2nd ed. Springer, New York, NY. https://doi.org/10.1007/978-0-387-40065-5.

Ooms, J., 2025a. jsonlite: A Simple and Robust JSON Parser and Generator for R, R package version 2.0.0, https://jeroen.r-universe.dev/jsonlite.

Ooms, J., 2025b. writexl: Export Data Frames to Excel xlsx Format, R package version 1.5.4, https://ropensci.r-universe.dev/writexl.

OpenAI, 2023. ChatGPT, Large language model (Mar 14 version), https://chat.openai.com/.

Orlowski, L.T., 2001. From Inflation Targeting to the Euro-Peg: A Model of Monetary Convergence for Transition Economies. Economic Systems 25(3), 233–251. https://doi.org/10.1016/S0939-3625(01)00020-6.

Osińska, M., 2006. Ekonometria finansowa. PWE, Warszawa.
Osińska, M., 1999. Wpływ oczekiwań na kształtowanie się indeksów giełdowych na przykładzie WIG. Prace i Materiały Instytutu Rozwoju Gospodarczego SGH (64), 85–89.
Osińska, M., Pietrzak, M.B., Żurek, M., 2011. Ocena wpływu czynników behawioralnych i rynkowych na postawy inwestorów indywidualnych na polskim rynku kapitałowym za pomocą modelu SEM. Przegląd Statystyczny 58(3-4), 175–194.
Osowska, E., Wójcik, P., 2024. Predicting the Reaction of Financial Markets to Federal Open Market Committee Post-Meeting Statements. Digital Finance 6(1), 145–175. https://doi.org/10.1007/s42521-023-00096-8.
Pace, L., 2012. Beginning R: An Introduction to Statistical Programming. Apress, New York, NY.
Papadamou, S., Sidiropoulos, M., Spyromitros, E., 2015. Central Bank Transparency and the Interest Rate Channel: Evidence from Emerging Economies. Economic Modelling 48, 167–174. https://doi.org/10.1016/j.econmod.2014.10.016.
Pearson, E.S., D'Agostino, R.B., Bowman, K.O., 1977. Tests for Departure from Normality: Comparison of Powers. Biometrika 64(2), 231–246. https://doi.org/10.1093/biomet/64.2.231.
Persson, E., 2025. ecb: Programmatic Access to the European Central Bank's Data Portal, R package version 0.4.3, https://github.com/expersso/ecb.
Pfaff, B., 2008. Analysis of Integrated and Cointegrated Time Series with R, 2nd ed, http://www.pfaffikus.de. Springer, New York, NY.
Plummer, M., Best, N., Cowles, K., Vines, K., Sarkar, D., Bates, D., Almond, R., Magnusson, A., 2024. coda: Output Analysis and Diagnostics for MCMC, R package version 0.19-4.1, https://CRAN.R-project.org/package=coda.
Przybyliński, M., Gorzałczyński, A., 2016. Zastosowanie tablic przepływów międzygałęziowych do modelowania procesów inflacyjnych. Gospodarka w Praktyce i Teorii 42(1), 49–60. https://doi.org/10.18778/1429-3730.42.04.
R Core Team, 2025. R: A Language and Environment for Statistical Computing, https://www.R-project.org/. R Foundation for Statistical Computing, Vienna, Austria.
Ramsey, J.B., 1969. Tests for Specification Errors in Classical Linear Least-Squares Regression Analysis. Journal of the Royal Statistical Society. Series B (Methodological) 31(2), 350–371. https://doi.org/10.1111/j.2517-6161.1969.tb00796.x.
Ren, K., 2021. rlist: A Toolbox for Non-Tabular Data Manipulation, R package version 0.4.6.2, https://renkun-ken.github.io/rlist/.
Robert, C.P., Casella, G., 2010. Introducing Monte Carlo Methods with R. Springer, New York, NY. https://doi.org/10.1007/978-1-4419-1576-4.
Robinson, D., Silge, J., 2024. tidytext: Text Mining using dplyr, ggplot2, and Other Tidy Tools, R package version 0.4.2, https://juliasilge.github.io/tidytext/.
Royston, P., 1995. Remark AS R94: A Remark on Algorithm AS 181: The W-test for Normality. Journal of the Royal Statistical Society: Series C (Applied Statistics) 44(4), 547–551. https://doi.org/10.2307/2986146.
Rubaszek, M., 2016. Forecasting the Yield Curve With Macroeconomic Variables. Econometric Research in Finance 1(1), 1–21.

