Peng Zhao

Working with Data in Public Health: A Practical Pathway with R

西安交通大学出版社
XI'AN JIAOTONG UNIVERSITY PRESS

Springer

Working with Data in Public Health: A Practical Pathway with R

Peng Zhao

Working with Data in Public Health: A Practical Pathway with R

西安交通大学出版社
XI'AN JIAOTONG UNIVERSITY PRESS

Springer

Peng Zhao
Department of Health and Environmental
Sciences
Xi'an Jiaotong-Liverpool University
Suzhou, China

ISBN 978-981-99-0134-0 ISBN 978-981-99-0135-7 (eBook)
https://doi.org/10.1007/978-981-99-0135-7

Jointly published with Xi'an Jiaotong University Press
The print edition is not for sale in China (Mainland). Customers from China (Mainland) please order the print book from: Xi'an Jiaotong University Press.

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Preface

Years of teaching R to undergraduates has struck me with the realization that even if students are well equipped with a lot of skills in R, they may find it difficult to pick out the proper tools when they need them. Conversely, if they conduct research without learning R, they may have no idea what R can achieve for them.

This book is neither a book for the R language, nor for statistics, nor for data science, nor for public health. It is a mixture of them. In this book, I would like to demonstrate how to apply R language to processing data in public health with practical examples. Thus, the preferable usage of this book is a textbook for teaching or learning R language or data science in public health, although it can also be used across disciplines.

Data are a set of values of numerical or categorical variables about one or more persons or objects. In scientific research, we process a lot of data before extending our knowledge and making decisions. In Xi'an Jiaotong-Liverpool University (XJTLU, Suzhou, China), undergraduate programs are carried out for data analysis in multiple disciplines. Most of the contents of this book originated from the undergraduate module DPH206 *Methods for Analyzing Public Health IV: Working in the Field: Data*. The Roman number *IV* indicates that this module is a component within a series for the students who are major in public health. Prerequisites for this module include basic knowledge in *Epidemiology*, *Biostatistics*, and *Qualitative and Quantitative Research Methods*. This book, however, gives examples as simple as possible so that the prerequisites are helpful rather than a must.

Public health is an important and ever-evolving field that requires the collection, analysis, and interpretation of data to inform decisions and policies that impact the health of individuals and communities. The ability to work effectively with data is therefore a crucial skill for any public health practitioner. As a result of the teaching and learning experience of DPH206, this book provides a practical way of conducting and analyzing research data. Students learn principles of working with data through the entire workflow from planning to presenting. For the convenience of the readers,

the R code and demonstration datasets in this book are available in the GitHub repository pzhaonet/dph206.[1] Readers are suggested to download the files in "inst/extdata/" and save them in a local folder "data/", which will be the default path to the data files in the demonstrations in this book. As a result of reading this book, readers will not only gain a solid understanding of the R language and its applications in public health but will also develop the skills necessary to work with data in a rigorous and reproducible manner. I hope that this book will serve as a valuable resource for students and practitioners alike and will inspire readers to continue exploring the rich intersection between data analysis and public health.

It could have been impossible to write this book without the collaboration with my students. I appreciate the contribution from the students during the years of my teaching, especially Hongran Ding, Xinwen Hu, Chenwenyu Li, Guiyu Liang, Longsheng Liu, Shiyu Ma, Jing Wang, and Shiqiang Wu.

Suzhou, China Peng Zhao
December 2022

[1] https://github.com/pzhaonet/dph206.

Contents

List of Figures

List of Tables

Chapter 1
Preparing Tools

1.1 Chapter Highlights

- Set up the R environment with useful packages for working with data in public health,
- The fundamental R operations, including assigning values, writing comments, using functions, loading packages and datasets, and
- Find help in the built-in documents and online resources.

1.2 Tools

Data, which serve as a link between the problem and the solution, describe what is known about the issue and place it in context. A handy tool for working with data plays a key role in program evaluation and inference in public health. Therefore, we must choose a data processing tool before doing anything with data. This stage might be incorporated into the data planning process instead, as tool selection is an element of the study design (see Chap. 2). If we chose it before establishing a plan, however, the tool may play an essential part in the planning process as well.

Data work can be done traditionally in a pen-and-paper mode. Statistical hypothesis testing, for example, might be performed using simple calculation with the assistance of look-up tables. Thanks to the rapid development of modern computer science and technology, it is now possible to select a preferred tool from among a large number of software packages. Microsoft Excel and comparable tools, such as LibreOffice Calc, are frequently used to manage spreadsheet data. In the field of public health, professional data analysis tools such as SPSS, Epi Info, and NVivo are particularly popular with graphic user interfaces (GUI). Aside from GUI-dominated software, programming languages such as MATLAB, Python, and R offer more powerful and versatile alternatives.

P. Zhao, *Working with Data in Public Health: A Practical Pathway with R*,
https://doi.org/10.1007/978-981-99-0135-7_1

This book focuses on R because of the extensive ecosystem of R packages and friendly community for wide support. The open-source project RStudio offers an enhanced integrated development interface (IDE) to R user, which simplifies the usage of R. When students' work is required to be reproducible, R Markdown is the preferred writing tool.

The R language (R Core Team 2022) was created as a software package for statistical analysis and has now evolved into an all-in-one data science solution. R was created by Ross Ihaka and Robert Gentleman in the Department of Statistics, University of Auckland, New Zealand, and has been maintained by a group (the R Core Team) since 1997. The majority of the team members are statisticians from universities such as Oxford University, Washington University, Wisconsin University, Iowa University, and others. Aside from that, many contributors from all over the world work on R by writing scripts, debugging applications, and writing documentation. R is free and open-source, and it is backed by a big community, unlike many other data tools.

R is based on the S programming language and Scheme, so it looks and works like S. R is an explanatory programming language. The commands in R are written as functions, and users can also invoke the programmes written in C, C++, or FORTRAN. R's data structure lends itself well to data science. R can handle all types of numerical and categorical data. Vector, matrix, data frame, list, array, factor, time series, and other data types are among them. In terms of the algorithm, R gives a wealth of functions for data tools that are simple to use. We may also create our own algorithms using the R rules.

R's official webpage provides a comprehensive introduction to the language, as well as seven handbooks that can be also found in the R installation path on our computers. In the appendix of the R-intro manual, there is a demonstration for beginners. Other handbooks among the seven are more suited to programmers than to beginners. Aside from the official R website, a number of learning materials are either accessible online for free or available in published hardcopies, written in English and other languages, such as *R for Beginners* (Paradis 2005). Although it is a bit out of date, *A Beginner's Guide to R* (Zuur et al. 2009) is still a most friendly book for beginners.[1] *R for Data Science* (Wickham and Grolemund 2016), a complete guide for the **tidyverse** community in R, is a modern comprehensive one. The R community contributes additional learning materials on the open forums and video channels.

We presume that the readers of this book have a fundamental understanding of R. That is, readers should be familiar with the basics of R, such as variable assignment, vectors, lists, data frames, and functions. Therefore, the purpose of this chapter is to provide the readers a rapid review of R in order to keep them on track.

[1] All the R codes in this book could be downloaded from https://highstat.com/index.php/a-beginners-guide-to-r. The book has been translated into other languages, including Chinese.

1.3 Set Up R

1.3.1 R

We could download the installation package of R, either for Microsoft Windows, macOS, or Linux, from any mirror site of the Comprehensive R Archive Network (CRAN), where we could find `Download and Install R`. Choose the proper link for our operation system.

- Windows users could click `Download R for Windows` and click `base`. In the new window, click `Download R-\*.\*.\* for Windows`, where `*.*.*` indicates the version, such as `4.2.2`. After the download is complete, run the installation file. There are various options throughout the installation process, and we may just use the default settings.
- Linux users could click `Download R for Linux (Debian, Fedora/Redhat, Ubuntu)`, and then choose the proper link for RedHat, Ubuntu, etc. We could either install R from the binary package or from the source code. The latter might require us to solve the dependencies.
- Mac users could click `Download R for macOS` and choose the `R-*.*.*.pkg` file for installation.

We have to keep in mind that R is in development. Usually, a new version of R is released on a seasonal basis. It is recommended to update R to the newest version.

1.3.2 RStudio

R comes with a minimal GUI by default. Although it works great, we recommend using the RStudio IDE to make R more user-friendly, especially for beginners. The RStudio IDE is open-source and cross-platform. We could just download the installation package from RStudio's official website and run it with the default settings.

After the installation of R and RStudio IDE, We can begin working with data. It is suggested to create an R project as the start. The steps are as follows:

1. Start RStudio IDE from the program list on our computer.
2. Click in the menu bar `File - New Project - New Directory - New project`.
3. Fill in the blanks with the project name as the new directory name and the path for saving the project.
4. Click the button `Create Project`.

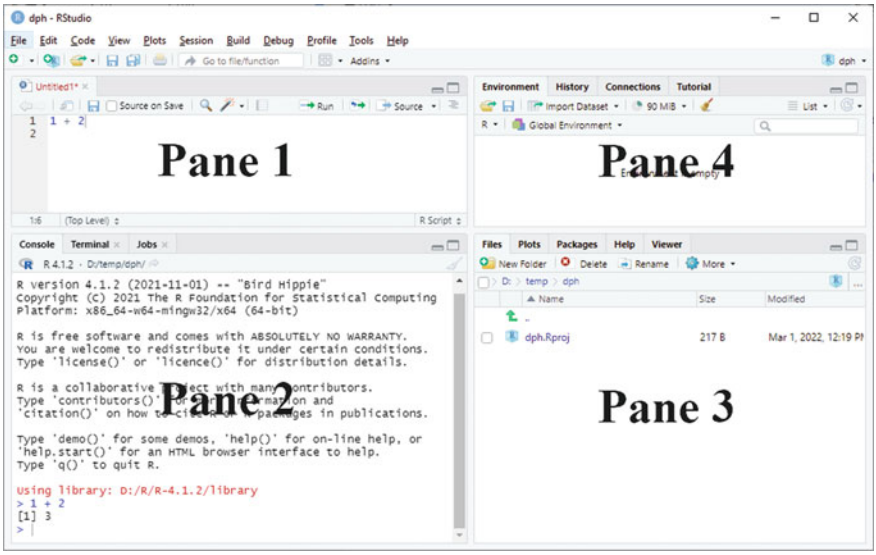


Fig. 1.1 The RStudio IDE with four panes

The new directory with supportive project files are created and the RStudio IDE interface is switched to the new project. We could find a `*.Rproj` file in the directory, where `*` is the project name.

Now we click in the menu bar `File - New File - R Script`, or press the hot key `ctrl + shift + n`, to create a script file. The RStudio IDE with four panes looks like Fig. 1.1.

The four panes are identified with 1–4 in an anti-clockwise order, and a hot key `ctrl+shift+pane_ID` (e.g. `ctrl+shift+1`) can zoom in or zoom out one of them. By default, Pane 1 is the source pane, which serves as the working desk where we write the R code. The R code is a sequence of programming statements that can be saved (`File - Save` or `ctrl+s`) and edited as a script file. We can write something like `1 + 2`, and click the `Run` button at the top bar of Pane 1 or press the hot key `ctrl+enter`. As a result, the command where the cursor stays is sent to and evaluated in Pane 2, which is called the console pane. We can also use Pane 2 for performing quick calculations that we do not have to save for future use. Pane 3 shows the file directories, plots, packages, and help documents. Pane 4 is mainly used for viewing the objects in the environment and command history. The layout of these four panes can be customized in the menu `Tools - Global Options - Pane Layout`.

We recommend just double-clicking the `*.Rproj` file to launch R and the RStudio IDE every time. The benefit is that the working directory is straightforward to manage. Traditionally, R users had to use the `getwd()` and `setwd()` functions to get and set the working directory, respectively. Beginners often overlooked this and became confused while importing or exporting files. Although these two functions are still mentioned in many R manuals, we nowadays seldom use them any more, because (1) the `*.Rproj` file simplifies the usage of working directory, and (2) changing the working directory while processing data is not a smart idea unless you have to. Therefore, it is recommended to group R scripts and products into R projects, keep relevant files in the project folder, and always start with the `*.Rproj` file.

The RStudio IDE provides a number of hot keys for accelerating the operation. The hot key `alt+shift+k` evokes a quick reference for keyboard shortcuts.

1.4 Programming Basics

Programming basics in R includes arithmetic and logical operations, assigning values, data types, using functions, etc. Here we show the most fundamental operations in the following simple example:

```
radius <- c(1, 2, 3, 4, 5) # assign values
area <- pi * x ^ 2 # pi is a built-in constant
plot(x = radius, y = area) # draw a figure
```

Explanations:

- This example calculates the areas of circles and visualize the relationship between the radius and the area.
- There are three lines in this example. Each line is a complete operation. Line 1 assigns a series of the values to an object called `radius`. Line 2 calculates the areas of circles with the radius, and assigns the results to an object called `area`. Line 3 draws a scatter plot of `area` as *y* and `radius` as *x*.
- The assignment operation is performed by an arrow (<-). We could use = instead, which is nothing different for the assignment operation.
- `c()` is a command or, more precisely, a function that combines multiple values in the parenthesis into one vector. The values are separated with commas, as required by `c()`. A function is a set of statements organized together to accomplish a specific task. All the R operations are performed or based on functions. Line 3 shows another function `plot()`, in which the objects `radius` and `area` are passed to the arguments `x` and `y`. Functions are a key feature in R.
- The text led by a hash (`#`) is a comment, which is ignored by R when executing the code. Comments can be any text, but usually they are used to explain the code and make it more readable to users.

These explanations may help you go through this book. R beginners could read the reference books we recommend for step-by-step learning, while it is also a good practice to learn straightforward from the scripts written by others.

1.5 R Packages

An R package refers to the fundamental unit of sharable codes, data, documentation and tests (Wickham 2015). R packages considerably enhance R's ability to work with data. The default installation of R automatically includes the **base**, **stats**, **graphics**, **grDevices**, and **datasets** packages, which provides many statistical models (e.g. linear and generalized linear models, non-linear simulation models, time series analysis, hypothesis tests), supports for graphics, and demonstration datasets. Other add-on packages must be installed manually when they are needed.

There are around 20,000 R add-on packages available on CRAN so far, and even more on Bioconductor, R-Forge, Omegahat, and GitHub for obtaining and sharing useful tools in R. CRAN provides a webpage of task views[2] as a guideline for finding packages in a given topic, where we could find helpful packages in the categories *Epidemiology* and *Survival* for public health. Most of the R packages are under 1 MB in size. R packages implement a large number of new statistical theories and methods as soon as they are produced, which makes R the industry leader in statistics and data science.

1.5.1 Installation

An add-on package has to be installed when we use it for the first time. The installation of R packages from CRAN is performed via the `install_packages()` function. For example, we could install the **remotes** package (Csárdi et al. 2021) like this:

```
install.packages('remotes')
```

The **remotes** package can install add-on packages from remote repositories rather than CRAN, which we will demonstrate in subsequent sections.

Multiple packages could be installed in one command. Let's install the **Epi** package (Carstensen and Plummer 2022) for research in Epidemiology, as well as the **data.table** (Dowle and Srinivasan, 2022) and the **tidyverse** packages (Wickham 2022) for data science:

```
install.packages(c('Epi', 'data.table', 'tidyverse'))
```

Every time when a new R session is started, we must load the add-on package into R before using it. There are often two ways of loading an add-on package. The

[2] https://cran.r-project.org/web/views/.

first is the `library()` function or the `require()` function, which loads all the functions of a given package. Let's load the **remotes** package:

```
library(remotes)
```

Now we can use any function provided by the **remotes** package. For example, **remotes** provides a function `install_github()`, which allows installing R packages on GitHub. Let's install the `rosr` package (Zhao 2021) in the GitHub repository `pzhaonet/rosr`, which provides a data management strategy (see Chap. 10):

```
install_github('pzhaonet/rosr')
```

The second way of loading the add-on package is the syntax `package_name::function_name()`, which only loads one single function of a package. The previous task can also be performed as:

```
remotes::install_github('pzhaonet/rosr')
```

This book uses many R packages for working with data in public health. We do not go over the installation and loading procedure again in the subsequent sections, but you should be aware of how to perform it. We will clarify a package on GitHub like `Github: package_repository`. If no clarification is given, it means that the package is available on CRAN.

1.5.2 Packages in Public Health

Although public health is not a topic listed in the CRAN task views, numerous R packages are often used in this field, and they are included in other general topics such as *Databases*, *ClinicalTrials*, *Epidemiology*, *ExperimentalDesign*, *Multivariate*, *SocialSciences*, *Survival*, etc. Beginners might take a quick glance at these pages to get a sense of what they can do. Here we list several commonly used packages in public health as follows.

- **Epi** (Carstensen and Plummer 2022) provides statistical analysis in epidemiology. Specifically, it offers a set of functions tailored towards demographic and epidemiological analysis in the Lexis diagram, i.e. register and cohort follow-up data, in particular representation, manipulation and simulation of multistate data. This package also contains functions for Age-Period-Cohort and Lee-Carter modeling, as well as a function for interval censored data. Additionally, it offers several useful features for data tabulation and visualization, and ships a diverse range of epidemiological datasets.
- **EpiModel** (Jenness et al. 2022) employs mathematical modeling techniques to simulate the dynamics of infectious diseases. Specifically, it enables the simulation of diverse models, including deterministic compartmental models, stochastic individual-contact models, and stochastic network models. These modeling

approaches allow for a comprehensive understanding of infectious disease dynamics, and their simulation can facilitate the development of effective intervention strategies to mitigate disease spread.

- **epibasix** (Rotondi 2018) provides elementary epidemiological functions for epidemiology and biostatistics.
- **epiDisplay** (Chongsuvivatwong 2022) explores epidemiological data and presents results.
- **epiR** (Stevenson et al. 2022) provides analysis tools for epidemiological and surveillance data. It includes functions for calculating sample sizes for cross-sectional, case-control, and cohort studies as well as functions for directly and indirectly adjusting measures of disease frequency, quantifying measures of association using one or more strata of count data presented in a contingency table, computing confidence intervals around incidence risk and incidence rate estimates, and adjusting measures of disease frequency. It offers surveillance tools to estimate surveillance system sensitivity, estimate the proper sample size for 1- and 2-stage representative freedom surveys, and facilitate scenario tree modeling analyses.
- **epitools** (Aragon 2020) provides tools, including methods for two-way and multi-way contingency tables, for training and practicing epidemiologists.
- **nCov2019** (Yu and Wu 2021) provides the real world COVID-19 outbreak data, which could be served as the learning materials for public health students and decision evidences for policy makers.
- **pubh** (Athens 2022) provides a variety of functions to describe and analyze epidemiological public health data and biostatistics science in a user-friendly way.
- **survival** (Therneau 2022) contains the essential survival analysis procedures, including definition of Surv objects, Kaplan-Meier and Aalen-Johansen (multi-state) curves, Cox models, and parametric accelerated failure time models.
- **survey** (Lumley 2021) analyzes complex survey samples, calculates summary statistics, and performs two-sample tests, rank tests, generalised linear models, cumulative link models, Cox models, loglinear models, and general maximum pseudolikelihood estimation for multistage stratified, cluster-sampled, unequally weighted survey samples. A variety of tools for variances by Taylor series linearisation or replicate weights, post-stratification, calibration, and raking, two-phase subsampling designs, and so on, are available as well.
- **surveillance** (Hoehle et al. 2022) provides statistical approaches for the modeling of continuous-time point processes of epidemic phenomena as well as for the monitoring and monitoring of time series of counts, proportions, and categorical data.
- **SPARSEMODr** (Mihaljevic 2022) uses stochastic disease models that are spatially explicit and have adjustable time windows to show how parameter values change when an outbreak occurs.
- **WHO** (Persson 2019) provides programmatic access to the World Health Organization API.

Some of these packages (e.g. **Epi**, **epiR**) contain extensive capabilities for data analysis in public health, while others focus on specific tasks (e.g. **nCov2019**, **WHO**). Almost all of them provide some demonstration datasets with examples that may be utilised to practise and learn R.

An appropriate citation is typically required when using an R package in publications. The `citation()` function provides the bibliographic information of the package. The returned text are usually given in two different formats. When there is only one reference, it returns the reference ready for general use in publications and a bibliographic entry in the BibTeX format:

```
citation('pubh')
```

```
##
## To cite package 'pubh' in publications use:
##
##   Athens J (2022). _pubh: A Toolbox for Public Health and
##   Epidemiology_. R package version 1.2.7,
##   <https://CRAN.R-project.org/package=pubh>.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {pubh: A Toolbox for Public Health and Epidemiology},
##     author = {Josie Athens},
##     year = {2022},
##     note = {R package version 1.2.7},
##     url = {https://CRAN.R-project.org/package=pubh},
##   }
```

The BibTeX entry could either be integrated into the R Markdown tools for academic writing (see Chap. 9), or be imported to any reference management software, such as EndNote and Zotero.

When there are multiple references, it returns a list of the references. For example, the citations for the **Epi** package:

```
citation('Epi')
```

```
##
## To cite Epi in publications use:
##
##   Bendix Carstensen, Martyn Plummer, Esa Laara, Michael Hills (2022).
##   Epi: A Package for Statistical Analysis in Epidemiology. R package
##   version 2.47. URL \url{https://CRAN.R-project.org/package=Epi}
##
## If you use Lexis objects/diagrams, please also cite:
##
##   Martyn Plummer, Bendix Carstensen (2011). Lexis: An R Class for
##   Epidemiological Studies with Long-Term Follow-Up. Journal of
##   Statistical Software, 38(5), 1-12. URL
##   \url{https://www.jstatsoft.org/v38/i05/.}
##
## For use of Lexis objects in multi-state models, please also cite:
##
##   Bendix Carstensen, Martyn Plummer (2011). Using Lexis Objects for
##   Multi-State Models in R. Journal of Statistical Software, 38(6),
##   1-18. URL \url{https://www.jstatsoft.org/v38/i06/.}
##
## To see these entries in BibTeX format, use 'print(<citation>,
## bibtex=TRUE)', 'toBibtex(.)', or set
## 'options(citation.bibtex.max=999)'.
```

includes multiple recommendations for citing the resources about the package itself, the utilization of Lexis objects/diagrams, and, more specifically, the use of Lexis objects in multi-state models. These bibliographic entries could be exported as BibTeX format with the `boBibtex()` function:

```
toBibtex(citation('Epi'))
```

```
## @Manual{,
##   title = {{Epi}: A Package for Statistical Analysis in Epidemiology},
##   author = {Bendix Carstensen and Martyn Plummer and Esa Laara and Michael Hills},
##   year = {2022},
##   note = {R package version 2.47},
##   url = {https://CRAN.R-project.org/package=Epi},
## }
##
## @Article{,
```

```
##   title = {{Lexis}: An {R} Class for Epidemiological Studies with Long-Term Follow-Up},
##   author = {Martyn Plummer and Bendix Carstensen},
##   journal = {Journal of Statistical Software},
##   year = {2011},
##   volume = {38},
##   number = {5},
##   pages = {1--12},
##   url = {https://www.jstatsoft.org/v38/i05/},
## }
##
## @Article{,
##   title = {Using {Lexis} Objects for Multi-State Models in {R}},
##   author = {Bendix Carstensen and Martyn Plummer},
##   journal = {Journal of Statistical Software},
##   year = {2011},
##   volume = {38},
##   number = {6},
##   pages = {1--18},
##   url = {https://www.jstatsoft.org/v38/i06/},
## }
```

The `print()` function returns both of the reference text and the BibTeX entries:

```
print(citation('Epi'), bibtex = TRUE)
```

If we would like to know how popular a package is, the `cranlogs::cran_downloads()` function gives the daily download times of a package within a given period:

```
epi_downloads <- cranlogs::cran_downloads('Epi')
epi_downloads
```

```
##         date count package
## 1 2022-12-11   173     Epi
```

It shows that the **Epi** package was download 173 times from CRAN on 2022–12-11.

The following code plots the daily downloads of the previously listed packages in the recent one year (see Fig. 1.2):

```
library(ggplot2)
library(dplyr)
cranlogs_cum <- function(x) {
  cranlogs::cran_downloads(
    x,
    from = Sys.Date() - 365,
    to = Sys.Date()) |>
    transform(CumDownload = cumsum(count))
}
c('Epi', 'EpiModel', 'epibasix', 'epiDisplay', 'epiR',
  'epitools', 'nCov2019', 'pubh', 'survival', 'survey',
  'surveillance', 'SPARSEMODr', 'WHO') |>
  lapply(cranlogs_cum) |>
  bind_rows() |>
  ggplot() +
  geom_line(aes(date, CumDownload)) +
  theme(axis.text.x = element_text(angle = 90)) +
  scale_y_continuous(labels = function(x) format(x,
  scientific = TRUE)) +
  facet_wrap(~package, scales = 'free_y')
```

As the most popular one among these packages, **survival** gains more than 1.5 million downloads, followed by **survey** (> 800 thousands) and **epiR** (>150 thousands).

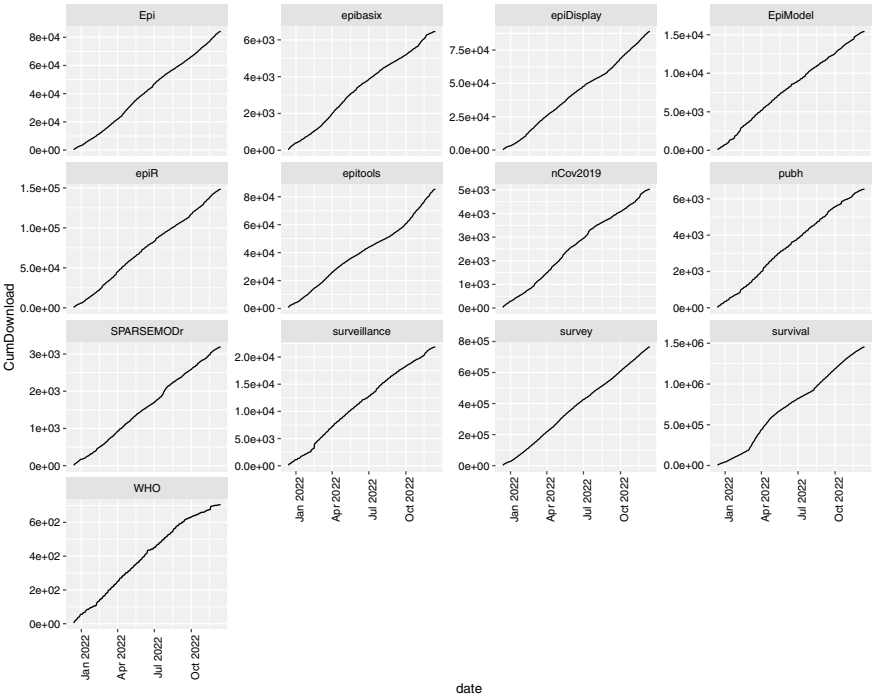


Fig. 1.2 Cumulative downloads of R packages on CRAN in the recent year

1.5.3 Datasets in Packages

The `data()` function can list the datasets of the packages that have been loaded:

```
data()
```

or the datasets of a given package. For example, the **Epi** package ships more than 30 datasets in epidemiology, such as birth data in Denmark, diet and heart data, thorotrast study data, etc. The `data()` function lists all the available datasets in a given installed package:

```
data(package = 'Epi')
```

The list of the shipped dataset is displayed in the top-left pane (Pane 1) of the RStudio IDE.

More frequently, we use this function for loading a dataset shipped by a certain package. For example, we can load the dataset *diet* in the **Epi** package into the R session:

```
data(diet, package = 'Epi')
```

With the `help()` function, we can find the meta data for a dataset, including the introduction to the dataset, the formats and units, as well as the data source. The following code shows how to find the meta data of the dataset *diet* in the **Epi** package:

```
help(diet, package = "Epi")
```

The default packages and their datasets are loaded automatically when R starts up. As a result, we can utilize any function or dataset in them without having to manually load them.

After the dataset is loaded, it is ready for use. Let's print the first 6 rows of *diet*:

```
head(diet) # print the first 6 rows
```

```
##    id        doe        dox        dob          y fail         job month
## 1 102 1976-01-17 1986-12-02 1939-03-02 10.8747433    0      Driver     1
## 2  59 1973-07-16 1982-07-05 1912-07-05  8.9691992    0      Driver     7
## 3 126 1970-03-17 1984-03-20 1919-12-24 14.0095825   13   Conductor     3
## 4  16 1969-05-16 1969-12-31 1906-09-17  0.6269678    3      Driver     5
## 5 247 1968-03-16 1979-06-25 1918-07-10 11.2744695   13 Bank worker     3
## 6 272 1969-03-16 1973-12-13 1920-03-06  4.7446954    3 Bank worker     3
##    energy  height   weight     fat    fibre   energy.grp chd
## 1 22.8601 181.610 88.17984  9.168 1.4000000 <=2750 KCals   0
## 2 23.8841 165.989 58.74120  9.651 0.9350001 <=2750 KCals   0
## 3 24.9537 152.400 49.89600 11.249 1.2480000 <=2750 KCals   1
## 4 22.2383 171.196 89.40456  7.578 1.5570000 <=2750 KCals   1
## 5 18.5402 177.800 97.07040  9.147 0.9910000 <=2750 KCals   1
## 6 20.3073 175.260 61.00920  8.536 0.7650000 <=2750 KCals   1
```

The list of the shipped dataset can also be assigned to an object for future use. For example,

```
ls_data <- data(package = c('Epi', 'pubh', 'NHANES', 'survey', 'boot'))
```

The names and brief descriptions of the datasets can be found in the matrix called `results` in `ls_data`, which we could view with the `View()` function.

```
View(ls_data$results)
```

From the third column of this matrix we can get all the dataset names:

```
ls_data$results[, 3]
```

```
##   [1] "B.dk"                          "BrCa"
##   [3] "DMconv"                        "DMepi"
##   [5] "DMlate"                        "DMrand"
##   [7] "M.dk"                          "N.dk"
##   [9] "S.typh"                        "Y.dk"
##  [11] "bdendo"                        "bdendo11"
##  [13] "births"                        "blcaIT"
##  [15] "brv"                           "diet"
##  [17] "ewrates"                       "gmortDK"
##  [19] "hivDK"                         "lep"
##  [21] "lungDK"                        "mortDK"
##  [23] "nickel"                        "occup"
##  [25] "pr"                            "st2alb"
##  [27] "st2clin"                       "steno2"
##  [29] "testisDK"                      "thoro"
##  [31] "Bernard"                       "Brenner"
##  [33] "Fentress"                      "Hodgkin"
##  [35] "Kirkwood"                      "Macmahon"
##  [37] "Oncho"                         "Roberts"
##  [39] "Rothman"                       "Sandler"
##  [41] "Sharples"                      "Thall"
##  [43] "Tuzson"                        "Vanderpump"
##  [45] "NHANES"                        "NHANESraw"
##  [47] "apiclus1 (api)"                "apiclus2 (api)"
##  [49] "apipop (api)"                  "apisrs (api)"
##  [51] "apistrat (api)"                "crowd"
##  [53] "election"                      "election_insample (election)"
##  [55] "election_jointHR (election)"   "election_jointprob (election)"
##  [57] "election_pps (election)"       "fpc"
##  [59] "hospital"                      "mu284"
##  [61] "nhanes"                        "scd"
##  [63] "yrbs"                          "acme"
```

Table 1.1 Datasets used in this book

Package	Dataset	Description
Boot	*melanoma*	Survival from Malignant Melanoma
Epi	*diet*	Diet and heart data
Epi	*DMepi*	Epidemiological rates for diabetes in Denmark 1996–2015
NHANES	*NHANES*	Body Shape and related measurements from the US National Health and Nutrition Examination Survey (NHANES) with adjusted weighting
pubh	*Oncho*	Onchocerciasis in Sierra Leone
survey	*api*	Student performance in California schools

```
##  [65] "aids"                "aircondit"
##  [67] "aircondit7"          "amis"
##  [69] "aml"                 "beaver"
##  [71] "bigcity"             "brambles"
##  [73] "breslow"             "calcium"
##  [75] "cane"                "capability"
##  [77] "catsM"               "cav"
##  [79] "cd4"                 "cd4.nested"
##  [81] "channing"            "city"
##  [83] "claridge"            "cloth"
##  [85] "co.transfer"         "coal"
##  [87] "darwin"              "dogs"
##  [89] "downs.bc"            "ducks"
##  [91] "fir"                 "frets"
##  [93] "grav"                "gravity"
##  [95] "hirose"              "islay"
##  [97] "manaus"              "melanoma"
##  [99] "motor"               "neuro"
## [101] "nitrofen"            "nodal"
## [103] "nuclear"             "paulsen"
## [105] "poisons"             "polar"
## [107] "remission"           "salinity"
## [109] "survival"            "tau"
## [111] "tuna"                "urine"
## [113] "wool"
```

Table 1.1 list the datasets which are used as examples in this book. The original goal of shipping data in R packages is to demonstrate how to use R functions. This philosophy may be extended to organizing and sharing data: We could store our own data in self-developed packages and load them and check their documentations with the R help system (see Chap. 10).

1.6 R Help

1.6.1 Built-in Documents

R provides a comprehensive and user-friendly help system for R packages and functions, which is a valuable resource for R users. We could see the help document for a package with the `help()` function:

```
help(package = "Epi")
```

which shows an overview with a brief description that the package **Epi** is used for *Statistical Analysis in Epidemiology*, links to the *DESCRIPTION file*, *User guides*, *package vignettes and other documentation*, and a list of functions in it.

Aside from providing an overview of a package, the `help()` function is more frequently used for checking the help document of a function or a dataset with the syntax of `help("function_name")` and `help("dataset_name")`, respectively.

```
help("install_github", package = "remotes")
help(DMepi, package = 'Epi')
```

The `Help` tab in the bottom-right pane (Pane 3) of RStudio displays the help document, including `Description`, `Usage`, `Examples`, and returned `Value` for functions and `Format` for datasets. The examples are the most helpful: all we have to do is copy and paste them into the console, and the results will be reproduced.

If the packages have been already loaded by `library()`, then we can skip the argument `package` in the `help()` function, or use `?` as a more convenient approach:

```
library(remotes)
help("install_github")
?install_github

library(Epi)
help(DMepi)
?DMepi
```

The RStudio IDE provides a shortcut for opening the help document: put the cursor anywhere within a function name in the topleft or the bottomleft pane, then press `F1` on the keyboard.

If we do not exactly know the function name, we can search all the help documents with some keywords by using the `help.search()` function or `??`. Suppose we forget the name of the R function for ANOVA. We could search the keywords "analysis of variance":

```
help.search("analysis of variance")
# or
??"analysis of variance"
```

We can find all the relevant functions about analysis of variance, including the function `aov()`. We may then utilise the methods introduced previously to learn more about them.

