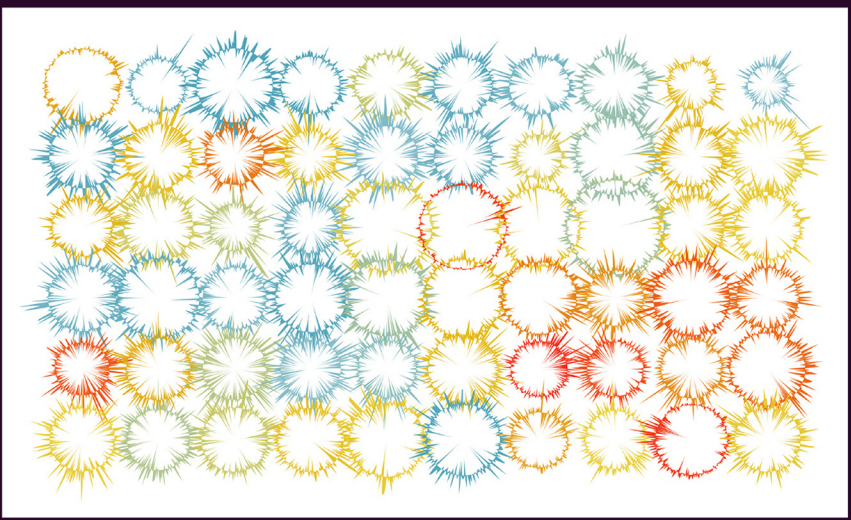
The R Series

Tidy Finance with R



Christoph Scheuch
Stefan Voigt
Patrick Weiss

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

Tidy Finance with R

This textbook lifts the curtain on reproducible finance and shows how to apply theoretical concepts from finance and econometrics by providing a fully transparent R code base. Focusing on coding and data analysis with R, we illustrate how students, researchers, data scientists, and professionals can conduct research in empirical finance from scratch. We start with a beginner-friendly introduction to the tidyverse family of R packages around which our approach revolves. We then show how to access and prepare common open-source (e.g., French data library, macroeconomic data) and proprietary financial data sources (e.g., CRSP, Compustat, Mergent FISD, and TRACE). We present data management principles using an SQLite database, which constitutes the basis for the applications presented in the subsequent chapters. The empirical applications range from key concepts of empirical asset pricing (e.g., beta estimation, portfolio sorts, performance analysis, and Fama-French factors) to modeling and machine learning applications (e.g., fixed effects estimation, clustering standard errors, difference-in-difference estimators, ridge regressions, Lasso, Elastic nets, random forests, and neural networks) and portfolio optimization techniques.

Highlights

1. Self-contained chapters on the most important applications and methodologies in finance, which can easily be used for the reader's research or as a reference for courses on empirical finance.
2. Each chapter is reproducible in the sense that the reader can replicate every single figure, table, or number by simply copy-pasting the code we provide.
3. A full-fledged introduction to machine learning with tidymodels based on tidy principles to show how factor selection and option pricing can benefit from Machine Learning methods.
4. A chapter that shows how to retrieve and prepare the most important datasets in the field of financial economics: CRSP and Compustat. The chapter also contains detailed explanations of the most relevant data characteristics.
5. Each chapter provides exercises that are based on established lectures and exercise classes and which are designed to help students to dig deeper. The exercises can be used for self-studying or as a source of inspiration for teaching exercises.

Chapman & Hall/CRC
The R Series

Recently Published Titles

Javascript for R
John Coene

Advanced R Solutions
Malte Grosser, Henning Bumann, and Hadley Wickham

Event History Analysis with R, Second Edition
Göran Broström

Behavior Analysis with Machine Learning Using R
Enrique Garcia Ceja

Rasch Measurement Theory Analysis in R: Illustrations and Practical Guidance for Researchers and Practitioners
Stefanie Wind and Cheng Hua

Spatial Sampling with R
Dick R. Brus

Crime by the Numbers: A Criminologist's Guide to R
Jacob Kaplan

Analyzing US Census Data: Methods, Maps, and Models in R
Kyle Walker

ANOVA and Mixed Models: A Short Introduction Using R
Lukas Meier

Tidy Finance with R
Christoph Scheuch, Stefan Voigt and Patrick Weiss

Deep Learning and Scientific Computing with R torch
Sigrid Keydana

Model-Based Clustering, Classification, and Density Estimation Using mclust in R
Lucca Scrucca, Chris Fraley, T. Brendan Murphy, and Adrian E. Raftery

Spatial Data Science: With Applications in R
Edzer Pebesma and Roger Bivand

For more information about this series, please visit: https://www.crcpress.com/Chapman--HallCRC-The-R-Series/book-series/CRCTHERSER

Tidy Finance with R

Christoph Scheuch
Stefan Voigt
Patrick Weiss

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
A CHAPMAN & HALL BOOK

Designed cover image: Christoph Scheuch, Stefan Voigt and Patrick Weiss

First edition published 2023
by CRC Press
6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487-2742

and by CRC Press
4 Park Square, Milton Park, Abingdon, Oxon, OX14 4RN

CRC Press is an imprint of Taylor & Francis Group, LLC

ISBN: 978-1-032-38933-2 (hbk)
ISBN: 978-1-032-38934-9 (pbk)
ISBN: 978-1-003-34753-8 (ebk)

DOI: 10.1201/b23237

Typeset in CMR10
by KnowledgeWorks Global Ltd.

Publisher's note: This book has been prepared from camera-ready copy provided by the authors.

Access the Support Material: github.com/voigtstefan/tidy_finance

Contents

Preface

Why Does This Book Exist?

Financial economics is a vibrant area of research, a central part of all business activities, and at least implicitly relevant to our everyday life. Despite its relevance for our society and a vast number of empirical studies of financial phenomena, one quickly learns that the actual implementation of models to solve problems in the area of financial economics is typically rather opaque. As graduate students, we were particularly surprised by the lack of public code for seminal papers or even textbooks on key concepts of financial economics. The lack of transparent code not only leads to numerous replication efforts (and their failures) but also constitutes a waste of resources on problems that countless others have already solved in secrecy.

This book aims to lift the curtain on reproducible finance by providing a fully transparent code base for many common financial applications. We hope to inspire others to share their code publicly and take part in our journey toward more reproducible research in the future.

Who Should Read This Book?

We write this book for three audiences:

- Students who want to acquire the basic tools required to conduct financial research ranging from undergrad to graduate level. The book's structure is simple enough such that the material is sufficient for self-study purposes.
- Instructors who look for materials to teach courses in empirical finance or financial economics. We provide plenty of examples and focus on intuitive explanations that can easily be adjusted or expanded. At the end of each chapter, we provide exercises that we hope inspire students to dig deeper.
- Data analysts or statisticians who work on issues dealing with financial data and who need practical tools to succeed.

What Will You Learn?

The book is currently divided into five parts:

- Chapter 1 introduces you to important concepts around which our approach to Tidy Finance revolves.
- Chapters 2–4 provide tools to organize your data and prepare the most common data sets used in financial research. Although many important data are behind paywalls, we start by describing different open-source data and how to download them. We then move on to prepare two of the most popular datasets in financial research: CRSP and Compustat. Then, we cover corporate bond data from TRACE. We reuse the data from these chapters in all subsequent chapters. Chapter 5 contains an overview of common alternative data providers for which direct access vie R packages exist.
- Chapters 6–11 deal with key concepts of empirical asset pricing, such as beta estimation, portfolio sorts, performance analysis, and asset pricing regressions.
- Chapters 12–15 apply linear models to panel data and machine learning methods to problems in factor selection and option pricing.
- Chapters 16–17 provide approaches for parametric, constrained portfolio optimization, and backtesting procedures.
  Each chapter is self-contained and can be read individually. Yet the data chapters provide an important background necessary for data management in all other chapters.

What Won't You Learn?

This book is about empirical work. While we assume only basic knowledge of statistics and econometrics, we do not provide detailed treatments of the underlying theoretical models or methods applied in this book. Instead, you find references to the seminal academic work in journal articles or textbooks for more detailed treatments. We believe that our comparative advantage is to provide a thorough implementation of typical approaches such as portfolio sorts, backtesting procedures, regressions, machine learning methods, or other related topics in empirical finance. We enrich our implementations by discussing the needy-greedy choices you face while conducting empirical analyses. We hence refrain from deriving theoretical models or extensively discussing the statistical properties of well-established tools.