Rubaszek, M., 2012. Modelowanie polskiej gospodarki z pakietem R. Oficyna Wydawnicza SGH, Warszawa.
Rubaszek, M., Skrzypczyński, P., 2008. On the Forecasting Performance of a Small-Scale DSGE Model. International Journal of Forecasting 24(3), 498–512. https://doi.org/10.1016/j.ijforecast.2008.05.002.
Ryan, J.A., Ulrich, J.M., 2024. xts: eXtensible Time Series, R package version 0.14.0, https://joshuaulrich.github.io/xts/.
Schloerke, B., Cook, D., Larmarange, J., Briatte, F., Marbach, M., Thoen, E., Elberg, A., Crowley, J., 2024. GGally: Extension to ggplot2, R package version 2.2.1, https://ggobi.github.io/ggally/.
Schumacker, R.E., Tomek, S.R., 2013. Understanding Statistics Using R. Springer, New York, NY. https://doi.org/10.1007/978-1-4614-6227-9.
Schwarz, G., 1978. Estimating the Dimension of a Model. The Annals of Statistics 6(2), 461–464. https://doi.org/10.1214/aos/1176344136.
Serwa, D., Wdowiński, P., 2017. Modeling Macro-Financial Linkages: Combined Impulse Response Functions in SVAR Models. Central European Journal of Economic Modelling and Econometrics 9(4), 323–357. https://doi.org/10.2139/ssrn.2845737.
Shapiro, S.S., Wilk, M.B., 1965. An Analysis of Variance Test for Normality (Complete Samples). Biometrika 52(3-4), 591–611. https://doi.org/10.1093/biomet/52.3-4.591.
Sharpe, W.F., 1964. Capital Asset Prices: A Theory of Market Equilibrium Under Conditions of Risk. The Journal of Finance 19(3), 425–442. https://doi.org/10.1111/j.1540-6261.1964.tb02865.x.
Sharpe, W.F., 1963. A Simplified Model for Portfolio Analysis. Management Science 9(2), 277–293. https://doi.org/10.1287/mnsc.9.2.277.
Sievert, C., 2020. Interactive Web-Based Data Visualization with R, plotly, and shiny. CRC Press, Boca Raton, FL.
Sievert, C., Parmer, C., Hocking, T., Chamberlain, S., Ram, K., Corvellec, M., Despouy, P., 2024. plotly: Create Interactive Web Graphics via plotly.js, R package version 4.10.4, https://plotly-r.com.
Signorell, A., 2024. DescTools: Tools for Descriptive Statistics, R package version 0.99.55, https://andrisignorell.github.io/DescTools/.
Smets, F., 1995. Central Bank Macroeconometric Models and the Monetary Policy Transmission Mechanism (BIS Conference Paper No. 394), Included in BIS Conference Volume No. 394: Financial Structure and the Monetary Policy Transmission Mechanism, March 1995, pp. 225–266, https://www.bis.org/publ/conf394.pdf. Bank for International Settlements, Basel.
Smets, F., Wouters, R., 2007. Shocks and Frictions in US Business Cycles: A Bayesian DSGE Approach. American Economic Review 97(3), 586–606. https://doi.org/10.1257/aer.97.3.586.
Spanos, A., 1986. Statistical Foundations of Econometric Modelling. Cambridge University Press, New York, NY.
Spector, P., 2008. Data Manipulation with R. Springer, New York. https://doi.org/10.1007/978-0-387-74731-6.