The `help.start()` function launches a webpage where you may browse manuals, references, and other resources. If we still can't locate what we're looking for, the best option is to use search engines or go to online forums like StackOverflow, where most beginner-level inquiries have previously been addressed.

1.6.2 Examples and Demonstrations

Most of R functions provide reproducible examples, while some package provide friendly demonstrations, to help users understand or learn directly how to use the functions or packages.

Examples can be evoked by the `example()` function. For instance, we could see how to use the `stat.table()` function of the **Epi** package:

```
example(stat.table, package = 'Epi')
```

Example code is sent to the console, where we can see the R commands are executed line by line and the results are displayed as well. The main argument of `example()` is the function name and the package name. If the package has been already loaded, then the argument `package` could be skipped.

The `demo()` function evokes the demonstrations, which are usually large groups and combinations for multiple functions that show the feature of a package. The usage of `demo()` is similar to the `data()` function. It shows the demonstrations provided by the packages loaded to the environment, when we leave the arguments as blank:

```
demo()
```

The list of the demos is displayed in the top-left pane (Pane 1) of the RStudio IDE.

The `demo()` function can show available demonstrations in one or more packages. For example, the following code returns the demonstrations provided by **epiDisplay** and **surveillance**.

```
demo(package = c('epiDisplay', 'surveillance'))
```

The **surveillance** package, for example, provides a demo named "biosurvbook," which is characterised as "Code from the book chapter on Danish mortality monitoring (Hoehle and Mazick, 2010)." We could use `demo()` to run this demo:

```
demo(biosurvbook, package = 'surveillance')
```

Follow the instruction shown in the console, and we see the commands with returned results and graphs, where we could learn how to use **surveillance**.

We could list all the available demonstrations in R with the help of the `.packages(all.available = TRUE)` function which returns the names of all the installed R packages:

```
demo(package = .packages(all.available = TRUE))
```

1.6.3 Asking Questions

Sometimes we are unlucky to find out that none of the approaches introduced in the previous sections is helpful and have to turn to colleagues, acquaintances, or a Q&A community for help. A face-to-face question is not difficult to ask, because our listeners communicate with us instantly and interactively, and we get immediate response and can rephrase our trouble till our listeners understand it. However, it is different to ask a question on the internet or to someone far away. We won't get immediate feedback. It takes time for an answer to appear.

Suppose we send an email to friends or post a question on a forum for help. In the first round, the readers will ask us what we exactly mean if our question is not clear enough. If we still don't ask the question properly in the second round, the readers probably loses patience and our question is never answered. We could attempt for a third round, but in most circumstances in the real world, no one listens any more. Some questions stay unanswered on forums for years without a single reply, and the person who asked them still has no idea why.

The golder rule of asking a question is simple: make yourself understandable to the audience. Your goal is to eliminate troubles rather than create more. Those who wish to assist you have time for you, but not much. Be kind to them. We suggest three simple rules (abbreviated as TES) for asking questions.

1. Express a clear **Title (T)**

A good title for the email or the post is what expresses the question in one sentence so as to save the reader's time. The readers may not even need to open the post if you craft a decent title, from which they understand what you're talking about. "A question," "Help," or "I don't understand this" are all awful titles. There is no valuable information in these titles. An appropriate title could be "Failed when replacing missing values with NA", or "the Epi package cannot be installed."

2. Provide a minimal reproducible **Example (E)**.

If you are having trouble with running R code, please give a minimal reproducible example (MRE). The term "reproducible" means that the readers may copy and paste your scripts into their console and experience the identical issues you did. Don't expect your audience to be experts in all topics, even if they are advanced R users. They have to, in most situations, test the R code. It takes time if your example is so incomplete that they have to infer what you mean. They are ANSWERING your questions rather than GUESSING them. Don't challenge their patience.

The term "minimal" means that your example should be as simple as possible. If your problem can be reproduced with a two-column data frame, don't use a three-column one. It absolutely helps others focus on your real issue.

Making a MRE is difficult for R beginners, but it is worthwhile. It aids in organizing your thoughts as well as brain training. If you overlook it and give a bad example, you'll end up wasting more time with your audience later. You will eventually lose those who are supposed to help you.

Sometimes you have to provide data in a MRE. You may either upload your data to a server (such as GitHub) and offer a link, or you can show the data using the `dput()` function. Another useful function is `reprex::reprex()`. Read its help document for details.

3. Provide your **Session information (S)**

Your MRE may not be able to reproduce your problem on other people's computers, most likely due to differences in your environment, such as R version, package version, operating system, and so on. It is a good practice to provide the session details.

The function of session information in R is `sessionInfo()`. Here is the session information of the computer which is used for writing this book:

```
sessionInfo()
```

```
## R version 4.2.2 (2022-10-31 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19044)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United Kingdom.utf8
## [2] LC_CTYPE=English_United Kingdom.utf8
## [3] LC_MONETARY=English_United Kingdom.utf8
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United Kingdom.utf8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] bookdown_0.30   digest_0.6.30   lifecycle_1.0.3 magrittr_2.0.3
##  [5] evaluate_0.18   rlang_1.0.6     stringi_1.7.8   cli_3.4.1
##  [9] rstudioapi_0.14 vctrs_0.5.1     rmarkdown_2.18  tools_4.2.2
## [13] stringr_1.5.0   glue_1.6.2      xfun_0.34       yaml_2.3.6
## [17] fastmap_1.1.0   compiler_4.2.2  htmltools_0.5.3 knitr_1.41
```

An advanced function, `devtools::session_info()`, gives more information in a more friendly way:

```
devtools::session_info()
```

```
## - Session info ---------------------------------------
##  setting  value
##  version  R version 4.2.2 (2022-10-31 ucrt)
##  os       Windows 10 x64 (build 19044)
##  system   x86_64, mingw32
##  ui       RTerm
##  language (EN)
##  collate  English_United Kingdom.utf8
##  ctype    English_United Kingdom.utf8
##  tz       Asia/Taipei
##  date     2022-12-13
##  pandoc   2.19.2 @ D:/R/RStudio/bin/quarto/bin/tools/ (via rmarkdown)
##
## - Packages -------------------------------------------
##  package     * version date (UTC) lib source
##  bookdown      0.30    2022-11-09 [1] CRAN (R 4.2.2)
##  cachem        1.0.6   2021-08-19 [1] CRAN (R 4.2.1)
##  callr         3.7.3   2022-11-02 [1] CRAN (R 4.2.2)
##  cli           3.4.1   2022-09-23 [1] CRAN (R 4.2.1)
##  crayon        1.5.2   2022-09-29 [1] CRAN (R 4.2.1)
##  devtools      2.4.5   2022-10-11 [1] CRAN (R 4.2.1)
##  digest        0.6.30  2022-10-18 [1] CRAN (R 4.2.1)
##  ellipsis      0.3.2   2021-04-29 [1] CRAN (R 4.2.1)
##  evaluate      0.18    2022-11-07 [1] CRAN (R 4.2.2)
```

```
##  fastmap       1.1.0   2021-01-25 [1] CRAN (R 4.2.1)
##  fs            1.5.2   2021-12-08 [1] CRAN (R 4.2.1)
##  glue          1.6.2   2022-02-24 [1] CRAN (R 4.2.1)
##  htmltools     0.5.3   2022-07-18 [1] CRAN (R 4.2.1)
##  htmlwidgets   1.5.4   2021-09-08 [1] CRAN (R 4.2.1)
##  httpuv        1.6.6   2022-09-08 [1] CRAN (R 4.2.1)
##  knitr         1.41    2022-11-18 [1] CRAN (R 4.2.2)
##  later         1.3.0   2021-08-18 [1] CRAN (R 4.2.1)
##  lifecycle     1.0.3   2022-10-07 [1] CRAN (R 4.2.1)
##  magrittr      2.0.3   2022-03-30 [1] CRAN (R 4.2.1)
##  memoise       2.0.1   2021-11-26 [1] CRAN (R 4.2.1)
##  mime          0.12    2021-09-28 [1] CRAN (R 4.2.0)
##  miniUI        0.1.1.1 2018-05-18 [1] CRAN (R 4.2.1)
##  pkgbuild      1.4.0   2022-11-27 [1] CRAN (R 4.2.2)
##  pkgload       1.3.2   2022-11-16 [1] CRAN (R 4.2.2)
##  prettyunits   1.1.1   2020-01-24 [1] CRAN (R 4.2.1)
##  processx      3.8.0   2022-10-26 [1] CRAN (R 4.2.1)
##  profvis       0.3.7   2020-11-02 [1] CRAN (R 4.2.1)
##  promises      1.2.0.1 2021-02-11 [1] CRAN (R 4.2.1)
##  ps            1.7.2   2022-10-26 [1] CRAN (R 4.2.1)
##  purrr         0.3.5   2022-10-06 [1] CRAN (R 4.2.1)
##  R6            2.5.1   2021-08-19 [1] CRAN (R 4.2.1)
##  Rcpp          1.0.9   2022-07-08 [1] CRAN (R 4.2.1)
##  remotes       2.4.2   2021-11-30 [1] CRAN (R 4.2.0)
##  rlang         1.0.6   2022-09-24 [1] CRAN (R 4.2.1)
##  rmarkdown     2.18    2022-11-09 [1] CRAN (R 4.2.2)
##  rstudioapi    0.14    2022-08-22 [1] CRAN (R 4.2.1)
##  sessioninfo   1.2.2   2021-12-06 [1] CRAN (R 4.2.1)
##  shiny         1.7.3   2022-10-25 [1] CRAN (R 4.2.1)
##  stringi       1.7.8   2022-07-11 [1] CRAN (R 4.2.1)
##  stringr       1.5.0   2022-12-02 [1] CRAN (R 4.2.2)
##  urlchecker    1.0.1   2021-11-30 [1] CRAN (R 4.2.1)
##  usethis       2.1.6   2022-05-25 [1] CRAN (R 4.2.1)
##  vctrs         0.5.1   2022-11-16 [1] CRAN (R 4.2.2)
##  xfun          0.34    2022-10-18 [1] CRAN (R 4.2.1)
##  xtable        1.8-4   2019-04-21 [1] CRAN (R 4.2.1)
##  yaml          2.3.6   2022-10-18 [1] CRAN (R 4.2.1)
##
##  [1] D:/R/library
##  [2] D:/R/R-4.2.2/library
##
## ------------------------------------------------------
```

Before you ask a question, remember the TES rules, which apply to both online and face-to-face situations. Asking smart questions will make your life and the lives of others much simpler and happier. It may be tough for beginners to come up with a decent question at first. Continue to practise until you've figured out how to avoid stupid questions. No book can cover everything about working with data, and one of the best instructors for learning it is asking smart questions.

1.7 Base R, data.table, and tidyverse

Many human languages have dialects. So does R. Base R is the default dialect when we open up R for the first time. Apart from base R, the most popular dialects are **data.table** (Dowle and Srinivasan 2022) and **tidyverse** (Wickham 2022) for data work. For a long time, base R was the only dialect available in R language. The **data.table** and **tidyverse** packages, on the other hand, have been developed in recent years. As all roads lead to Rome, these various approaches with quite different features lead to the same goal in data work with R. Here we use a simple example to show how they work.

For instance, we would like to pick out the subjects with the maximum dietary fibre intake in each job group in the *diet* dataset, and add one more column as `fibre/fat`. Let's do this task with the base R functions. As there is no existing function in base R that can do this job, we have to combine multiple steps by ourselves:

```
# 1. Create the index for the subjects:
index <- 1:nrow(diet)
# 2. Create a new variable 'fibre-fat':
diet$fibre_fat <- paste0(diet$fibre, '/', diet$fat)
# 3. Find which is the maximum dietary fibre in take in each group:
tb <- tapply(diet$fibre, diet$job, which.max)
# 4. Find the row number of the maximum dietary fibre in take in each group:
rowi <- sapply(names(tb),
               function(x) index[diet$job == x][tb[names(tb) == x]])
# 5. Display the result:
diet[rowi, c('job', 'fibre_fat')]
```

```
##            job    fibre_fat
## 257     Driver 4.595/16.658
## 298   Conductor 3.352/14.173
## 305 Bank worker 5.351/14.386
```

The comments in the code explain what each step does. More explanations are as follows:

1. The index created in Step 1 is used for locating the target rows in Step 4.
2. The `paste0()` creates a new variable *fibre-fat* with our desired format.
3. The `tapply()` function applies the `which.max()` function to each category of the qualitative variable `fibre`, and returns the location of the maximum value in each group.
4. The `sapply()` function applies a self-defined function, which locates the categorical maximum value in the entire data frame, to each job.

The key step (Step 4) is to find the rows in which the grouped maximum dietary fibre intake values are located. Alternative ways with base R functions might exist, but they can hardly be much simpler. Here is another way:

```
# 1. Create a new variable 'fibre-fat':
diet$fibre_fat <- paste0(diet$fibre, '/', diet$fat)
# 2. Define a function for finding the maximum fat in 'job-fat'
find_max <-  function(x)
  x[which.max(data.frame(strsplit(x, '/'))[1, ])]
# 3. Apply the function 'find_max' to 'job-fat' grouped by 'job'
tb <- tapply(diet$fibre_fat, diet$job, find_max)
# 4 Display the result
data.frame(job = names(tb), fibre_fat = tb, row.names = NULL)
```

```
##          job    fibre_fat
## 1     Driver 4.595/16.658
## 2   Conductor 3.352/14.173
## 3 Bank worker 5.351/14.386
```

In this demonstration, the combination of the `which.max()`, `data.frame()`, and `strsplit()` makes up the self-defined function `find_max()`, which can pick the maximum dietary fibre intake out of *fibre-fat*. The `tapply()` function apply the `find_max()` function to each category of the qualitative variable *job*. Finally, the `data.frame()` function organizes and prints the result in a friendly way.

The **data.table** package provides another way:

```
# 1. Load the package:
library(data.table)
# 2. Converts diet from a data.frame into a data.table:
setDT(diet)
# 3. Do the task and display the result:
diet[,.SD[fibre == max(fibre, na.rm = TRUE),
          .(fibre_fat = paste0(fibre, "/", fat))],
     by = .(job)]
```

Compared to the base R method, the **data.table** method is much faster and more ink-saving: the core code is only one line (Step 3). As long as the dataset is converted into the proper format (*data.table*), we can use the syntax of the **data.table** package for many complicated calculations in an easy way. Note that "data.table" is both the package name as well as the specific class of the data defined by the package. Two remarkable features of the **data.table** package are (1) processing data in the *data.table* format and (2) using the dot `.` as the leading operator.

The **tidyverse** method, however, is totally different:

```
# 1. Load the packages:
library(tidyverse)
# 2. Specify the data frame:
diet %>%
# 3. Group the dataset with job:
  group_by(job) %>%
# 4. Pick out the target rows:
  filter(fibre == max(fibre, na.rm = TRUE)) %>%
# 5. Create the target columns:
  transmute(job = job,
            fibre_fat = paste0(fibre, '/', fat))
```

```
## # A tibble: 3 x 2
## # Groups:   job [3]
##   job         fibre_fat
##   <fct>       <chr>
## 1 Driver      4.595/16.658
## 2 Conductor   3.352/14.173
## 3 Bank worker 5.351/14.386
```

Tidyverse is technically a family of multiple R packages that share some common rules, and the **tidyverse** package is a "manager" for installing and loading the core Tidyverse packages, which can be found with the `tidyverse_packages()` function:

```
tidyverse::tidyverse_packages()
```

```
##  [1] "broom"         "cli"           "crayon"        "dbplyr"
##  [5] "dplyr"         "dtplyr"        "forcats"       "ggplot2"
##  [9] "googledrive"   "googlesheets4" "haven"         "hms"
## [13] "httr"          "jsonlite"      "lubridate"     "magrittr"
## [17] "modelr"        "pillar"        "purrr"         "readr"
## [21] "readxl"        "reprex"        "rlang"         "rstudioapi"
## [25] "rvest"         "stringr"       "tibble"        "tidyr"
## [29] "xml2"          "tidyverse"
```

When we run `library(tidyverse)`, all these packages are loaded.

Tidyverse also has two remarkable features: (1) it is designed for processing data frames, and (2) it uses the pipe operator `%>%` or `|>` very frequently, even sometimes it might be unnecessary. The pipe operator passes the result returned by the current function as an input to the function that comes next, which makes the data processing procedure very fluent. In this book, we use the built-in pipe operator `|>` instead of `%>%` wherever applicable.

All the three methods have their own pros and cons. The most traditional one is base R, for which we could find learning material in almost every single book about R. It is also the most flexible: we could (almost) complete any complicated data work step by step with base R. Furthermore, the skills we learn in base R are more transferable to other languages. However, we often have to think about how to combine the base R functions into an integrated project. The **data.table** package is much advanced and fast, but we have to be familiar with its syntax and rules. **Tidyverse** is the most popular method nowadays. In most cases, we can find tidyverse solutions to a question when we search the internet for help.

In this book, we use both the base R method and the tidyverse method for demonstrations.

1.8 Exercises

1. RStudio IDE is not the only graphic user interface of using R. Find at least two alternatives. What are their pros and cons?
2. In RStudio IDE, what is the hotkey for inserting a comment line as a section label? What is a section label used for?
3. What arguments can be used in the function `plot()`? Give an example.
4. Install the **epiR** package. Answer the following questions:

- How many times has **epiR** been downloaded from CRAN since it was available?
- Who are the authors of **epiR**? How do you cite it in a journal paper?
- What functions does **epiR** include? What are they used for?
- What datasets does **epiR** ship?
- What is the dataset *epi.incin* about? How many observations and variables are contained in it?

References

Aragon, Tomas J. 2020. *epitools: Epidemiology Tools*. https://CRAN.R-project.org/package=epitools.

Athens, Josie. 2022. *pubh: A Toolbox for Public Health and Epidemiology*. https://CRAN.R-project.org/package=pubh.

Carstensen, Bendix, and Martyn Plummer. 2022. *epi: Statistical Analysis in Epidemiology*. http://bendixcarstensen.com/Epi/.

Chongsuvivatwong, Virasakdi. 2022. *epiDisplay: Epidemiological Data Display Package*. https://CRAN.R-project.org/package=epiDisplay.

Csárdi, Gábor, Jim Hester, Hadley Wickham, Winston Chang, Martin Morgan, and Dan Tenenbaum. 2021. *remotes: R Package Installation from Remote Repositories, Including GitHub*. https://CRAN.R-project.org/package=remotes.

Dowle, Matt, and Arun Srinivasan. 2022. *data.table: Extension of 'Data.frame'*. https://CRAN.R-project.org/package=data.table.

Hoehle, Michael, Sebastian Meyer, and Michaela Paul. 2022. *surveillance: Temporal and Spatio-Temporal Modeling and Monitoring of Epidemic Phenomena*. https://surveillance.R-Forge.R-project.org/.

Jenness, Samuel, Steven M. Goodreau, Martina Morris, Adrien Le Guillou, and Chad Klumb. 2022. *epiModel: Mathematical Modeling of Infectious Disease Dynamics*. http://www.epimodel.org/.

Lumley, Thomas. 2021. *survey: Analysis of Complex Survey Samples*. http://r-survey.r-forge.r-project.org/survey/.

Mihaljevic, Joseph. 2022. *SPARSEMODr: SPAtial Resolution-SEnsitive Models of Outbreak Dynamics*. https://github.com/NAU-CCL/SPARSEMODr.

Paradis, Emmanuel. 2005. *R for Beginners*. Institut des Sciences de l'Evolution: Université Montpellier II.

Persson, Eric. 2019. *WHO: R Client for the World Health Organization API*.

R Core Team . 2022. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Rotondi, Michael A. 2018. *epibasix: Elementary Epidemiological Functions for Epidemiology and Biostatistics*. https://CRAN.R-project.org/package=epibasix.

Stevenson, Mark, Telmo Nunes, Cord Heuer, Jonathon Marshall, Javier Sanchez, Ron Thornton, Jeno Reiczigel, Jim Robison-Cox, Paola Sebastiani, and Peter Solymos. 2022. *epiR: Tools for the Analysis of Epidemiological Data*. https://fvas.unimelb.edu.au/research/groups/veterinary-epidemiology-melbourne.

Therneau, Terry M. 2022. *survival: Survival Analysis*. https://github.com/therneau/survival.

Wickham, Hadley, and Garrett Grolemund. 2016. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. O'Reilly Media, Inc.

Wickham, Hadley. 2015. *R Packages: Organize, Test, Document, and Share Your Code*. O'Reilly Media, Inc.

Wickham, Hadley. 2022. *tidyverse: Easily Install and Load the Tidyverse*. https://CRAN.R-project.org/package=tidyverse.
Yu, Guangchuang, and Tianzhi Wu. 2021. *nCov2019: Exploring COVID-19 Statistics*. https://github.com/YuLab-SMU/nCov2019.
Zhao, Peng. 2021. *rosr: Create Reproducible Research Projects*. https://github.com/pzhaonet/rosr.
Zuur, Alain F., Elena N. Ieno, Erik H.W.G. Meesters, et al. 2009. *A Beginner's Guide to R*. Springer.

Chapter 2
Planning Data

2.1 Chapter Highlights

- Make sense of the importance of research design and data planning,
- Describe the basic process of making a plan for working with data,
- Conduct literature review with R packages, and
- Illustrate the data plan with mind maps and Gannt charts.

2.2 Research Design

Research design refers to the overall strategy to lead the research process by planning processes from formulating research questions to accomplishing research goals. It includes the determination of the study type, research questions, hypotheses, variables of interest, the time-line and location, resources and participants, methods of processing, etc.

A research design centres on a research question, such as "what is the purpose of the study?" or "what do you wish to estimate or predict?" The general steps in the design process suggested by Guest and Namey (2015) is illustrated in Fig. 2.1. To identify a question, start with a broad research subject and then focus it down to a more particular issue. Next, perform a literature evaluation in order to identify the research gap and formulate the research question. After the question is specified, it is time to plan the research. The research method (qualitative, qualitative, or mixed-method) should be chosen based on the research question. Then, details should be taken into consideration, including sampling method, sample size, inclusion criteria, research instruments, independent and dependent variables, data analysis method, and so on.

P. Zhao, *Working with Data in Public Health: A Practical Pathway with R*,
https://doi.org/10.1007/978-981-99-0135-7_2

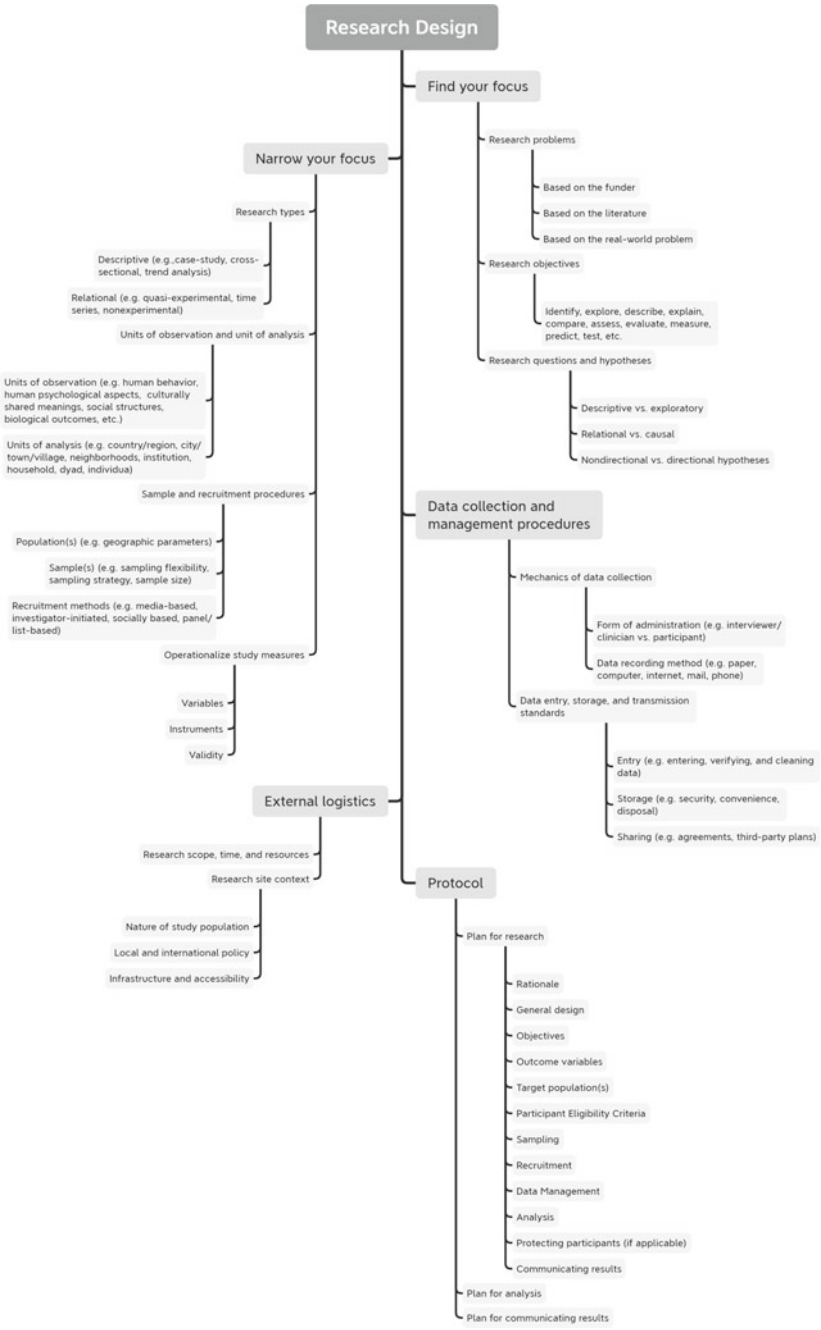


Fig. 2.1 Basic steps in the project design process. Adapted from Guest and Namey (2015)

2.3 Literature Review

2.3.1 Relevant R Packages

A research design begins with finding the research topic (see Fig. 2.1). This task is achieved via literature review, which is an overview of previously published works on a particular subject. It gives us a broad perspective of current knowledge and helps us to spot relevant ideas, methodologies, and research gaps.

A literature review does more than list sources of publications; it analyses, synthesizes, and critically evaluates them to provide a comprehensive picture of the current state of knowledge on the topic. It is based on reference management, including searching, recording, utilizing and, managing bibliographic citations. Many databases, including Web of Science, Scopus, ScienceDirect, PubMed, and Google Scholar, are available for searching for journals and papers with keywords in a certain topic, and provide source libraries with rich bibliographic information. A number of R packages support these reference libraries for literature reviews and reference management, which is summarized in the following list:

- **bibliometrix** (Aria and Cuccurullo 2022, GitHub massimoaria/bibliometrix) imports bibliographic data and performs bibliometric analysis such as co-citation, coupling, scientific collaboration analysis and co-word analysis.
- **fulltext** (Chamberlain 2022, GitHub ropensci/fulltext) searches across and gets full text for Open Access (OA) and closed journals.
- **rbibutils** (Boshnakov and Putman 2022, GitHub GeoBosh/rbibutils) reads and writes BibTeX files, and converts between bibliography formats, including 'Bibtex', 'Biblatex', 'PubMed', 'Endnote', and 'Bibentry'.
- **rcrossref** (Chamberlain et al. 2022, GitHub ropensci/rcrossref) is a client for various CrossRef APIs, including metadata search with their old and newer search APIs. It gets citations in various formats (including 'bibtex', 'citeproc-json', 'rdf-xml', etc.), converts DOIs to PMIDs, and vice versa. Additionally, it gets citations for DOIs and links to full text of articles when available.
- **RefManageR** (McLean 2022, GitHub ropensci/RefManageR) imports and works with bibliographic references.
- **scholar** (Yu, Keirstead, and Jefferis 2022, GitHub jkeirstead/scholar) analyzes citation data from Google Scholar.
- **wosr** (Baker 2018, GitHub vt-arc/wosr) is a client to the Web of Science and Incites APIs.

2.3.2 The bibliometrix Package

Comprehensive literature analysis can be conducted with the R **bibliometrix** package (Aria and Cuccurullo 2022), which is an analytical tool for quantitative research

in scientometrics and bibliometrics. Although based on R, it provides a friendly application (built with the R **shiny** package) to non-programmers for bibliometric analysis and building data matrices for co-citation, coupling, scientific collaboration analysis and co-word analysis with statistical and graphical packages. Here we briefly introduce how to use it with a demonstration library

We can start the application with one command in R:

```
bibliometrix::biblioshiny()
```

This function evokes the computer's default web browser, which displays a webpage-based application. A menu bar is shown at the to of the application interface (Fig. 2.2), which provides access to almost all the functionality of **bibliometrix**. The menu bar is followed by the title, the citation, the logo, the features, and the instruction of how to use this application from top to bottom. The workflow chart helps understand the internal procedure of how the application works. At the bottom of the page, an example gives a step-by-step instruction of how to conduct the bibliometric analysis with a demonstration library file.

The analysis is conducted as follows:

1. Download the demonstration library file.
2. Click the `Data - Import or Load files` menu.
3. In the left sidebar, choose `Import raw file(s)` and `Web of Science (WoS/WoK)` as the database.
4. Click the `Browse` button and choose the demonstration library. Then the upload starts.
5. When the upload is completed, click the `Start` button. Then the bibliometric analysis starts and all the analysis is conducted automatically.

A wide table is shown in the main panel. Each row in this table is a bibliographic entry of a publication (e.g. an article or a book), and each column is a field of the references, such as the Digital Object Identifier (`DOI`), authors (`AU`), author full names (`AF`), cited references (`CR`), and so on. A complete list of the abbreviations for the reference fields is available on the website of the Web of Science.[1]

The bibliometric analysis results, including the overview, the metadata focus analysis, and the knowledge focus analysis, are hidden in the tabs listed in the menu bar. We could simply click the tabs and see the result tables and graphs. Figure 2.3 shows the available results in a logical structure.

The demonstration library was obtained from the search in Web of Science Core Collection about all the articles published by the *Journal of Informetrics* from 2007 to 2017. Apart from Web of Science, the **bibliometrix** package can analyze the searched results by the SCOPUS, PubMed, Digital Science Dimensions, and Cochrane databases.

The bibliometric analysis provided by the **bibliometrix** package summarizes vast amounts of bibliometric data and exhibits the state of the intellectual structure and developing trends of a research topic or area. It is a helpful tool for the literature review in the research design stage, especially when the scope of review is wide and the reference library is too large for manual review.

[1] https://images.webofknowledge.com/images/help/WOS/hs_wos_fieldtags.html.

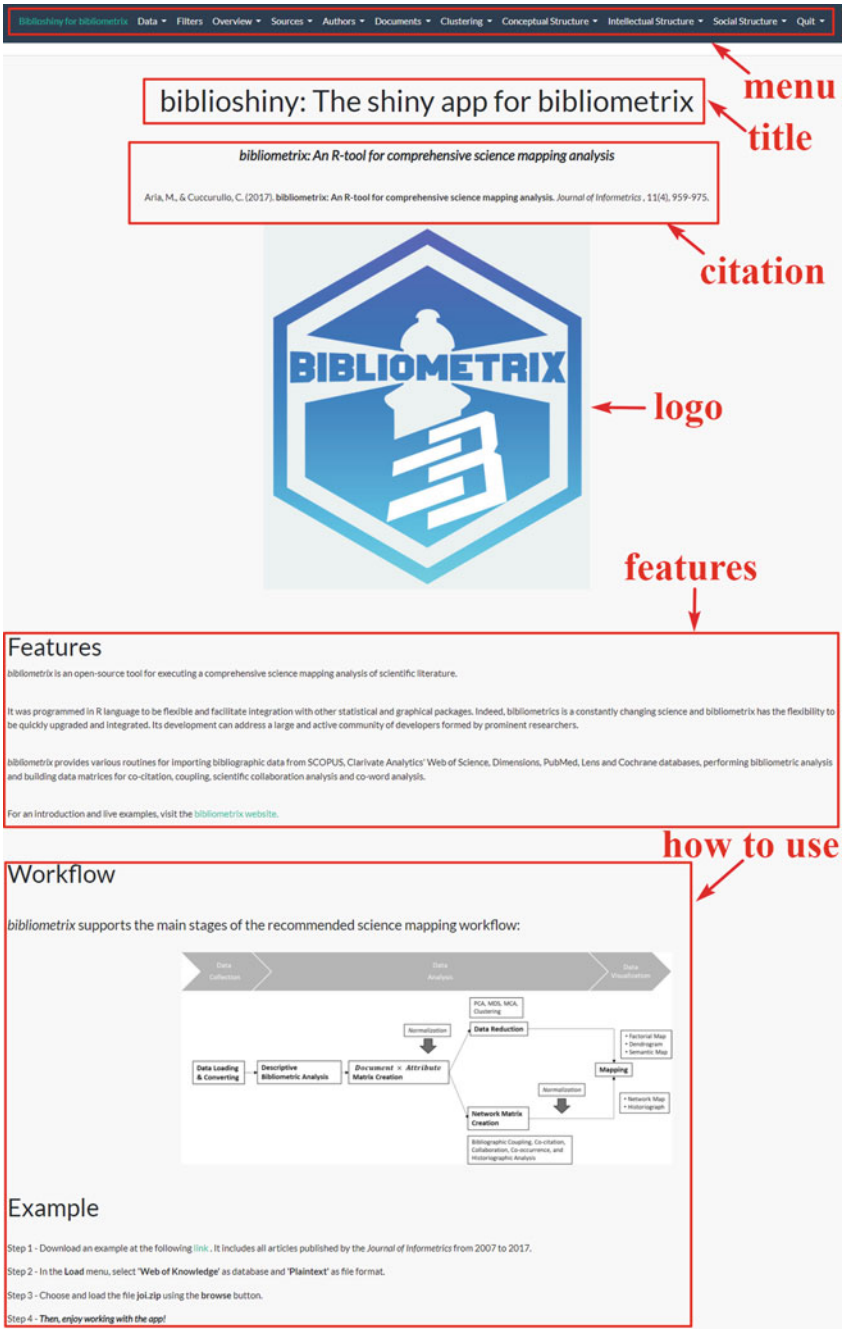


Fig. 2.2 The homepage of the **bibliometrix** application

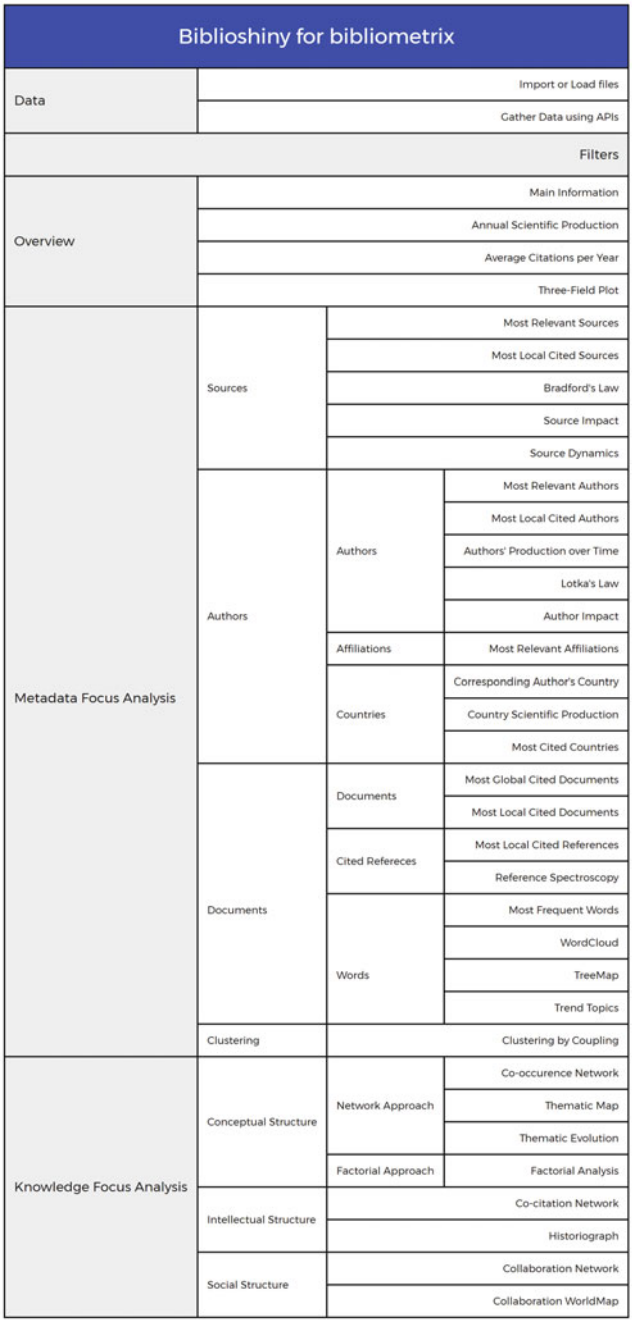


Fig. 2.3 The structure of the **bibliometrix** application

2.3.3 The scholar Package

When we are interested in a certain scientist, such as your supervisor or a co-author, and want to have a whole picture of her/his research, the **scholar** package (Yu,Keirstead, and Gregory Jefferis 2022) could be a tool for obtaining and analyzing citation data of a scholar who has a profile page on Google Scholar.