Our book is close in spirit to other books that provide fully reproducible code for financial applications. We view them as complementary to our work and want to highlight the differences:

- Regenstein Jr (2018) provides an excellent introduction and discussion of different tools for standard applications in finance (e.g., how to compute returns and sample standard deviations of a time series of stock returns). In contrast, our book clearly focuses on applications of the state-of-the-art for academic research in finance. We thus fill a niche that allows aspiring researchers or instructors to rely on a well-designed code base.
- Coqueret and Guida (2020) constitute a great compendium to our book with respect to applications related to return prediction and portfolio formation. The book primarily targets practitioners and has a hands-on focus. Our book, in contrast, relies on the typical databases used in financial research and focuses on the preparation of such datasets for academic applications. In addition, our chapter on machine learning focuses on factor selection instead of return prediction.

Although we emphasize the importance of reproducible workflow principles, we do not provide introductions to some of the core tools that we relied on to create and maintain this book:

- Version control systems such as Git[1] are vital in managing any programming project. Originally designed to organize the collaboration of software developers, even solo data analysts will benefit from adopting version control. Git also makes it simple to publicly share code and allow others to reproduce your findings. We refer to Bryan (2022) for a gentle introduction to the (sometimes painful) life with Git.
- Good communication of results is a key ingredient to reproducible and transparent research. To compile this book, we heavily draw on a suite of fantastic open-source tools. First, Wickham (2016) provides a highly customizable yet easy-to-use system for creating data visualizations. Wickham and Grolemund (2016) provide an intuitive introduction to creating graphics using this approach. Second, in our daily work and to compile this book, we used the markdown-based authoring framework described in Xie et al. (2018) and Xie et al. (2020). Markdown documents are fully reproducible and support dozens of static and dynamic output formats. Lastly, Xie (2016) tremendously facilitates authoring markdown-based books. We do not provide introductions to these tools, as the resources above already provide easily accessible tutorials.

- Good writing is also important for the presentation of findings. We neither claim to be experts in this domain nor do we try to sound particularly academic. On the contrary, we deliberately use a more colloquial language to describe all the methods and results presented in this book in order to allow our readers to relate more easily to the rather technical content. For those who desire more guidance with respect to formal academic writing for financial economics, we recommend Kiesling (2003), Cochrane (2005), and Jacobsen (2014), who all provide essential tips (condensed to a few pages).

[1] https://git-scm.com/

Why R?

We believe that R (R Core Team, 2022) is among the best choices for a programming language in the area of finance. Some of our favorite features include:

- R is free and open-source, so that you can use it in academic and professional contexts.
- A diverse and active online community works on a broad range of tools.
- A massive set of actively maintained packages for all kinds of applications exists, e.g., data manipulation, visualization, machine learning, etc.
- Powerful tools for communication, e.g., Rmarkdown and shiny, are readily available.
- RStudio is one of the best development environments for interactive data analysis.
- Strong foundations of functional programming are provided.
- Smooth integration with other programming languages, e.g., SQL, Python, C, C++, Fortran, etc.

For more information on why R is great, we refer to Wickham et al. (2019).

Why Tidy?

As you start working with data, you quickly realize that you spend a lot of time reading, cleaning, and transforming your data. In fact, it is often said that more than 80% of data analysis is spent on preparing data. By *tidying data*, we want to structure data sets to facilitate further analyses. As Wickham (2014) puts it:

> [T]idy datasets are all alike, but every messy dataset is messy in its own way. Tidy datasets provide a standardized way to link the structure of a dataset (its physical layout) with its semantics (its meaning).

In its essence, tidy data follows these three principles:

1. Every column is a variable.

2. Every row is an observation.
3. Every cell is a single value.

Throughout this book, we try to follow these principles as best as we can. If you want to learn more about tidy data principles in an informal manner, we refer you to this vignette[2] as part of Wickham and Girlich (2022).

In addition to the data layer, there are also tidy coding principles outlined in the tidy tools manifesto[3] that we try to follow:

1. Reuse existing data structures.
2. Compose simple functions with the pipe.
3. Embrace functional programming.
4. Design for humans.

In particular, we heavily draw on a set of packages called the `tidyverse`[4] (Wickham et al., 2019). The `tidyverse` is a consistent set of packages for all data analysis tasks, ranging from importing and wrangling to visualizing and modeling data with the same grammar. In addition to explicit tidy principles, the `tidyverse` has further benefits: (i) if you master one package, it is easier to master others, and (ii) the core packages are developed and maintained by the Public Benefit Company Posit. These core packages contained in the `tidyverse` are: `ggplot2` (Wickham, 2016), `dplyr` (Wickham et al., 2022a), `tidyr` (Wickham and Girlich, 2022), `readr` (Wickham et al., 2022c), `purrr` (Henry and Wickham, 2020), `tibble` (Müller and Wickham, 2022), `stringr` (Wickham, 2019), and `forcats` (Wickham, 2021).

Throughout the book we use the native pipe `|>`, a powerful tool to clearly express a sequence of operations. Readers familiar with the `tidyverse` may be used to the predecessor `%>%` that is part of the `magrittr` package. For all our applications, the native and `magrittr` pipe behave identically, so we opt for the one that is simpler and part of base R. For a more thorough discussion on the subtle differences between the two pipes, we refer to the second edition[5] of Wickham and Grolemund (2016).

Prerequisites

Before we continue, make sure you have all the software you need for this book:

[2] https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html
[3] https://tidyverse.tidyverse.org/articles/manifesto.html
[4] https://tidyverse.tidyverse.org/index.html
[5] https://r4ds.hadley.nz/workflow-pipes.html

- Install R and RStudio.[6] To get a walk-through of the installation for every major operating system, follow the steps outlined in this summary.[7] The whole process should be done in a few clicks. If you wonder about the difference: R is an open-source language and environment for statistical computing and graphics, free to download and use. While R runs the computations, RStudio is an integrated development environment that provides an interface by adding many convenient features and tools. We suggest doing all the coding in RStudio.
- Open RStudio and install the `tidyverse`. Not sure how it works? You will find helpful information on how to install packages in this brief summary[8].

If you are new to R, we recommend starting with the following sources:

- A very gentle and good introduction to the workings of R can be found in the form of the weighted dice project[9]. Once you are done setting up R on your machine, try to follow the instructions in this project.
- The main book on the `tidyverse`, Wickham and Grolemund (2016), is available online and for free: R for Data Science[10] explains the majority of the tools we use in our book.
- If you are an instructor searching to effectively teach R and data science methods, we recommend taking a look at the excellent data science toolbox[11] by Mine Cetinkaya-Rundel.[12]
- RStudio provides a range of excellent cheat sheets[13] with extensive information on how to use the `tidyverse` packages.

Colophon

This book was written in RStudio using `bookdown` (Xie, 2016). The website is hosted with GitHub Pages. The complete source is available from GitHub[14]. We generated all plots in this book using `ggplot2` and its classic dark-on-light theme (`theme_bw()`).

This version of the book was built with R (R Core Team, 2022) version 4.2.1 (2022-06-23, Funny-Looking Kid) and the following packages:

[6] https://rstudio-education.github.io/hopr/starting.html#starting
[7] https://rstudio-education.github.io/hopr/starting.html#starthng
[8] https://rstudio-education.github.io/hopr/packages2.html
[9] https://rstudio-education.github.io/hopr/project-1-weighted-dice.html
[10] https://r4ds.had.co.nz/introduction.html
[11] https://datasciencebox.org/
[12] https://mine-cr.com/about/
[13] https://www.rstudio.com/resources/cheatsheets/
[14] www.github.com/voigtstefan/tidy_finance

Package	Version
alabama	2022.4-1
bookdown	0.29
broom	1.0.1
dbplyr	2.2.1
devtools	2.4.4
dplyr	1.0.10
fixest	0.10.4
forcats	0.5.2
frenchdata	0.2.0
furrr	0.3.1
ggplot2	3.3.6
glmnet	4.1-4
googledrive	2.0.0
hardhat	1.2.0
jsonlite	1.8.0
kableExtra	1.3.4
keras	2.9.0
knitr	1.40
lmtest	0.9-40
lubridate	1.8.0
purrr	0.3.4
quadprog	1.5-8
ranger	0.14.1
readr	2.1.2
readxl	1.4.1
renv	0.15.5
rlang	1.0.6
rmarkdown	2.16
RPostgres	1.4.4
RSQLite	2.2.17
sandwich	3.0-2
scales	1.2.1
slider	0.2.2
stringr	1.4.1
tibble	3.1.8
tidymodels	1.0.0
tidyquant	1.0.5
tidyr	1.2.1
tidyverse	1.3.2
timetk	2.8.1

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

Author biographies

Christoph Scheuch is the Director of Product at the social trading platform wikifolio.com. He is responsible for product planning, execution, and monitoring and manages a team of data scientists to analyze user behavior and develop data-driven products. Christoph is also an external lecturer at the Vienna University of Economics and Business, where he teaches finance students how to manage empirical projects.