Stelmasiak, D., Szafrański, G., 2016. Forecasting the Polish Inflation Using Bayesian VAR Models with Seasonality. Central European Journal of Economic Modelling and Econometrics 8(1), 21–42. https://doi.org/10.24425/cejeme.2016.119185.
Stock, J.H., Watson, M.W., 2001. Vector Autoregressions. Journal of Economic Perspectives 15(4), 101–115. https://doi.org/10.1257/jep.15.4.101.
Strawiński, P., Ślepaczuk, R., 2008. Analysis of High-Frequency Data on the Warsaw Stock Exchange in the Context of Efficient Market Hypothesis. Journal of Applied Economic Sciences 3(5), 306–319.
Suess, E.A., Trumbo, B.E., 2010. Introduction to Probability Simulation and Gibbs Sampling with R. Springer, New York. https://doi.org/10.1007/978-0-387-68765-0.
Szafranek, K., Hałka, A., 2017. Determinants of Low Inflation in an Emerging, Small Open Economy: A Comparison of Aggregated and Disaggregated Approaches (NBP Working Paper No. 267), https://static.nbp.pl/publikacje/materialy-i-studia/267_en.pdf. Narodowy Bank Polski.
Szafranek, K., Szafrański, G., Leszczyńska-Paczesna, A., 2024. Inflation returns. Revisiting the role of external and domestic shocks with Bayesian structural VAR. International Review of Economics & Finance 93, 789–810. https://doi.org/10.1016/j.iref.2024.03.054.
Szafrański, G., 2013. Syntetyczne wskaźniki wyprzedzające w prognozowaniu inflacji w Polsce. Przegląd Statystyczny 60(3), 395–416. https://doi.org/10.59139/ps.2013.03.7.
Sznajderska, A., 2013. On the Empirical Evidence of Asymmetric Effects in the Polish Interest Rate Pass-Through. The Journal of Economic Asymmetries 10(2), 78–93. https://doi.org/10.1016/j.jeca.2013.11.001.
Taylor, J.B., 1995. The Monetary Transmission Mechanism: An Empirical Framework. Journal of Economic Perspectives 9(4), 11–26. https://doi.org/10.1257/jep.9.4.11.
Teetor, P., 2011. R Cookbook: Proven Recipes for Data Analysis, Statistics, and Graphics. O'Reilly Media, Sebastopol, CA.
Trapletti, A., Hornik, K., 2024. tseries: Time Series Analysis and Computational Finance, R package version 0.10-58, https://CRAN.R-project.org/package=tseries.
Tsay, R.S., 2014. Multivariate Time Series Analysis: With R and Financial Applications. Wiley, Hoboken, NJ.
Tsay, R.S., 2010. Analysis of Financial Time Series, 3rd ed, Wiley Series w Probability i Statistics. Wiley, Hoboken, NJ.
Turlach, B.A., 2019. quadprog: Functions to Solve Quadratic Programming Problems, R package version 1.5-8, https://CRAN.R-project.org/package=quadprog.
Ugarte, M.D., Militino, A.F., Arnholt, A.T., 2015. Probability and Statistics with R. Chapman & Hall/CRC, Boca Raton, FL.
Vasicek, O., 1977. An Equilibrium Characterization of the Term Structure. Journal of Financial Economics 5(2), 177–188. https://doi.org/10.1016/0304-405X(77)90016-2.
Verbeek, M., 2017. A Guide to Modern Econometrics, 5th ed. Wiley, Hoboken, NJ.
Verzani, J., 2014. Using R for Introductory Statistics, 2nd ed. Chapman & Hall/CRC, Boca Raton, FL.

Wdowiński, P., 2020. Wstęp do programowania i analizy danych w języku R. Wydawnictwo Uniwersytetu Łódzkiego, Łódź. https://doi.org/10.18778/8220-278-6.
Wdowiński, P., 2010. Modele kursów walutowych. Wydawnictwo Uniwersytetu Łódzkiego, Łódź. https://doi.org/10.18778/7525-466-2.
Wdowiński, P., 2004. Determinants of Country Beta Risk in Poland (No. 1120), CESifo Working Paper, https://www.cesifo.org/en/publikationen/2004/working-paper/determinants-country-beta-risk-poland. Center for Economic Studies; ifo Institute (CESifo), Munich.
Wdowiński, P., 2002. Wielorównaniowy model cen i płac przeciętnych: analiza symulacyjna, w: Milo, W. (red.), Prognozowanie i symulacja. Wydawnictwo Uniwersytetu Łódzkiego, Łódź.
Wdowiński, P., Małecka, M., 2010. Asymmetry in Volatility: A Comparison of Developed and Transition Stock Markets (No. 2974), CESifo Working Paper, https://hdl.handle.net/10419/39039. Center for Economic Studies; ifo Institute (CESifo), Munich.
Wdowiński, P., Wrzesiński, D., 2004. Czynniki ryzyka rynku kapitałowego w Polsce. Acta Universitatis Lodziensis. Folia Oeconomica (177), 23–41.
Wdowiński, P., Zglińska-Pietrzak, A., 2005. The Warsaw Stock Exchange Index WIG: Modelling and Forecasting, w: Issues in Modeling, Forecasting and Decision-Making in Financial Markets, Acta Universitatis Lodziensis. Folia Oeconomica. Wydawnictwo Uniwersytetu Łódzkiego, Łódź, s. 115–127.
Wdowiński, P., Zglińska-Pietrzak, A., Tomasik, J., 1997. Kilka uwag na temat interpretacji parametrów strukturalnych modelu ekonometrycznego, w: Milo, W. (red.), Badania ekonometryczne. Wydawnictwo Uniwersytetu Łódzkiego, Łódź, s. 125–135.
Welfe, A., 2000. Modeling Inflation in Poland. Economic Modelling 17(3), 375–385. https://doi.org/10.1016/S0264-9993(99)00044-9.
Welfe, A., Karp, P., Kębłowski, P., 2004. Modelling Polish Economy: An Application of SVEqCM, w: Handbook of Comparative Economic Policies. s. 557–602. https://doi.org/10.1016/S0573-8555(04)69008-0.
Welfe, A., Majsterek, M., 2002. Wage and Price Inflation in Poland in the Period of Transition: The Cointegration Analysis. Economics of Planning 35, 205–219. https://doi.org/10.1023/A:1022420518609.
Welfe, A., Majsterek, M., 2001. Długookresowy model sprzężenia inflacyjnego w gospodarce polskiej okresu transformacji. Ekonomista (5), 619–633.
Wesołowski, G., 2016. Do Long Term Interest Rates Drive GDP and Inflation in Small Open Economies? Evidence from Poland (NBP Working Paper No. 242), https://static.nbp.pl/publikacje/materialy-i-studia/242_en.pdf. Narodowy Bank Polski.
White, H., 1980. A Heteroskedasticity-Consistent Covariance Matrix Estimator and a Direct Test for Heteroskedasticity. Econometrica 48(4), 817–838. https://doi.org/10.2307/1912934.
Wickham, H., 2024. rvest: Easily Harvest (Scrape) Web Pages, R package version 1.0.4, https://rvest.tidyverse.org/.
Wickham, H., 2023a. forcats: Tools for Working with Categorical Variables (Factors), R package version 1.0.0, https://forcats.tidyverse.org/.