Before the analysis is conducted, we have to know the scientist's Google Scholar ID, which could be found in the URL of the scientist's Google Scholar profile page. For example, the URL of a scientist's Google Scholar profile page is https://scholar.google.com/citations?user=JoN88RAAAAAJ, indicating the ID is the string after `user=` in the link, i.e. `JoN88RAAAAAJ`. Alternatively, the `get_scholar_id()` function could retrieve the ID from the scientist's name:

```
library(scholar)
gsid <- get_scholar_id(first_name = "marius", last_name = "wamsiedel")
gsid
```

```
## [1] "JoN88RAAAAAJ"
```

The **scholar** package mainly contains functions for exploring the citation data of a person, a journal, or a publication. For convenience, we create self-defined functions that combine these functions into three groups:

```
# functions for a person
gs_person <- function(id){
  list(citation = get_citation_history(id),
       coauthor = get_coauthors(id),
       n_article = get_num_articles(id),
       n_journal = get_num_distinct_journals(id),
       n_top = get_num_top_journals(id),
       oldest = get_oldest_article(id),
       profile = get_profile(id),
       pub = get_publications(id))
}

# functions for a journal
gs_journal <- function(journal){
  list(IF = get_impactfactor(journal),
       rank = get_journalrank(journal))
}

# functions for a publication
gs_pub <- function(id, pubid){
  list(cite = get_article_cite_history(id, pubid),
       authors = get_complete_authors(id, pubid))
}
```

- `gs_person()` groups the person-centred functions, which can retrieve historical citation data, the network of co-authors, the total publication number, the total distinct journal number, the top journal number, the year of the oldest article, the profile information, and the publication list for a scientist.

- `gs_journal()` groups the journal-centred functions, which can retrieve the cites, impact factor (IF), the Eigenfactor score, the rank, the ISSN, the publisher, and so on, of a journal.
- `gs_pub()` groups the publication-centred functions, which can retrieve the historic annual citations and the complete author list of a publication.

Each of these three functions returns a list. Let's get the data for a person:

```
ls_person <- gs_person(gsid)
```

The publications are saved as a data frame in `ls_person$pub`:

```
names(ls_person$pub)
```

```
## [1] "title"   "author"  "journal" "number"  "cites"   "year"    "cid"
## [8] "pubid"
```

Let's get the data for the journal and the publication in the first row:

```
ls_journal <- gs_journal(ls_person$pub$journal[1])
ls_pub <- gs_pub(gsid, ls_person$pub$pubid[1])
```

These data could be used for further analysis. For example, the following code plots the annual citations of the scientist, the top six journals with the most of the scientist's publications, the annual citations of the chosen publication, and the network of the co-authors (Fig. 2.4).

```
library(ggplot2)
library(dplyr)
library(patchwork)

p_citation <-
  ls_person$citation |>
  ggplot(aes(year, cites)) + geom_bar(stat = 'identity')

p_pub <-
  ls_pub$cite |>
  ggplot(aes(year, cites)) +
  geom_segment(aes(xend = year, yend = 0), size=1, color='darkgrey') +
  geom_point(size=3, color='firebrick')

p_journal <- ls_person$pub |>
  filter(journal != '') |>
  group_by(journal) |>
  summarise(n = length(journal)) |>
  arrange(desc(n)) |>
  mutate(journal = reorder(journal, n)) |>
  head() |>
  ggplot() +
  geom_bar(aes(x = journal, y = n), stat = 'identity') +
  coord_flip()

p_colab <-
  plot_coauthors(ls_person$coauthor, size_labels = 3) +
```

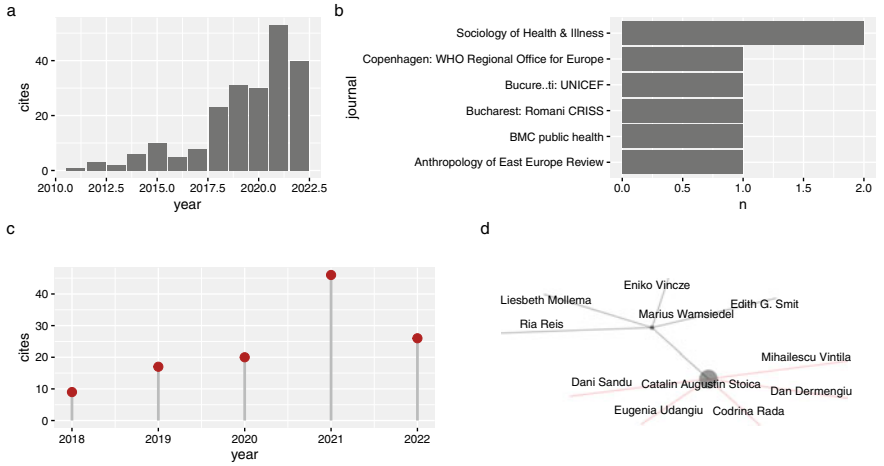


Fig. 2.4 Explore the Google Scholar data. **a** The annual citations of a scientist. **b** The top six journals with the most of the scientist's publications. The annual citations of the chosen publication. **d** The network of the co-authors

```
  labs(title = '')

(p_citation | p_journal) / (p_pub | p_colab) + plot_annotation(tag_levels = 'A')
```

Apart from the three groups of the functions, the **scholar** package provides other functions for comparing the citation records and careers of multiple scholars, predicting the h-index for a scholar, retrieving a Google Scholar Page storing session cookies, and so on.

2.4 Establish a Data Plan

2.4.1 Workflow

Among research design, a data plan, including data collection, interpretation, analysis, visualization, and discussion, has an important impact on the reliability of the obtained results. Making a data plan enables researchers to focus on research methodologies that are appropriate for the research topic and build their research for success. According to data types, public health research can be roughly categorized as quantitative research and qualitative research. Typically, public health research focuses on health determinants, with the findings being applied to policies and treatments to address real-world health issues. An effective data plan is essential for ensuring the evidence's validity and dependability.

However, the raw data created by all kinds of field work may be stored in a variety of forms in most cases. Those data may not be used by researchers if they are not summarised properly with the help of appropriate tools. As a result, the following considerations are critical at the beginning and throughout the data planning process.

1. **Data source**. The first step in data planning is to understand the data source. Due to the varied methodologies that are employed, the same data from different health agencies may differ. For example, health care data come from medical records department, outpatient clinics, private clinics and hospitals, pathology laboratories, autopsy services, death certificates, health insurance, screening programmes, etc. From a management perspective, it is decisive to understand that the relevance of data sources, data usage and distribution policies might vary depending on who publishes the data, which can have a substantial impact on potential data management strategies.
2. **Data format**. Raw data come from a variety of sources, and the data format must be consistent before it can be processed for further analysis. Because it is vital to present the data to the end users, data of various forms should be converted into agreed ones. The decoding of the raw data's original format, as well as the correction of the data's internal structure, are both part of this process.
3. **Data update frequency**. It is crucial to keep some sorts of data up to date. Therefore, understanding the data refresh rate is very useful for designing user interface updates. The "frequency" of interface update is not always equal to the frequency of data update, but it can "adapt" to a lower update frequency based on user requirements. The data format and update frequency are also influenced by the data usage time. The more data there is, the longer it takes to get it and show it on the interface, which has an impact on how the data is presented.

A data plan must be established within a research design. After a general search plan is made, a detailed data plan could be established (Fig. 2.5). The first step is clarify what are the resources to collect data, what are relevant data, and how to collect them. We must decide how to sample data, who will collect the data, and how the data are recorded and stored. If possible, we can even fake some dummy data for practice just like a rehearsal before a stage performance, which might be useful and helpful for a better data plan at a low cost.

As soon as the raw data are collected, we can prepare them for further work. The raw data might be stored in a variety of formats, and proper skills for importing them into the data processing tools (i.e. R environment in this book) would be required. The real-world data are usually dirty, and the cleaning process is often necessary.

When the clean data is ready, we can move to the data exploration stage. In public health, we use many statistics tools for describing and exploring data. Visualization is often carried out simultaneously so as to display data summaries and analysis results, find any trend, correlation or pattern. Data exploration is followed tightly by modelling and evaluating data.

The final step of working with data is publishing them. Data are presented in a variety of forms such as statements, tables, and graphs, following associated rules

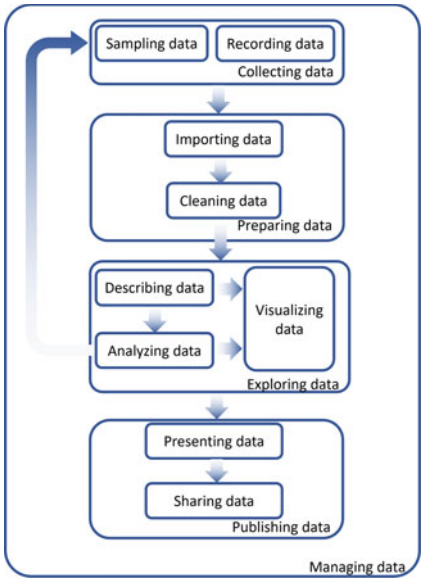


Fig. 2.5 Typical stages of working with data

and conventions, and shared in posters, slides, journal papers, theses, etc. The entire data workflow must be integrated in the framework of data management.

Planning data is the bridge to link a research question to the research objectives. Therefore, the setup of a data plan should be systemic that can cover the whole procedure of the research and start at the research question as the very early stage. A good data plan is the foundation of every public health study, as it helps researchers better understand how to gather and evaluate data in order to address research gaps on a number of issues and discover the best solution to their research goal and concerns. It can increase the efficacy of research data, and saves resource investment (money, workforce, time, etc.). Most public health studies, in particular, are social science-based and collect data over a lengthy period of time. The progress of the study may be hampered if problems are detected and rectified during the project's execution, and data planning can expose a variety of issues at an early stage. Data planning can also guarantee that the theoretical and conceptual framework is in line with the study goal.

In general, planning data can be regarded as a foundation and framework established to find solution to a specific scientific question. Data plans, which are based on the study's purpose and methodology, give a strong framework for scientific research and aid in the study's efficiency.

2.4.2 Mind Maps

A mind map is a diagram that displays a hierarchical structure and relationships between elements of a larger whole. It normally has a central concept, with major and minor ideas that branch out. As it simulates the way of the human brain's thinking, it is very helpful for making a research design and a data plan. Figure 2.1 is a mind map demonstration.

Traditional mind maps may be drawn by hand using coloured pens and paper, but modern information technology has made it more common and convenient to build mind maps using desktop computer software or webware (e.g., FreeMind, XMind, iMindMap, and MindMeister). The R **mindr** package (Zhao 2021) can generate reproducible mind maps, which can either be saved in a form that is supported by common mind map computer software, or be visualized interactively in web browsers.

The following R code creates a mind map for the framework of processing data with R suggested by Beaumont (2015).

```
library(mindr)
demo_txt <- c(
  '# Data',
  '## Creating',
  '## Importing',
  '### Files',
  '### Via applications',
  '## Saving/exporting',
  '## R script files',
  '# Manupulation',
  '## Columns (variables)',
  '## Rows (cases/subjects)',
  '# R Environment',
  '## Non-CRAN packages',
  '## Workspaces',
  '## Ojbects',
  '## History',
  '# Code development',
  '## RStudio',
  '## Notepad++'
)
mindmap_demo <- mm(from = demo_txt,
                   output_type = c('widget', 'mindmap'),
                   root = 'Processing data with R')
```

The argument `from` specifies the input text given in Markdown syntax, i.e. # leads a title of level 1, ## for level 2, ### for level 3, etc. (see Chap. 9). The argument `output_type` here indicates that we would like a HTML widget (`'widget'`) and a mind map that can be opened with other mind map software (`'mindmap'`). The argument `root` specifies what is shown in the centre of the mind map (Fig. 2.6).

The output is a list containing two forms of mind maps. i.e. a widget (`mindmap_demo$widget`) and a character vector (`mindmap_demo$mindmap`). The widget can be displayed directly in the built-in Viewer in the bottom right pane of RStudio IDE:

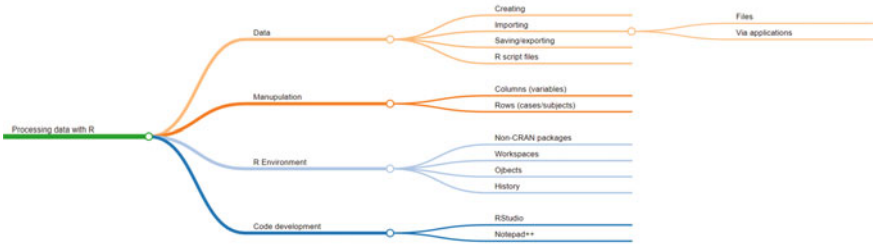


Fig. 2.6 A HTML widget for a mind map created by the **mindr** package

```
mindmap_demo$widget
```

This widget is interactive: we could click the nodes to expand or hide branches, or use the mouse to zoom it in or out. We can click in the Viewer menu `Export - Save as Web Page`, and then an independent mind map file in .html is created, which can be viewed with a web browser or embedded into a webpage. Alternatively, it can also be saved as a HTML file via the **htmlwidgets** package:

```
htmlwidgets::saveWidget(mindmap_demo$widget, file = 'mindmap_demo.html')
```

By default, the widget is displayed in a colourful theme. The theme and the size of the mind map can be customized. See the help document of **mindr** for more details. R Markdown users can insert such an interactive mind map in their reproducible research as well (see Chap. 9).

The character vector, which is included in `mindmap_demo`, could be exported as a plain text file:

```
writeLines(mindmap_demo$mindmap, 'mindmap_demo.mm')
```

The exported `.mm` file could either be opened by FreeMind or imported to XMind, and displayed and edited as an elegant mind map.

Although typing them in R code is not difficult, a more convenient way is saving the input text (`demo_txt`) in a plain text file, which can be edited with Markdown editors (e.g. Typora, Mark Text, RStudio IDE) and imported into R with the `readLines()` function.

The advantages of drawing mind maps with **mindr** are as follows:

1. The input plain text is software-independent, which means that you can edit it with any text editor.
2. As the input text is in Markdown syntax, you can extend it further into a data report with R Markdown.
3. Version control software can be used for tracking the change in the input file and for collaborating with others.

2.4.3 Gannt Charts

A Gantt chart is often used for illustrating a project schedule, i.e. the time and duration of activities in a data plan visually. It helps the creator not only have a overview of the timeline, but also communicate with the collaborators and supervisors about the milestones of the project.

R provides a few elegant ways to generate a Gantt chart. For example, the **DiagrammeR** package (Iannone 2022) provides a powerful function `mermaid()`, which generates many types of HTML diagrams, including Gannt charts, with the JavaScript library mermaid.js.

Suppose we have a data frame with a project schedule, which is created with the **tibble** package (Müller and Wickham 2022):

```
x <- tibble::tribble(
  ~Start,        ~End,         ~Activity,        ~Status,
  "2021-01-01", "2021-03-31", "Research design", 'done',
  "2021-04-01", "2021-09-30", "Lab preparation", 'done',
  "2021-10-01", "2021-10-31", "Outdoor test",    'active',
  "2021-11-01", "2023-01-31", "Collect data",    'crit, active',
  "2021-05-01", "2022-04-30", "R package dev.", 'active',
  "2023-02-01", "2023-08-31", "Analyze data",    '',
  "2023-06-01", "2024-01-01", "Manuscript",      ''
  )
```

Then we can create a Gannt chart (Fig. 2.7) after a little transformation of the data frame:

```
library(DiagrammeR)
create_gannt_txt <- function(x){
         paste(
           paste0(x$Activity, ':',
                  x$Status, ifelse(x$Status == '', '', ', '),
                  1:nrow(x), ', ', x$Start, ',', x$End),
           collapse = '\n')
}
gannt_txt <- paste('gantt',
                   'dateFormat  YYYY-MM-DD',
                   'section Basic Tasks',
                   create_gannt_txt(x[1:4,]),
                   'section Products',
                   create_gannt_txt(x[5:7,]),
                   sep = '\n')
mermaid(gannt_txt)
```

Another option is using the **ggplot2** package (Wickham et al. 2022) as follows.

```
library(ggplot2)
library(tidyr)
plot_gannt <- function(x){
  acts <- x$Activity
  gannt <- gather(x, "state", "date", 1:2) %>%
    transform(
      Activity=factor(Activity, acts[length(acts):1]),
      date = as.Date(date)) %>%
```

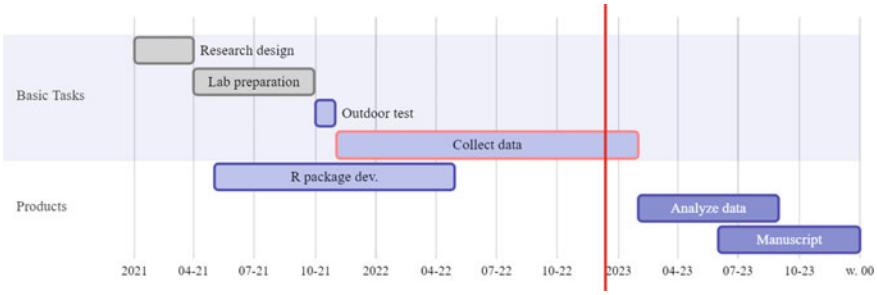


Fig. 2.7 A Gannt chart for a project generated with the **DiagrammeR** package

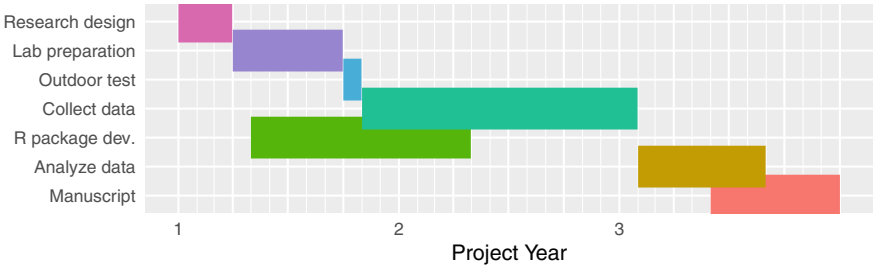


Fig. 2.8 A Gannt chart for a project generated with the **ggplot2** package

```
    ggplot(aes(date, Activity, color = Activity)) +
    geom_line(size = 10) +
    scale_x_date(
      breaks = seq.Date(
        as.Date(x$Start[1]),
        as.Date(x$End[nrow(x)]), "quarter"),
      minor_breaks = seq.Date(
        as.Date(x$Start[1]),
        as.Date(x$End[nrow(x)]), "month"),
      labels=c(1, "", "", "", 2, "", "", "", 3, "", "", "", "")) +
    theme(legend.position = "", axis.ticks = element_blank())+
    labs(x = "Project Year", y = "")
  gannt
}
plot_gannt(x)
```

It uses the **tidyr** package (Wickham and Girlich 2022) for transforming the table, and the **ggplot2** package for making a Gannt chart (Fig. 2.8).

Aside from **DiagrammeR** and **ggplot2**, alternative ways for plotting Gannt chars are as follows.

- The **plotrix** package (Lemon et al. 2021) provides a function `gantt.chart()` which can generate nice static Gannt chars as well.

- The **plan** package (Kelley and Schmitt 2022) provides functions for planning and describing projects and monitoring the progress towards completions of individual tasks within projects.
- The **PlotPrjNetworks** package (Muñoz 2015) provides a function `GanttChart()`.
- The **plotly** package (Sievert et al. 2022) can generate interactive HTML Gannt charts.[2]

2.5 Exercises

1. College students are sometimes burdened by mental diseases. According to studies, college students prefer to seek help from mental health specialists, use campus psychological counselling facilities, and use mobile applications. However, preferences and relevance levels for mental healthcare service qualities remain uncertain. In developing countries, students' utilization and preferences for mental health treatments may differ from those in developed countries. Students' choice of mental disease services is also influenced by regional economic level, public and self-mental disease stigmas, and the characteristics of colleges.
   Make a research design and data plan for answering the following question: what attributes of mental healthcare services influence the a student's decision to use them?

   - Search relevant publications, and use the **bibliometrix** package for a bibliometric analysis.
   - Draw a mind map for the research design.
   - Draw a Gannt chart for the data plan.

2. The **fulltext** package (Chamberlain 2022) is a tool for searching journal articles and text-mining. It can search for and fetch articles, get links for full text articles, extract text from articles, and download supplementary materials from papers. Read the online manual[3] of **fulltext** and do the following exercises:

   - Search for the keyword *COVID-19* across PLOS, Crossref, and arXiv preprint server.
   - Retrieve the abstracts of the ten returned articles from PLOS and read them.
   - Download the full text of the ten returned articles from PLOS.
   - Extracting the title and the keywords of the ten returned articles. What topics do people study about COVID-19?

3. The **mindmap** package can extract the outline of a .pdf document and convert it into a mind map. Find how to use this feature for grasping the main points of journal articles of your interest.

[2] https://plotly.com/r/gantt/.

[3] https://books.ropensci.org/fulltext.

4. Plot Fig. 2.7 with the **plotly** package. What are the pros and cons of an interactive Gannt chart?

References

Aria, Massimo, and Corrado Cuccurullo. 2022. *bibliometrix: Comprehensive Science Mapping Analysis*. https://CRAN.R-project.org/package=bibliometrix.

Baker, Christopher. 2018. *Wosr: Clients to the Web of Science and InCites APIs*. https://vt-arc.github.io/wosr/index.html.

Beaumont, Robin. 2015. *Health Science Statistics Using R and R Commander*. Scion.

Boshnakov, Georgi N, and Chris Putman. 2022. *rbibutils: Read Bibtex Files and Convert Between Bibliography Formats*. https://CRAN.R-project.org/package=rbibutils.

Chamberlain, Scott. 2022. *fulltext: Full Text of Scholarly Articles Across Many Data Sources*.

Chamberlain, Scott, Hao Zhu, Najko Jahn, Carl Boettiger, and Karthik Ram. 2022. *rcrossref: Client for Various CrossRef 'APIs'*. https://CRAN.R-project.org/package=rcrossref.

Guest, G, and E. Namey. 2015. *Public Health Research Methods*. Book. Thousands Oaks, California: SAGE Publications.

Iannone, Richard. 2022. *DiagrammeR: Graph/Network Visualization*. https://github.com/rich-iannone/DiagrammeR.

Kelley, Dan, and Frank Schmitt. 2022. *plan: Tools for Project Planning*. https://github.com/dankelley/plan.

Lemon, Jim, Ben Bolker, Sander Oom, Eduardo Klein, Barry Rowlingson, Hadley Wickham, Anupam Tyagi, and et al. 2021. *plotrix: Various Plotting Functions*. https://CRAN.R-project.org/package=plotrix.

McLean, Mathew W. 2022. *RefManageR: Straightforward BibTeX and BibLaTeX Bibliography Management*. https://github.com/ropensci/RefManageR/.

Müller, Kirill, and Hadley Wickham. 2022. *tibble: Simple Data Frames*. https://CRAN.R-project.org/package=tibble.

Muñoz, Celigueta, Javier. 2015. *PlotPrjNetworks: Useful Networking Tools for Project Management*. https://CRAN.R-project.org/package=PlotPrjNetworks.

Sievert, Carson, Chris Parmer, Toby Hocking, Scott Chamberlain, Karthik Ram, Marianne Corvellec, and Pedro Despouy. 2022. *plotly: Create Interactive Web Graphics via plotly.js*. https://CRAN.R-project.org/package=plotly.

Wickham, Hadley, and Maximilian Girlich. 2022. *tidyr: Tidy Messy Data*. https://CRAN.R-project.org/package=tidyr.

Wickham, Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, and Dewey Dunnington. 2022. *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. https://CRAN.R-project.org/package=ggplot2.

Yu, Guangchuang, James Keirstead, and Gregory Jefferis. 2022. *Analyse Citation Data from Google Scholar*. https://CRAN.R-project.org/package=scholar.

Zhao, Peng. 2021. *mindr: Generate Mind Maps with R*. https://github.com/pzhaonet/mindr.

Chapter 3
Collecting Data

3.1 Chapter Highlights

- Make sense of the importance of data sampling strategy and recording methods in research,
- Describe the characteristics (pros and cons) of different data sampling methods, and
- Use the R language to apply those methods to research.

3.2 Sampling Data

We live in an age of information overload, and it is often difficult to collect data that are useful for relevant study. Data come from a variety of sources, including original data obtained from field studies and existing data sources. For example, public health surveillance refers to "the ongoing, systematic collection, analysis, and interpretation of health-related data essential to planning, implementation, and evaluation of public health practice" (Gregg 2008), which provides data from continuous monitoring of disease, or emerging health related issues. As a result, public health actions are taken and the surveillance results reflect the effectiveness of those actions. During a field investigation, public health researchers create surveys that may gather data in a structured and systematic manner, and the survey results might be applied to the entire population if a typical sample is chosen.

For health related purpose, data normally come from the following sources:

- Individual persons, for personal information,
- Health-care providers and facilities, such as physician offices, hospitals, outpatient departments, emergency departments, inpatient settings, laboratories, etc., and
- Environmental conditions, such as the air quality, water quality, etc.

P. Zhao, *Working with Data in Public Health: A Practical Pathway with R*,
https://doi.org/10.1007/978-981-99-0135-7_3

In special cases, data collected for non-health-related-purposes are also used to illustrate health related issues, such as administrative actions, financial transactions, etc.

Most of studies in public health are focused on a group of individuals. Because it is impossible to gather data from every individual, a subset of the group is invited to participate in the study. The sample refers to the selected subset, whereas the population, or target population in specialised study, refers to the whole participants to be examined. The phrase *target population* is frequently used to indicate the aspects that suit a study's aims (Guest and Namey 2015). Depending on the study aims, the typical target population in public health research can be persons (e.g., patients, health specialists), objects (e.g., alcohol, drugs), and locations (e.g., clinics, communities). For example, in a research about the mental stress of nurses working in the clinics in St. Louis, all nurses working in such clinics may be considered the target population (Missouri Department of Health and Senior Services 2015). Why don't we simply choose all of these nurses? Choosing all elements of the target population to include in the study is called census. It has the advantage of completely reflecting all elements that fit the research objective, but it can be too time-consuming and expensive. In the previous example, there were 16,581 registered nurses in St. Louis in 2015, which would be very difficult to be all included in data collection.

When compared to a census, sampling costs less money, time, and manpower, making it more practicable and common in research. However, sampling has a significant drawback known as sampling error. When the selected sample does not reflect the complete target population, a divergence between the sample statistic and the population parameter arises. As a result, the sample findings cannot exactly represent those of the target population. In other words, it would be questionable to use the findings from the sample to estimate the situation of its target population.

The sample must be cautiously selected in order to achieve valid results. Otherwise, it is debatable whether the findings from the samples can be applied to the general population. A well-known case was the US presidential election of 1936 between Alf Landon and Franklin Roosevelt. The famous magazine *Literary Digest* conducted a poll with a sample size of around 2.4 million people about voting preferences, based on which they predicted that Landon would win but he failed. The failed prediction resulted from their sampled individuals, who were chosen from telephone subscribers and automobile owners, namely only wealthier people. Another example was given by Kinsey 1948 and 1953, who interviewed more than 10,000 men and women about their sexual behaviour and attitudes. This interview, however, did not give each individual the same chance of being chosen. Therefore, the findings can hardly be extrapolated to the population.

The following sections describe the two main types of sampling methods: probability sampling and non-probability sampling.

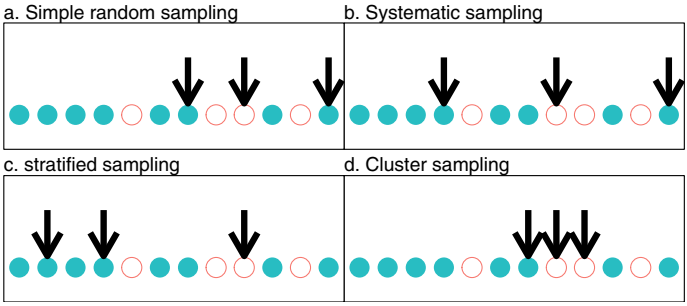


Fig. 3.1 Methods of probability sampling

3.2.1 Probability Sampling

Probability sampling selects individuals using random techniques, providing each member of the target population a known chance of being included in the sample. The major methods of probability sampling include simple random sampling, systematic sampling, stratified sampling, and cluster sampling (Fig. 3.1).

3.2.1.1 Simple Random Sampling

Simple random sampling is a sampling strategy that ensures that every member of the target population has an equal and independent probability of being chosen. Here we simulate and visualize the procedure of simple random sampling with an example.

First of all, define the target population. Suppose we are interested in a target population composed of 4 females and 8 males, identified with 1 to 12. We create a categorical object with 12 elements (`1` means female, and `2` means male) ordered in a random sequence.:

```
N <- 12
id <- 1:N
gender <-  factor(sample(rep(c(1, 2), c(4, 8)), 12))
```

Step 2 is to identify or develop a sampling frame, i.e. a list of all the members (N) in the target population. We could make up a data frame:

```
x <- data.frame(id = id,
                yi = 1,
                gd = gender)
```

The column `yi` is meaningless and only for plotting, while `gd` indicates the gender. Now we plot the population:

```
library(ggplot2)
p <- ggplot(x) +
 geom_point(aes(id, yi, colour = gd, shape = gd), size = 6) +
 labs(x = '', y = '') +
```

```
 scale_shape_manual(values = c(1, 16)) +
 lims(y = c(0.5, 2.5)) +
 theme_void() +
 theme(panel.border = element_rect(fill = NA), legend.position = "none")
p
```

Before we visualize the sampling results, we create a self-defined function which can add arrows into a graph at the given coordinates:

```
ggsample <- function(x, i) {
  # x is the input data frame
  # i is the index of the chosen individuals
  geom_segment(aes(x = id, y = 2, xend = id, yend = 1.2),
               data = x[i, ], size = 2, arrow = arrow())
}
```

Step 3 is to make a decision on the sample size (n). Here we assume the sample size is 3 just for simplification.

```
n <- 3
```

The final step is to randomly select the sample using lottery method or random number method:

```
i_sim <- sample(1:N, n)
p + ggsample(x, i_sim)
```

`i_sim` is the index of the chosen individuals, which are indicated with arrows in the graph.

Simple random sampling is a typical sample approach in public health and an excellent choice if a sampling frame is available. This is a simple and straightforward procedure. It is also feasible to assess the sampling error using this approach. Furthermore, it is equitable since each member of the target population has an equal and independent chance of being chosen. However, simple random sampling may be expensive and time-consuming, especially for a large population. Furthermore, the chosen sample may not be sufficiently representative. For example, if we repeat the R code in Step 4 multiple times, it occasionally happens that no female is chosen, or even no male is chosen. This problem could be solved by stratified sampling (see Sect. 3.2.1.3).

3.2.1.2 Systematic Sampling

Systematic sampling is a sampling approach in which the initial element of the sample is chosen at random, and then the other items are chosen at a defined and periodic interval. We inherit the previous simple example and simulate the systematic sampling as follows.

The first few steps are the same as the simple random sampling, including defining the target population, identifying or developing a sampling frame, determining the number of elements in the sampling frame and the sample size.

Then, the sampling interval could be calculated as $i = N/n$.

```
i <- N %/% n
```

Next step is to randomly select a number (r) from 1 through i.

```
r <- sample(1:i, 1)
```

The final step is to choose the r-th, $r + i$-th, $r + 2i$-th, … elements as the sample:

```
i_sys <- seq(r, r + i * (n - 1), i)
```

These steps could be combined into one self-defined function:

```
systematic_sampling <- function(N, n) {
  i <- N %/% n
  r <- sample(1:i, 1)
  seq(r, r + i * (n - 1), i)
}
i_sys <- systematic_sampling(N, n)
p + ggsample(x, i_sim)
```

`i_sys` is the index of the chosen individuals, which are indicated with arrows in the graph.