Stefan Voigt is Assistant Professor of Finance at the Department of Economics at the University of Copenhagen and a research fellow at the Danish Finance Institute. His research focuses on blockchain technology, high-frequency trading, and financial econometrics. Stefan's research has been published in the leading finance and econometrics journals. He received the Danish Finance Institute teaching award 2022 for his courses for students and practitioners on empirical finance based on this book.

Patrick Weiss is a postdoctoral researcher at the Vienna University of Economics and Business and an external lecturer at Reykjavík University. His research activity centers around the intersection of empirical asset pricing and corporate finance. Patrick is especially passionate about empirical asset pricing and has published research in a top journal in financial economics.

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

Part I

Getting Started

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

1

Introduction to Tidy Finance

The main aim of this chapter is to familiarize yourself with the tidyverse. We start by downloading and visualizing stock data from Yahoo!Finance. Then we move to a simple portfolio choice problem and construct the efficient frontier. These examples introduce you to our approach of *Tidy Finance*.

1.1 Working with Stock Market Data

At the start of each session, we load the required packages. Throughout the entire book, we always use the `tidyverse` (Wickham et al., 2019). In this chapter, we also load the convenient `tidyquant` package (Dancho and Vaughan, 2022a) to download price data. This package provides a convenient wrapper for various quantitative functions compatible with the `tidyverse`.

You typically have to install a package once before you can load it. In case you have not done this yet, call `install.packages("tidyquant")`. If you have trouble using `tidyquant`, check out the corresponding documentation.[1]

```
library(tidyverse)
library(tidyquant)
```

We first download daily prices for one stock market ticker, e.g., the Apple stock, *AAPL*, directly from the data provider Yahoo!Finance. To download the data, you can use the command `tq_get`. If you do not know how to use it, make sure you read the help file by calling `?tq_get`. We especially recommend taking a look at the examples section of the documentation. We request daily data for a period of more than 20 years.

```
prices <- tq_get("AAPL",
  get = "stock.prices",
  from = "2000-01-01",
```

[1] https://cran.r-project.org/web/packages/tidyquant/vignettes/TQ00-introduction-to-tidyquant.html

```
  to = "2021-12-31"
)
prices
```

```
# A tibble: 5,535 x 8
  symbol date        open  high   low close    volume adjusted
  <chr>  <date>     <dbl> <dbl> <dbl> <dbl>     <dbl>    <dbl>
1 AAPL   2000-01-03 0.936 1.00  0.908 0.999 535796800    0.853
2 AAPL   2000-01-04 0.967 0.988 0.903 0.915 512377600    0.781
3 AAPL   2000-01-05 0.926 0.987 0.920 0.929 778321600    0.793
4 AAPL   2000-01-06 0.948 0.955 0.848 0.848 767972800    0.724
5 AAPL   2000-01-07 0.862 0.902 0.853 0.888 460734400    0.759
# ... with 5,530 more rows
```

`tq_get` downloads stock market data from Yahoo!Finance if you do not specify another data source. The function returns a tibble with eight quite self-explanatory columns: `symbol`, `date`, the market prices at the `open`, `high`, `low`, and `close`, the daily `volume` (in the number of traded shares), and the `adjusted` price in USD. The adjusted prices are corrected for anything that might affect the stock price after the market closes, e.g., stock splits and dividends. These actions affect the quoted prices, but they have no direct impact on the investors who hold the stock. Therefore, we often rely on adjusted prices when it comes to analyzing the returns an investor would have earned by holding the stock continuously.

Next, we use the `ggplot2` package (Wickham, 2016) to visualize the time series of adjusted prices in Figure 1.1. This package takes care of visualization tasks based on the principles of the grammar of graphics (Wilkinson, 2012).

```
prices |>
  ggplot(aes(x = date, y = adjusted)) +
  geom_line() +
  labs(
    x = NULL,
    y = NULL,
    title = "Apple stock prices between beginning of 2000 and end of 2021"
  )
```

Instead of analyzing prices, we compute daily net returns defined as $r_t = p_t/p_{t-1} - 1$, where p_t is the adjusted day t price. In that context, the function `lag()` is helpful, which returns the previous value in a vector.

```
returns <- prices |>
  arrange(date) |>
  mutate(ret = adjusted / lag(adjusted) - 1) |>
```

FIGURE 1.1
Prices are in USD, adjusted for dividend payments and stock splits.

```
  select(symbol, date, ret)
returns
```

```
# A tibble: 5,535 x 3
  symbol date           ret
  <chr>  <date>       <dbl>
1 AAPL   2000-01-03 NA
2 AAPL   2000-01-04 -0.0843
3 AAPL   2000-01-05  0.0146
4 AAPL   2000-01-06 -0.0865
5 AAPL   2000-01-07  0.0474
# ... with 5,530 more rows
```

The resulting tibble contains three columns, where the last contains the daily returns (`ret`). Note that the first entry naturally contains a missing value (`NA`) because there is no previous price. Obviously, the use of `lag()` would be meaningless if the time series is not ordered by ascending dates. The command `arrange()` provides a convenient way to order observations in the correct way for our application. In case you want to order observations by descending dates, you can use `arrange(desc(date))`.

For the upcoming examples, we remove missing values as these would require separate treatment when computing, e.g., sample averages. In general, however, make sure you understand why `NA` values occur and carefully examine if you can simply get rid of these observations.

```
returns <- returns |>
  drop_na(ret)
```

Next, we visualize the distribution of daily returns in a histogram in Figure 1.2. For convenience, we multiply the returns by 100 to get returns in percent for the visualizations. Additionally, we add a dashed line that indicates the 5 percent quantile of the daily returns to the histogram, which is a (crude) proxy for the worst return of the stock with a probability of at most 5 percent. The 5 percent quantile is closely connected to the (historical) value-at-risk, a risk measure commonly monitored by regulators. We refer to Tsay (2010) for a more thorough introduction to stylized facts of returns.

```
quantile_05 <- quantile(returns |> pull(ret) * 100, probs = 0.05)

returns |>
  ggplot(aes(x = ret * 100)) +
  geom_histogram(bins = 100) +
  geom_vline(aes(xintercept = quantile_05),
    linetype = "dashed"
  ) +
  labs(
    x = NULL,
    y = NULL,
    title = "Distribution of daily Apple stock returns in percent"
  )
```

Here, `bins = 100` determines the number of bins used in the illustration and hence implicitly the width of the bins. Before proceeding, make sure you understand how to use the geom `geom_vline()` to add a dashed line that indicates the 5 percent quantile of the daily returns. A typical task before proceeding with *any* data is to compute summary statistics for the main variables of interest.

```
returns |>
  mutate(ret = ret * 100) |>
  summarize(across(
    ret,
    list(
      daily_mean = mean,
      daily_sd = sd,
      daily_min = min,
      daily_max = max
    )
  ))
```

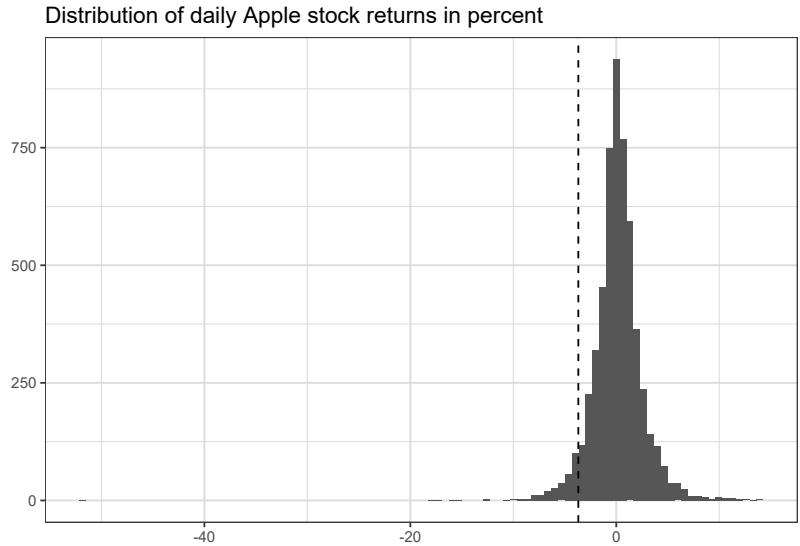


FIGURE 1.2
The dotted vertical line indicates the historical 5 percent quantile.

```
# A tibble: 1 x 4
  ret_daily_mean ret_daily_sd ret_daily_min ret_daily_max
           <dbl>        <dbl>         <dbl>         <dbl>
1          0.130         2.52         -51.9          13.9
```

We see that the maximum *daily* return was 13.905 percent. Perhaps not surprisingly, the average daily return is close to but slightly above 0. In line with the illustration above, the large losses on the day with the minimum returns indicate a strong asymmetry in the distribution of returns.