Wickham, H., 2023d. httr: Tools for Working with URLs and HTTP, R package version 1.4.7, https://httr.r-lib.org/.
Wickham, H., 2023c. stringr: Simple, Consistent Wrappers for Common String Operations, R package version 1.5.1, https://stringr.tidyverse.org.
Wickham, H., 2023b. tidyverse: Easily Install and Load the Tidyverse, R package version 2.0.0, https://tidyverse.tidyverse.org.
Wickham, H., 2020. reshape2: Flexibly Reshape Data: A Reboot of the Reshape Package, R package version 1.4.4, https://github.com/hadley/reshape.
Wickham, H., 2019. Advanced R, 2nd ed, Chapman & Hall/CRC The R Series. Chapman & Hall/CRC, Boca Raton, FL.
Wickham, H., Bryan, J., 2023. readxl: Read Excel Files, R package version 1.4.3, https://readxl.tidyverse.org.
Wickham, H., Çetinkaya-Rundel, M., Grolemund, G., 2023a. R for Data Science: Import, Tidy, Transform, Visualize, and Model Data, 2nd ed. O'Reilly Media, Sebastopol, California.
Wickham, H., Chang, W., Henry, L., Pedersen, T.L., Takahashi, K., Wilke, C., Woo, K., Yutani, H., Dunnington, D., van den Brand, T., 2024a. ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics, R package version 3.5.1, https://ggplot2.tidyverse.org.
Wickham, H., François, R., Henry, L., Müller, K., Vaughan, D., 2023b. dplyr: A Grammar of Data Manipulation, R package version 1.1.4, https://dplyr.tidyverse.org.
Wickham, H., Grolemund, G., 2016. R for Data Science: Visualize, Model, Transform, Tidy, and Import Data. O'Reilly Media, Sebastopol, CA.
Wickham, H., Henry, L., 2023. purrr: Functional Programming Tools, R package version 1.0.2, https://purrr.tidyverse.org/.
Wickham, H., Hester, J., Bryan, J., 2024b. readr: Read Rectangular Text Data, R package version 2.1.5, https://readr.tidyverse.org.
Wickham, H., Hester, J., Chang, W., Bryan, J., 2022. devtools: Tools to Make Developing R Packages Easier, R package version 2.4.5, https://devtools.r-lib.org/.
Wickham, H., Vaughan, D., Girlich, M., 2024c. tidyr: Tidy Messy Data, R package version 1.3.1, https://tidyr.tidyverse.org.
Wiley, G.M., Wiley, J.F., 2016. Advanced R: Data Programming and the Cloud. Apress, New York, NY.
Wirth, N., 1989. Algorytmy + struktury danych = programy. Wydawnictwa Naukowo–Techniczne, Warszawa.
Wooldridge, J.M., 2020. Introductory Econometrics: A Modern Approach, 7th ed. Cengage, Boston, MA.
Wójcik, S., 2015. Prognozowanie inflacji w Polsce na podstawie modeli autoregresji wektorowej. Wiadomości Statystyczne 60(1), 28–41. https://doi.org/10.59139/ws.2015.01.3.
Wu, L., 2024. Interest Rate Modeling: Theory and Practice, 3rd ed, Chapman & Hall/CRC Financial Mathematics Series. Chapman & Hall/CRC, Boca Raton, FL. https://doi.org/10.1201/9781003389101.