Systematic sampling spreads the selected observations over the target population, which is more representative than simple random sampling. It is also simpler and quicker to carry out. Nevertheless, systematic sampling might result in a biased sample when there is a periodical pattern in the population and this pattern coincides with the sampling interval (Guest and Namey 2015).

3.2.1.3 Stratified Sampling

Stratified sampling is a sampling approach in which the target population is first separated into strata (i.e., mutually exclusive and homogenous groups), and then a simple random sample based on a defined number or proportion is picked out from each stratum. The strata are usually created based on the demographic data's common qualities.

We inherit the previous simple example and simulate the stratified sampling as follows.

After defining the target population, we must identify the stratification variable(s) and determine the number of strata. In this example, the gender is the stratification variable and the number of strata is two.

Then, identify or develop a sampling frame and separate the sampling frame into strata based on the stratification variable(s). As mentioned before, the gender is labelled as 1 and 2 for separating the frame.

Finally, determine the sample size for each stratum and randomly select the sample from each stratum. Suppose we select 30% of the observations in each stratum. The **dplyr** package (Wickham et al. 2022) provides a function `sample_frac()` for this step:

```
library(dplyr)
x_str <- x |>
  group_by(gd) |>
  sample_frac(size = 0.3)
i_str <- x_str$id
p + ggsample(x, i_str)
```

Sometimes, we would like to take a stratified sample of a certain number of each group. For example, we sample 2 observations from each gender group:

```
x_str <- x |>
  group_by(gd) |>
  sample_n(size = 2)
i_str <- x_str$id
```

Stratified sampling reduces sampling error and ensures the representation of subgroups. However, it is more complicated and time-consuming since it requires a sampling frame with sufficient information.

3.2.1.4 Cluster Sampling

Cluster sampling is a sampling method that randomly selects naturally existing clusters of items.

The first few steps are the same as the simple random sampling, including defining the target population and the sample size. Instead of listing all the observations in a frame, we only have to determine the number of clusters, and randomly select the targeted number of clusters.

Based on the previous example, we suppose every neighboured n observations are in one cluster, and we want to sample one cluster from all:

```
n_clu <- 1
sample_clu <- sample(1: (N / n), n_clu)
i_clu <- (n * (sample_clu - 1) + 1) : (n * sample_clu)
```

We could create a self-defined function for cluster sampling:

```
cluster_sampling <- function(N, n, n_clu) {
  sample_clu <- sample(1: (N / n), n_clu)
  c(sapply(sample_clu, function(sample_clu) (n * (sample_clu - 1) + 1) : (n * sample_clu)))
}
i_clu <- cluster_sampling(N, n, n_clu)
```

Now the sample could be visualized:

```
p + ggsample(x, i_clu)
```

The cluster sampling does not require a sampling frame and it is easy to administrate. However, when clusters are sufficiently different from one another, it might result in more sampling error than simply random sampling.

3.2.2 Non-probability Sampling

In non-probability sampling, no random process is used in the sampling process. Each member in the target population does not necessarily have an equal probability of being chosen in the sample. Non-probability sampling approaches mainly include availability sampling, purposive sampling, quota sampling, and respondent-assisted sampling. As they don't need the assistance by a programming language, here we only describe them briefly.

- The availability sampling approach chooses a sample depending on the availability and convenience of the participants and researchers. It is the simplest method of sampling, requiring the fewest resources in terms of time, money, and manpower. All we have to do is define the target population, figure out how to recruit the sample, and determine the sample size. It does, however, have a number of significant flaws. The most significant drawback is that researchers have limited control over the sample characteristics. The individuals who are difficult to locate may also be excluded from the research. All of these factors combine to make this the least reliable sampling approach.
- The purposive sampling approach chooses a sample depending on how well it fits the inclusion and exclusion criteria of the research. We must determine the inclusion and exclusion criteria, followed by sampling, in addition to the standard stages of identifying the target population and determining the sample size. It is the most common non-probability sampling approach employed by researchers. Using this strategy, researchers may select parts of the target population depending on the study plan, without having to learn all about the target population.
- The quota sampling approach chooses a sample depending on how well it meets the research's inclusion and exclusion criteria from a created mutually exclusive sub-group, based on a specified proportion. It might be compared to a combination of purposive sampling and stratified sampling. It allows for the inclusion of large numbers of people from diverse parts of the target population in the sample. However, more resources and up-to-date information about the target population are required.
- The respondent-assisted sampling approach chooses a member from the population with the assistance of a previously chosen member. We also have to define the target population and identify the inclusion and exclusion criteria before recruiting and selecting the initial individual from the population and collecting data. The key step is to invite the chosen individual to choose referrals, which should be continued until no new referrals come in. Using the respondent-assisted sampling, the recruitment becomes easier, and the rare population can be included. Nevertheless, bias may be produced by the initial sample, as they are not likely to refer those who they dislike.

Due to the lack of random processes, all non-probability sampling systems have several similar drawbacks. The sample is chosen at the discretion of the researcher. The sample may or may not be representative.

3.3 Recording Data

Recording data means documenting data in a variety of forms, such as writing on paper and audio/video recording, in order to preserve it for later use. There are distinctly different ways to record data for a study. For example, if we want to measure the sleepless behaviour in college students, we could collect those data by observing the students with our eyes, by using equipment to measure their sleeping time, by surveying or interviewing them, or in other ways. As with most data recording techniques, each method has pros and cons. Observations are straightforward but might not always be quite practical. Equipment are precise and consistent but might be time and cost consuming. Surveys with questionnaires can be quick and inexpensive but need a careful and proper design. At present, using phone, computer, and internet to record data has become increasingly common.

Data can be categorized in different aspects. Their types can be primary or secondary. The unit of data can be individual level on persons or houses, and community level on groups of people. From an aspect of data sources in public health, data include demographic characteristics, geographic characteristics, socioeconomic characteristics, health data, and environmental data. For more details, health data include notifiable diseases, laborators specimens, vital records, sentinel surveillance, disease registries, periodic surveys, special studies, and administrative data systems. Environmental data can be ambient air quality, toxic release inventory, ground water sampling, drinking water databases, and meteorological records. Not all types of data have to be recorded. Careful considerations must be carried out about the budget constraints, time and labor, available technology, further updates, security, user-friendliness, etc.

3.3.1 Surveys and Questionnaires

Surveys are a common way of collecting data in public health. A survey, including interviews, questionnaires, and instruments or inventories, collects data of each individual in a sample and attempts to estimate opinions, attitudes, and characteristics of a population.

Questionnaire is a principle data recording method for a survey. A questionnaire is a list of questions or forms to be completed. A well-structured questionnaire can be used as a scientific instrument to obtain data from large numbers of participants quickly. Nowadays, questionnaires can be classified as paper questionnaire, mail questionnaire, and web-based questionnaire (Guest and Namey 2015). Paper questionnaire has higher validity and accuracy, while mail and web-based questionnaire are more convenient and flexible (e.g. SurveyMonkey,[1] MikeCRM,[2] qualtrics,[3] Wen-

[1] https://www.surveymonkey.com/.

[2] http://mikecrm.com/.

[3] https://livpsych.eu.qualtrics.com/.

juanxing[4]). Furthermore, some key points need to be paid attention to when developing the survey questionnaire. Double-barreled questions should be avoided and a combination of positive and negative worded items could be utilized (Rattray and Jones 2007). Additionally, the validity and reliability of the questionnaire should also be illustrated and tested.

A well-constructed questionnaire could provide data that can be easily organized and tabulated for further analysis. Some specific survey platforms could export the data into a form that could be imported in R. For instance, the SurveyMonkey platform could export the data in a .csv file which could be input into R (see Chap. 4).

3.3.2 Desktop Questionnaire Platform

Epi Info is a comprehensive software package for electronic questionnaire design, word-processing, database construction, data analysis, and visualization in public health. It is developed by Centers for Disease Control and Prevention (CDC), US. Epi info is open source and cross-platform for Microsoft Windows, Android, and IOS, as along with an online version OPenEpi. The data can be saved either in Microsoft Access (.mdb) or Microsoft SQL Server Database, which can then be easily imported into R.

The interface of Epi Info for generating a questionnaire is shown in Fig. 3.2. A task bar and a menu bar are displayed on the top of the window, where users can find all the functions, such as creating or opening a project. The main part of the window is divided into two panels aligned side by side. The left panel is the project explorer, including the page names of the questionnaire, *Fields* you can drag to the pages, and *Templates* which combine some fields for common uses. The right panel displays the questionnaire pages as a demonstration.

The demonstrative questionnaire in Fig. 3.2 starts with a title *Parent School Asthma Pre-Intervention Survey*. Sometimes, the title is followed by a motivation paragraph which encourages participants to join and also clarifies the study aims. The main body of the questionnaire contains three sections, including the demographic information, the medical information on Page 2, and the asthma symptoms information on Page 3. The layout of all the fields on the questionnaire is flexible for users. The pages can be printed out directly with a satisfactory layout.

Once the data are recorded in Epi Info, we can do statistical calculations for the sample, clean, transform, and analyze data, and visualize analytical result with tables and charts either in Epi Info or in R.

[4] https://www.wjx.cn/.

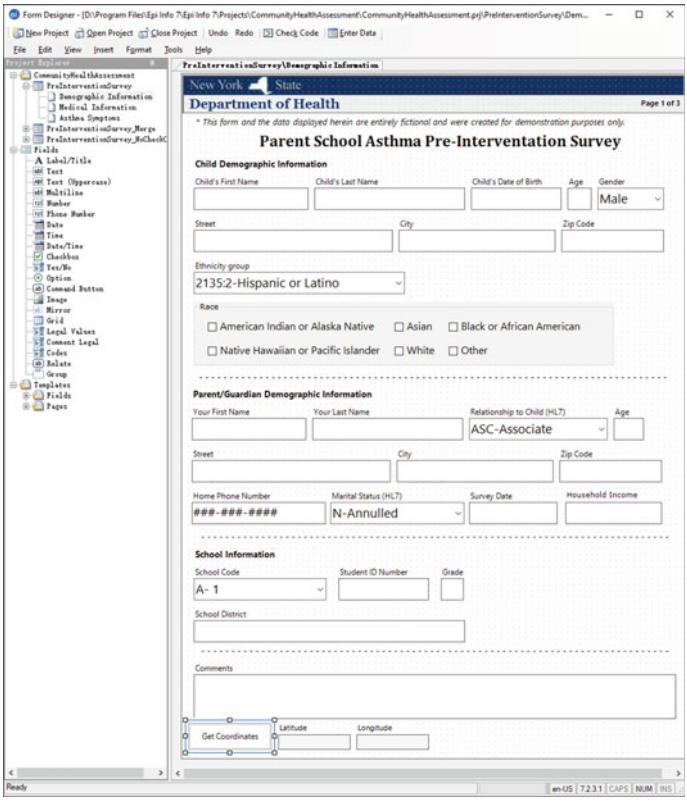


Fig. 3.2 Create a questionnaire with Epi Info

3.3.3 Online Questionnaire Platform

SurveyMonkey is a popular online platform for generating, distributing, and summarizing questionnaires. After signing up for free, you can start creating a questionnaire either from a scratch or from a template. If you have the contacts of the subjects, you can choose distributing your questionnaire to them; otherwise, you can pay for targeted responses around the world.

The interface of creating questionnaire on SurveyMonkey is shown in Fig. 3.3. Similarly to Epi Info, the side-by-side panels display the available components (left) and the questionnaire page. As the questions are listed on webpages, you don't have to think much about the printed-out layout.

The survey results, including the summary data and the individual response data, could be exported in Microsoft Excel (.xlsx), comma separated value (.csv), and SPSS data (.sav). All these formats can be imported into R for further analysis.

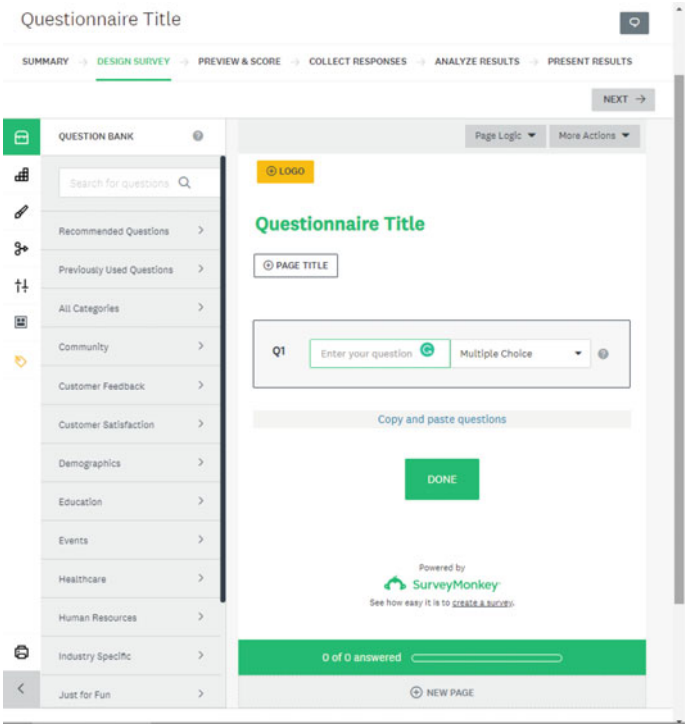


Fig. 3.3 Create a questionnaire with SurveyMonkey

3.3.4 Databases and Tables

After the data have been recorded, an important step is to organize the data in secure database systems in place along with proper data management and quality control procedures.

A database is an organized collection of information consisting of one or multiple tables, which are usually linked to each other. A table is a rectangular structure consisting of rows (also called records) and columns (called fields). For example, a table of patients could have the fields of name, social security number or ID, gender, date of birth, height, etc. Each row is for one patient.

A data dictionary or code book often accompanies a table. The code book defines and names the fields, as well as states the attributes, formats, and permissible ranges of the variables. The attributes clarify whether the variables are numeric, characteristic, logical, date and time. Data coding standard are often used in public health, such as diagnosis codes (ICD-10), medical procedures codes (CPT), national drug codes (U.S. FDA), logical Observations Identifier Names and Codes (LONIC), Systematized Nomenclature of Medicine (SNOMED), and Health Level 7 standard (HL-7).

3.3.5 Using Secondary Data

Secondary data are those collected by someone else, such as other researchers, studies, and organizations. The general sources of secondary data are public records, personal documents, and physical evidence. The public records mainly contain journal articles, official statements, magazines, publications of government or other authority, etc. Sometimes, unpublished sources can also be applied, such as data collected by teachers, professors, and private enterprises. Document review is a typical method to record secondary data. A lot of information regarding public health is available in the public domain. This information includes census data, health statistics, and national survey results. We can use such data for describing public health information and can ask questions regarding possible relationships among variables. As this kind of study depends on these archived data, it is called archival research.

Using secondary data can save much time and money, and it provides possibility for single or a group of researchers to access to data with greater scope and higher quality. For example, census records provide data of entire population of a country. However, secondary data have drawbacks. As it is not us who collect the data ourselves, we must figure out the methodology used to collect the data in details, including whether the sampling procedures are involved and adequate, who and how they collect the data, how the data are stored, and whether there is quality control. We have to find answers to these questions before data exploration and analysis. As those questionnaires designed by others may not be tailored for our research questions, we must keep in mind that the data may not answer the questions for our own research, and the data might be inaccurate or out of date. When using secondary data, we must follow the guide or rules provided by the owner of the data. For instance, when we submit our research proposal or clarify our motivation, we must provide the consent forms or fulfil other requirements of the database.

3.4 Exercises

You conduct a study about mental pressure in undergraduates in your college, which is composed of 150 freshmen, 130 sophomores, 120 juniors, and 100 seniors. Describe how to sample the data with probability sampling methods.

References

Gregg, Michael. 2008. *Field Epidemiology*. USA: Oxford University Press.

Guest, G., and E. Namey. 2015. *Public Health Research Methods*. Book. Thousands Oaks, California: SAGE Publications.

Missouri Department of Health and Senior Services. 2015. "Missouri Nursing Workforce 2015." https://health.mo.gov/living/families/primarycare/pdf/MissouriNursingWorkforce2015.pdf.

Rattray, J., and M. C. Jones. 2007. Essential Elements of Questionnaire Design and Development. *Journal of Clinical Nursing* 16 (2): 234–43. https://doi.org/10.1111/j.1365-2702.2006.01573.x.

Wickham, Hadley, Romain François, Lionel Henry, and Kirill Müller. 2022. *dplyr: A Grammar of Data Manipulation*. https://CRAN.R-project.org/package=dplyr.

Chapter 4
Importing and Exporting Data

4.1 Chapter Highlights

- Import and export common data files with appropriate R functions, and
- Obtain useful datasets from online sources via R packages.

4.2 Import Data Manually

Importing and exporting data can be regarded as the entrance and exit, respectively, for the data work. These two steps connect our data work with the upstream and downstream collaborators.

In most cases in public health, we deal with rectangular datasets (i.e., a table composed with columns and rows, in which all columns have equal rows). Small rectangular datasets can be imported into R by manually typing. Let's use an dataset from a cohort study with the demographic variable 'gender' as an exposure variable and the 'eating disorder' as a response variable (see Table 4.1).

It is a two-by-two contingency table, which you could simply import into R as a data frame like this:

```
dtf_eating <- data.frame(
  Outcome = c('Eating disorder', 'No eating disorder'),
  Girls = c(100, 14900),
  Boys = c(25, 9975))
```

Table 4.1 An example of small rectangular dataset

Outcome	Girls	Boys
Eating disorder	100	25
No eating disorder	14,900	9975

P. Zhao, *Working with Data in Public Health: A Practical Pathway with R*,
https://doi.org/10.1007/978-981-99-0135-7_4

The **tibble** package allows you to type it in a straightforward way:

```
dtf_eating <- tibble::tribble(
  ~Outcome,              ~Girls, ~Boys,
  'Eating discorder',     100,      25,
  'No eating discorder', 14,900,  9975
)
```

Small non-rectangular datasets could be manually imported as vectors or lists. For example, we have Sample 1 of the heights of three people, and Sample 2 of the heights of four people. We could import them as two vectors by typing:

```
sample1 <- c(162, 173, 181)
sample2 <- c(155, 168, 172, 180)
```

or as one list by typing:

```
ls_sample <- list(sample1 = c(162, 173, 181),
                  sample2 = c(155, 168, 172, 180))
ls_sample
```

```
## $sample1
## [1] 162 173 181
##
## $sample2
## [1] 155 168 172 180
```

In most cases, non-rectangular datasets could be transformed into equivalent rectangular datasets:

```
dtf_sample <- data.frame(group = rep(c('sample1', 'sample2'), c(3, 4)),
                         value = c(162, 173, 181, 155, 168, 172, 180))
dtf_sample
```

```
##     group value
## 1 sample1   162
## 2 sample1   173
## 3 sample1   181
## 4 sample2   155
## 5 sample2   168
## 6 sample2   172
## 7 sample2   180
```

Rectangular datasets are recommended in R, as many packages and functions (especially the **tidyverse** family) support them very well.

Manually typing works fine for small datasets or for temporary use. Although it is quite handy, it is tedious, time-consuming and error-prone, especially when dealing with large data sets. No matter they are small or large, it is suggested to save the datasets as local files and import them with the methods described in the subsequent sections.

4.3 Using RStudio Dialogues

Our data are often stored as files in local disks. Most of the data files in public health are stored as plain text or Excel spreadsheets. Some of them are saved or exported by SPSS, SAS, and Stata. R beginners, who are not familiar with the R functions, could use RStudio IDE dialogues for importing these data files into R.

RStudio IDE provides three entrances for importing data files via dialogue boxes. Figure 4.1 shows a screenshot indicating how to do it. We could either (1) click in the menu bar `File - Import Dataset`, or (2) click the button `Import Dataset` in the top-right pane. Click the appropriate data source in the dialogue box that pops up. Then we can browse in the computer and choose the target data file. Alternatively, we could (3) find the target file in the bottom-right pane and click it, which opens a dialogue box with two options for common dataset files: `View File` opens the file in the same way as double click it in the default file browser, while `Import Dataset` import it into R just like the entrances (1) and (2).

Figure 4.2 shows the graphic interface for importing a data file in .xlsx. The dataset is previewed in the upper panel as a data frame. The import options, shown in the bottom panel, allow us to customize:

- `Name`, the name of the object, which the data will be assigned to,
- `Sheet`, where we could choose the sheet name (if there are multiple sheets in the .xlsx file),
- `Range`, the range of the cells to be imported,
- `Max Rows`, the maximum row number to be imported,
- `Skip`, the number of the beginning rows that are skipped,
- `NA`, the strings representing missing values,
- `First Rows as Names`, whether use the first row as column names, and
- `Open Data Viewer`, whether view the data frame in the RStudio IDE Viewer after importing.

The code of these actions is shown in the bottom-right box. The interfaces for importing data files in other formats (such as .csv, SPSS, SAS, Stata) are slightly different, depending on the formats.

Note that the entrance (3) cannot import a .txt file into R; it only opens it in a plain text editor (Fig. 4.3 left), while the entrances (1) and (2) can import a .txt file into R as a data frame (Fig. 4.3 right). We could customize:

- `Name`, the name of the object, which the data will be assigned to,
- `Encoding`, the encoding of the source file, usually UTF-8,
- `Heading`, the heading, i.e. whether to handle the first row as column names,
- `Row names`, whether to name the rows automatically, or with the first column, or with number,
- `Separator`, the column separator, which could be white space, comma, semi-colon, or tab,
- `Decimal`, the decimal separator, which couldeither be period or comma,

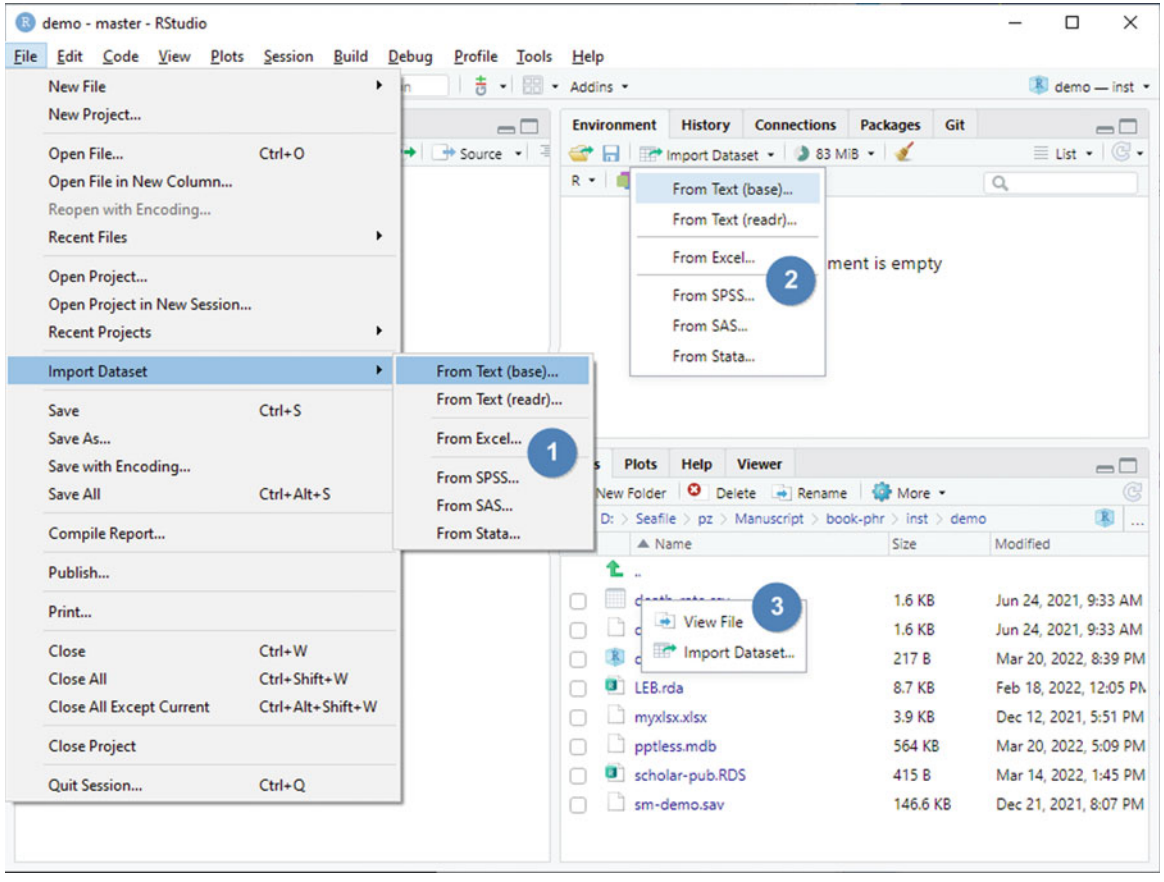


Fig. 4.1 Import data into R via graphical dialogues in RStudio IDE

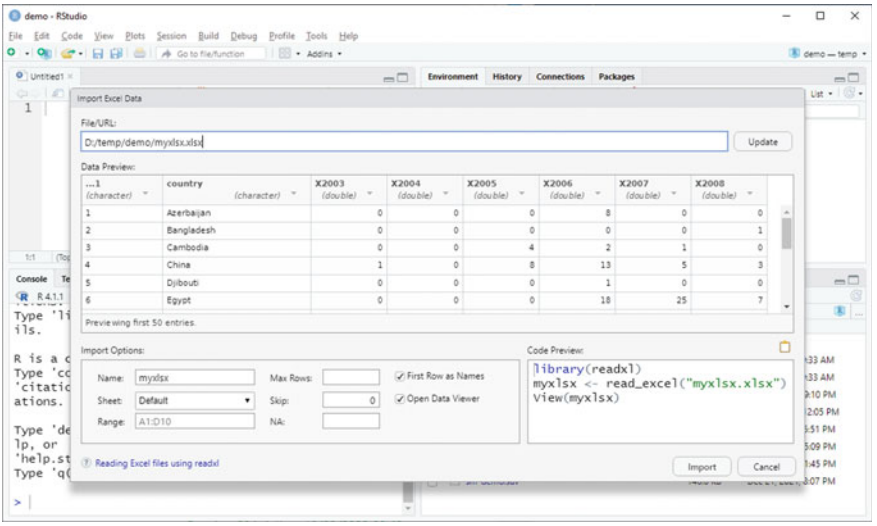


Fig. 4.2 Costomize the settings of importing data into R via graphical dialogues in RStudio

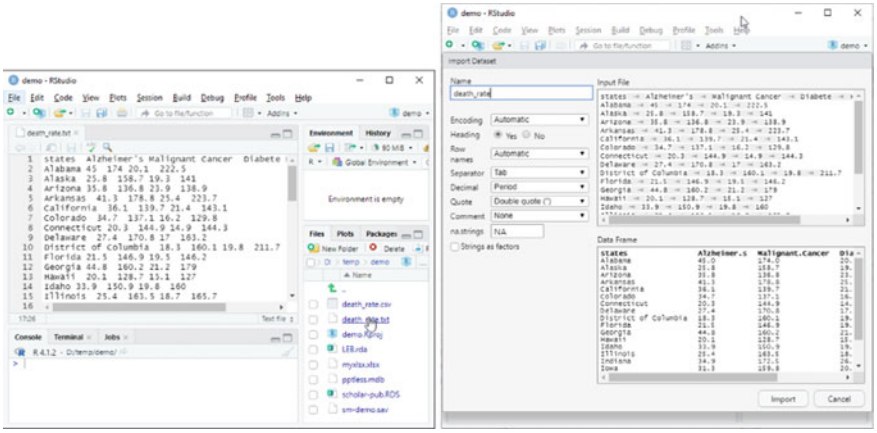


Fig. 4.3 Open/import .txt files via graphical diologues in RStudio IDE

- `Quote`, the quoting character for characters, which could be `"` or `'`,
- `Comment`, the indicator with which the rows are not imported as data, including `#`, `!`, `%`, `@`, `/`, or `~`,
- `na. strings`, the strings representing missing values, and
- `Strings as factors`, whether to convert character vectors into factors.

RStudio dialogues can do most of the importing-data tasks in public health, and this method is convenient and friendly to R beginners. However, it is limited to only a few data formats with limited options. For instance, the options for importing .txt

files are technically a subset of the arguments of the `read.table()` function. Furthermore, the operations of selecting the options and customizing the import settings are not reproducible. We would suggest that R beginners should learn the code provided by the dialogue boxes, and use the functions introduced in the subsequent sections whenever it is applicable.

4.4 Comma/Tab Separated Values

Rectangular spreadsheets are recommended to be saved and distributed in plain text format, such as comma separated values format (.csv) or tab separated ones. These files are software independent and widely supported by a variety of software packages. They are used for exchanging and converting data between various spreadsheet programs for decades. In a .csv file, each line contains one record, ended with a line break, and there is one or more fields separated by commas within the header and each record. Note that each line must contain the same number of fields throughout the file. The tab separated data files are an alternative to .csv, only except that tab separated data files are often friendly to human's eyes.

A .csv file can be imported and exported by the `read.csv()` function and the `write.csv()` function, respectively, and tab separated files by the `read.delim()` function and `write.table(sep = '\t')` function (there is surprisingly no function called `write.delim()` in base R packages), respectively. Technically, `read.csv()` and `read.delim()` are two wrappers of `read.table()`, and `write.csv()` is a wrapper of `write.table()`. Files with other delimiters can be imported/exported via the general data importing/exporting functions `read.table()`/`write.table()`, in which the argument `sep` is used to specify the delimiter to separate data.

Questionnaires on the SurveyMonkey (see Sect. 3.3.3) can be exported as .csv files. Here we use a demonstrative summary file `sm-demo.csv`. Open it with a text editor or spreadsheet viewer. The first two rows act as meta data, which we should skip when importing them:

```
dtf_csv <- read.csv('data/sm-demo.csv', skip = 2, header = FALSE)
# or
dtf_csv <- read.table('data/sm-demo.csv', skip = 2, header = FALSE, sep = ',')
```

We skip the first two rows because they are improper to serve as column names.

The data frame `dtf_csv` can be exported as a new .csv file:

```
write.csv(dtf_csv, 'data/sm-demo-new.csv', row.names = FALSE)
# or
write.table(dtf_csv, 'data/sm-demo-new.csv', row.names = FALSE, sep = ',')
```

The argument `row.names` is set as `FALSE` because they are redundant and we do not need them.

Some external packages can do the similar job in a better way. For example, the **readr** package (Wickham et al. 2022) provides the functions `read_csv()`/`write_csv()`, `read_delim()`/`write.delim()`, and `read_table()` (again, there is surprisingly no function called `write_table()` in **readr**), which not only often run faster (especially for large datasets) than base R functions but also provide other interesting features such as progress bar. Importing data with **readr** is included in the options of the RStudio dialogues as well. Alternatively, the **data.table** package provides a pair of functions `fread()`/`fwrite()`, which uses multiple CPUs, if any, to accelerate the speed, and can automatically detect the delimiter.

4.5 Default Formats in R

The default data formats in R are .rds and .rda/.RData, while .rda is just a short name for .RData.

An .rds file stores one single object, which could be imported/exported with the `readRDS()`/`saveRDS()` functions.

```
bf <- readRDS('data/bf.rds')
saveRDS(bf, 'data/bf2.rds')
```

The .rds file does not include the object name, therefore we can assign the imported data into an object with any name allowed.

An .rda/.RData file stores a workspace with one or multiple objects. It can be imported/exported with a pair of functions `load()`/`save()`. For example, we could check what objects we have in the current workspace:

```
ls()
```

```
[1] "bf"  "dtf_csv"  "dtf_eating" "dtf_sample" "ls_sample" "sample1" "sample2"
```

We could export any of these objects (e.g. `bf`,`ls_sample`, `sample1`) into an .rda file:

```
save(bf, ls_sample, sample1, file = 'data/save.rda')
```

The exported data file contains a mixture of a data frame (`bf`), a list (`ls_sample`), and a vector (`sample1`), with their names. Now let's quit R or RStudio, restart it, and import the save.rda file either with the RStudio menu `Session - Load Workspace` or with the `load()` function:

```
load('data/save.rda')
```

It does not need the assignment operation: The three objects are automatically restored in the R environment. It is convenient, but we must be very careful with it. If the object with the same name is already assigned, the previous value is lost when loading the new one without an explicit message. For example:

```
sample1 <- 1
load('data/save.rda')
```

Then the value `1` for `sample1` is gone.

The `save.image()` function is a wrapper for saving everything in the workspace:

```
save.image('save.rda')
```

The .rds and .rda/.RData data files can preserve the structure of the data in processing them with R. They are often used locally as intermediate products, as they might not be supported by other software. Anyway, they are the safest and stablest formats for sharing the data with other R users.

4.6 Other Software Dependent Data Files

Software dependent data refer to the data formats originated in data-processing software, such as .xlsx in Microsoft Excel and .sav in SPSS. The .rds/.rda/.RData files are software dependent data as well, as they are generated by R. Other software dependent data cannot be imported into R directly by using base R functions.

To import these types of data, one solution is converting them to software-independent format (e.g. .csv) before importing them into R. Usually, the software itself can perform the conversion with a click in its menu (often in *File - Export* or *Save as*).

Alternatively, many R packages are available for different data formats. For example, the package **foreign** (R Core Team 2022) provides a series of function for importing data stored by some versions of 'Epi Info' (`read.epiinfo()`), 'Minitab' (`read.mtp()`), 'S' (`read.S()`), 'SAS' (`read.ssd()` and `read.xport()`), 'SPSS' (`read.spss()`), 'Stata' (`read.dta()`), 'Systat' (`read.systat()`), 'Weka' (`read.arff()`), and 'dBase' (`read.dbf()`) files. Two other packages, **readxl** (Wickham and Bryan 2022) and **xlsx** (Dragulescu and Arendt 2020), can import Microsoft Excel spreadsheets (.xlsx).

Here we use the demonstrative exported files, i.e. sm-demo.xlsx by MicroSoft Excel and sm-demo.sav by SPSS, from SurveyMonkey (see Sect. 3.3.3).

The `read_xlsx()` function in the **readxl** package and the `read.xlsx()` function in the **xlsx** package can import the .xlsx file:

```
dtf_xlsx <- xlsx::read.xlsx("data/sm-demo.xlsx", sheetIndex = 1, header = FALSE, startRow = 3)
# or
dtf_xlsx <- readxl::read_xlsx("data/sm-demo.xlsx", col_names = FALSE, skip = 2)
```

Compared to **readxl**, **xlsx** can not only import .xlsx files into R, but also export data into .xlsx files:

```
xlsx::write.xlsx(dtf_xlsx, 'data/sm-demo2.xlsx')
```

The **xlsx** package, however, depends on Java. The right version of Java (32 bit or 64 bit) must be pre-installed before you can properly load **xlsx**.