You can also compute these summary statistics for each year individually by imposing `group_by(year = year(date))`, where the call `year(date)` returns the year. More specifically, the few lines of code below compute the summary statistics from above for individual groups of data defined by year. The summary statistics, therefore, allow an eyeball analysis of the time-series dynamics of the return distribution.

```
returns |>
  mutate(ret = ret * 100) |>
  group_by(year = year(date)) |>
  summarize(across(
    ret,
    list(
      daily_mean = mean,
      daily_sd = sd,
      daily_min = min,
      daily_max = max
```

```
    ),
    .names = "{.fn}"
  )) |>
  print(n = Inf)
```

```
# A tibble: 22 x 5
    year daily_mean daily_sd daily_min daily_max
   <dbl>      <dbl>    <dbl>     <dbl>     <dbl>
 1  2000   -0.346       5.49    -51.9      13.7
 2  2001    0.233       3.93    -17.2      12.9
 3  2002   -0.121       3.05    -15.0       8.46
 4  2003    0.186       2.34     -8.14     11.3
 5  2004    0.470       2.55     -5.58     13.2
 6  2005    0.349       2.45     -9.21      9.12
 7  2006    0.0949      2.43     -6.33     11.8
 8  2007    0.366       2.38     -7.02     10.5
 9  2008   -0.265       3.67    -17.9      13.9
10  2009    0.382       2.14     -5.02      6.76
11  2010    0.183       1.69     -4.96      7.69
12  2011    0.104       1.65     -5.59      5.89
13  2012    0.130       1.86     -6.44      8.87
14  2013    0.0472      1.80    -12.4       5.14
15  2014    0.145       1.36     -7.99      8.20
16  2015    0.00199     1.68     -6.12      5.74
17  2016    0.0575      1.47     -6.57      6.50
18  2017    0.164       1.11     -3.88      6.10
19  2018   -0.00573     1.81     -6.63      7.04
20  2019    0.266       1.65     -9.96      6.83
21  2020    0.281       2.94    -12.9      12.0
22  2021    0.133       1.58     -4.17      5.39
```

In case you wonder: the additional argument `.names = "{.fn}"` in `across()` determines how to name the output columns. The specification is rather flexible and allows almost arbitrary column names, which can be useful for reporting. The `print()` function simply controls the output options for the R console.

1.2 Scaling Up the Analysis

As a next step, we generalize the code from before such that all the computations can handle an arbitrary vector of tickers (e.g., all constituents of an index). Following

tidy principles, it is quite easy to download the data, plot the price time series, and tabulate the summary statistics for an arbitrary number of assets.

This is where the `tidyverse` magic starts: tidy data makes it extremely easy to generalize the computations from before to as many assets as you like. The following code takes any vector of tickers, e.g., `ticker <- c("AAPL", "MMM", "BA")`, and automates the download as well as the plot of the price time series. In the end, we create the table of summary statistics for an arbitrary number of assets. We perform the analysis with data from all current constituents of the Dow Jones Industrial Average index.[2]

```
ticker <- tq_index("DOW")
ticker
```

```
# A tibble: 30 x 8
  symbol company          ident~1 sedol weight sector share~2 local~3
  <chr>  <chr>            <chr>   <chr>  <dbl> <chr>    <dbl> <chr>
1 UNH    UnitedHealth Gr~ 91324P~ 2917~ 0.115  Healt~ 5715189 USD
2 GS     Goldman Sachs G~ 38141G~ 2407~ 0.0659 Finan~ 5715189 USD
3 HD     Home Depot Inc.  437076~ 2434~ 0.0608 Consu~ 5715189 USD
4 MCD    McDonald's Corp~ 580135~ 2550~ 0.0535 Consu~ 5715189 USD
5 MSFT   Microsoft Corpo~ 594918~ 2588~ 0.0535 Infor~ 5715189 USD
# ... with 25 more rows, and abbreviated variable names
#   1: identifier, 2: shares_held, 3: local_currency
```

Conveniently, `tidyquant` provides a function to get all stocks in a stock index with a single call (similarly, `tq_exchange("NASDAQ")` delivers all stocks currently listed on the NASDAQ exchange).

```
index_prices <- tq_get(ticker,
  get = "stock.prices",
  from = "2000-01-01",
  to = "2022-12-31"
)
```

The resulting tibble contains 163613 daily observations for 30 different corporations. Figure 1.3 illustrates the time series of downloaded *adjusted* prices for each of the constituents of the Dow Jones index. Make sure you understand every single line of code! (What are the arguments of `aes()`? Which alternative `geoms` could you use to visualize the time series? Hint: if you do not know the answers try to change the code to see what difference your intervention causes.)

[2] https://en.wikipedia.org/wiki/Dow_Jones_Industrial_Average

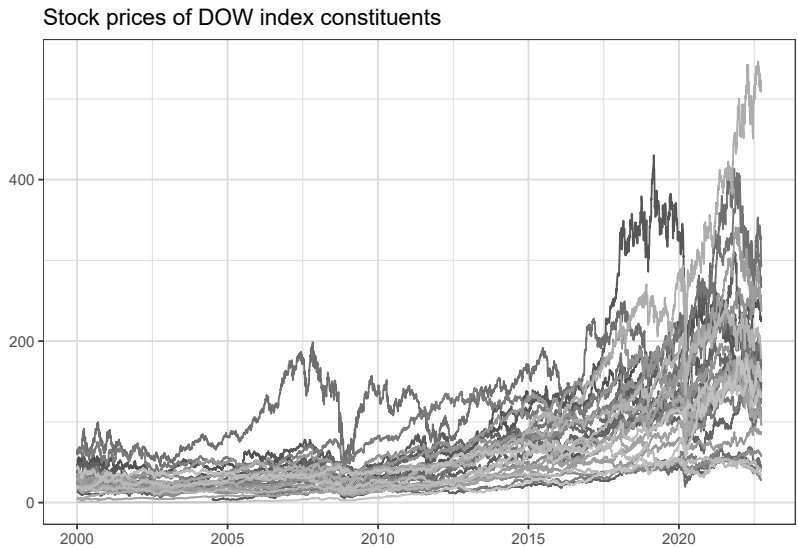


FIGURE 1.3
Prices in USD, adjusted for dividend payments and stock splits.

```
index_prices |>
  ggplot(aes(
    x = date,
    y = adjusted,
    color = symbol
  )) +
  geom_line() +
  labs(
    x = NULL,
    y = NULL,
    color = NULL,
    title = "Stock prices of DOW index constituents"
  ) +
  theme(legend.position = "none")
```

Do you notice the small differences relative to the code we used before? `tq_get(ticker)` returns a tibble for several symbols as well. All we need to do to illustrate all tickers simultaneously is to include `color = symbol` in the `ggplot2` aesthetics. In this way, we generate a separate line for each ticker. Of course, there are simply too many lines on this graph to identify the individual stocks properly, but it illustrates the point well.