Xie, Y., 2025a. bookdown: Authoring Books and Technical Documents with R Markdown, R package version 0.45, https://github.com/rstudio/bookdown.
Xie, Y., 2025b. knitr: A General-Purpose Package for Dynamic Report Generation in R, R package version 1.50, https://yihui.org/knitr/.
Xie, Y., 2025c. tinytex: Helper Functions to Install and Maintain TeX Live, and Compile LaTeX Documents, R package version 0.58, https://github.com/rstudio/tinytex.
Xie, Y., 2016. Bookdown: Authoring Books and Technical Documents with R Markdown. Chapman & Hall/CRC, Boca Raton, FL.
Xie, Y., 2015. Dynamic Documents with R and knitr. Chapman & Hall/CRC, Boca Raton, Florida.
Xie, Y., Allaire, J.J., Grolemund, G., 2018. R Markdown: The Definitive Guide, https://bookdown.org/yihui/rmarkdown. Chapman & Hall/CRC, Boca Raton, FL.
Zeileis, A., Grothendieck, G., Ryan, J.A., 2023. zoo: S3 Infrastructure for Regular and Irregular Time Series (Z's Ordered Observations), R package version 1.8-12, https://zoo.R-Forge.R-project.org/.
Zhang, G.P., Patuwo, B.E., Hu, M.Y., 1998. Forecasting with Artificial Neural Networks: The State of the Art. International Journal of Forecasting 14(1), 35–62. https://doi.org/10.1016/S0169-2070(97)00044-7.
Zhu, H., 2025. kableExtra: Construct Complex Table with kable and Pipe Syntax, R package version 1.4.0.12, http://haozhu233.github.io/kableExtra/.
Ziarko-Siwek, U., Kamiński, M., 2003. Empiryczna weryfikacja teorii oczekiwań terminowej struktury stóp procentowych w Polsce (No. 159), Materiały i Studia. Narodowy Bank Polski, Warszawa.

Indeks

Piotr Wdowiński – doktor habilitowany nauk ekonomicznych, profesor Uniwersytetu Łódzkiego, w latach 2012–2016 dyrektor Instytutu Ekonometrii UŁ, w latach 2016–2024 kierownik Katedry Ekonometrii UŁ. Autor monografii i publikacji naukowych w recenzowanych czasopismach. Redaktor naczelny czasopisma „Econometric Research in Finance". W latach 2002–2016 organizator międzynarodowych konferencji naukowych pod nazwą „Forecasting Financial Markets and Economic Decision-Making (FindEcon)". Specjalizuje się w modelowaniu i prognozowaniu ekonometrycznym procesów makroekonomicznych i finansowych, a także w zagadnieniach stabilności finansowej, polityki makroostrożnościowej i ryzyka kredytowego.

Liniowe modele ekonometryczne. Zaawansowane zastosowania języka R to kompleksowe opracowanie, które łączy teorię ekonometrii z praktyką programowania w R. Autor w przystępny sposób wprowadza w zagadnienia procesów stochastycznych, liniowych modeli ekonometrycznych oraz metod estymacji i weryfikacji modeli, wzbogacając je licznymi przykładami kodu i analizami empirycznymi. Pokazuje, jak wykorzystać R do przetwarzania danych, automatyzacji raportów w formacie R/Markdown oraz tworzenia interaktywnych aplikacji w R/Shiny. Szczególną wartością publikacji jest integracja klasycznych metod ekonometrycznych z nowoczesnymi narzędziami wizualizacji i prognozowania danych, co czyni ją praktycznym przewodnikiem dla badaczy, analityków i studentów kierunków ekonomicznych oraz finansowych. To pozycja dla wszystkich, którzy chcą opanować zaawansowane metody analizy danych, modelowania i prognozowania ekonometrycznego w środowisku R, łącząc solidne podstawy teoretyczne z umiejętnościami praktycznymi.

WYDAWNICTWO
UNIWERSYTETU
ŁÓDZKIEGO

wydawnictwo.uni.lodz.pl
ksiegarnia@uni.lodz.pl
(42) 665 58 63

Książka dostępna również jako e-book

ISBN 978-83-8331-974-2