The `read.spss()` function in the **foreign** package can import the .sav file:

```
dtf_spss <- foreign::read.spss('data/sm-demo.sav', to.data.frame = TRUE)
```

The Epi Info questionnaire data (see Sect. 3.3.2) are saved either in Microsoft Access (.mdb) or SQL database. Here we give an example for the .mdb database. The **RODBC** package (Ripley and Lapsley 2021) provides a series of function for fetching data from .mdb database:

```
library(RODBC)
mymdb <- odbcConnectAccess2007("data/pptless.mdb")
sqlTables(mymdb)
dtf_mdb <- sqlFetch(mymdb, "pgcert1")
```

To our experience, if you do not know how to import a data file into R, you can search the internet with the key words "R language import data" with your desired format, and you will be able to find the solution.

4.7 Online Sources

We can download secondary data from online source with the help of R packages. For example, the **WHO** package is an R client for the World Health Organization (WHO) API. **WHO** provides access to the datasets from WHO. You can install it:

```
remotes::install_github("expersso/WHO")
```

There are two functions in **WHO**: `get_codes()` gets all codes and metadata for WHO series:

```
library(WHO)
codes <- get_codes()
```

In the object `codes` there are 2808 datasets available with a label, a brief description, and a URL for each. If you want to import the dataset *Under-five mortality rate*, which is labeled as 'MDG_0000000007', then apply the `get_data()` function to the label:

```
whod <- get_data("MDG_0000000007")
```

Another example is the package **nCov2019**, which can download the COVID-19 pandemic data.

```
remotes::install_github("GuangchuangYu/nCov2019")
library(nCov2019)
ncovr_data <- load_nCov2019()
```

As each of these R package is developed by different authors, inconsistency in the usage of these functions might exist, including the argument types of the functions and the formats of the returned values. Thus, we have to read the help document carefully before using them.

4.8 Data Shipped with R Packages

Some datasets are shipped with R packages, which means that those datasets are installed into your computer simultaneously when you install the packages. They can be used as demonstrations in teaching and learning. The function `data()` loads them into R and the `help()` function displays the metadata. See Chap. 1 for the details.

Some packages, however, ship some datasets in a folder named as `extdata` in the installation path. For example, in the installation path of the **MSG** package (Xie and Zhao 2021), we can see the shipped dataset files:

```
list.files(file.path(system.file(package = "MSG"), 'extdata'))
```

```
##  [1] "AgriComp.R"        "ChinaPop.R"        "cities.txt"        "graphnr.csv"
##  [5] "HighFreq100.rda"   "Napoleon.csv"      "SongWords.rda"     "stem-islands.txt"
##  [9] "stem-poisson.txt" "troops.txt"
```

These data files could differ in formats, and we should use the importing approaches described in the previous sections. The complete path of such a file could either be located with the file browser or with the following functions:

```
file.path(system.file(package = "MSG"), 'extdata', 'covidcountries.rds')
```

```
## [1] "D:/R/library/MSG/extdata/covidcountries.rds"
```

We could develop our own R packages for shipping datasets. See Sect. 10.5.5.

4.9 Exercises

1. Apart from `read.csv()` and `read.delim()`, there are more wrappers for `read.table()`, such as:

```
read.delim2()
read.csv2()
read.fwf()
```

   Find out what each of them is used for, and what is the difference between them.
2. A demonstration dataset "sm-demo.csv" was downloaded from the Survey Monkey. Import it into R with the RStudio dialogue. There are two options: (1) From Text (base), and (2) From Text (readr). What is the difference between them? After importing it, export this dataset as .xlsx document.

Table 4.2 Meta data for the *births* dataset

Names	Description	Unit
id	Identity number for mother and baby	
bweight	Birth weight of baby	Grams
lowbw	Indicator for birth weight less than 2500 g	
gestwks	Gestation period	Weeks
preterm	Indicator for gestation period less than 37 weeks	
matage	Maternal age	Years
hyp Indicator for maternal hypertension		
sex	Sex of baby: 1:Male, 2:Female	

3. The *births* dataset from the **Epi** package contains data from 500 singleton births in a London Hospital. See the details about it in the help document. Generate a data frame as the metadata like Table 4.2 with a spreadsheet programme (e.g. Excel or LibreOffice).
4. The China Health and Nutrition Survey (CHNS) is an ongoing international collaborative project for investigating how China's social and economic transformations are influencing the population's health and nutritional condition. This project provides a website[1] where we could obtain survey datasets with their documents. Download a dataset of your interest. What format is it? Import it into R.

References

Dragulescu, Adrian, and Cole Arendt. 2020. *xlsx: Read, Write, Format Excel 2007 and Excel 97/2000/XP/2003 Files*. https://github.com/colearendt/xlsx.

R Core Team. 2022. *foreign: Read Data Stored by Minitab, S, SAS, SPSS, Stata, Systat, Weka, dBase,...*https://CRAN.R-project.org/package=foreign.

Ripley, Brian, and Michael Lapsley. 2021. *RODBC: ODBC Database Access*. https://CRAN.R-project.org/package=RODBC.

Wickham, Hadley, and Jennifer Bryan. 2022. *readxl: Read Excel Files*. https://CRAN.R-project.org/package=readxl.

Wickham, Hadley, Jim Hester, and Jennifer Bryan. 2022. *readr: Read Rectangular Text Data*. https://CRAN.R-project.org/package=readr.

Xie, Yihui, and Peng Zhao. 2021. *MSG: Data and Functions for the Book Modern Statistical Graphics*. https://github.com/yihui/MSG.

[1] https://www.cpc.unc.edu/projects/china.

Chapter 5
Cleaning Data

5.1 Chapter Highlights

- Understand the necessity and importance of data cleaning,
- Describe the common steps for data cleaning, and
- Use the relevant R packages and functions for data cleaning in research.

5.2 Introduction

The major work with data is focused on summarizing, exploring, analyzing, and modelling data under the assumption that the input data is clean, which means that the data is error-free, in correct formats, with no missing fields, and with correct labels and codes, or, briefly speaking, ready for further analysis. Unfortunately, this does not happen in the real-world data we get in public health. Typically, we get a data file with no headers, incorrect data types, incorrect category labels, and unknown or unexpected character encoding. Even after we've resolved these concerns and obtained theoretically valid data, there are still certain errors, such as a negative age, a child who has a driver's licence, or data that are missing.

Data cleaning, which converts raw data into consistent data that can be further analyzed, is one of the most important processes in data work. The goal of data cleaning is to increase the reliability of subsequent analysis and outcomes. While data cleaning might not be the core of data work in public health, it shouldn't be disregarded because it has the power to make or break statistics.

Data cleaning in practice is often more time-consuming than data exploration and analysis in public health. A list of recommended practices to keep in mind when cleaning raw data will help us avoid missing any of the critical data cleaning phases to the greatest extent feasible (Van der Loo and De Jonge 2018).

P. Zhao, *Working with Data in Public Health: A Practical Pathway with R*,
https://doi.org/10.1007/978-981-99-0135-7_5

In the subsequent sections, we demonstrate the common steps with R with a small example. We'll go over the data cleaning list step by step with a demo data file *data_cleaning_example1.txt*, which includes three columns (age, body height, gender) in a study.

```
myfile <- "data/data_cleaning_example1.txt"
```

5.3 File Information

Before getting our hands dirty with data cleaning, we must first know the dataset well, from file sizes to data types. If applicable, a quick visual inspection would give us a general impression about how the dataset looks like.

First of all, we use the `file.info()` function to see the size of the data file before opening it:

```
file.info(myfile)
```

```
##                                size isdir mode               mtime
## data/data_cleaning_example1.txt   85 FALSE  666 2022-04-01 15:49:45
##                                           ctime               atime exe
## data/data_cleaning_example1.txt 2022-04-03 17:49:58 2022-12-13 22:02:00  no
```

The output information gives us an impression about:

- `size`, the file size (85 bytes),
- `isdir`, whether it is a directory or not,
- `mode`, the file permissions (`666` means all users can read and write the file[1]),
- `mtime`, the last modification time,
- `ctim`, the last status change time,
- `atime`, the last access time, and
- `exe`, the sort of executable.

Apart from `file.info()`, other functions, including `file.size()`, `file.mode()`/ `file.access()`, and `file.mtime()`, can check the file size, permissions, and the last modification time, respectively. They help us understand the details about the data file and make a decision for the next step: If the size is too large (e.g., `size` in GB), it might be not applicable to open it with Excel. If the dataset was shared by someone else long time ago while it has recently been modified (`mtime`), it might be not the very file we intend to import.

[1] The three digits indicate the file permission for the file owner, the owner's group, and other users. Each digit indicate the *read* (or *r*, which adds 4 to its total), the *write* (or *w*, which adds 2), and the *execute* (or *x*, which adds 1 to its total) permission. As $6 = 4 + 2$, it means the users have the *read* and *write* permissions. More details refer to https://en.wikipedia.org/wiki/File-system_permissions.

In most cases in public health, the contents in the data files could be visually inspected before importing them. As the size of the demonstration file is small, we could now double click it with the file browser and open it with the default programme such as Excel for .xlsx files. We can alternatively use the function `file.show()` to open it:

```
file.show(myfile)
```

As the demo file is a .txt file, it is opened with the default text file editor (while a .csv file is often opened by Excel). We can see that there is no header (column names) in the file, there are six rows, and the columns are separated with comma, which determines how to import it into R (see Chap. 4): using `read.csv()`, `data.table::fread()`, or `readr::read_csv()`:

```
ex1 <- read.csv(myfile, header = FALSE)
```

5.4 Dimensions and Structure

After importing the data, we usually would like to see a whole picture of the dataset. It is often performed by clicking the arrow before the object name `ex1` in the top-right *Environment* pane or with the `str()` function:

```
str(ex1)
```

```
## 'data.frame':    6 obs. of  3 variables:
##  $ V1: int  32 20 19 21 -9999 23
##  $ V2: chr  "5.8" "62feet" "5.6" "6.1" ...
##  $ V3: chr  " M" " M " " Female" " M" ...
```

It shows the dimension (6 observations/rows of 3 variables/columns), the data structure type (data frame), the column names (`V1`, `V2`, `V3`), and the data type of each column (integer for `V1` and character for `V2` and `V3`). Note that if we import the data with other functions than `read.csv()`, the `str()` function returns the information in different formats. The function `readr::read_csv()` itself not only imports the data, but also returns a message of the data structure, including the dimension and the variable type of each column.

The dimension and the column names could alternatively obtained with the `dim()` and the `names()` functions:

```
dim(ex1) # dimension
```

```
## [1] 6 3
```

```
names(ex1) # or colnames(ex1)
```

```
## [1] "V1" "V2" "V3"
```

The `str()` function returns the first few values of each column as well. It won't hurt anyone if we double inspect the entire dataset either by printing the data in the R console:

```
ex1
```

```
##      V1     V2      V3
## 1    32    5.8       M
## 2    20 62feet      M
## 3    19    5.6  Female
## 4    21    6.1       M
## 5 -9999    5.7     fem
## 6    23    5.6  Female
```

or double clicking the object name `ex1` in the top-right pane of the RStudio IDE, or running the `View()` function:

```
View(ex1)
```

In the data viewer, we could stroll the window horizontally or vertically, click a column name for ordering, search with a certain keyword in the search bar, or filter any row(s) with the `Filter` button. We can now see some structural flaws in the dataset, including:

1. mislabelled variables (i.e. the columns are not named in the raw dataset and the names `V1`, `V2`, and `V3` are automatically named when the dataset is imported),
2. erroneous data type (i.e. the body height `V2` should be numeric rather than character), and
3. text inconsistencies (i.e. *female* is recorded in multiple forms).

The text inconsistencies might be not easy to detect, especially for large datasets. The `unique()` function could show all the elements in an object with duplicated ones removed:

```
unique(ex1$V3)
```

```
## [1] " M"      " M "     " Female" " fem"
```

For showing the unique elements in each column, we use `sapply()` or `lapply()`:

```
sapply(ex1, unique) # or lapply(ex1, unique)
```

```
## $V1
## [1]    32    20    19    21 -9999    23
##
```

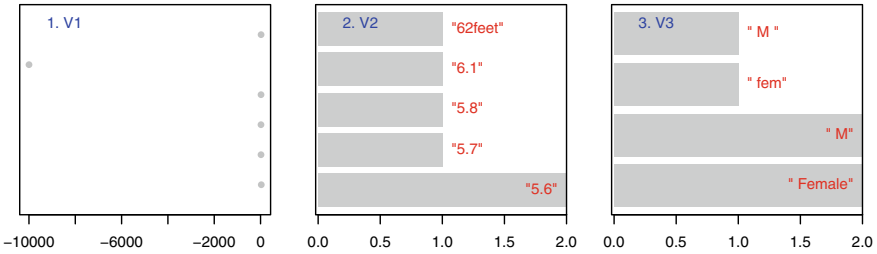


Fig. 5.1 Demonstration of visualizing unique values in each column of a data frame

```
## $V2
## [1] "5.8"    "62feet" "5.6"    "6.1"    "5.7"
##
## $V3
## [1] " M"      " M "     " Female" " fem"
```

We can see that there are actually leading and trailing white spaces in the strings! R treats " `M`" and " `M` " as two different strings. They can hardly be detected by eyes in the data viewer or external software (such as Excel). As we often use categorical data in public health, such text inconsistencies often exist, including leading and trailing white spaces, double spaces between words, mixing with '`l`' (the lower case of '`L`') and `I` (the upper case of '`i`').

The **pzhaonet/fecitr** package (Zhao 2022) provides a function `plot_summary()`, which draws a graph for each column individually (Fig. 5.1):

```
fecitr::plot_summary(ex1, if_quote = TRUE)
```

The variable names are shown as the titles for the sub-figures of Fig. 5.1. The numeric column (`V1`) is visualized as a one-dimensional scatter plot, and character columns (`V2` and `V3`) are visualized as bar charts that show the frequency of each unique string. The three structural flaws can all be seen in it.

5.5 Mislabelled Variables

Data analysis could be annoyed or even misled by mislabelled variables. The column names are assigned as `V1`, `V2`, and `V3` automatically when importing the dataset with `read.csv()` as we set `header = FALSE`. We can rename them:

```
names(ex1) <- c("age", "height", "gender")
```

The **dplyr** package renames the columns in another way, which is sometimes more explicit:

```
library(dplyr)
ex1 <- rename(ex1, age = V1, height = V2, gender = V3)
```

Alternatively, the renaming step can be included in the importing step:

```
ex1 <- read.csv(myfile, col.names = c("age", "height", "gender"))
```

Section 10.5.3 introduces the `sjlabelled::var_label()` function (Lüdecke 2022), which adds a label as attribute to the variable. We could use it in this step as:

```
ex1 <- sjlabelled::var_labels(ex1,
                              age = "years",
                              height = "inches")
str(ex1)
```

```
## 'data.frame':    6 obs. of  3 variables:
##  $ age   : int  32 20 19 21 -9999 23
##   ..- attr(*, "label")= chr "years"
##  $ height: chr  "5.8" "62feet" "5.6" "6.1" ...
##   ..- attr(*, "label")= chr "inches"
##  $ gender: chr  " M" " M " " Female" " M" ...
```

5.6 Incorrect Data Types

It is sometimes confusing when we talk about *data types*. In statistics, the term *data type* usually refers to the classification of quantitative/numerical and qualitative/categorical data. In R, *data type* is often mixed with *data structure type* (also called *object type*), probably because of the similarity in their callings and some confusing functions which return sometimes the data types and sometimes the object types (we will talk about this later). The *object type* refers to the classification of vector, matrix, array, data frame, list, and so on. A vector is a sequence of data elements of the same basic (also called *atomic*) data type (also called *data class*). The atomic data type of a vector is very important because it defines what we could do with the object. There are six atomic data types in R, including *numeric*, *integer*, *logical*, *character*, *complex*, and *raw*, among which we often process the first three ones, i.e.:

- numeric values (e.g., `1`, `2.4`, `-6`), with which we could do mathematical operations,
- logical values (e.g., `TRUE`, `FALSE`), with which we could do logical operations, and
- character values (e.g., `'hello'`, `'data/data_cleaning_example1.txt'`, `'Female'`), which we could split and concatenate. Character values are often recorded as factors.

Vector is the simplest object type, and all other object types are built on vectors in R. Briefly speaking,

- a matrix is similar to a vector (i.e., a sequence of data elements of the same basic data type) but organized in row(s) and column(s),

- an array is similar to a vector (i.e., a sequence of data elements of the same basic data type) but organized in more than one dimensions,
- a data frame is one vector or a combination of multiple vectors with the same length in the same or different data types, organized in row(s) and column(s), and
- a list is a wrapped combination of one or multiple objects in any types.

Thus, a matrix is a special case of an array, and a data frame is a special case of a list.

Let's get back to our demonstration. The object type and the data type of each column in `ex1` could be obtained separately with the following functions:

```
class(ex1) # object type
```

```
## [1] "data.frame"
```

```
class(ex1$age)
```

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

The data type "`integer`" is a subset of "`numeric`".

R provides a series of functions for logically checking the object types and data types. For example, the functions `is.matrix()`, `is.array()`, `is.data.frame()`, and `is.list()` check if an object is a matrix, an array, a data frame, and a list, respectively:

```
is.matrix(ex1)
```

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

```
is.data.frame(ex1)
```

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

```
is.list(ex1)
```

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

and the functions `is.numeric()`, `is.logical()`, and `is.character()` check if an atomic data type is numeric, logical, and character, respectively:

```
is.numeric(ex1$age)
```

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

```
is.character(ex1$age)
```

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

Type `is.` in RStudio IDE and we can see a list of relevant functions.

A convenient and fast way for checking the data type of each column in a data frame is:

```
sapply(ex1, class) #class of each column
```

```
##         age      height      gender
##   "integer" "character" "character"
```

R functions, when importing the data, are usually able to automatically classify the variables into correct data types. The data types for age and gender are properly classified, but the type of height is improper: It should be numeric, while it is incorrectly recognized as character (`chr`). The `read.csv()` sees a value `5.7*` with a redundant `*` and converts the whole column of age into characters.

Object and data types are usually corrected or converted with R functions beginning with `as.`. Similarly to the names of the `is.` functions, the functions `as.matrix()`, `as.array()`, `as.data.frame()`, and `as.list()` convert an object to a matrix, an array, a data frame, or a list, which is explicitly indicated in the function name:

```
as.matrix(ex1)
```

```
##      age     height   gender
## [1,] "   32" "5.8"     " M"
## [2,] "   20" "62feet" " M "
## [3,] "   19" "5.6"     " Female"
## [4,] "   21" "6.1"     " M"
## [5,] "-9999" "5.7"     " fem"
## [6,] "   23" "5.6"     " Female"
```

```
as.list(ex1)
```

```
## $age
## [1]    32    20    19    21 -9999    23
## attr(,"label")
## [1] "years"
##
## $height
## [1] "5.8"    "62feet" "5.6"    "6.1"    "5.7"    "5.6"
## attr(,"label")
## [1] "inches"
##
## $gender
## [1] " M"      " M "     " Female" " M"      " fem"    " Female"
```

and the functions `as.numeric()`, `as.character()`, and `as.logical()` convert the data type into numeric, character, or logical, respectively:

```
as.character(ex1$age)
```

```
## [1] "32"    "20"    "19"    "21"    "-9999" "23"
```

```
as.numeric(ex1$height)
```

```
## [1] 5.8 NA 5.6 6.1 5.7 5.6
## Warning message:
## NAs introduced by coercion
```

The warning indicates that the string `62feet` cannot be converted into numeric with the `as.numeric()` function. Instead, a simple way to remove non-numeric characters before or after the number is the `readr::parse_number()` function:

```
ex1$height_cor <- ex1$height
ex1$height_cor <- readr::parse_number(ex1$height_cor)
class(ex1$height_cor)
```

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

The **readr** package provides a series of functions beginning with `parse_` for different objectives, which are self-explanatory in the function names:

- Parsing numeric values:
  - `parse_double()`
  - `parse_integer()`
  - `parse_number()`
- Parsing logical values:
  - `parse_logical()`
- Parsing character values:
  - `parse_character()`
  - `parse_factor()`
- Parsing temporal values:
  - `parse_date()`
  - `parse_datetime()`
  - `parse_time()`
- Parsing automatically:
  - `parse_guess()`

The type of gender in the demonstration data should be categorized as a factor, while the values are inconsistent. We will correct the data types after the text inconsistency problem is solved.

5.7 Text Inconsistencies

People often record one single thing in multiple ways, which computers do not like for processing data. As categorical variables are often used in public health, text inconsistencies in the raw data are common, which causes problems with downstream data processing. We can see this inconsistency in the demonstrative dataset. The variable gender is recorded as "`M`" and "`M `" for "male", and "`Female`" and "`fem `" for "female". Both within-group inconsistency and between-group inconsistency exist. The aim of this step in cleaning data is to use only "`male`" and "`female`" as the values for gender.

For a small dataset, it is straightforward and fast if we simply correct them manually:

```
ex1$gender_cor <- c('male', 'male', 'female', 'male', 'female', 'female')
```

But for a large dataset, it might be labour consuming and fallible. Thus, it is suggested to use relevant functions.

As mentioned before, leading and trailing white spaces are very difficult to detect by eyes. They can be removed with the argument `strip.white = TRUE` in `read.csv()` when importing data.[2] Alternatively, this job could be done with the functions `readr::parse_character()`, or `stringr::str_trim()`, or the `trimws()` function:

```
trimws(ex1$gender)
```

```
## [1] "M"      "M"      "Female" "M"      "fem"    "Female"
```

Most generally, the fundamental function for character replacement is `gsub()`:

```
ex1$gender_cor <- gsub("(^[[:space:]]+|[[:space:]]+$)", "", ex1$gender)
unique(ex1$gender_cor)
```

```
## [1] "M"      "Female" "fem"
```

where `[:space:]` is a pre-defined character class that matches space characters. `+` means one or more continuous identical characters. `^` and `$` mean matching the beginning and the end of the string, respectively.

The matching and replacement of characters in R is powerful and complicated. Base R provides a series of functions to do this job, including `grep()`, `grepl()`, `agrep()`, `agrepl()`, `regexpr()`, `gregexpr()`, `regexec()`, and `gregexec()` for matching, `sub()` and `gsub()` for replacement. These functions are able to do all the character correction jobs, while they might not be easy for

[2] By default, the `readr::read_csv()` and `data.table::fread()` functions trim the leading and trailing spaces from each field before parsing it.

beginners. The **stringi** (Gagolewski 2022) and **stringr** (Wickham 2022) packages provide a series of wrapper functions for common use.

Now we have only three different values for gender: "M", "fem", and "Female". It is fine to replace them using the `ifelse()` function, as "M" stands for "male" and all other values are "female":

```
ifelse(ex1$gender_cor == "M",
       "male",
       "female")
```

```
## [1] "male"   "male"   "female" "male"   "female" "female"
```

A more explicit way is using the `switch()` function:

```
sapply(ex1$gender_cor,
       function(x)
         switch(x,
                "M" = "male",
                "Female" = "female",
                "fem" = "female")
)
```

```
##        M        M   Female        M      fem   Female
##   "male"   "male" "female"   "male" "female" "female"
```

If there is a more complicated mapping, the function `match()` works better with a lookup table:[3]

```
tbl_ref <- data.frame(
  tibble::tribble(
    ~old,    ~new,
    'm',     'male',
    'man',   'male',
    'male',  'male',
    'f',     'female',
    'fem',   'female',
    'woman','female',
    'female','female'
  )
)
ex1$gender_cor <- tbl_ref[match(tolower(ex1$gender_cor), tbl_ref$old), 'new']
```

The table `tbl_ref` contains all the possible mapping strings. Here we use the `tolower()` function for converting the original characters into lower-case characters which can be found in the lookup table, and the `match()` function returns the index of the target in the table.

When the text is consistent, we can convert the character variable, if it is used to categorize the data, into a factor:

[3] In practice, we could create this table in a spreadsheet program like Excel.

```
ex1$gender_cor <- as.factor(ex1$gender_cor)
str(ex1)
```

```
## 'data.frame':    6 obs. of  5 variables:
##  $ age       : int  32 20 19 21 -9999 23
##   ..- attr(*, "label")= chr "years"
##  $ height    : chr  "5.8" "62feet" "5.6" "6.1" ...
##   ..- attr(*, "label")= chr "inches"
##  $ gender    : chr  " M" " M " " Female" " M" ...
##  $ height_cor: num  5.8 62 5.6 6.1 5.7 5.6
##  $ gender_cor: Factor w/ 2 levels "female","male": 2 2 1 2 1 1
```

5.8 Anomalies

Anomalies in data can exist in two forms: incorrect or invalid values (i.e., values that make no sense) and outliers (i.e., extremely skewed values that are either too high or too low). Anomalies are different from text inconsistencies: inconsistent text could be easily corrected, while anomalies could not. For example, we can infer that the gender recorded as "fem" means "female" exclusively, while we cannot know exactly what gender it is from a recoded value of "Z". Therefore, we should detect the anomalies first, and then decide what to do with them.

Incorrect or invalid values

In the demonstration dataset, the value `-9999` is apparently invalid for the human age. This might come from a missing value. We can either replace it with `NA` individually:

```
ex1$age_cor <- ex1$age
ex1[ex1$age_cor == -9999, 'age_cor'] <- NA
```

or, more generally, set a range for meaningful values (e.g. from 0 to 150 years for human's age) and exclude those beyond the range:

```
ex1[ex1$age_cor < 0 | ex1$age_cor > 150, 'age_cor'] <- NA
```

Unlike explicitly impossible values that should be marked undoubtedly, some values might be inferred according to some supplemental knowledge. For example, although it makes no sense as human's body height, the value "62" might be 6.2 with a missing decimal point, according to the context. We must be careful about these values if we replace "62" with "6.2" temporarily, and it must be kept in mind in further analysis. A good practice is making a note for future reference. Here we generate a new column for recording the note:

```
ex1$note <- NA
ex1[ex1$height_cor == 62, 'note'] <- 'add a decimal point'
ex1[ex1$height_cor == 62, 'height_cor'] <- 6.2
```

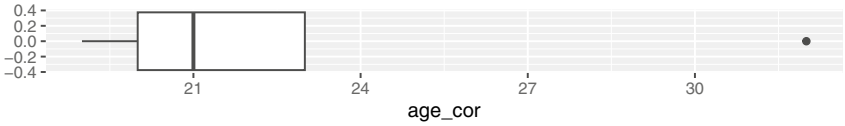


Fig. 5.2 Check outliers with the box plot

Outliers

Outliers are the data points that differ considerably from other values. They do not necessarily mean errors, but their existence could cause serious problems in statistical analysis.

A common and universal method for detecting outliers in unimodal and symmetrically distributed data is the box plot (see Sect. 8.7): The upper threshold is computed as the third quartile (Q_3) plus the interquartile range (IQR) times a multiplier, i.e. $Q_3 + k \cdot IQR$ and the lower threshold as the first quartile (Q_1) minus it, i.e. $Q_1 - k \cdot IQR$. Then the data beyond the range of these two thresholds are labelled as outliers. In a box plot, the outliers are indicated as points outside of the box. We could use the base R function `boxplot()` or the `geom_boxplot()` function from the **ggplot2** package:

```
library(ggplot2)
ex1 |>
  ggplot() +
  geom_boxplot(aes(age_cor))
```

The value "32" is marked as an outlier, which does not mean that this value is so wrong that we must remove it. It only indicates that this value might be different from other values and might have severe influence on further analysis (Fig. 5.2).

The function `boxplot.stats()` can numerically detect the box-plot-based outliers:

```
ex_box <- boxplot.stats(ex1$age_cor)
ex_box
```

```
## $stats
## [1] 19 20 21 23 23
##
## $n
## [1] 5
##
## $conf
## [1] 18.88021 23.11979
##
## $out
## [1] 32
```

where the returned results include `stats` (the five values of the lower threshold, Q_1, the median, Q_3 and the upper threshold), `n` (the number of non-outlier data points), `conf` (the confidence interval, i.e. median $\pm 1.58 IQR/\sqrt{n}$), and `out` (the values of

the outliers). We could generate a new column in the data frame `ex1` for labelling the outlier:

```
ex1$age_outlier <- 0
ex1$age_outlier[ex1$age_cor %in% ex_box$out] <- 1
```

The multiplier k is arbitrary, usually assigned as 1.5 by default in many functions. They can be changed, depending on how relaxed we would like to detect the outliers. We could enlarge the tolerance window for non-outlier values with a larger k:

```
boxplot(ex1$age_cor, range = 3)
ex1 |>
  ggplot() +
  geom_boxplot(aes(age_cor), coef = 3)
boxplot.stats(ex1$age_cor, coef = 3)
```

The **outliers** package (Komsta 2022) provides a series of functions for χ^2-test (`chisq.out.test()`) , Cochrant test (`cochran.test()`), Dixon test (`dixon.test()`), and Grubbs test (`grubbs.test()`) for outliers. It also provides a handy function `outlier()` for finding the values far from the mean:

```
outliers::outlier(ex1$age_cor)
```

```
## [1] 32
```

Other packages, such as **anomalize** (Dancho and Vaughan 2020), **extremevalues** (van der Loo 2020), and **mvoutlier** (Filzmoser and Gschwandtner 2021), as well as other functions, such as `car::outlierTest()` (Fox et al. 2022), provide a number of methods based on a variety of models (e.g. distance, dispersion, depth or density based) for outlier detection in univariate and multivariate data, which can be used for more complicated situations.

5.9 Missing Values

Missing values often take place due to skipped questions in questionnaires used in public health (i.e., if one replies "yes" to a question, one can skip the some subsequent question) rather than manual mistakes. If our dataset has such missing values, we have to make a decision whether to preserve these observations for future work.

People or computer software often record missing data with impossible values or labels as indicators. The indicators for missing values should be clarified in the protocol during the data plan phase. The following indicators are commonly used for missing values: `9999999` (either fewer or more 9s, negative or positive), `NA`, `N.A.`, `Not Available`, `-`, `/`, blank.

Missing values are recorded as NA (not characters) in R. In a consistent way, many functions allow users to decide how to treat missing values with a logical argument na.rm, which determines whether NAs are removed before the function proceeds. For instances, compare:

```
mean(ex1$age_cor)
```

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

```
mean(ex1$age_cor, na.rm = TRUE)
```

```
## [1] 23
```

Here is an incomplete collection of the functions that support na.rm:

```
mean()
sd()
var()
sum()
colSums()
rowSums()
colMeans()
rowMeans()
```

Therefore, it is suggested to replace all the NA indicators in the raw data with NA before further analysis.

If we know the indicators in advance, we could mark the NA values when importing the data with the na.strings argument (suppose we use both "-9999" and "N.A." for NAs):

```
read.csv(myfile,
         header = FALSE,
         na.strings = c("-9999", "N.A."))
```

The function anyNA() finds whether at least one missing value exists:

```
anyNA(ex1)
```

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

It can be used for finding which rows or columns contain NAs:

```
which(apply(ex1, 1, anyNA))
```

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

```
which(apply(ex1, 2, anyNA))
```

```
## age_cor    note
##       6       7
```

The `is.na()` function returns logical values about whether an element is NA in a vector, matrix, array, or a data frame:

```
is.na(ex1)
```

```
##        age height gender height_cor gender_cor age_cor  note age_outlier
## [1,] FALSE  FALSE  FALSE      FALSE      FALSE   FALSE  TRUE       FALSE
## [2,] FALSE  FALSE  FALSE      FALSE      FALSE   FALSE FALSE       FALSE
## [3,] FALSE  FALSE  FALSE      FALSE      FALSE   FALSE  TRUE       FALSE
## [4,] FALSE  FALSE  FALSE      FALSE      FALSE   FALSE  TRUE       FALSE
## [5,] FALSE  FALSE  FALSE      FALSE      FALSE    TRUE  TRUE       FALSE
## [6,] FALSE  FALSE  FALSE      FALSE      FALSE   FALSE  TRUE       FALSE
```

We can check how many NAs in total in the dataset:

```
sum(is.na(ex1))
```

```
## [1] 6
```

For a large dataset, remove the rows that have NA values may be an appropriate strategy. The `na.omit()` function can do this job:

```
ex2 <- na.omit(ex1)
```

However, we have to be mindful that vital information may be lost.

There are lots of R package available for dealing with missing data. The CRAN Task View page collects a list of these package and introduces them in categories for a variety of usages.[4]

5.10 Other Operations

The previous sections introduce common steps for data cleaning with a very small demonstration dataset in public health. In fact, other operations might be necessary, such as checking duplicated identifiers, converting date/time records, and transforming character encoding (especially for the languages using multibyte characters such as Chinese). Data cleaning is quite case specific, depending on what data we have in a certain project and what the research objectives are. It is always recommended to make a protocol for data cleaning during the planning stage and follow it throughout the data workflow.

[4] https://cran.r-project.org/web/views/MissingData.html.

5.11 Exercises

1. The R function `class()` returns the class of an object. There are two other functions `typeof()` and `mode()` which do similar jobs. What are the differences between them? Compare them with the following code:

```
lapply(ex1, class)
lapply(ex1, typeof)
lapply(ex1, mode)
```

2. The **deducorrect** package (van der Loo et al. 2015) is a collection of methods for automated data cleaning. What kind of data cleaning work can it do? Give examples and discuss its usage in your research.
3. A study is conducted to investigate the prevalence of certain mental health disorders within the tech industry in 2016. The data set is available by the Open Sourcing Mental Illness, Ltd.[5] It contains 1433 observations and 63 variables from questionnaires. Import it into R and carry out the data cleaning procedure.

References

Dancho, Matt, and Davis Vaughan. 2020. *anomalize: Tidy Anomaly Detection*. https://github.com/business-science/anomalize.

Filzmoser, Peter, and Moritz Gschwandtner. 2021. *mvoutlier: Multivariate Outlier Detection Based on Robust Methods*. http://cstat.tuwien.ac.at/filz/.

Fox, John, Sanford Weisberg, and Brad Price. 2022. *car: Companion to Applied Regression*. https://CRAN.R-project.org/package=car.

Gagolewski, Marek. 2022. stringi: Fast and portable character string processing in R. J Stat Softw 103(2):1–59. https://doi.org/10.18637/jss.v103.i02.

Komsta, Lukasz. 2022. *outliers: Tests for Outliers*. https://CRAN.R-project.org/package=outliers.