The same holds for stock returns. Before computing the returns, we use `group_by(symbol)` such that the `mutate()` command is performed for each

symbol individually. The same logic also applies to the computation of summary statistics: `group_by(symbol)` is the key to aggregating the time series into ticker-specific variables of interest.

```
all_returns <- index_prices |>
  group_by(symbol) |>
  mutate(ret = adjusted / lag(adjusted) - 1) |>
  select(symbol, date, ret) |>
  drop_na(ret)

all_returns |>
  mutate(ret = ret * 100) |>
  group_by(symbol) |>
  summarize(across(
    ret,
    list(
      daily_mean = mean,
      daily_sd = sd,
      daily_min = min,
      daily_max = max
    ),
    .names = "{.fn}"
  )) |>
  print(n = Inf)
```

```
# A tibble: 30 x 5
   symbol daily_mean daily_sd daily_min daily_max
   <chr>       <dbl>    <dbl>     <dbl>     <dbl>
 1 AMGN       0.0466     1.97     -13.4      15.1
 2 AXP        0.0508     2.30     -17.6      21.9
 3 BA         0.0528     2.23     -23.8      24.3
 4 CAT        0.0645     2.04     -14.5      14.7
 5 CRM        0.113      2.69     -27.1      26.0
 6 CSCO       0.0289     2.38     -16.2      24.4
 7 CVX        0.0514     1.76     -22.1      22.7
 8 DIS        0.0435     1.94     -18.4      16.0
 9 DOW        0.0414     2.63     -21.7      20.9
10 GS         0.0525     2.32     -19.0      26.5
11 HD         0.0517     1.94     -28.7      14.1
12 HON        0.0479     1.94     -17.4      28.2
13 IBM        0.0247     1.65     -15.5      12.0
14 INTC       0.0285     2.36     -22.0      20.1
15 JNJ        0.0399     1.22     -15.8      12.2
16 JPM        0.0544     2.43     -20.7      25.1
17 KO         0.0318     1.32     -10.1      13.9
```

```
18 MCD        0.0519     1.48     -15.9      18.1
19 MMM        0.0368     1.50     -12.9      12.6
20 MRK        0.0340     1.68     -26.8      13.0
21 MSFT       0.0513     1.93     -15.6      19.6
22 NKE        0.0711     1.92     -19.8      15.5
23 PG         0.0355     1.34     -30.2      12.0
24 TRV        0.0536     1.84     -20.8      25.6
25 UNH        0.0986     1.98     -18.6      34.8
26 V          0.0900     1.90     -13.6      15.0
27 VZ         0.0236     1.51     -11.8      14.6
28 WBA        0.0258     1.81     -15.0      16.6
29 WMT        0.0302     1.50     -11.4      11.7
30 AAPL       0.124      2.51     -51.9      13.9
```

Note that you are now also equipped with all tools to download price data for *each* ticker listed in the S&P 500 index with the same number of lines of code. Just use `ticker <- tq_index("SP500")`, which provides you with a tibble that contains each symbol that is (currently) part of the S&P 500. However, don't try this if you are not prepared to wait for a couple of minutes because this is quite some data to download!

1.3 Other Forms of Data Aggregation

Of course, aggregation across variables other than `symbol` can also make sense. For instance, suppose you are interested in answering the question: are days with high aggregate trading volume likely followed by days with high aggregate trading volume? To provide some initial analysis on this question, we take the downloaded data and compute aggregate daily trading volume for all Dow Jones constituents in USD. Recall that the column `volume` is denoted in the number of traded shares. Thus, we multiply the trading volume with the daily closing price to get a proxy for the aggregate trading volume in USD. Scaling by `1e9` (R can handle scientific notation) denotes daily trading volume in billion USD.

```
volume <- index_prices |>
  group_by(date) |>
  summarize(volume = sum(volume * close / 1e9))

volume |>
  ggplot(aes(x = date, y = volume)) +
  geom_line() +
  labs(
    x = NULL, y = NULL,
```

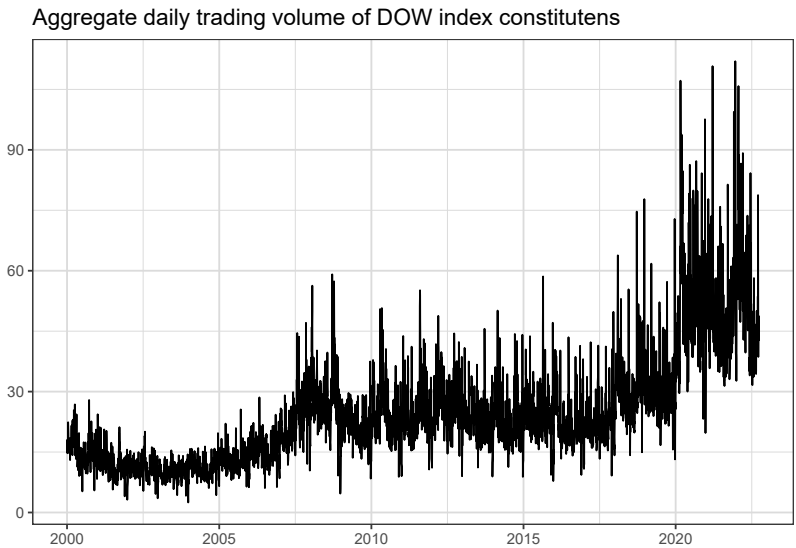


FIGURE 1.4
Total daily trading volume in billion USD.

```
    title = "Aggregate daily trading volume of DOW index constitutens"
  )
```

Figure 1.4 indicates a clear upward trend in aggregated daily trading volume. In particular, since the outbreak of the COVID-19 pandemic, markets have processed substantial trading volumes, as analyzed, for instance, by Goldstein et al. (2021). One way to illustrate the persistence of trading volume would be to plot volume on day t against volume on day $t-1$ as in the example below. In Figure 1.5, we add a dotted 45°-line to indicate a hypothetical one-to-one relation by `geom_abline()`, addressing potential differences in the axes' scales.

```
volume |>
  ggplot(aes(x = lag(volume), y = volume)) +
  geom_point() +
  geom_abline(aes(intercept = 0, slope = 1),
    linetype = "dashed"
  ) +
  labs(
    x = "Previous day aggregate trading volume",
    y = "Aggregate trading volume",
    title = "Persistence in daily trading volume of DOW index constituents"
  )
```

```
Warning: Removed 1 rows containing missing values (geom_point).
```

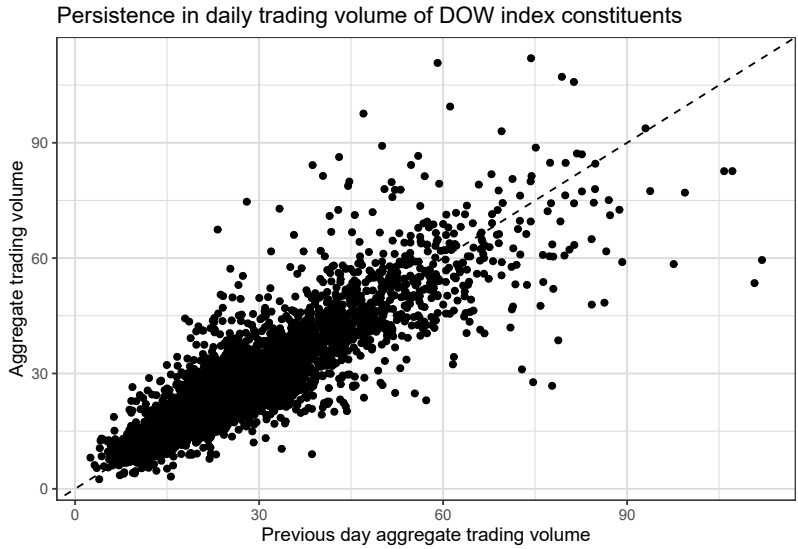


FIGURE 1.5
Total daily trading volume in billion USD.

Do you understand where the warning `## Warning: Removed 1 rows containing missing values (geom_point).` comes from and what it means? Purely eye-balling reveals that days with high trading volume are often followed by similarly high trading volume days.

1.4 Portfolio Choice Problems

In the previous part, we show how to download stock market data and inspect it with graphs and summary statistics. Now, we move to a typical question in Finance: how to allocate wealth across different assets optimally. The standard framework for optimal portfolio selection considers investors that prefer higher future returns but dislike future return volatility (defined as the square root of the return variance): the *mean-variance investor* (Markowitz, 1952).

An essential tool to evaluate portfolios in the mean-variance context is the *efficient frontier*, the set of portfolios which satisfies the condition that no other portfolio exists with a higher expected return but with the same volatility (the square root of the variance, i.e., the risk), see, e.g., Merton (1972). We compute and visualize the efficient frontier for several stocks. First, we extract each asset's *monthly* returns. In order to keep things simple, we work with a balanced panel and exclude DOW

constituents for which we do not observe a price on every single trading day since the year 2000.

```
index_prices <- index_prices |>
  group_by(symbol) |>
  mutate(n = n()) |>
  ungroup() |>
  filter(n == max(n)) |>
  select(-n)

returns <- index_prices |>
  mutate(month = floor_date(date, "month")) |>
  group_by(symbol, month) |>
  summarize(price = last(adjusted), .groups = "drop_last") |>
  mutate(ret = price / lag(price) - 1) |>
  drop_na(ret) |>
  select(-price)
```

Here, `floor_date()` is a function from the `lubridate` package (Grolemund and Wickham, 2011), which provides useful functions to work with dates and times.