Lüdecke, Daniel. 2022. *sjlabelled: Labelled Data Utility Functions*. https://strengejacke.github.io/sjlabelled/.

van der Loo, Mark, and Edwin De Jonge. 2018. *Statistical Data Cleaning with Applications in R*. Wiley.

van der Loo, Mark, Edwin de Jonge, and Sander Scholtus. 2015. *deducorrect: Deductive Correction, Deductive Imputation, and Deterministic Correction*. https://github.com/data-cleaning/deducorrect.

van der Loo, Mark. 2020. *extremevalues: Univariate Outlier Detection*. https://www.github.com/markvanderloo/extremevalues.

Wickham, Hadley. 2022. *stringr: Simple, Consistent Wrappers for Common String Operations*. https://CRAN.R-project.org/package=stringr.

Zhao, Peng. 2022. *fecitr: Functions for Feciting Graphs*. https://github.com/pzhaonet/fecitr.

[5] Download from: https://www.kaggle.com/datasets/osmi/mental-health-in-tech-2016.

Chapter 6
Describing Data

6.1 Chapter Highlights

1. Describe numerical data with the measures of the centre, spread, and shape of the distribution,
2. Describe categorical data with frequency tables,
3. Calculate statistics for grouped data with R functions, and
4. Illustrate the data descriptions in graphs.

6.2 Categorical Data

Data are often difficult for human brains to understand, especially when we would like to know the whole picture of the data. Usually, we have to technically reduce the raw data down to a short summary, so that we could make informed decisions from them.

Categorical data or qualitative data are those that describe qualities or characteristics, such as gender, preference, age group, and job. Categorical data are often summarized for the count or (relative) frequency in contingency tables, also called cross-tabulations, and the most widely used statistical technique for the analysis of contingency tables is the χ^2 test (see Sect. 7.3.3).

In this chapter we use the dataset *diet* shipped with the **Epi** package for demonstrations:

```
data(diet, package = "Epi")
```

It contains 14 variables for 337 subjects, from dietary questionnaires in a cohort study of the incidence of coronary heart disease (CHD). It is a mixture of numerical and categorical data.

P. Zhao, *Working with Data in Public Health: A Practical Pathway with R*,
https://doi.org/10.1007/978-981-99-0135-7_6

Table 6.1 Categorical variables in the diet dataset (the first 6 rows)

Fail	Job	Energy.grp	chd
0	Driver	<=2750 KCals	0
0	Driver	<=2750 KCals	0
13	Conductor	<=2750 KCals	1
3	Driver	<=2750 KCals	1
13	Bank worker	<=2750 KCals	1
3	Bank worker	<=2750 KCals	1

We use the categorical variables `fail` (status on exit), `job`, `energy.grp` (grouped high daily energy intake), and `chd` (CHD event) in the *diet* dataset as an example (Table 6.1):

6.2.1 Contingency Tables

The `table()` function calculates the frequency of one or multiple categorical variables:

```
tab1 <- table(diet$job) # one-way table
tab1
```

```
##
##      Driver   Conductor Bank worker
##         102          84         151
```

```
tab2 <- table(diet[, c('job', 'chd')]) # two-way table
tab2
```

```
##              chd
## job             0   1
##   Driver       90  12
##   Conductor    70  14
##   Bank worker 131  20
```

```
tab3 <- table(diet[, c('job', 'chd', 'energy.grp')]) # three-way table
tab3
```

```
## , , energy.grp = <=2750 KCals
##
##              chd
## job            0  1
##   Driver      40  8
##   Conductor   31  8
##   Bank worker 56 12
##
## , , energy.grp = >2750 KCals
```

```
##
##              chd
## job            0  1
##   Driver      50  4
##   Conductor   39  6
##   Bank worker 75  8
```

and the `prop.table()` function converts the counts in a contingency table into relative frequencies. We could choose whether to calculate the relative percentages as the count divided by the entire sum, or the row sums (`margin = 1`), or the column sums (`margin = 2`).

```
tabp2 <- prop.table(tab2)
tabp2
```

```
##              chd
## job                   0          1
##   Driver      0.26706231 0.03560831
##   Conductor   0.20771513 0.04154303
##   Bank worker 0.38872404 0.05934718
```

```
tabp21 <- prop.table(tab2, margin = 1)
tabp21
```

```
##              chd
## job                   0         1
##   Driver      0.8823529 0.1176471
##   Conductor   0.8333333 0.1666667
##   Bank worker 0.8675497 0.1324503
```

```
tabp22 <- prop.table(tab2, margin = 2)
tabp22
```

```
##              chd
## job                   0         1
##   Driver      0.3092784 0.2608696
##   Conductor   0.2405498 0.3043478
##   Bank worker 0.4501718 0.4347826
```

The `pubh::cross_tbl()` function generates a two-way contingency table for both the frequency and the relative frequency with marginal sums (see Sect. 6.2.2), such as Table 6.2:

```
library(pubh)
diet[, c('job', 'chd')] |>
  sjlabelled::var_labels(job = "Job", chd = "CHD") |>
  cross_tbl(by = "chd") |>
  theme_pubh()
```

We give the variables labels (see Chap. 5) which could be displayed in the output table. The `theme_pubh()` function controls the style of the table.

Table 6.2 A contingency table created by the `cross_tbl()` function

	CHD	CHD	
Job	**0**, N = 291	**1**, N = 46	**Overall**, N = 337
Driver	90 (31%)	12 (26%)	102 (30%)
Conductor	70 (24%)	14 (30%)	84 (25%)
Bank worker	131 (45%)	20 (43%)	151 (45%)

You might find that the more variables are involved, the more difficult to display, further analyze, and visualize the returned table. The `ftable()` function displays the contingency tables in a "flat" way, i.e. a matrix whose rows and columns correspond to unique combinations of the levels of the row and column variables, which is friendly to human eyes and recommended as the layout of tables in publications:

```
ftable(tab3)
```

```
##                 energy.grp <=2750 KCals >2750 KCals
## job         chd
## Driver      0                        40          50
##             1                         8           4
## Conductor   0                        31          39
##             1                         8           6
## Bank worker 0                        56          75
##             1                        12           8
```

Alternatively, a contingency table can be displayed in a long version as a data frame for convenience (Table 6.3):

```
dtf3 <- data.frame(tab3)
```

The long table is recommended for further analysis and visualization in R, as many packages (especially the **tidyverse** family) are designed for them. For example, the `dplyr::count()` function works similarly to `table()` but returns a long table, or data frame, by default:

```
library(dplyr)
dtf2 <- diet |>
  count(job, chd)
```

Then the long table can be further used for calculating the relative frequency:

```
dtfp2 <- transform(dtf2, prop = prop.table(n))
```

A long table can be converted back to a contingency table with the `xtab()` function. Let's convert `dtf2` back to `tab2`:

```
xtab2 <- xtabs(n ~ job + chd, dtf2)
```

Table 6.3 A long table converted from a contigency table

Job	chd	Energy.grp	Freq
Driver	0	<=2750 KCals	40
Conductor	0	<=2750 KCals	31
Bank worker	0	<=2750 KCals	56
Driver	1	<=2750 KCals	8
Conductor	1	<=2750 KCals	8
Bank worker	1	<=2750 KCals	12
Driver	0	>2750 KCals	50
Conductor	0	>2750 KCals	39
Bank worker	0	>2750 KCals	75
Driver	1	>2750 KCals	4
Conductor	1	>2750 KCals	6
Bank worker	1	>2750 KCals	8

The first argument in `xtab()` is a formula in the syntax of the column name for the frequency against (~) the names of the categorical variables (separated by "+") . An abbreviated version is "`n ~ .`", where "`.`" means all other columns except the one before "~".

This conversion could also be performed with the `dplyr::pivot_wider()` function,[1] which convert a long table into a wide one:

```
dtf2w <- pivot_wider(dtf2, names_from = chd, values_from = n)
```

The difference from `xtab()` is that `pivot_wider()` returns a data frame with the categorical variable's levels as the column names, while `xtab()` returns a table.

6.2.2 Marginal Statistics

Marginal statistics are those statistics which appear in the margins of a contingency table. They describe the characteristics of each row and each column of the table. In most cases in public health, tables with marginal statistics (mainly marginal totals/means) could tell us the overall value for a specific level of some variables.

The marginal totals and means for rows and columns can be calculated with the following functions:

```
rowSums(tab2)
```

```
##      Driver   Conductor Bank worker
##         102          84         151
```

[1] The partner of `pivot_wider()` is `pivot_long()`, which converts wide tables into long ones.

```
rowMeans(tab2)
```

```
##      Driver   Conductor Bank worker
##        51.0        42.0        75.5
```

```
colSums(tab2)
```

```
##   0   1
## 291  46
```

```
colMeans(tab2)
```

```
##        0        1
## 97.00000 15.33333
```

If we calculate other statistics than the sums and the means for the rows and columns, we could use the `apply()` function, which is a universal function for any marginal statistics:

```
apply(tab2, MARGIN = 1, max) # maximum for each row
```

```
##      Driver   Conductor Bank worker
##          90          70         131
```

```
apply(tab2, MARGIN = 2, range) # range for each column
```

```
##       chd
##         0  1
##    [1,]  70 12
##    [2,] 131 20
```

If we would like to display the marginal statistics in the contingency table, use `addmargins()`, which by default calculates the row sums and column sums:

```
addmargins(tab2)
```

```
##              chd
## job             0   1 Sum
##   Driver       90  12 102
##   Conductor    70  14  84
##   Bank worker 131  20 151
##   Sum         291  46 337
```

It also supports other marginal statistics if using other functions:

```
addmargins(tab2, FUN = max)
```

```
## Margins computed over dimensions
## in the following order:
## 1: job
## 2: chd
##               chd
## job             0   1 max
##   Driver       90  12  90
##   Conductor    70  14  70
##   Bank worker 131  20 131
##   max         131  20 131
```

The **Epi** package provides a function `stat.table()`, which serves as an extension of `table()` (and `tapply()`, see Sect. 6.4). It can generate contingency tables with marginal statistics and nice headers:

```
library(Epi)
stat.table(list(job, energy.grp), data = diet, margins = TRUE)
```

```
##  ----------------------------------------
##                -------energy.grp--------
##  job             <=2750   >2750   Total
##                   KCals   KCals
##  ----------------------------------------
##  Driver              48      54     102
##  Conductor           39      45      84
##  Bank worker         68      83     151
##
##  Total              155     182     337
##  ----------------------------------------
```

6.3 Numerical Data

Numerical data or quantitative data are the data that can be quantified in numerical values, such as age, height, weight, and temperature. Numerical data can be summarized with a limited number of values which indicate what a typical value is, how broad the data spread, and some further information.

Here we pick out the numeric variables `y` (number of years at risk), `energy` (total energy intake, in kCal per day/100), `height` (in cm), `weight` (kg), `fat` (at intake, in 10 g/day), and `fibre` (dietary fibre intake, 10 g/day) as a numerical subset of *diet*.

In R, the `summary()` function is the most convenient and straightforward way for summarizing data:

```
dietn <- diet[, c('y', 'energy', 'height', 'weight', 'fat', 'fibre')]
summary(dietn)
```

```
##        y                energy          height          weight
##  Min.   : 0.2875   Min.   :17.48   Min.   :152.4   Min.   : 46.72
##  1st Qu.:10.7762   1st Qu.:25.37   1st Qu.:168.9   1st Qu.: 64.64
##  Median :15.4606   Median :28.03   Median :173.0   Median : 72.80
##  Mean   :13.6607   Mean   :28.29   Mean   :173.4   Mean   : 72.54
##  3rd Qu.:17.0431   3rd Qu.:31.10   3rd Qu.:177.8   3rd Qu.: 79.83
##  Max.   :20.0411   Max.   :43.96   Max.   :190.5   Max.   :106.14
##                                    NA's   :5       NA's   :4
##       fat             fibre
##  Min.   : 7.26   Min.   :0.605
##  1st Qu.:11.12   1st Qu.:1.359
##  Median :12.59   Median :1.673
##  Mean   :12.75   Mean   :1.719
##  3rd Qu.:14.01   3rd Qu.:1.935
##  Max.   :21.63   Max.   :5.351
##                  NA's   :4
```

The outputs include `Min.` (minimum), `1st Qu.` (the first quartile), `Median`, `Mean` (the arithmetic mean), `3rd Qu.` (the third quartile), `Max.` (the maximum), and the number of `NA`s.

6.3.1 Central Tendency

The mean and the median are typical values for the measures of the central tendency. If we would like to use only one value to summarize the data, the mean or the median might be the best choice.

Three types of means are often used in mathematics, namely the arithmetic mean (i.e. the sum divided by the count, $\mu = \Sigma x_i / n$), the harmonic mean (i.e. the reciprocal of the arithmetic mean of the reciprocal of each given value, $H = n / \Sigma \frac{1}{x_i}$), and the geometric mean (i.e. the n-th root of the product of all, $G = \sqrt[n]{\Pi x_i}$). By default, we mean the arithmetic mean when we talk about the *mean* in public health. We can calculate the arithmetic mean of a variable, e.g. the height in the *diet* dataset, with the `mean()` function:

```
mean(dietn$height, na.rm = TRUE)
```

```
## [1] 173.3579
```

Outliers could have unfair influence on the arithmetic mean. A method for eliminating the influence of outliers is using the trimmed mean (also called a truncated mean), which removes a fraction (0–0.5) from each end of the observations before calculating the arithmetic mean. This can be done with the `trim` argument in `mean()`:

```
mean(dietn$height, trim = 0.1, na.rm = TRUE)
```

```
## [1] 173.4288
```

There are no build-in function in R for calculating the harmonic mean or the geometric mean. For calculating the harmonic mean, we simply follow the definition:

```
n <- sum(!is.na(dietn$height))
n / sum(1/dietn$height, na.rm = TRUE)
```

```
## [1] 173.1196
```

```
# or
1 / mean(1/dietn$height, na.rm = TRUE)
```

```
## [1] 173.1196
```

For calculating the geometric mean, we could use the `prod()` function to get the product of multiple values, according the definition of G:

```
x <- 1:10
prod(x) ^ (1 / 10)
```

```
## [1] 4.528729
```

However, sometime it causes problems when the product of the numbers exceeds the computer's capacity:

```
prod(dietn$height, na.rm = TRUE) ^ (1/n)
```

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

Alternatively, we can calculate it like this:

```
exp(mean(log(dietn$height), na.rm = TRUE))
```

```
## [1] 173.2391
```

The median, also called Q_2 and P_{50}, is the middle value of an ordered set. If n is odd, the median is the $(n+1)/2$ -th largest observation; If n is even, the median is the mean of the $n/2$ -th and the $(n+1)/2$ -th largest observations. R provides the `median()` function to do this job:

```
median(dietn$height, na.rm = TRUE)
```

```
## [1] 172.9994
```

The mean is the best way to estimate a subject of a population, while the median is the safest. However, both are insufficient and unsatisfactory for the whole picture of the data because of the lack of the variance or the spread.

6.3.2 Spread

The generic measure of the spread of the data is the maximum and the minimum:

```
max(dietn$height, na.rm = TRUE)
```

```
## [1] 190.5
```

```
min(dietn$height, na.rm = TRUE)
```

```
## [1] 152.4
```

The `range()` function gives both:

```
range(dietn$height, na.rm = TRUE)
```

```
## [1] 152.4 190.5
```

However, the term *range* actually means the difference between the maximum and the minimum:

```
max(dietn$height, na.rm = TRUE) - min(dietn$height, na.rm = TRUE)
```

```
## [1] 38.1
```

```
# or
diff(range(dietn$height, na.rm = TRUE))
```

```
## [1] 38.1
```

Although the range can be used for all kind of numerical variables, it only informs us the upper and lower limits of the data. A more informative measure of the spread is the variance ($\sigma^2 = \frac{\Sigma(x_i-\mu)^2}{N}$ for a population, and $s^2 = \frac{\Sigma(x_i-\bar{x})^2}{n-1}$ for a sample) and the standard deviation (σ, s).

```
var(dietn$height, na.rm = TRUE)
```

```
## [1] 41.09623
```

```
sd(dietn$height, na.rm = TRUE)
```

```
## [1] 6.410634
```

The coefficient of variation is defined as $c_v = \sigma/\mu$ for a population and $c_v = s/\bar{x}$ for a sample, which measures the relative spread of the data.

The variance and standard deviation are paired with the mean. If the data follow the normal distribution, we can easily estimate in mind how many values are located around a certain distance from the mean (e.g. the *68-95-99.7* law[2]).

If the data do not follow normal distribution, the percentiles or the quartiles, paired with the median, make more sense for describing the spread. The p-th percentile is the value P_p such that $p\%$ of the sample points are less than or equal to P_p. For example, the 10th percentile of the height data (P_{10}) is:

```
quantile(dietn$height, probs = 0.10, na.rm = TRUE)
```

```
##   10%
## 165.1
```

which means 10% of the height values are less than or equal to 165.1 cm.

Based on P_p, three quartiles are defined as $Q_1 = P_{25}$, $Q_2 = P_{50} =$ median, $Q_3 = P_{75}$:

```
quantile(dietn$height, probs = c(0.25, 0.5, 0.75), na.rm = TRUE)
```

```
##      25%      50%      75%
## 168.9100 172.9994 177.8000
```

and the interquartile range is defined as IQR $= Q_3 - Q_1$:

```
IQR(dietn$height, na.rm = TRUE)
```

```
## [1] 8.89
```

which means that 50% of the data are distributed around the median within 8.89 cm.

R provides a function `fivenum()`, which is a wrapper of `quantile(probs = seq(0, 1, 0.25))` and returns the minimum, the three quartiles, and the maximum, in a fast way:

[2] 68%, 95%, and 99.7% of the values lie within one, two, and three standard deviations from the mean in a normal distribution, respectively.

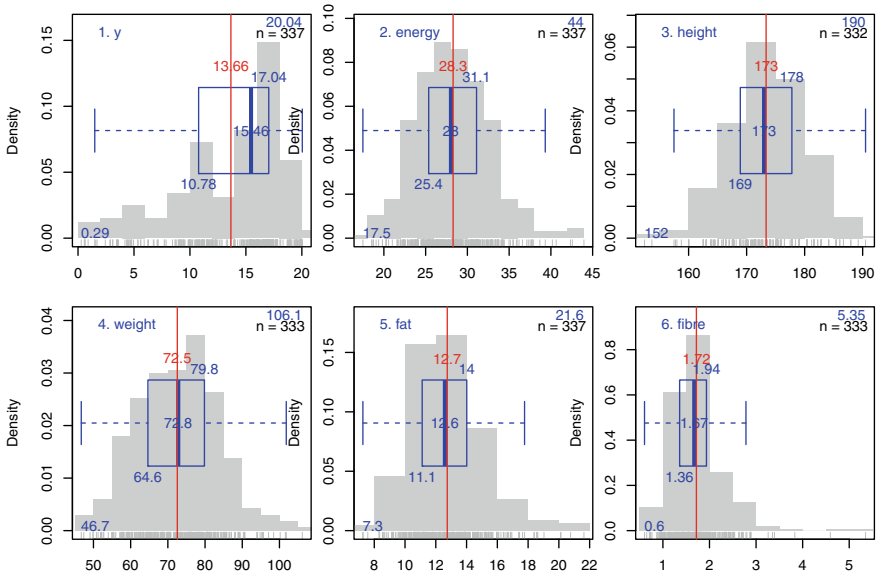


Fig. 6.1 Distribution of the numerical variables in the *diet* dataset

```
fivenum(dietn$height, na.rm = TRUE)
```

```
## [1] 152.4000 168.9100 172.9994 177.8000 190.5000
```

6.3.3 Distribution Shape

The shape of the numerical data means how the distribution or pattern looks like. Usually the shape can be described in three aspects, i.e. the mode, the skewness, and the kurtosis. The shape is a very helpful assistance in choosing proper statistics, including the measures of centres and spreads, for describing data.

The mode

The mode indicates how many centres the data points gather around. Usually we can see it as the peak number in a histogram for one-dimensional numeric data. Figure 6.1 visualizes the demonstration dataset in histograms, where `energy`, `height`, `weight`, `fat`, and `fibre` show a uni-modal (single peak) shape, while `y` shows a multi-modal (more than one peaks) shape. The means (red lines and numbers) well represent the centre of the uni-modal data, but poorly provide sufficient information for multi-modal data.

The skewness

The skewness describes the asymmetry of the shape. The two sides of a symmetrical distribution are mirror images of each other, and the skewness reflects the tendency of the data to be more frequent around the right or left ends of the x-axis. It is calculated as[3]:

$$b_1 = \frac{n}{(n-1)(n-2)} \sum_{i=1}^{n} (\frac{x_i - \bar{x}}{s})^3$$

The `e1071::skewness()` function (Meyer et al. 2022) can calculate the skewness of a vector, and the `psych::skew()` function (Revelle 2022) can calculate the skewness for each variable in a data frame or a matrix:

```
e1071::skewness(dietn$height, na.rm = TRUE)
```

```
## [1] -0.1457102
```

```
psych::skew(dietn)
```

```
## [1] -0.9003218  0.4410730 -0.1457102  0.1349350  0.6222846  1.6854345
```

```
# or
apply(dietn, MARGIN = 2, e1071::skewness, na.rm = TRUE)
```

```
##          y     energy     height     weight        fat      fibre
## -0.9003218  0.4410730 -0.1457102  0.1349350  0.6222846  1.6854345
```

The normal distribution has a perfect symmetric shape with a skewness of 0. A negative skewness (`y`) stands for negatively skewed data (i.e. the tail on the left side of the histogram is longer than the right), where the mean is smaller than the median; skewness around 0 (`height` and `weight`) for symmetrical data, where the mean is (nearly) the same as the median; and positive (`energy`, `fat`, and `fibre`) for positively skewed data (i.e. skewed to the right), where the mean is greater than the median. For skewed data, the median may be a more appropriate measure of central tendency than the mean.

The kurtosis

The kurtosis describes the flatness or sharpness of a peak compared to the normal distribution. It is calculated as

[3] Different definition and formulas exist for calculating the skewness and kurtosis. Here we use the same ones as in MINITAB and BMDP. See the help documents of `e1071::skewness()` and `e1071::kurtosis()` for more details.

$$b_2 = \frac{n(n+1)}{(n-1)(n-2)(n-3)} \sum_{i=1}^{n} \left[\frac{x_i - \bar{x}}{s} \right]^4 - \frac{3(n-1)^2}{(n-2)(n-3)}$$

which can be conducted with `e1071::kurtosis()` or `psych::kurtosi()`:

```
e1071::kurtosis(dietn$height, na.rm = TRUE)
```

```
## [1] 0.0467094
```

```
psych::kurtosi(dietn)
```

```
##          y     energy     height     weight        fat      fibre
##  0.0128214  0.4859871  0.0467094 -0.1381723  0.7395585  6.8383764
```

The normal distribution has a kurtosis of 0, which is called a Mesokurtic shape. A negative kurtosis stands for a flatter peak (a Platykurtic shape), and positive for a shaper one (a Leptokurtic shape) than the normal distribution.

6.3.4 *Extended Summaries*

The **psych** package gives a more powerful summary function `describe()`, which produces the most frequently requested statistics in public health, especially in psychometric and psychology studies (Table 6.4):

```
library(psych)
dtf_desc <- describe(diet, quant = c(0.25, 0.75), IQR = TRUE)
```

The returned result is a data frame with the rows as all the variables in the data set and columns as the statistics, including the number of valid cases (`n`), the mean, the standard deviation (`sd`), the median, the trimmed mean with trim defaulting to 0.1 (`trimmed`), the median absolute deviation (`mad`), the minimum (`min`), the maximum (`max`), the range, the skewness (`skew`), the kurtosis, the standard error (`se`), IQR, Q_1 (`Q0.25`), and Q_3 (`Q0.75`). The quantiles and IQR are not calculated by default, therefore we have to use the argument `quant` and `IQR` to activate them. Compared to the built-in `summary()` function, `describe()` organizes more descriptions for the data in a data frame which is friendly both to the users and to the computers for further use. It can also do the grouped summary (see Sect. 6.4):

```
dtf_des1 <- describe(diet ~ job)
dtf_des2 <- describe(diet ~ job + fail)
```

The returned result is a list, containing multiple data frames, each of which is a descriptive data frame for a group.

Table 6.4 A summary table for the *diet* data. Generated by the `psych::describe()` function

	sd	Median	Trimmed	Mad	Min	Max	Range	Skew	Kurtosis	se	IQR	Q0.25	Q0.75
id	97	169	169	125	1	337	336	0	−1	5	168	85	253
Doe	NA	NA	NaN	NA	Inf	−Inf	−Inf	NA	NA	NA	NA	NA	NA
Dox	NA	NA	NaN	NA	Inf	−Inf	−Inf	NA	NA	NA	NA	NA	NA
Dob	NA	NA	NaN	NA	Inf	−Inf	−Inf	NA	NA	NA	NA	NA	NA
y	5	15	14	4	0	20	20	−1	0	0	6	11	17
Fail	4	0	1	0	0	15	15	2	2	0	0	0	0
Job*	1	2	2	1	1	3	2	0	−2	0	2	1	3
Month	4	6	6	6	1	12	11	0	−1	0	7	3	10
Energy	4	28	28	4	17	44	26	0	0	0	6	25	31
Height	6	173	173	6	152	190	38	0	0	0	9	169	178
Weight	11	73	72	11	47	106	59	0	0	1	15	65	80
Fat	2	13	13	2	7	22	14	1	1	0	3	11	14
Fibre	1	2	2	0	1	5	5	2	7	0	1	1	2
Energy.grp*	0	2	2	0	1	2	1	0	−2	0	1	1	2
chd	0	0	0	0	0	1	1	2	2	0	0	0	0

The **pastecs** package (Grosjean and Ibanez 2018) provides a function `stat.desc()` for descriptive statistics on a data frame or time series:

```
dtf_pastecs <- pastecs::stat.desc(diet, norm = TRUE)
```

The returned result is a data frame with the rows as statistics and the columns as all the variables in the data set. The statistics include the number of valid values (`nbr.val`), null values (`nbr.null`), and missing values (`nbr.na`), the minimum (`min`), the maximum (`max`), the range, the sum, the median, the mean, the standard error on the mean (`SE.mean`), the confidence interval of the mean at the *p* level (`CI.mean.0.95`), the variance (`var`), the standard deviation (`std.dev`), the variation coefficient (i.e. the standard deviation divided by the mean, `coef.var`), the skewness and its significant criterium (`skew.2SE`), the kurtosis and its significant criterium (`kurt.2SE`), the statistic of a Shapiro-Wilk test of normality (`normtest.W`) and its associated probability (`normtest.p`).

6.4 Grouped Summary

Numerical data in public health are often classified into groups according to the categorical variables. For example, in previous sections, we summarize all the values of height in the *diet* dataset with typical values such as the mean and the standard deviation, but we often need the centre and the spread of the height for each class of the gender, age group, or job.

The built-in function in R for group summary is `tapply()` (table apply). Here we calculate the mean height for each job:

```
tapply(diet$height, diet$job, mean, na.rm = TRUE)
```

```
##      Driver   Conductor Bank worker
##    172.2177    169.3553    176.4090
```

or the mean height for each job and energy group:

```
tapply(diet$height, list(diet$job, diet$energy.grp), mean, na.rm = TRUE)
```

```
##             <=2750 KCals >2750 KCals
## Driver          172.1469    172.2807
## Conductor       168.2216    170.3601
## Bank worker     175.7561    176.9125
```

A limitation of `tapply()` is that only one statistics could be calculated, i.e. the mean in the previous example. The `Epi::stat.table()` function can do the group summary for multiple statistics. The following code calculates the means and standard deviations of the height, grouped by the job and energy group with marginal values:

```
stat.table(list(job, energy.grp),
           contents = list(mean(height, na.rm = TRUE),
                           sd(height, na.rm = TRUE)),
           data = diet, margins = TRUE)
```

```
##  ----------------------------------------
##                -------energy.grp--------
##  job             <=2750   >2750   Total
##                   KCals   KCals
##  ----------------------------------------
##  Driver          172.15  172.28  172.22
##                    5.14    5.36    5.23
##
##  Conductor       168.22  170.36  169.36
##                    5.34    4.59    5.04
##
##  Bank worker     175.76  176.91  176.41
##                    7.43    5.41    6.37
##
##
##  Total           172.66  173.94  173.36
##                    6.93    5.90    6.41
##  ----------------------------------------
```

It supports most of the common functions for statistics, including `count()`, `mean()`, `weighted.mean()`, `sum()`, `quantile()`, `median()`, `IQR()`, `max()`, `min()`, `ratio()`, `percent()`, and `sd()`. If we would like to get more statistics, the **dplyr** package might be the most powerful way with the `group_by()` and `summarise()` functions for group summary.

```
diet |>
  group_by(job, energy.grp) |>
  summarise(mean = mean(height, na.rm = TRUE),
            sd = sd(height, na.rm = TRUE),
            median = median(height, na.rm = TRUE))
```

```
## # A tibble: 6 x 5
## # Groups:   job [3]
##   job         energy.grp    mean    sd median
##   <fct>       <fct>        <dbl> <dbl>  <dbl>
## 1 Driver      <=2750 KCals  172.  5.14   173.
## 2 Driver      >2750 KCals   172.  5.36   173.
## 3 Conductor   <=2750 KCals  168.  5.34   169.
## 4 Conductor   >2750 KCals   170.  4.59   171.
## 5 Bank worker <=2750 KCals  176.  7.43   177.
## 6 Bank worker >2750 KCals   177.  5.41   178.
```

The marginal relative frequency calculation of `dtf2` can then be calculated as:

```
dtf2td1 <- dtf2 |>
  group_by(chd) |>
  mutate(prop = prop.table(n))
```

```
dtf2td2 <- dtf2 |>
  group_by(job) |>
  mutate(prop = prop.table(n))
```

6.5 Visualization

The functions we have introduced previously summarize the data into numbers and/or organize them into tables. The numbers give us an abstract overview of a data set, but might be not easy to draw a whole picture. It would be a good practice to visualize the dataset in graphs.

R provides numerous functions and packages for data visualization (see Chap. 8). Here we introduce a wrapper function `plot_summary()` from the **fecitr** package (GitHub: pzhaonet/fecitr), which can visualize the summary report of a data frame. We pick out two numeric variables and two categorical ones in *diet* for demonstration (Fig. 6.2):

```
library(fecitr)
diet[, c('height', 'weight', 'job', 'energy.grp')] |>
  plot_summary(if_box = TRUE)
```

By default, it plots the raw data of each numeric variable in a dot chart (with the horizontal axis as the value and the vertical axis as the index), and the values of each categorical variable in a bar chart (with the horizontal axis as the frequency and the vertical axis as the levels). We set the argument `if_box = TRUE` so as to draw a box plot of the distribution for each numerical variable with the minimum, Q_1, the median, Q_3, and the maximum aligned from bottom-left to top-right, as well as the mean at the top.

The `plot_summary()` can plot the histograms of the numeric data instead of the dot chart. Other arguments, including showing the curve of the normal distribution (`if_normline`) and showing the sample size and skewness (`if_skewness`), allow us see more descriptions of the dataset (Fig. 6.3). More information about data visualization is described in Chap. 8.

```
diet[, c('height', 'weight')] |>
  plot_summary(base = 'hist', if_box = TRUE,
               if_normline = TRUE, if_skewness = TRUE)
```

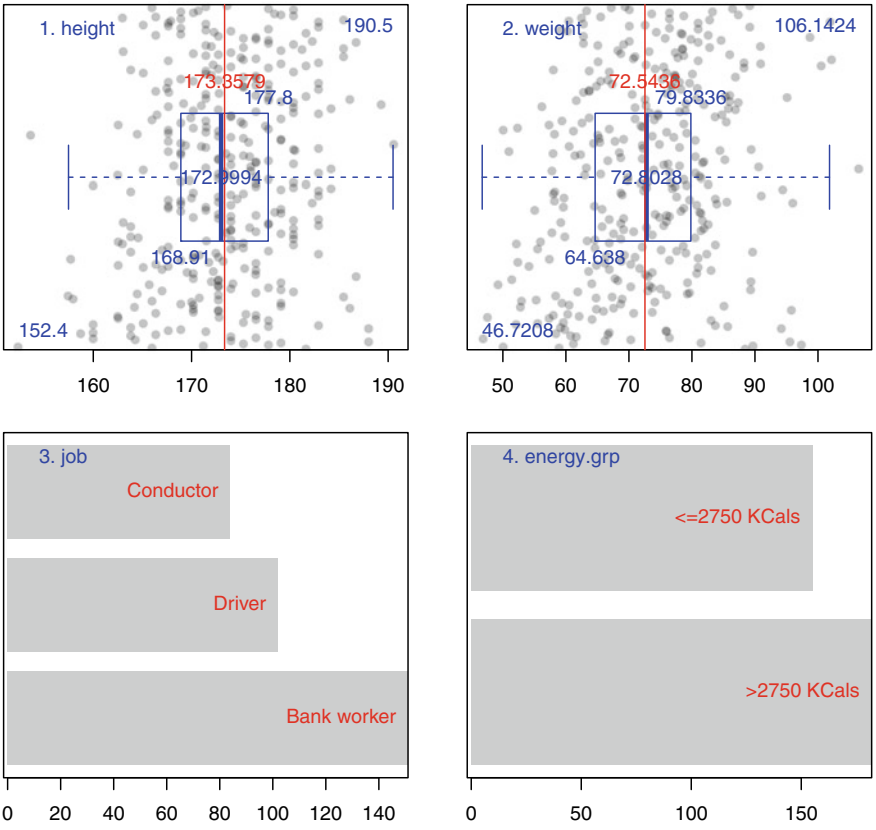


Fig. 6.2 Visualization of a subset of the raw *diet* dataset

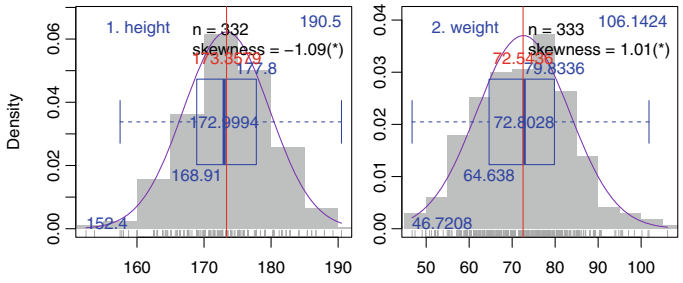


Fig. 6.3 Visualization of a subset of the *diet* data with histograms and normal distribution curves

6.6 Exercise

1. Body mass index (BMI) is a person's weight in kilograms divided by the square of height in meters. BMI is an inexpensive and easy screening method for weight category—underweight, healthy weight, overweight, and obesity. For adults 20 years old and older, BMI is interpreted using standard weight status categories (see Table 6.5). These categories are the same for men and women of all body types and ages.
   Convert the height data in the *diet* data of the **Epi** package into BMI and make a contingency table (with marginal sums) with it.
2. The *births* dataset from the **Epi** package is introduced in Chap. 4.
   - Generate a contingency table for the count of the sex and the maternal hypertension, with marginal sums.
   - Calculate the mean birth weight by gender.
   - Calculate the means and the standard deviations of birth weight by the combination of gender and maternal hypertension group with marginal statistics.
3. A case-control study of (o)esophageal cancer in a department of France provided a dataset *esoph*, which is available as a data frame in the built-in R installation. It contains the frequency of the subjects in the case group (`ncases`) and in the control group (`ncontrols`) for the grouped age, alcohol consumption, and tobacco consumption. It is a long table. Convert it into a contingency table in a wide version.
4. The dataset *melanoma*, shipped by the **boot** package (Canty and Ripley 2021), came from a study on patients with malignant melanoma in Denmark. 205 patients had their tumour removed by surgery during the period 1962–1977 and were followed until the end of 1977. Seven variables for each patient were recorded, including the survival time (`time`, in days) since the operation, the patient's status (`status`) at the end of the study, the patient's sex (`sex`), the patient's age at the time of the operation (`age`), the year of operation (`year`), the tumour thickness (`thickness`, in mm), and the existence of ulceration (`ulcer`). Describe the numerical variables and the categorical variables in the dataset.