Next, we transform the returns from a tidy tibble into a $(T \times N)$ matrix with one column for each of the N tickers and one row for each of the T trading days to compute the sample average return vector

$$\hat{\mu} = \frac{1}{T}\sum_{t=1}^{T} r_t$$

where r_t is the N vector of returns on date t and the sample covariance matrix

$$\hat{\Sigma} = \frac{1}{T-1}\sum_{t=1}^{T}(r_t - \hat{\mu})(r_t - \hat{\mu})'.$$

We achieve this by using `pivot_wider()` with the new column names from the column `symbol` and setting the values to `ret`. We compute the vector of sample average returns and the sample variance-covariance matrix, which we consider as proxies for the parameters of the distribution of future stock returns. Thus, for simplicity, we refer to Σ and μ instead of explicitly highlighting that the sample moments are estimates. In later chapters, we discuss the issues that arise once we take estimation uncertainty into account.

```
returns_matrix <- returns |>
  pivot_wider(
    names_from = symbol,
    values_from = ret
```

```
  ) |>
  select(-month)

Sigma <- cov(returns_matrix)
mu <- colMeans(returns_matrix)
```

Then, we compute the minimum variance portfolio weights ω_{mvp} as well as the expected portfolio return $\omega_{\text{mvp}}'\mu$ and volatility $\sqrt{\omega_{\text{mvp}}'\Sigma\omega_{\text{mvp}}}$ of this portfolio. Recall that the minimum variance portfolio is the vector of portfolio weights that are the solution to

$$\omega_{\text{mvp}} = \arg\min w'\Sigma w \text{ s.t. } \sum_{i=1}^{N} w_i = 1.$$

The constraint that weights sum up to one simply implies that all funds are distributed across the available asset universe, i.e., there is no possibility to retain cash. It is easy to show analytically that $\omega_{\text{mvp}} = \frac{\Sigma^{-1}\iota}{\iota'\Sigma^{-1}\iota}$, where ι is a vector of ones and Σ^{-1} is the inverse of Σ.

```
N <- ncol(returns_matrix)
iota <- rep(1, N)
mvp_weights <- solve(Sigma) %*% iota
mvp_weights <- mvp_weights / sum(mvp_weights)

tibble(
  average_ret = as.numeric(t(mvp_weights) %*% mu),
  volatility = as.numeric(sqrt(t(mvp_weights) %*% Sigma %*% mvp_weights))
)
```

```
# A tibble: 1 x 2
  average_ret volatility
        <dbl>      <dbl>
1     0.00770     0.0315
```

The command `solve(A, b)` returns the solution of a system of equations $Ax = b$. If `b` is not provided, as in the example above, it defaults to the identity matrix such that `solve(Sigma)` delivers Σ^{-1} (if a unique solution exists).
Note that the *monthly* volatility of the minimum variance portfolio is of the same order of magnitude as the *daily* standard deviation of the individual components. Thus, the diversification benefits in terms of risk reduction are tremendous!

Next, we set out to find the weights for a portfolio that achieves, as an example, three times the expected return of the minimum variance portfolio. However, mean-variance investors are not interested in any portfolio that achieves the required return but rather in the efficient portfolio, i.e., the portfolio with the lowest standard

deviation. If you wonder where the solution ω_{eff} comes from: The efficient portfolio is chosen by an investor who aims to achieve minimum variance *given a minimum acceptable expected return* $\bar{\mu}$. Hence, their objective function is to choose ω_{eff} as the solution to

$$\omega_{\text{eff}}(\bar{\mu}) = \arg\min w'\Sigma w \text{ s.t. } w'\iota = 1 \text{ and } \omega'\mu \geq \bar{\mu}.$$

The code below implements the analytic solution to this optimization problem for a benchmark return $\bar{\mu}$, which we set to 3 times the expected return of the minimum variance portfolio. We encourage you to verify that it is correct.

```
mu_bar <- 3 * t(mvp_weights) %*% mu

C <- as.numeric(t(iota) %*% solve(Sigma) %*% iota)
D <- as.numeric(t(iota) %*% solve(Sigma) %*% mu)
E <- as.numeric(t(mu) %*% solve(Sigma) %*% mu)

lambda_tilde <- as.numeric(2 * (mu_bar - D / C) / (E - D^2 / C))
efp_weights <- mvp_weights +
  lambda_tilde / 2 * (solve(Sigma) %*% mu - D * mvp_weights)
```

1.5 The Efficient Frontier

The mutual fund separation theorem states that as soon as we have two efficient portfolios (such as the minimum variance portfolio w_{mvp} and the efficient portfolio for a higher required level of expected returns $\omega_{\text{eff}}(\bar{\mu})$, we can characterize the entire efficient frontier by combining these two portfolios. That is, any linear combination of the two portfolio weights will again represent an efficient portfolio. The code below implements the construction of the *efficient frontier*, which characterizes the highest expected return achievable at each level of risk. To understand the code better, make sure to familiarize yourself with the inner workings of the `for` loop.

```
c <- seq(from = -0.4, to = 1.9, by = 0.01)
res <- tibble(
  c = c,
  mu = NA,
  sd = NA
)

for (i in seq_along(c)) {
  w <- (1 - c[i]) * mvp_weights + (c[i]) * efp_weights
  res$mu[i] <- 12 * 100 * t(w) %*% mu
```

```
  res$sd[i] <- 12 * sqrt(100) * sqrt(t(w) %*% Sigma %*% w)
}
```

The code above proceeds in two steps: First, we compute a vector of combination weights c and then we evaluate the resulting linear combination with $c \in \mathbb{R}$:

$$w^* = cw_{\text{eff}}(\bar{\mu}) + (1-c)w_{mvp} = \omega_{\text{mvp}} + \frac{\lambda^*}{2}\left(\Sigma^{-1}\mu - \frac{D}{C}\Sigma^{-1}\iota\right)$$

with $\lambda^* = 2\frac{c\bar{\mu}+(1-c)\bar{\mu}-D/C}{E-D^2/C}$. where $C = \iota'\Sigma^{-1}\iota$, $D = \iota'\Sigma^{-1}\mu$, and $E = \mu'\Sigma^{-1}\mu$. Finally, it is simple to visualize the efficient frontier alongside the two efficient portfolios within one powerful figure using `ggplot2` (see Figure 1.6). We also add the individual stocks in the same call. We compute annualized returns based on the simple assumption that monthly returns are independent and identically distributed. Thus, the average annualized return is just 12 times the expected monthly return.

```
res |>
  ggplot(aes(x = sd, y = mu)) +
  geom_point() +
  geom_point(
    data = res |> filter(c %in% c(0, 1)),
    size = 4
  ) +
  geom_point(
    data = tibble(
      mu = 12 * 100 * mu,
      sd = 12 * 10 * sqrt(diag(Sigma))
    ),
    aes(y = mu, x = sd), size = 1
  ) +
  labs(
    x = "Annualized standard deviation (in percent)",
    y = "Annualized expected return (in percent)",
    title = "Efficient frontier for DOW index constituents"
  )
```

The line in Figure 1.6 indicates the efficient frontier: the set of portfolios a mean-variance efficient investor would choose from. Compare the performance relative to the individual assets (the dots) – it should become clear that diversifying yields massive performance gains (at least as long as we take the parameters Σ and μ as given).

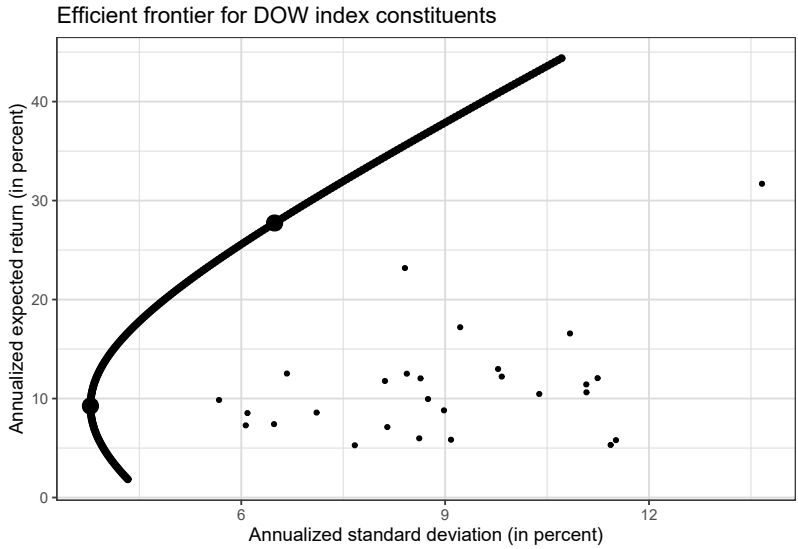


FIGURE 1.6
The big dots indicate the location of the minimum variance and efficient tangency portfolios, respectively. The small dots indicate the location of the individual constituents.

1.6 Exercises

1. Download daily prices for another stock market ticker of your choice from Yahoo!Finance with `tq_get()` from the `tidyquant` package. Plot two time series of the ticker's un-adjusted and adjusted closing prices. Explain the differences.
2. Compute daily net returns for the asset and visualize the distribution of daily returns in a histogram. Also, use `geom_vline()` to add a dashed red line that indicates the 5 percent quantile of the daily returns within the histogram. Compute summary statistics (mean, standard deviation, minimum and maximum) for the daily returns
3. Take your code from before and generalize it such that you can perform all the computations for an arbitrary vector of tickers (e.g., `ticker <- c("AAPL", "MMM", "BA")`). Automate the download, the plot of the price time series, and create a table of return summary statistics for this arbitrary number of assets.
4. Consider the research question: Are days with high aggregate trading volume often also days with large absolute price changes? Find an appropriate visualization to analyze the question.

5. Compute monthly returns from the downloaded stock market prices. Compute the vector of historical average returns and the sample variance-covariance matrix. Compute the minimum variance portfolio weights and the portfolio volatility and average returns. Visualize the mean-variance efficient frontier. Choose one of your assets and identify the portfolio which yields the same historical volatility but achieves the highest possible average return.
6. In the portfolio choice analysis, we restricted our sample to all assets trading every day since 2000. How is such a decision a problem when you want to infer future expected portfolio performance from the results?
7. The efficient frontier characterizes the portfolios with the highest expected return for different levels of risk, i.e., standard deviation. Identify the portfolio with the highest expected return per standard deviation. Hint: the ratio of expected return to standard deviation is an important concept in Finance.

Part II

Financial Data

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com

2

Accessing & Managing Financial Data

In this chapter, we suggest a way to organize your financial data. Everybody, who has experience with data, is also familiar with storing data in various formats like CSV, XLS, XLSX, or other delimited value storage. Reading and saving data can become very cumbersome in the case of using different data formats, both across different projects and across different programming languages. Moreover, storing data in delimited files often leads to problems with respect to column type consistency. For instance, date-type columns frequently lead to inconsistencies across different data formats and programming languages.

This chapter shows how to import different open source data sets. Specifically, our data comes from the application programming interface (API) of Yahoo!Finance, a downloaded standard CSV file, an XLSX file stored in a public Google Drive repository, and other macroeconomic time series. We store all the data in a *single* database, which serves as the only source of data in subsequent chapters. We conclude the chapter by providing some tips on managing databases.

First, we load the global packages that we use throughout this chapter. Later on, we load more packages in the sections where we need them.

```
library(tidyverse)
library(lubridate)
library(scales)
```

The package `lubridate` provides convenient tools to work with dates and times (Grolemund and Wickham, 2011). The package `scales` (Wickham and Seidel, 2022) provides useful scale functions for visualizations.

Moreover, we initially define the date range for which we fetch and store the financial data, making future data updates tractable. In case you need another time frame, you can adjust the dates below. Our data starts with 1960 since most asset pricing studies use data from 1962 on.

```
start_date <- ymd("1960-01-01")
end_date <- ymd("2021-12-31")
```

2.1 Fama-French Data

We start by downloading some famous Fama-French factors (e.g., Fama and French, 1993) and portfolio returns commonly used in empirical asset pricing. Fortunately, there is a neat package by Nelson Areal[1] that allows us to access the data easily: the `frenchdata` package provides functions to download and read data sets from Prof. Kenneth French finance data library[2] (Areal, 2021).

```
library(frenchdata)
```

We can use the main function of the package to download monthly Fama-French factors. The set *3 Factors* includes the return time series of the market, size, and value factors alongside the risk-free rates. Note that we have to do some manual work to correctly parse all the columns and scale them appropriately, as the raw Fama-French data comes in a very unpractical data format. For precise descriptions of the variables, we suggest consulting Prof. Kenneth French's finance data library directly. If you are on the site, check the raw data files to appreciate the time you can save thanks to `frenchdata`.

```
factors_ff_monthly_raw <- download_french_data("Fama/French 3 Factors")
factors_ff_monthly <- factors_ff_monthly_raw$subsets$data[[1]] |>
  transmute(
    month = floor_date(ymd(str_c(date, "01")), "month"),
    rf = as.numeric(RF) / 100,
    mkt_excess = as.numeric(`Mkt-RF`) / 100,
    smb = as.numeric(SMB) / 100,
    hml = as.numeric(HML) / 100
  ) |>
  filter(month >= start_date & month <= end_date)
```

It is straightforward to download the corresponding *daily* Fama-French factors with the same function.

```
factors_ff_daily_raw <- download_french_data("Fama/French 3 Factors [Daily]")
factors_ff_daily <- factors_ff_daily_raw$subsets$data[[1]] |>
  transmute(
    date = ymd(date),
    rf = as.numeric(RF) / 100,
    mkt_excess = as.numeric(`Mkt-RF`) / 100,
    smb = as.numeric(SMB) / 100,
```

[1] https://github.com/nareal/frenchdata/
[2] https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html

```
    hml = as.numeric(HML) / 100
  ) |>
  filter(date >= start_date & date <= end_date)
```

In a subsequent chapter, we also use the 10 monthly industry portfolios, so let us fetch that data, too.

```
industries_ff_monthly_raw <- download_french_data("10 Industry Portfolios")
industries_ff_monthly <- industries_ff_monthly_raw$subsets$data[[1]] |>
  mutate(month = floor_date(ymd(str_c(date, "01")), "month")) |>
  mutate(across(where(is.numeric), ~ . / 100)) |>
  select(month, everything(), -date) |>
  filter(month >= start_date & month <= end_date)
```

It is worth taking a look at all available portfolio return time series from Kenneth French's homepage. You should check out the other sets by calling `get_french_data_list()`.

2.2 q-Factors

In recent years, the academic discourse experienced the rise of alternative factor models, e.g., in the form of the Hou et al. (2014) *q*-factor model. We refer to the extended background[3] information provided by the original authors for further information. The *q* factors can be downloaded directly from the authors' homepage from within `read_csv()`.

We also need to adjust this data. First, we discard information we will not use in the remainder of the book. Then, we rename the columns with the "R_"-prescript using regular expressions and write all column names in lowercase. You should always try sticking to a consistent style for naming objects, which we try to illustrate here – the emphasis is on *try*. You can check out style guides available online, e.g., Hadley Wickham's `tidyverse` style guide.[4]

```
factors_q_monthly_link <-
  "http://global-q.org/uploads/1/2/2/6/122679606/q5_factors_monthly_2021.csv"

factors_q_monthly <- read_csv(factors_q_monthly_link) |>
  mutate(month = ymd(str_c(year, month, "01", sep = "-"))) |>
```

[3] http://global-q.org/background.html
[4] https://style.tidyverse.org/index.html

```
  select(-R_F, -R_MKT, -year) |>
  rename_with(~ str_remove(., "R_")) |>
  rename_with(~ str_to_lower(.)) |>
  mutate(across(-month, ~ . / 100)) |>
  filter(month >= start_date & month <= end_date)
```

2.3 Macroeconomic Predictors

Our next data source is a set of macroeconomic variables often used as predictors for the equity premium. Welch and Goyal (2008) comprehensively reexamine the performance of variables suggested by the academic literature to be good predictors of the equity premium. The authors host the data updated to 2021 on Amit Goyal's website.[5] Since the data is an XLSX-file stored on a public Google drive location, we need additional packages to access the data directly from our R session. Therefore, we load `readxl` to read the XLSX-file (Wickham and Bryan, 2022) and `googledrive` for the Google drive connection (D'Agostino McGowan and Bryan, 2021).

```
library(readxl)
library(googledrive)
```

Usually, you need to authenticate if you interact with Google drive directly in R. Since the data is stored via a public link, we can proceed without any authentication.

```
drive_deauth()
```

The `drive_download()` function from the `googledrive` package allows us to download the data and store it locally.

```
macro_predictors_link <-
  "https://docs.google.com/spreadsheets/d/1OArfD2Wv9IvGoLkJ8JyoXS0YMQLDZfY2"

drive_download(
  macro_predictors_link,
  path = "data/macro_predictors.xlsx"
)
```

Next, we read in the new data and transform the columns into the variables that we later use:

[5] https://sites.google.com/view/agoyal145

1. The dividend price ratio (`dp`), the difference between the log of dividends and the log of prices, where dividends are 12-month moving sums of dividends paid on the S&P 500 index, and prices are monthly averages of daily closing prices (Campbell and Shiller, 1988; Campbell and Yogo, 2006).
2. Dividend yield (`dy`), the difference between the log of dividends and the log of lagged prices (Ball, 1978).
3. Earnings price ratio (`ep`), the difference between the log of earnings and the log of prices, where earnings are 12-month moving sums of earnings on the S&P 500 index (Campbell and Shiller, 1988).
4. Dividend payout ratio (`de`), the difference between the log of dividends and the log of earnings (Lamont, 1998).
5. Stock variance (`svar`), the sum of squared daily returns on the S&P 500 index (Guo, 2006).
6. Book-to-market ratio (`bm`), the ratio of book value to market value for the Dow Jones Industrial Average (Kothari and Shanken, 1997)
7. Net equity expansion (`ntis`), the ratio of 12-month moving sums of net issues by NYSE listed stocks divided by the total end-of-year market capitalization of NYSE stocks (Campbell et al., 2008).
8. Treasury bills (`tbl`), the 3-Month Treasury Bill: Secondary Market Rate from the economic research database at the Federal Reserve Bank at St. Louis (Campbell, 1987).
9. Long-term yield (`lty`), the long-term government bond yield from Ibbotson's Stocks, Bonds, Bills, and Inflation Yearbook (Welch and Goyal, 2008).
10. Long-term rate of returns (`ltr`), the long-term government bond returns from Ibbotson's Stocks, Bonds, Bills, and Inflation Yearbook (Welch and Goyal, 2008).
11. Term spread (`tms`), the difference between the long-term yield on government bonds and the Treasury bill (Campbell, 1987).
12. Default yield spread (`dfy`), the difference between BAA and AAA-rated corporate bond yields (Fama and French, 1989).
13. Inflation (`infl`), the Consumer Price Index (All Urban Consumers) from the Bureau of Labor Statistics (Campbell and Vuolteenaho, 2004).

For variable definitions and the required data transformations, you can consult the material on Amit Goyal's website[6].

```
macro_predictors <- read_xlsx(
  "data/macro_predictors.xlsx",
  sheet = "Monthly"
) |>
  mutate(month = ym(yyyymm)) |>
  mutate(across(where(is.character), as.numeric)) |>
```

[6] https://sites.google.com/view/agoyal145

```
  mutate(
    IndexDiv = Index + D12,
    logret = log(IndexDiv) - log(lag(IndexDiv)),
    Rfree = log(Rfree + 1),
    rp_div = lead(logret - Rfree, 1), # Future excess market return
    dp = log(D12) - log(Index), # Dividend Price ratio
    dy = log(D12) - log(lag(Index)), # Dividend yield
    ep = log(E12) - log(Index), # Earnings price ratio
    de = log(D12) - log(E12), # Dividend payout ratio
    tms = lty - tbl, # Term spread
    dfy = BAA - AAA # Default yield spread
  ) |>
  select(month, rp_div, dp, dy, ep, de, svar,
    bm = `b/m`, ntis, tbl, lty, ltr,
    tms, dfy, infl
  ) |>
  filter(month >= start_date & month <= end_date) |>
  drop_na()
```

Finally, after reading in the macro predictors to our memory, we remove the raw data file from our temporary storage.

```
file.remove("data/macro_predictors.xlsx")
```

```
[1] TRUE
```

2.4 Other Macroeconomic Data

The Federal Reserve bank of St. Louis provides the Federal Reserve Economic Data (FRED), an extensive database for macroeconomic data. In total, there are 817,000 US and international time series from 108 different sources. As an illustration, we use the already familiar `tidyquant` package to fetch consumer price index (CPI) data that can be found under the CPIAUCNS[7] key.

```
library(tidyquant)

cpi_monthly <- tq_get("CPIAUCNS",
```

[7] https://fred.stlouisfed.org/series/CPIAUCNS

```
  get = "economic.data",
  from = start_date,
  to = end_date
) |>
  transmute(
    month = floor_date(date, "month"),
    cpi = price / price[month == max(month)]
  )
```

To download other time series, we just have to look it up on the FRED website and extract the corresponding key from the address. For instance, the producer price index for gold ores can be found under the PCU2122212122210[8] key. The `tidyquant` package provides access to around 10,000 time series of the FRED database. If your desired time series is not included, we recommend working with the `fredr` package (Boysel and Vaughan, 2021). Note that you need to get an API key to use its functionality. We refer to the package documentation for details.

2.5 Setting Up a Database

Now that we have downloaded some (freely available) data from the web into the memory of our R session let us set up a database to store that information for future use. We will use the data stored in this database throughout the following chapters, but you could alternatively implement a different strategy and replace the respective code.

There are many ways to set up and organize a database, depending on the use case. For our purpose, the most efficient way is to use an SQLite[9] database, which is the C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine. Note that SQL[10] (Structured Query Language) is a standard language for accessing and manipulating databases and heavily inspired the `dplyr` functions. We refer to this tutorial[11] for more information on SQL.

There are two packages that make working with SQLite in R very simple: `RSQLite` (Müller et al., 2022) embeds the SQLite database engine in R, and `dbplyr` (Wickham et al., 2022b) is the database back-end for `dplyr`. These packages allow to set up a database to remotely store tables and use these remote database tables as if they are

[8] https://fred.stlouisfed.org/series/PCU2122212122210
[9] https://www.sqlite.org/index.html
[10] https://en.wikipedia.org/wiki/SQL
[11] https://www.w3schools.com/sql/sql_intro.asp

in-memory data frames by automatically converting `dplyr` into SQL. Check out the `RSQLite`[12] and `dbplyr`[13] vignettes for more information.

```
library(RSQLite)
library(dbplyr)
```

An SQLite database is easily created – the code below is really all there is. You do not need any external software. Note that we use the `extended_types=TRUE` option to enable date types when storing and fetching data. Otherwise, date columns are stored and retrieved as integers. We will use the resulting file `tidy_finance.sqlite` in the subfolder `data` for all subsequent chapters to retrieve our data.

```
tidy_finance <- dbConnect(
  SQLite(),
  "data/tidy_finance.sqlite",
  extended_types = TRUE
)
```

Next, we create a remote table with the monthly Fama-French factor data. We do so with the function `dbWriteTable()`, which copies the data to our SQLite-database.

```
  dbWriteTable(tidy_finance,
    "factors_ff_monthly",
    value = factors_ff_monthly,
    overwrite = TRUE
  )
```

We can use the remote table as an in-memory data frame by building a connection via `tbl()`.

```
factors_ff_monthly_db <- tbl(tidy_finance, "factors_ff_monthly")
```

All `dplyr` calls are evaluated lazily, i.e., the data is not in our R session's memory, and the database does most of the work. You can see that by noticing that the output below does not show the number of rows. In fact, the following code chunk only fetches the top 10 rows from the database for printing.

```
factors_ff_monthly_db |>
  select(month, rf)
```

[12] https://cran.r-project.org/web/packages/RSQLite/vignettes/RSQLite.html
[13] https://db.rstudio.com/databases/sqlite/

```
# Source:   SQL [?? x 2]
# Database: sqlite 3.39.3 [data/tidy_finance.sqlite]
  month          rf
  <date>      <dbl>
1 1960-01-01 0.0033
2 1960-02-01 0.0029
3 1960-03-01 0.0035
4 1960-04-01 0.0019
5 1960-05-01 0.0027
# ... with more rows
```

If we want to have the whole table in memory, we need to `collect()` it. You will see that we regularly load the data into the memory in the next chapters.

```
factors_ff_monthly_db |>
  select(month, rf) |>
  collect()
```

```
# A tibble: 744 x 2
  month          rf
  <date>      <dbl>
1 1960-01-01 0.0033
2 1960-02-01 0.0029
3 1960-03-01 0.0035
4 1960-04-01 0.0019
5 1960-05-01 0.0027
# ... with 739 more rows
```

The last couple of code chunks is really all there is to organizing a simple database! You can also share the SQLite database across devices and programming languages.

Before we move on to the next data source, let us also store the other five tables in our new SQLite database.

```
  dbWriteTable(tidy_finance,
    "factors_ff_daily",
    value = factors_ff_daily,
    overwrite = TRUE
  )

  dbWriteTable(tidy_finance,
    "industries_ff_monthly",
    value = industries_ff_monthly,
    overwrite = TRUE
  )
```