Table 6.5 Classification of adult's weight based on BMI

BMI	Weight status
Below 18.5	Underweight
18.5–24.9	Healthy weight
25.0–29.9	Overweight
30.0 and Above	Obesity

5. The **pubh** package provides a function `estat()`, which calculates descriptives of numerical variables. Read the help document and use it for describing the datasets demonstrated in this Chapter.

References

Canty, Angelo, and Brian Ripley. 2021. *boot: Bootstrap R (S-Plus) Functions*. https://CRAN.R-project.org/package=boot.

Grosjean, Philippe, and Frederic Ibanez. 2018. *pastecs: Package for Analysis of Space-Time Ecological Series*. https://github.com/phgrosjean/pastecs.

Meyer, David, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and Friedrich Leisch. 2022. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*. https://CRAN.R-project.org/package=e1071.

Revelle, William. 2022. *psych: Procedures for Psychological, Psychometric, and Personality Research*. https://CRAN.R-project.org/package=psych.

Chapter 7
Analyzing Data

7.1 Chapter Highlights

- Understand the probability distributions and illustrate them with R functions,
- Use hypothesis test functions in public health research, and
- Apply common regression models used in public health research.

7.2 Probability Distribution Functions in R

In the process of analyzing data, we infer conclusions for the population from the data of the sample with inferential statistical tools. Inferential statistics stands on the shoulder of probability distributions. The normal distribution, χ^2 distribution, and Student's t distribution are the most commonly used distributions.

Every distribution that R handles has four functions beginning with a root name (Table 7.1). The root name is prefixed by one of the letters:

- `d` for "density", the probability density function (PDF).
- `p` for "probability", the cumulative distribution function (CDF).
- `q` for "quantile", the inverse of CDF.
- `r` for "random", a random variable having the specified distribution.

The relationships of these four functions are illustrated in Fig. 7.1, showing the standard normal distribution as an example. As the root name for the normal distribution is `norm`, the `dnorm()` function (Fig. 7.1a) gives the probability density function of $x_1 = 1$ in a normal distribution:

P. Zhao, *Working with Data in Public Health: A Practical Pathway with R*,
https://doi.org/10.1007/978-981-99-0135-7_7

Table 7.1 Root names of the distribution functions in R

Distribution	Root name	PDF	CDF	Quantile function	Random function
Beta	`beta`	`dbeta`	`pbeta`	`qbeta`	`rbeta`
Binomial	`binom`	`dbinom`	`pbinom`	`qbinom`	`rbinom`
Cauchy	`cauchy`	`dcauchy`	`pcauchy`	`qcauchy`	`rcauchy`
Chi-square	`chisq`	`dchisq`	`pchisq`	`qchisq`	`rchisq`
Exponential	`exp`	`dexp`	`pexp`	`qexp`	`rexp`
F	`f`	`df`	`pf`	`qf`	`rf`
Gamma	`gamma`	`dgamma`	`pgamma`	`qgamma`	`rgamma`
Geometric	`geom`	`dgeom`	`pgeom`	`qgeom`	`rgeom`
Hypergeometric	`hyper`	`dhyper`	`phyper`	`qhyper`	`rhyper`
Logistic	`logis`	`dlogis`	`plogis`	`qlogis`	`rlogis`
Log normal	`lnorm`	`dlnorm`	`plnorm`	`qlnorm`	`rlnorm`
Negative binomial	`nbinom`	`dnbinom`	`pnbinom`	`qnbinom`	`rnbinom`
Normal	`norm`	`dnorm`	`pnorm`	`qnorm`	`rnorm`
Poisson	`pois`	`dpois`	`ppois`	`qpois`	`rpois`
Student t	`t`	`dt`	`pt`	`qt`	`rt`
Studentized range	`tukey`	`dtukey`	`ptukey`	`qtukey`	`rtukey`
Uniform	`unif`	`dunif`	`punif`	`qunif`	`runif`
Weibull	`weibull`	`dweibull`	`pweibull`	`qweibull`	`rweibull`
Wilcoxon rank sum statistic	`wilcox`	`dwilcox`	`pwilcox`	`qwilcox`	`rwilcox`
Wilcoxon signed rank statistic	`signrank`	`dsignrank`	`psignrank`	`qsignrank`	`rsignrank`

```
x1 <- 1
d1 <- dnorm(x = x1, mean = 0, sd = 1)
d1
```

```
## [1] 0.2419707
```

The `pnorm()` function (Fig. 7.1b) gives the cumulative probability density within the range of $< x_1$ in a normal distribution:

```
p1 <- pnorm(q = x1, mean = 0, sd = 1)
p1
```

```
## [1] 0.8413447
```

which equals to the integral of the PDF (i.e. the shaded area in Fig. 7.1a):

```
integrate(dnorm, -Inf, x1)
```

```
## 0.8413448 with absolute error < 1.5e-05
```

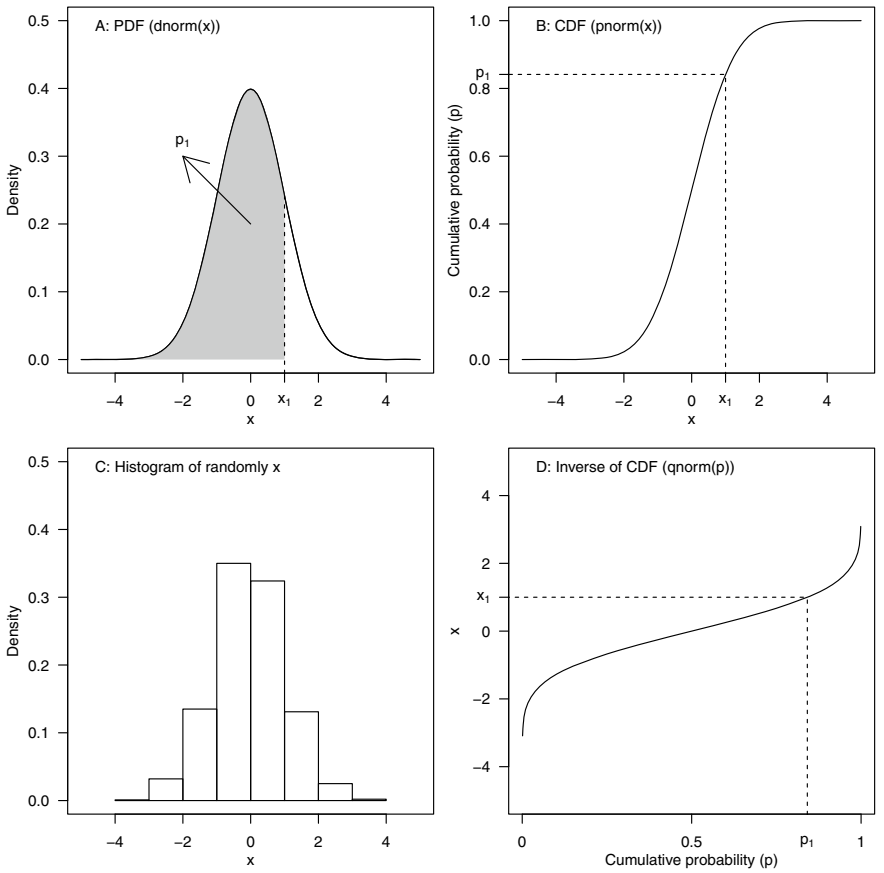


Fig. 7.1 The relationships of the four probability functions for standard normal distribution

For a given cumulative probability, we could use the `qnorm()` function (Fig. 7.1c) to calculate $x1$:

```
qnorm(p1, mean = 0, sd = 1)
```

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

The `rnorm()` function generates random values in a normal distribution:

```
rnorm(n = 1,mean = 0, sd = 1)
```

```
## [1] 1.022202
```

which could be visualized in a histogram (Fig. 7.1d) that follows the PDF.

7.3 Hypothesis Tests

7.3.1 R Functions and Common Steps

A hypothesis is a statement about a property of a population, and a hypothesis test is the procedure to test the statement. In public health, we often use parametric tests (including z-test, t-test, F-test, and χ^2-test) and non-parametric tests (including the Mann-Whitney U-test, Wilcoxon signed-ranks test, Kruskai-Wallis test, and Friedman test) for a variety of research purposes. The usage of these tests and the relative R functions are listed in Table 7.2.

The general procedure of hypothesis test with R functions usually consists of three main steps (Triola et al. 2019).

Table 7.2 R functions for hypothesis tests

Hypothesis test	Usage	R function
z-test	Parametric test for one or two population means when the population standard deviation is known	TeachingDemos::z.test()
Student's t-test	Parametric test for one or two population means when the population standard deviation is unknown	t.test()
F-test	Parametric test for two population variances	var.test()
χ^2-test	Parametric test for the observed and expected frequency	chisq.test()
ANOVA	Parametric test for two or more population means	aov()
Mann-Whitney U-test	Non-parametric test for two population medians based on two independent samples	wilcox.test()
Wilcoxon signed-ranks test	Non-parametric test for two population medians based on paired samples	wilcox.test()
Kruskai-Wallis test	Non-parametric test for multiple population medians based on independent samples	kruskal.test()
Friedman test	Non-parametric test for multiple population medians based on matched samples	friedman.test()

Step 1. Construct the hypotheses and determine the significance level. A hypothesis test starts with a null hypothesis (H_0) and an alternative hypothesis (H_1 or H_A). H_0, usually consisting of equality (i.e. $\leq$, $\geq$, =), is set up for the express purpose of being discredited, while H_1, the complement of H_0, is a statement of what we would like to prove is true. There are two types of hypothesis test, namely one-sided/tailed or two-sided/tailed test. A one-sided test means the rejection region is in the right or the left tail of the probability density distribution curve (i.e. H_1: the population parameter is either greater or less than a certain value), while a two-sided test means the rejection region is in both of the right and the left tails (i.e. H_1: the population parameter is not equal to a certain value). The hypothesis type could be clarified with the common argument `alternative` with a possible value of "two.sided", "less", and "greater", of the R functions listed in Table 7.2.

The significance level (α) of the test is another thing to be determined in this step. The significance level is the maximum allowable probability of Type I error, i.e. the probability of rejecting H_0 when H_0 is true. It could be declared with the argument `conf.level` ($1-\alpha$) in the R functions. In public health, $\alpha = 0.05$ (`conf.level = 0.95`) is a common choice, but researchers can specify α values according to different research purposes.

Step 2. Choose the test statistic and calculate its value and the p-value. As long as we find the suitable hypothesis test according to the research objectives (see Table 7.2), we could apply the chosen R function to the data, which returns the value of the test statistic (usually listed as the first value in the output report) and the p-value, which is the probability of obtaining the value of the test statistics that is at least as extreme as the statistics getting from the sample data based on the assumption that H_0 is true.

Step 3. Interpret the report and make a decision.
The returned report by R functions is usually friendly to interpret, as both of the p-value and H_1 are explicitly stated. We usually use the p-value to decide whether to reject H_0 or not. If the p-value is less than α, then we reject H_0, otherwise we do not.

The subsequent sections explains the details of the reports by the hypothesis functions.

7.3.2 Student's t-Test

Student's t-test is mainly used for a small sample size ($n < 30$) to test the mean of a population with normal distributions when the population standard deviation σ is unknown. The t-test can be categorized into one-sample t-test and two-sample t-test, the latter of which includes paired test and unpaired test. For unpaired t-test, the calculation for equal variances is different from that for unequal variances.

The usage of the R function `t.test()` for one-sample test is simple:

```
t.test(x, alternative, mu, conf.level)
```

where `x` is the sample data, `alternative` is H_1 (`alternative = "two.sided"` for two-sided test, `alternative = "greater"` and `alternative = "less"` for one-sided test), `mu` is the mean for testing, and `conf.level` is the confidence level. For example, we test whether the mean weight of the energy group $\leq$ 2750 KCals is less than 75 kg at $\alpha = 0.1$ with the following code:

```
x <- diet$weight[as.numeric(diet$energy.grp) == 1]
t.test(x, alternative = 'less', mu = 75, conf.level = 0.9)
```

The `t.test()` function can be used with two equivalent methods for two-sample test.

Method 1 uses the sample data values organized in one vector for Sample 1 (`x`) and another vector for Sample 2 (`y`):

```
t.test(x, y, alternative, conf.level, var.equal, ...)
```

where `var.equal` determines whether the variances of the two populations are equal.

Method 2 uses the sample data values for both samples organized in one vector, and a categorical vector with the same length for distinguishing the two samples:

```
t.test(sample_data ~ category, alternative, conf.level, var.equal, ...)
```

For example, we test whether the mean weights of two the energy groups are equal at $\alpha = 0.05$, assuming the variances are not equal:

```
# method 1
y <- diet$weight[as.numeric(diet$energy.grp) == 2]
t.test(x, y, alternative = 't', var.equal = FALSE)
# method 2
t.test(diet$weight ~ diet$energy.grp, alternative = 't', var.equal = FALSE)
```

Method 2 is recommended, as the categories are explicitly stated in the report, which is easier for the readers to understand.

When `paired = TRUE`, the paired t-test is performed, and them the argument `mu` is the difference between the two means for paired samples.

7.3.3 χ^2-Test

The χ^2-test, based on the χ^2 distribution, is used to test the divergence between the expected and observed frequencies. It can be mainly categorized into one-category test and two-category test. The R function for χ^2-test is `chisq.test()`.

One-category χ^2-test examines whether the frequency distribution follows a certain theory. The usage of `chisq.test()` is for one-category χ^2-test is:

```
chisq.test(x, p, ...)
```

where `x` is a numeric vector, and `p` is the theoretical probability.

For example, Mendelian inheritance predicts that the ratio of red to white to pink flowered plants should be 1:1:2 after cross-pollination of white and read sweet peas, and a biologist's experiment shows the number of red-, white- and pink-flowered plants were 90, 101, and 192. We could perform the one-category χ^2-test as follows:

```
chisq.test(x = c(90, 101, 192), p = c(0.25, 0.25, 0.5))
```

The two-category test can be performed by `chisq.test()` with two equivalent methods.

Method 1 uses the sample data values organized in a contingency table. Let's use the *diet* dataset for testing whether the occurrence of coronary heart disease (CHD) is associated with the job. We could first make a contingency table, and then use `chisq.test()`:

```
tbl <- table(diet[, c('job', 'chd')])
chisq.test(tbl)
```

Method 2 uses the sample data directly as the input rather than the contingency table:

```
chisq.test(x = diet$job, y = diet$chd)
```

In public health, we often deal with 2×2 contingency tables, and an argument `correct` could be used for the Yates' correction. Here we use the dataset *Oncho* shipped by **pubh** as an example. This dataset, containing 1302 rows and 7 variables, came from a study of onchocerciasis in Sierra Leone. The subjects' living areas of "Savannah" or "Rainforest" (the column `area`) and their status of "Infected" or "Not-infected" (the column `mf`) were recorded. Thus, the frequency table for and living areas and the infection statuses is a 2×2 contingency table, and the Yates' correction should be applied:

```
library(pubh)
data(Oncho)
chisq.test(x = Oncho$area, y = Oncho$mf, correct = TRUE)
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  Oncho$area and Oncho$mf
## X-squared = 56.275, df = 1, p-value = 6.301e-14
```

Alternatively, the **pubh** package provides a convenient function `contingency()` for χ^2-test for a 2×2 contingency table:

```
contingency(mf ~ area, data = Oncho)
```

```
##             Outcome
## Predictor    Infected Not-infected
##   Rainforest      541          213
##   Savannah        281          267
##
##              Outcome +    Outcome -      Total        Inc risk *        Odds
## Exposed +          541          213        754              71.8        2.54
## Exposed -          281          267        548              51.3        1.05
## Total              822          480       1302              63.1        1.71
##
## Point estimates and 95% CIs:
## -------------------------------------------------------------------
## Inc risk ratio                                 1.40 (1.27, 1.54)
## Odds ratio                                     2.41 (1.92, 3.04)
## Attrib risk in the exposed *                   20.47 (15.20, 25.75)
## Attrib fraction in the exposed (%)            28.53 (21.56, 34.89)
## Attrib risk in the population *                11.86 (6.92, 16.79)
## Attrib fraction in the population (%)         18.78 (13.59, 23.65)
## -------------------------------------------------------------------
## Uncorrected chi2 test that OR = 1: chi2(1) = 57.151 Pr>chi2 = <0.001
## Fisher exact test that OR = 1: Pr>chi2 = <0.001
##  Wald confidence limits
##  CI: confidence interval
##  * Outcomes per 100 population units
##
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  dat
## X-squared = 56.275, df = 1, p-value = 6.301e-14
```

The result shows the contingency table first, followed by the measures of association and then the χ^2-test result.

7.3.4 Analysis of Variance (ANOVA)

ANOVA is one of the most widely used statistical techniques in public health. ANOVA partitions the total variation in a data set into two or more components, and ascertains the contributions of each of these sources to the total variation. ANOVA can be mainly categorized into one-way and two-way. The one-way ANOVA examines three or more population means via analyzing the sample means, and the two-way ANOVA is usually conducted to examine separate or combined effects of two categorical independent variables, which is called factors, on the continuous dependent variable.

ANOVA is performed with the `aov()` function. If we are lucky, the dataset is organized a long table, i.e. a table with the numerical variable in one column and the categorical variable in another. The *diet* dataset is an example. Then `aov()` could be used directly like this:

```
summary(aov(weight ~ job, diet))
```

```
##              Df Sum Sq Mean Sq F value   Pr(>F)
## job           2   5964  2982.1   30.18 9.19e-13 ***
```

```
## Residuals   330  32608    98.8
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 4 observations deleted due to missingness
```

However, the dataset is not organized in a long version in many occasions. For example, the following dataset, which is a wide table, shows a certain numerical variable for three treatments:

```
dtf_wide <- data.frame(Treatment1 = c(9, 9, 10),
                       Treatment2 = c(12, 12, 13),
                       Treatment3 = c(12, 13, 14))
```

Before using `aov()` for testing if the three population means are equal, the dataset must be converted into a long table, which could be performed with the `stack()` function[1]:

```
dtf_long <- stack(dtf_wide)
myaov <- aov(values ~ ind, data = dtf_long)
```

The returned result of `aov()` is usually rearranged by `summary()` so as to display the p-value:

```
summary(myaov)
```

7.4 Regressions

7.4.1 Common Regression Models

Regression is a statistical tool that examines the strength and character of the relationship between two or more variables of interest in a mathematical way. Regression analysis allows us to determine whether a factor matters, or which factors matter the most and which factors can be ignored, and how they have influence on each other.

In a regression analysis, the main variable that we attempt to understand or predict is called the dependent variable (usually denoted by Y), while one or a series of other variables that we hypothesize have influence on the dependent variable are called independent variables (usually denoted by X). An example is the study on the effect of the level of cigarette consumption (or more factors) on the lung cancer rate. The lung cancer rate is the dependent variable Y, and the level of cigarette consumption is the independent variable X. The purpose of a regression analysis is to find how the dependent variable is influenced by the independent variable(s), based on a comprehensive dataset to work with. Here we introduce some common used regression models in public health, including the linear regression model, the logistic regression model, and the Cox regression model.

[1] The partner function `unstack()` converts a long table to a wide one. Section 6.2.1 mentions other ways for this conversion.

7.4.2 Linear Regression Model

The linear regression model includes the simple linear regression (SLR) model and the multiple linear regression (MLR) model. The SLR model demonstrates the correlation between the dependent and independent variables. It can also estimate the value of the dependent variable at the certain value of the independent variable. An example is to investigate the relationship between people's age and bone density. The MLR model uses several independent variables to predict the outcome of a dependent variable. An example is to investigate how sex, exercise, and air pollution level could affect people's life expectancy.

In the term *simple linear regression model*, the word *simple* means there is only one predictor, while *linear* means the relationship of Y on X is represented by a straight line. The form of the SLR model can be expressed as:

$$Y = \alpha + \beta X + \epsilon$$

where α is the intercept, β is the slope, and ϵ is the error term.

For example, we carry out a SLR analysis on the influence of the age on the average systolic blood pressure (BPSysAve) based on the *NHANES* dataset shipped by the **NHANES** package (Pruim 2015). We randomly pick out 60 observations from the dataset for a convenient demonstration.

```
data(NHANES, package = 'NHANES')
dtf <- NHANES[NHANES$Age > 17, c('Age', 'BPSysAve', 'Gender')]
dtf_sub <- dtf[sample(1:nrow(dtf), 60), ]
```

In R, linear regression models can be performed with the `lm()` (short for *linear model*) function.

```
lm_ba <- lm(BPSysAve ~ Age, data = dtf_sub)
```

Although the regressed coefficients (i.e. the intercept and the slope) can be viewed by printing `lm_ba`, more regression information can be extracted with the `summary()` function:

```
summary(lm_ba)
```

```
##
## Call:
## lm(formula = BPSysAve ~ Age, data = dtf_sub)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -21.667  -9.815  -0.652  10.168  41.612
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 102.3386     5.1322   19.94  < 2e-16 ***
## Age           0.4385     0.1105    3.97 0.000221 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 14.25 on 52 degrees of freedom
##   (6 observations deleted due to missingness)
## Multiple R-squared:  0.2326, Adjusted R-squared:  0.2179
## F-statistic: 15.76 on 1 and 52 DF,  p-value: 0.0002214
```

The output is separated into four parts:

1. The first part is the formula, indicating the regression model (`lm`), the dependent variable (`BPSysAve`), the independent variable (`Age`), and the data source (`dtf_sub`).
2. The second part is the residuals, displayed as the five numbers of the distribution. The residual is the difference between the observation and the regressed value. Smaller residuals mean better regression.
3. The third part is the fitted coefficients, including the estimated intercept and slope with their standard errors, the t-test statistics, the p-value (`Pr(>|t|)`), and the significant level (explained at the foot of the table). Three stars (***) here means that the test result is significant at $\alpha = 0.001$, which is explained in the line `Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1`.
4. The fourth part is overall fit information, including the residual standard error, R^2 (the percentage of the variability in the data that is accounted fo by the model), the F-test statistics with the p-value.

An scientific manuscript sometimes has to report the confidence intervals for the intercept and the slope, which could be obtained with the `confint()` function:

```
confint(lm_ba, level = 0.95)
```

```
##                  2.5 %      97.5 %
## (Intercept) 92.0400121 112.6371226
## Age          0.2168953   0.6601752
```

Alternatively, the `pubh::glm_coef()` function gives a more friendly table for the intercept and the slope with the standard errors and p-values (Table 7.3):

```
dtf_glm <- pubh::glm_coef(lm_ba, alpha = 0.05, type = 'ext')
```

Apart from the SLR model, `glm_coef()` supports the output of MLR model, generalised linear model (such as the logistic regression model), and the Cox Regression Model as well as some other models.

Table 7.3 Summary of a linear regression model. Generated by the `pubh::glm_coef()` function

Parameter	Coefficient	Std. Error	Lower CI	Upper CI	Pr(>|t|)
(Intercept)	102.34	5.13	92.04	112.64	< 0.001
Age	0.44	0.11	0.22	0.66	< 0.001

Table 7.4 Summary for the confidence interval of the coefficient of determination for a linear regression model. Generated by the `psychometric::CI.Rsqlm()` function

Rsq	SErsq	LCL	UCL
0.232626	0.0944306	0.0475454	0.4177067

Table 7.5 Prediction with the confidence interval for a linear regression model

Age	BPSysAve	ymin	ymax
20	111.1093	104.6921	117.5264
30	115.4946	110.6493	120.3399
40	119.8800	115.9304	123.8295
50	124.2653	120.0766	128.4541
60	128.6507	123.2361	134.0652
70	133.0360	125.9007	140.1714

The confidence interval of R^2 can be obtained with the `psychometric::CI.Rsqlm()` function (Fletcher 2022):

```
dtf_ci <- psychometric::CI.Rsqlm(lm_ba)
```

The obtained linear regression model could be used to predict the value of the dependent variable for a given value of the independent variable. The `jtools::make_predictions()` function (Long 2022) can do this job and gives the mean predictions and the 95% confidence intervals in an easy way (Tables 7.4 and 7.5):

```
dtf_pred <- jtools::make_predictions(lm_ba, new_data = data.frame(Age = 2:7 * 10))
```

The relationship of the dependent variable and the independent variable is often visualized in a scatter plot with the SLR model as a fitted straight line (sometimes with confidence intervals). The **ggplot2** package can do this job smoothly (Fig. 7.2):

```
library(ggplot2)
ggplot(dtf_sub, aes(Age, BPSysAve)) + geom_point() + geom_smooth(method = 'lm')
```

The `UsingR::simple.lm()` function (Verzani 2022) can visualize the diagnostics of the model with the confidence and prediction intervals, the residuals against the fitted values, the distribution of the residuals, and the Quantile-Quantile (QQ) plot for the residuals (Fig. 7.3):

```
dtf_naomit <- na.omit(dtf_sub)
UsingR::simple.lm(dtf_naomit$Age, dtf_naomit$BPSysAve, show.ci = TRUE, show.residuals = TRUE)
```

Set the argument `show.residuals = FALSE` and we will get only the first sub-figure, which should be sufficient for reporting the simple linear regression output in most times.

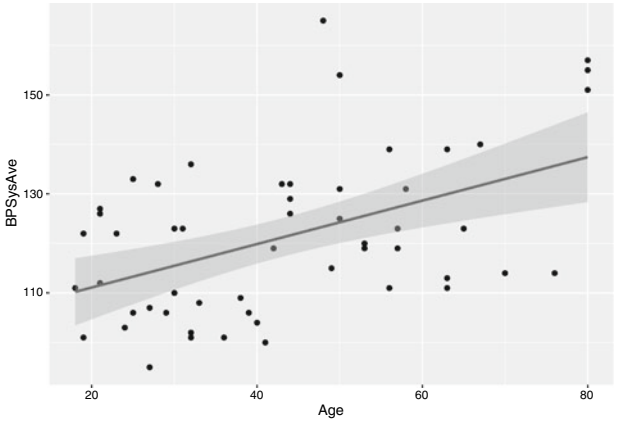


Fig. 7.2 Visualization of a simple linear regression model with a scatterplot and a fitted line

In the previous demonstration, we studied the effect of the age on the average systolic blood pressure. However, the age is not the only influential variable on the average systolic blood pressure. If we classify the dataset according to the gender, we can see that the fitted line for female has a visually different intercept and slope from that for the male, which indicates an interaction between the average systolic blood pressure and the gender. In other words, the independent variable has different impacts on the dependent variable across the levels of the categorical variable. For such cases, we could use the multiple linear regression (MLR) model.

The form of the MLR can be express as:

$$Y = \alpha + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + ... + \beta_t X_t + \epsilon$$

We assume that the average systolic blood pressure is dependent on both the gender and the age in a MLR:

$$P = \alpha + \beta_A A + \beta_G G + \epsilon$$

where P is the expected average systolic blood pressure and A is the age. We can use the `lm()` function for estimating the parameters (Table 7.6):

```
lm_bag <- lm(BPSysAve ~ Age + Gender, data = dtf)
summary(lm_bag)
```

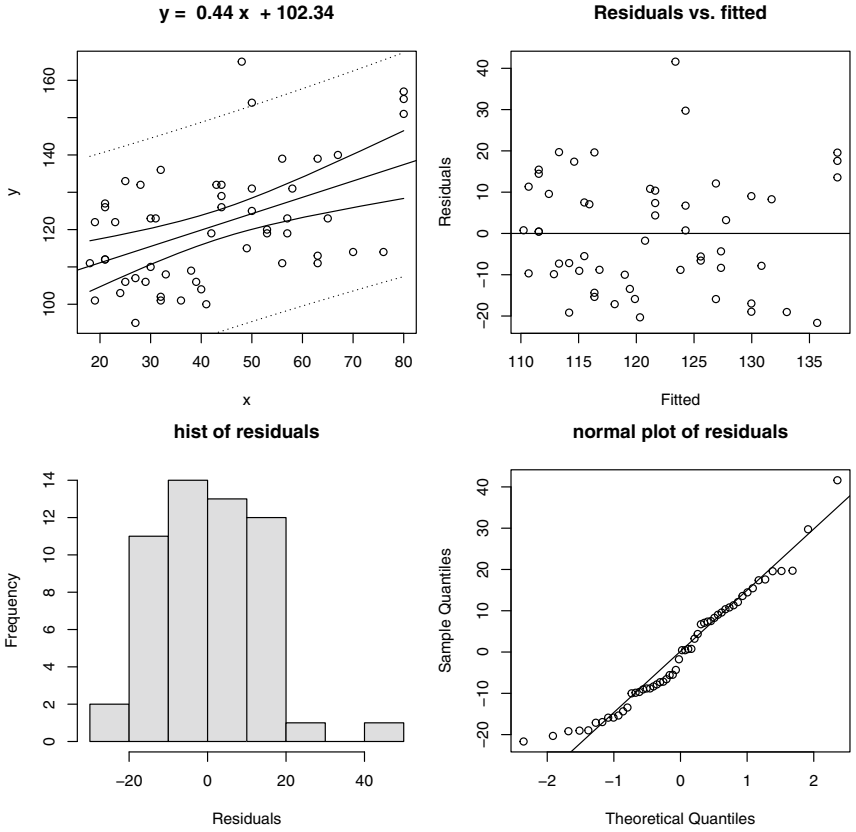


Fig. 7.3 Visualization of the diagnostics of a simple linear regression model

```
##
## Call:
## lm(formula = BPSysAve ~ Age + Gender, data = dtf)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -55.310  -9.216  -1.090   8.234 103.941
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 99.31118    0.55162  180.04   <2e-16 ***
## Age          0.41360    0.01039   39.82   <2e-16 ***
## Gendermale   4.56528    0.36201   12.61   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.35 on 7202 degrees of freedom
##   (276 observations deleted due to missingness)
## Multiple R-squared:  0.1911, Adjusted R-squared:  0.1909
## F-statistic: 850.9 on 2 and 7202 DF,  p-value: < 2.2e-16
```

Table 7.6 A look-up table showing the levels of a factor

Level	Value
Female	0
Male	1

The output format is similar to the previous example of the SLR regression, only except for two additional rows, beginning with `Gendermale`, in the part for the coefficients. Thus, the parametrized equation is:

$$P = 99.3 + 0.414 \cdot A + 4.56 \cdot G$$

where G is the gender (1 for male and 0 for female). It is a little bit tricky in R to make clear of the value of a categorical variable. By default, the levels of the factor are ordered alphabetically. Thus, `female` and `male` are labelled as Level 1 and Level 2, which can be converted into integers:

```
as.integer(factor(levels(dtf$Gender)))
```

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

We can create a small table as a dictionary showing the numerically matched value of each gender:

```
dtf_dict <-  data.frame(level = levels(dtf$Gender),
                        value = as.integer(factor(levels(dtf$Gender))) - 1)
```

In the this model, we do not take into account the interaction of the independent variables, i.e. the slope of the dependent variable against one independent variable does not change with the other dependent variable, which might be improper. For such a case, the MLR model can take the categorical variable with the interaction, i.e. the gender in this example, which is indicated with an asterisk (`*`) rather than a plus in the formula:

```
lm_bagi <- lm(BPSysAve ~ Age * Gender, data = dtf)
summary(lm_bagi)
```

```
##
## Call:
## lm(formula = BPSysAve ~ Age * Gender, data = dtf)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -51.731  -9.298  -1.030   8.112 102.977
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    93.57319    0.71146  131.52   <2e-16 ***
## Age             0.53544    0.01413   37.90   <2e-16 ***
```

```
## Gendermale     16.53181    1.01714   16.25   <2e-16 ***
## Age:Gendermale -0.25878    0.02059  -12.57   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.18 on 7201 degrees of freedom
##   (276 observations deleted due to missingness)
## Multiple R-squared:  0.2085, Adjusted R-squared:  0.2082
## F-statistic: 632.3 on 3 and 7201 DF,  p-value: < 2.2e-16
```

These two rows, beginning with `Gendermale` and `Age:Gendermale` in the part for the coefficients, show the impact of the gender on the relationship between the average systolic blood pressure and the age. For the female, the average systolic blood pressure can be estimated as:

$$P_{female} = 93.6 + 0.535 \cdot A_{female}$$

For the gender of male, the equation is:

$$P_{male} = (93.6 + 16.5) + (0.535 - 0.259) \cdot A_{male}$$

We can combine these two equations into one as:

$$P = 93.6 + 0.535 \cdot A + 16.5 \cdot G - 0.259 \cdot A \cdot G$$

The `confint()`, `psychometric::CI.Rsqlm()`, and `jtools::make_predictions()` functions can be applied to the multiple regression model as well to obtain the confidence intervals for the regressed intercept, the slope, R^2, and the predictions. Note that the linear regression model gives two R^2: multiple R^2 and adjusted R^2. For interpreting the MLR model, we should use the adjusted R^2, which has been corrected with taking into account the influence of adding independent variables.

7.4.3 Logistic Regression Model

The logistic regression model is a type of the generalized linear model (GLM), which is an extension to the linear regression model and allows the dependent variables to be binary or dichotomous. It is widely used in public health, especially in epidemiological studies. An example is to investigate the relationship between the age, sex and whether the people have hearing loss or not. The logistic regression model is well suited for the screening of risk factors in epidemiology. As public health research often requires categorical variables, such as infected and uninfected, survival and death, which violate the assumptions of linear regression models. Therefore, the logistic regression model can provide more meaningful results for public health research. Logistic regression models can be categorized into unconditional

logistic regression and conditional logistic regression. Both assume that the data obeys Bernoulli distribution (either positive or negative, with the probability sum of 1). For example, in a study on the individual's odds of getting lung cancer, the independent variable is smoking years (continuous), while the dependent variable is the lung caner status (yes/no, binary).

The form of the logistic regression model can be express as:

$$\text{logit}(\pi) = \log\left(\frac{\pi}{1-\pi}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + ... + \beta_t X_t$$

where π is the probability that an event occurs, and $\frac{\pi}{1-\pi}$ is then the odds ratio. If there is only one independent variable, it is called a simple logistic regression model. If there are multiple independent variables, it is called a multiple logistic regression model.

Here we use the *melanoma* dataset from the **boot** package for demonstration of how to apply the logistic model. This dataset is introduced in Chap. 6.

```
data(melanoma, package = "boot")
```

For demonstrating the simple logistic regression model, we assume that the thickness of the tumour is the only independent variable and the patient's status is the dependent variable. For simplicity, we select the patients with the status "3", which indicates that they died from causes unrelated to their melanoma.

```
melanoma2 <- melanoma[melanoma$status != 3, c('status', "thickness")]
melanoma2$status[melanoma2$status == 2] <- 0
```

Then the logistic regression model is performed with the `glm()` function:

```
glm_st <- glm(status~thickness, data = melanoma2, family = "binomial")
summary(glm_st)
```

```
##
## Call:
## glm(formula = status ~ thickness, family = "binomial", data = melanoma2)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8844  -0.7485  -0.6729   1.2162   1.8588
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.61134    0.25496  -6.320 2.62e-10 ***
## thickness    0.24853    0.06354   3.911 9.18e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 232.84  on 190  degrees of freedom
## Residual deviance: 213.45  on 189  degrees of freedom
## AIC: 217.45
##
## Number of Fisher Scoring iterations: 4
```

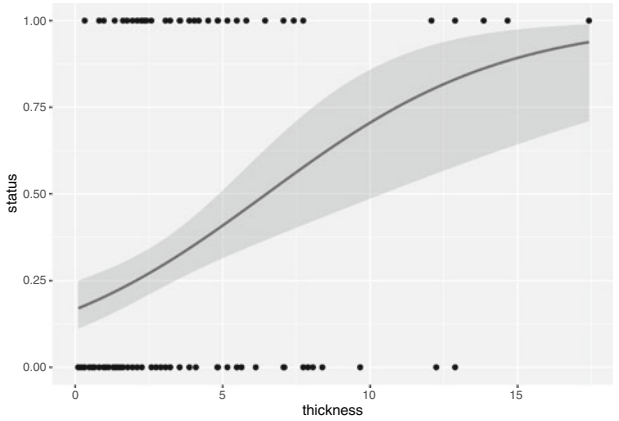


Fig. 7.4 Visualization of a logistic regression model

The confidence intervals for the estimated slope and intercept along with the odds rations can be obtained:

```
confint(glm_st)
```

```
##                  2.5 %     97.5 %
## (Intercept) -2.1348608 -1.1315007
## thickness    0.1319243  0.3825038
```

```
exp(confint(glm_st))
```

```
##                2.5 %    97.5 %
## (Intercept) 0.118261 0.3225488
## thickness   1.141022 1.4659504
```

The fitted curve can be visualized either with the basic R functions:

```
plot(melanoma2$thickness, melanoma2$status)
curve(predict(glm_st, data.frame(thickness = x), type = 'response'), add = TRUE)
```

or with the **ggolot2** functions (Fig. 7.4):

```
ggplot(melanoma2, aes(thickness, status)) +
  geom_point() +
  geom_smooth(method = "glm", method.args = list(family = "binomial"))
```

For demonstrating the multiple logistic regression model, we assume that the thickness of the tumour and the existence of the ulceration are the independent variables.

```
melanoma3 <- melanoma[melanoma$status != 3, c('status', "thickness", "ulcer")]
melanoma3$status[melanoma3$status == 2] <- 0
```

Now we apply the `glm()` function with multiple independent variables:

```
glm_stu <- glm(status~thickness + ulcer, data = melanoma3, family = "binomial")
summary(glm_stu)
```

```
##
## Call:
## glm(formula = status ~ thickness + ulcer, family = "binomial",
##     data = melanoma3)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7332  -0.6453  -0.5229   1.0787   2.0725
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.07388    0.31013  -6.687 2.28e-11 ***
## thickness    0.15782    0.06313   2.500 0.012425 *
## ulcer        1.39220    0.37229   3.740 0.000184 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 232.84  on 190  degrees of freedom
## Residual deviance: 198.98  on 188  degrees of freedom
## AIC: 204.98
##
## Number of Fisher Scoring iterations: 4
```

From the results of the logistic analysis we can see that the patient's status was significant for both the tumour thickness and the existence of ulceration. The confidence intervals for the estimated slope and intercept along with the odds rations can be obtained in the same way as shown previously.

7.4.4 Cox Regression Model

The Cox regression model or the proportional hazards model is used to predict the probability of the event of interest that occurs at a given time. An example is to estimate the probability of patients with gastric cancer who will die within a year. The Cox regression model is usually used in survival analysis in public health, especially in the epidemiology. Survival analysis aims at evaluating the difference in survival probabilities between two or more circumstances (Parodi et al. 2008). For instance, survival analysis could be applied to assess the influence of various risk factors, such as pollutants, genetic factors and so on, on exposed groups compared with unexposed ones. The outcome of interest in survival analysis is the **event**, which could be deaths, recurrence of specific diseases, and so on. The data that measure the time for the event is called **event time**. For example, the cancer event data could be the

time from surgery to death. It is critical to clearly state the event, the event time, and when the period of observation starts and finishes. Additionally, what makes survival analysis be significantly different from other analysis is the fact called **censoring**, which means only some individuals experienced the event due to some reasons, so other individuals' survival times are unknown (Clark et al. 2003).

Survival data could be described and modelled by survival probability and hazard probability. The survival probability could be denoted by $S(t)$, which means the probability that an individual survives from the original observation time to the specific future time t ($S(t) = Pr(T > t)$, where T is the observation time period). In particular, the survival probability is 1 when $t = 0$, i.e. $S(t_0) = 1$.

The hazard probability could be denoted by $h(t)$ or $\lambda(t)$, which measures the probability that the event of an individual occurs at the specific time t during the observation period. Generally speaking, the survival probability illustrates the cumulative non-occurrence while the hazard probability reflects the incident event rate.

The Cox regression model is a semi-parametric model that could quantify the effect size of one or multiple variables, expressed as (Bogaerts and Lesaffre 2018):

$$h(t|X_i) = h_0(t) \exp(\beta_1 X_{i1} + \cdots + \beta_p X_{ip})$$

where $\beta_1, \beta_2, ...\beta_m$ are the partial regression coefficients of the independent variable, and $h_0(t)$ is the underlying baseline hazard rate when $X_i = 0$. The hazard rate is the instantaneous probability density of death at the current time point t for individuals who have survived time t, further indicating the estimated hazard.

The hazard ratio (HR) is the measure of the relative survival experience in two groups, and could be calculated by Clark et al. (2003):

$$HR = \frac{O_1/E_1}{O_2/E_2}$$

$HR > 1$ means the variable is the risk factor, $HR = 1$ means the variable has no effect, and $HR < 1$ means the variable is the protective factor.

The **survival** package (Therneau 2022) is a useful tool to carry out the survival analysis and the Cox regression. Basically, **survival** could create survival objects, estimate the survival probability and time, compare the survival time between various groups, and assess the effect of some factors on the survival probability based on the Cox regression model. The Cox regression model could be developed via the `coxph()` and `cox.zph()` functions.

Here we use a dataset called *lung* from the **survival** package as an example. The dataset came from a study about the patients with advanced lung cancer from the North Central Cancer Treatment Group. The variables `time`, `status`, and `sex` indicate the survival time in days, the censoring status (1 = censored, 2 = dead), and the sex (male = 1, female = 2).

The `coxph()` function analyzes the effect of the factor of interest on samples' survival probability based on the Cox regression model.

```
library(survival)
```

```
coxph(Surv(time, status) ~ target variable, data = data set)
```

For example:

```
coxph(Surv(time, status) ~ sex, data = lung)
```

```
## Call:
## coxph(formula = Surv(time, status) ~ sex, data = lung)
##
##        coef exp(coef) se(coef)      z       p
## sex -0.5310    0.5880   0.1672 -3.176 0.00149
##
## Likelihood ratio test=10.63  on 1 df, p=0.001111
## n= 228, number of events= 165
```

The `coef` column shows the coefficient of 'sex' which is -0.5310. It implies the survival probability of females is higher than males. Additionally, `exp(coef) = 0.5880` indicates that the risk of death for females decreases by 41.20% (1–58.80%) compared to males. The p-value 0.00149 also shows the statistical significance.

The `cox.zph()` function tests the fitness of the Cox regression model.

```
cox.zph (coxph(formula, data=data set))
```

The `formula` uses the `Surv` statement. For example:

```
cox.zph(coxph(Surv(time, status) ~ sex, data = lung))
```

```
##        chisq df     p
## sex     2.86  1 0.091
## GLOBAL  2.86  1 0.091
```

The result indicates that the p-value of sex is 0.091, which has no statistical significance. Thus, it implies 'sex' meets the assumption of proportional hazards.

7.5 Exercises

The dataset *S.typh* shipped by the **Epi** package comes from a study on a Salmonella Typhimurium outbreak, which took place in a county in Denmark, 1996. Telephone interviews were carried out in the participants about their food intake during the last two weeks. We assume that the food types have effect on the Salmonella Typhinurium. Conduct an unconditional logistic regression.

References

Bogaerts, K., A. Komarek, and E. Lesaffre. 2018. *Survival Analysis with Interval-Censored Data : A Practical Approach with Examples in R, SAS and BUGS.* Chapman & Hall/CRC Interdisciplinary Statistics Series. CRC Press/Taylor; Francis Group. https://doi.org/10.1201/9781315116945.

Clark, T.G., M.J. Bradburn, S.B. Love, and D.G. Altman. 2003. Survival Analysis Part I: Basic Concepts and First Analyses. *British Journal of Cancer* 89 (2): 232. https://doi.org/10.1038/sj.bjc.6601118.

Fletcher, Thomas D. 2022. *psychometric: Applied Psychometric Theory*. https://CRAN.R-project.org/package=psychometric.

Long, Jacob A. 2022. *jtools: Analysis and Presentation of Social Scientific Data.* https://jtools.jacob-long.com.

Parodi, Stefano, Ezio Bottarelli, et al. 2008. Survival Analysis in Epidemiology: A Brief Introduction. *Annali Della Facoltá Di Medicina Veterinaria, Universitá Di Parma* 28: 17–42.

Pruim, Randall. 2015. *NHANES: Data from the US National Health and Nutrition Examination Study*. https://CRAN.R-project.org/package=NHANES.

Therneau, Terry M. 2022. *survival: Survival Analysis*. https://github.com/therneau/survival.

Triola, Marc M., Mario F. Triola, and Jason Roy. 2019. *Biostatistics for the Biological and Health Sciences.* Pearson Education Limited.

Verzani, John. 2022. *UsingR: Data Sets, Etc. For the Text "Using R for Introductory Statistics", Second Edition*. https://CRAN.R-project.org/package=UsingR.

Chapter 8
Visualizing Data

8.1 Chapter Highlights

- Make sense of the role that data visualization plays in academic research,
- Choose appropriate graphics for visualizing categorical and numerical data, and
- Generate common graphics, including pie charts, bar chats, doc chats, strip charts, histograms, box and violin plots, and scatterplots, with R functions.

8.2 Introduction

Data visualization refers to the representation of data in a pictorial or graphical format. In most cases, human brains are unable to process a dataset as efficiently as a picture which contains the same information. As it is said that *a picture is worth a thousand words*, good data visualizations may represent information that is difficult to describe in words.

Data visualization is important because it has a lot of advantages. Visualizing vast volumes of complicated data with charts or graphs is easier than going over spreadsheets or statements. When compared to words, graphs focus readers' attention to the most important messages. Data visualization can translate complex data linkages and data-driven insights into an easy-to-understand format. With the assistance of data visualization, researchers can better discover and explain patterns, trends, and relationships. The modern computer techniques allow interactive visualization for data, with which we can drill down into charts and graphs for more details by changing what data we see and how it's processed. In public health, data visualisation is used in teaching and learning, presentations, dissertations, journal publications, and project applications.

Academic graphs are the visual representation of experimental and theoretical data that provide evidence for supporting a paper's findings. They play an essential role in academic publications, where the experimental results are usually the focus of

P. Zhao, *Working with Data in Public Health: A Practical Pathway with R*,
https://doi.org/10.1007/978-981-99-0135-7_8

the study. Many academic journals provide guidelines for graphics in the submission. As universal requirements, the graphs should:

- fulfil the specifications of the journal in terms of format and resolution (including but not limited to the colour, size, resolution, font and size of labels in the graphs),
- clearly express the information in the data,
- completely reflect the relevant information of the data, and
- be simple and aesthetically appealing to the readers.

Academic graphs can be roughly classified as black-and-white graphs and coloured graphs. In black-and-white charts, data are often presented with shapes or filled textures. Many traditional journals do not accept coloured graphs not only for reducing printing costs, but also for being portable in different media. Online publications often use coloured graphics, which are more aesthetically pleasing, and the colours tend to enrich the expression of the data. We must keep it in mind that abusing colours might be problematic for some readers, especially for colour-blind people.

8.3 Plotting Systems in R

R supports a variety of plotting systems for data visualization, and each system is able to generate a variety of types of graphics. The base R plotting system is a set of traditional tools for plotting graphics. In recently years, many more advanced R packages, such as the **ggplot2** and **plotly** packages, have been developed for plotting elegant and modern graphs, which are very powerful and helpful in public health studies. With these systems, we can visualize the data in all kinds of graphics for different usages in public health.

8.3.1 Base R

Base R functions are a conventional way for data visualization for decades. In Chap. 6, we demonstrate using the `fecitr::plot_summary()` function for a preliminary visualization of the *diet* dataset. This function integrates base R plotting functions of scatterplots, box plots, bar charts, histograms, and density curves, for one dimensional data.

The base R plotting functions are categorized as high-level functions and low-level ones (Table 8.1). The high-level functions work alone and establish the coordinate systems as well as decide the graphic type and the major plotting elements. The low-level functions cannot work alone. They are used to add elements to the existing graphics generated by the high-level functions. We can see some examples which show how they work:

```
demo(graphics)
demo(persp)
demo(image)
```

Table 8.1 Common base R plotting functions

Function	Description
`barplot()`	Generate a bar chart
`boxplot()`	Generate a box plot (also called box-and-whisker plot)
`contour()`	Generate a contour plot
`coplot()`	Generate a conditional plot
`curve()`	Generate a curve corresponding to a function
`dotchart()`	Generate a Cleveland dot plot
`hist()`	Generate a histogram
`image()`	Generate a grid of colored or gray-scale rectangles with colors corresponding to the values
`matplot()`	Wrapper of `plot()` for plotting columns of one matrix against columns of another
`pairs()`	Generate a matrix of scatterplots
`persp()`	Generate a perspective plot of a surface
`pie()`	Generate a pie chart
`plot()`	Generic plotting function, depending on the object's class
`stripchart()`	Generate a one dimensional scatter plot
`stars()`	Generate a star plot or segment diagram of a multivariate data set
`symbols()`	Generate a plot with symbols (circles, squares, rectangles, stars, thermometers, and boxplots) representing data values
`ts.plot()`	Plot multiple Time Series
`abline()`	Add straight lines to a graph
`arrows()`	Add arrows to a graph
`axis()`	Add an axis to a graph
`box()`	Add a surrounding box to a graph
`legend()`	Add legends to a graph
`lines()`	Add connected line segments to a graph
`mtext()`	Add text to the margins of a graph
`points()`	Add points to a graph
`polygon()`	Add polygons to a graph
`rect()`	Add a rectangle to a graph
`rug()`	Add short lines representing data values to a graph
`segments()`	Add line segments to a graph
`text()`	Add text to a graph

8.3.2 ggplot2

The **ggplot2** package, unlike the base R plotting functions, visualizes data by layers, separating data-related plots from data-independent plots in the plotting process, making the plotting method more flexible and the resulting graph more aesthetically pleasing. The dataset usually has to be prepared as a data frame before being visualized with **ggplot2**. The plotting procedure often starts with the function `ggplot()`

as the initialization, followed by layers with + as a connector between them. The layers could either add elements above the previous layer or customize the settings of the graphic (Table 8.2). The following code is a minimal example for generating a scatterplot for the *diet* dataset in the **Epi** package:

```
data(diet, package = "Epi")
library(ggplot2)
ggplot(diet) + # initialization
  geom_point(aes(height, weight)) + # add scatter points
  theme_bw() # set the theme
```

Some examples of **ggplot2** can be shown as:

```
example(qplot)
```

Lots of R packages, based on **ggplot2**, have been developed for creating many kinds of graphs. For example, the `ggpairs()` function in the **GGally** package (Schloerke et al. 2021) is recommended for visualizing the relationships of paired variables. It integrates bar plots for one dimensional qualitative data, distribution density curves for one dimensional quantitative data, histograms and box plots for quantitative data grouped by the qualitative variable (if any), and scatterplots for each pair of quantitative variables in the data set. Here is an example using the *diet* dataset in the **Epi** package (Fig. 8.1):

```
library(GGally)
p1 <- diet[, c('height', 'weight', 'fat', 'fibre', 'chd')] |>
  transform(chd = factor(chd)) |>
  ggpairs(aes(colour = chd, alpha = 0.1)) +
  theme_bw()
p1
```

8.3.3 plotly

Data visualization is often limited by spatial limitations of charts, but interactive graphs allow the viewer to zoom in on areas of interest and hover over information, which means that a wealth of details can be retained in interactive graphs. The WHO COVID-19 Dashboard[1] is a good example, which allows the viewer to hover over a country/area to obtain information about that area (including confirmed case and mortality). The **plotly** package can generate a variety of elegant interactive graphs, which could be embed in webpages and interact with local or online users. Furthermore, interactive graphs attract readers attention to the graphs, and ensures the clarity of the graphs. It provides two universal functions, i.e. `plot_ly()` for directly plotting graphs, and `ggplotly()` for converting a **ggplot2** graphic into a **plotly** one:

```
library(plotly)
ggplotly(p1)
```

[1] https://covid19.who.int/.

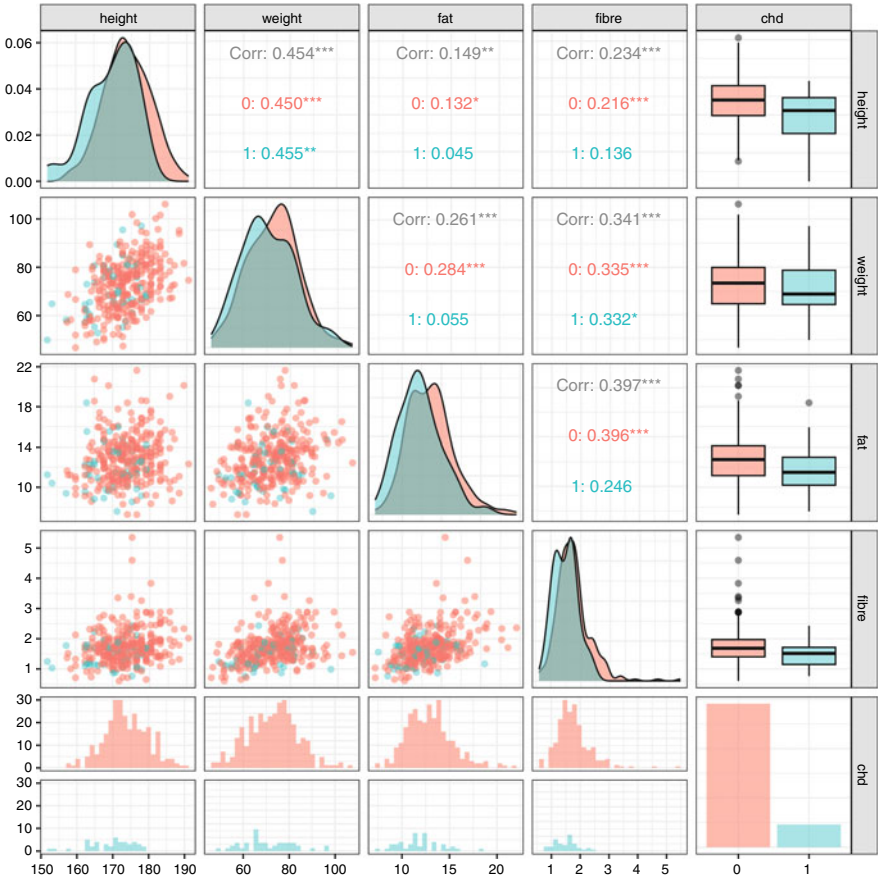


Fig. 8.1 Visualization of the *diet* dataset with a pair plot generated by the **GGally** package

We can see more demonstrations of these plotting methods with:

```
library(plotly)
demo('sf-plotly-3D-globe')
```

8.3.4 Other Systems

The **lattice** package (Sarkar 2021) is a traditional plotting system based on the **grid** package. It features the ability of plotting graphs grouped by categorical variables. For example, we can draw box plots of the fat intake (numerical) for each gender (categorical) and CHD event (categorical) with the `bwplot()` function (Fig. 8.2) .

Table 8.2 Common **ggplot2** plotting functions

Function	Description
`geom_abline()`, `geom_hline()`, `geom_vline()`	Straight lines
`geom_area()`, `geom_ribbon()`	Area and ribbon plot
`geom_bar()`	Bar chart
`geom_histogram()`, `geom_freqpoly()`, `geom_bin2d()`, `geom_hex()`	1D/2D histogram or frequency polygons
`geom_boxplot()`	Box plot
`geom_contour()`	Contour plot
`geom_crossbar()`, `geom_errorbar()`, `geom_errorbarh()`, `geom_linerange()`, `geom_pointrange()`	Data value with intervals
`geom_density()`, `geom_density2d()`	Distribution density curve/surface
`geom_line()`, `geom_path()`	Line chart
`geom_point()`	Scatterplot
`geom_polygon()`, `geom_rect()`, `geom_tile()`	Polygons/rectangles
`geom_rug()`	Rugs
`geom_segment()`, `geom_curve()`	Line segments and curves
`geom_smooth()`, `geom_quantile()`	Fitted lines
`geom_text()`	Text

```
library(lattice)
bwplot(job ~ fat | factor(chd), data = diet)
```

The **lattice** package provides a series of conditional functions for plotting scatterplots (`xyplot()`), dot charts (`dotplot()`), bar charts (`barchart()`), and strip charts (`stripplot()`), and histograms(`histogram()`).

The **dygraphs** package (Vanderkam et al. 2018) is an R interface to the *dygraphs* JavaScript library, which generates interactive visualization time series data. We give an example in Sect. 8.8.4.

The **rgl** package (Adler and Murdoch 2022) provides a series of functions for plotting 3D graphs, such as scatterplots, line charts, and perspective plots of a surface. These graphs are interactive: we could zoom in and out, rotate, and move them with the mouse. They are sometimes very helpful when exploring data. We give an example in Sect. 8.8.3.

In the subsequent sections, we focus on the features of common graphics and how to plot them with the base R functions as well as the **ggplot2** and **plotly** packages.

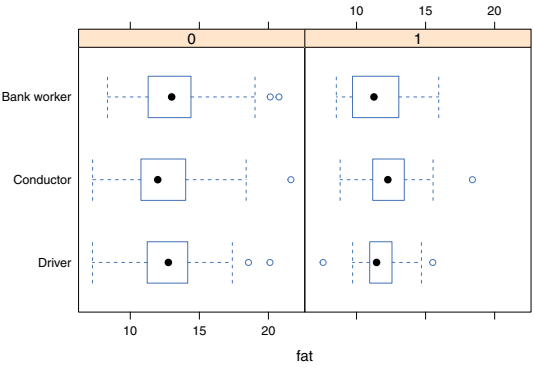


Fig. 8.2 A box plot generated by the **lattice** package

8.4 Pie Charts

Pie charts are often used for displaying the counts or relative frequency (percentage) of qualitative variables. A pie chart is a circle divided into sections that represent parts of a whole as slices of a pie. It can be regarded as the graphical form of a one-way frequency table.

We use the frequency table `tab1` generated in Chap. 6 for demonstration. The base R function `pie()` generates a pie chart for the (relative) frequency of each status:

```
tab1 <- sort(tab1, decreasing = TRUE)
pie_label <- paste0(names(tab1), ': ', tab1,
                    ' (', round(prop.table(tab1) * 100), '%)')
pie(tab1, labels = pie_label, density = 6, angle = 120 * 1:3)
```

We generate the labels for the pie chart before using the `pie()` function, in which we set the density and the angles of the shading lines for the three slices (Fig. 8.3a). Other arguments allow customizing the filling colours, the radius of the pie, the edges, the border, and so on.

The **ggplot2** package does not originally support pie charts. Therefore, we have to create a bar plot first and change it to a pie chart with the function

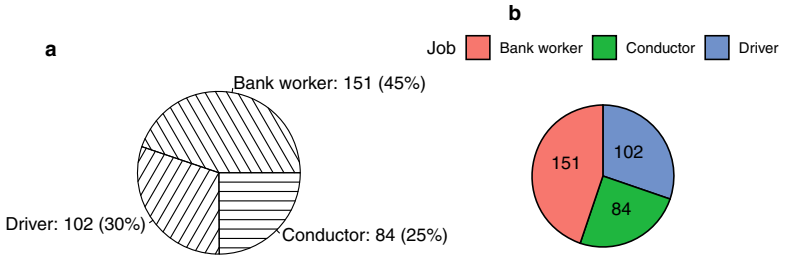


Fig. 8.3 Pie charts of the status in the *melanoma* dataset

`coord_polar()` and remove the axes as well as other redundant elements with the function `theme_void()`:

```
dtf1 <- data.frame(n = c(tab1), Job = names(tab1))
library(ggplot2)
ggplot(dtf1, aes(x = "", y = n, fill = Job))+
  geom_bar(stat = "identity", width = 1) +
  coord_polar(theta = 'y', start = 0) +
  theme_void()
```

which seems lengthy. Instead, the `ggpubr::ggpie()` function (Kassambara 2022) generates the pie chart in a simple way (Fig. 8.3b):

```
library(ggpubr)
ggpie(dtf1, "n", fill = "Job")
```

The **plotly** package generates interactive pie charts with the `plot_ly()` function:

```
library(plotly)
plot_ly(data.frame(tab1), labels = ~Var1, values = ~Freq, type = 'pie')
```

We can see the level of the status, the count, and the relative frequency if the cursor is placed over the pie chart. Many interactive buttons are displayed on the top of the graph, which allow us to zoom in and out of specific areas by clicking on them. Also, by clicking on the legend on the right side, we can make the image display only the data of the individual category.

Pie charts could be plotted in 3D form with the `plotrix::pie3D()` function:

```
library(plotrix)
pie3D(tab1, labels = pie_label, explode = 0.1)
```

Pie charts work well for the qualitative data with only a few levels. It is often argued that pie charts are a bad idea for data visualization as humans are not so sensitive to angles/areas as to positions. Three dimensional pie charts are even worse. Especially when we have many groups, a pie chart could be a disaster, as the readers could be confused in distinguishing them and feeling hard to get the differences in the sizes of the pie slices. However, pie charts are still often seen not only in public health but also in other research areas. Bar plots are often a better alternative than pie charts for displaying the same information.

8.5 Bar Charts

8.5.1 For One-Way Frequency Tables

Bar charts are used to compare values across a few categories. They usually serve as the graphical form of one-way or two-way frequency tables.

Bar charts can be generated with the base R function `barplot()`, or the **ggplot2** function `geom_bar()`, or the **plotly** method:

```
barplot(tab1, horiz = TRUE, las = 1)
dtf1 |>
  mutate(Job = reorder(Job, -n)) |>
  ggplot() +
  geom_bar(aes(x = Job, y = n), stat = 'identity') +
  coord_flip()
plot_ly(x = dtf1$n, y = dtf1$Job, type = "bar", orientation = 'h')
```

The bars are placed horizontally (via the argument `horiz` of `barplot()`, `orientation` of `plot_ly()`, and the function `coord_flip()` for **ggplot2**) for both the aesthetic and economic purposes: vertical bars might be too wide and would occupy too much space in the height, and the labels for the categorical variable might be too long to be shown at the horizontal axis, as there are only three bars in Fig. 8.4. Compared to the pie charts, it is easier to identify the difference in the lengths of bars.

The `geom_bar()` function accepts the data frame with the tabulated data as input (which is a more common way to use it), thus no statistic transformation is performed (`stat = 'identity'`). Alternatively, it can merge the tabulation calculation with the plotting in one step:

```
ggplot(diet) + geom_bar(aes(job))
```

8.5.2 *For Two-Way Contingency Tables*

Bar charts are more often used for visualizing two-way contingency tables. The plotting functions are the same as for the one-way contingency tables, but with more customization:

```
barplot(tab2, xlab = 'CHD',
        density = 6, angle = 120 * 1:3, beside = TRUE, col = 1,
        legend.text = TRUE, args.legend = list(
          x =  8, y = 100,
          legend = dimnames(tab2)$job,
          bty = 'n',
          density = 15, angle = 120 * 1:3))
ggplot(diet) +
  geom_bar(aes(chd, fill = job), position = 'dodge')
plot_ly(dtf2, x = ~chd, y = ~n, color = ~job, type = "bar")
```

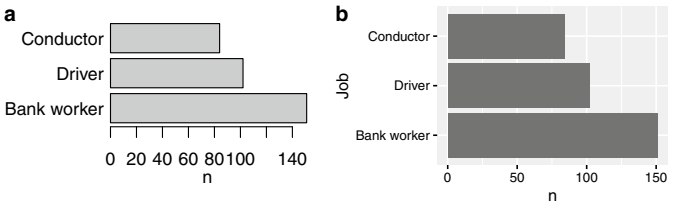


Fig. 8.4 Bar charts for a one-way frequency table

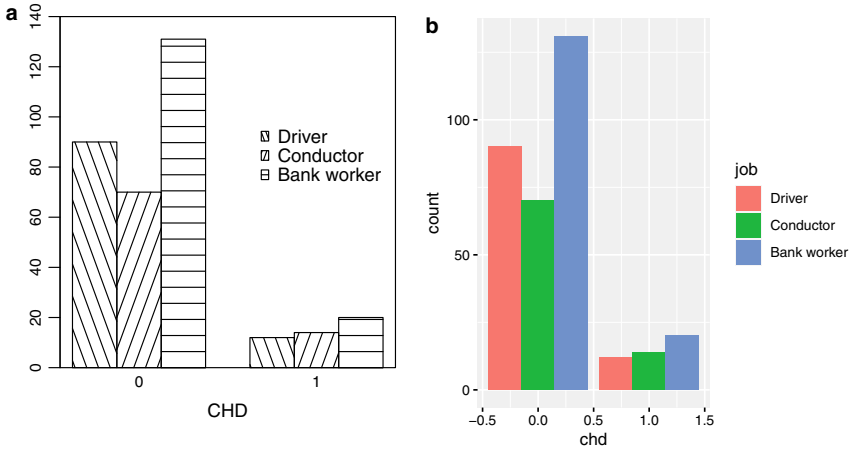


Fig. 8.5 Bar charts for a two-way contingency table

The figures plotted by the base R function and **ggplot2** are shown in Fig. 8.5. As there are two categorical variables, we have to use textures or colours to fill the bars for distinguishing them. In `barplot()`, the input data is a pre-calculated matrix with the frequency (`tab2`), while the arguments `density`, `angle`, and `col` determine how the filling textures look like, and `legend.text` and `args.legend` determine how the legend looks like. In the **ggplot2** functions, the input data is a data frame with the raw data (`diet`), and the argument `fill` determines filling the bars with colours according to `job`. In `plot_ly()`, the input data is a pre-calculated data frame with the categorical variable's levels and frequency (`dtf2`), and `color` works the same way as `fill` in `geom_bar()`.

The bars are placed side by side, as we set `beside = TRUE` and `position = 'dodge'`. If we want to highlight the marginal sums of a categorical variable, we could use stacked bars by using their default values (simply remove them from the function). The **plotly** package, however, plots side-by-side bars by default, and we have to use the `layout()` function for changing the bar mode as `'stack'`:

```
barplot(tab2, xlab = 'CHD')
ggplot(diet) + geom_bar(aes(chd, fill = job))
plot_ly(dtf2, x = ~chd, y = ~n, color = ~job, type = "bar")  |>
  layout(barmode = 'stack')
```

8.5.3 *For Multiple-Way Tables*

Three- or four-way tables can be visualized in grouped bar plots, which can be generated with the `facet_wrap()` or the `facet_grid()` function in the **ggplot2**

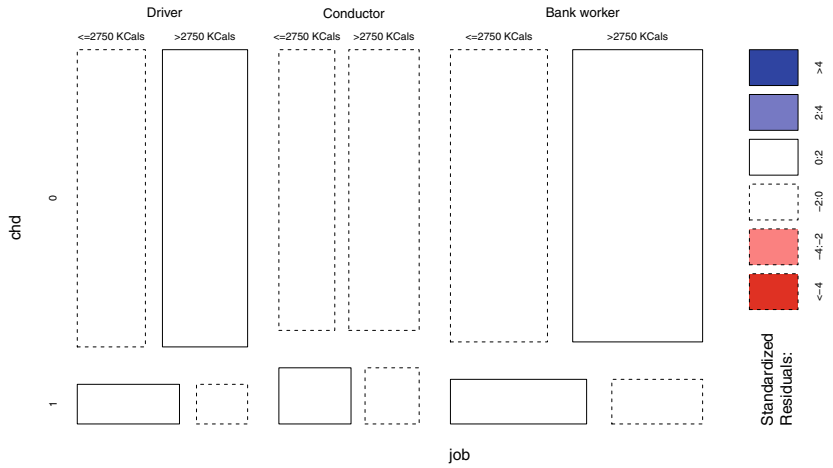


Fig. 8.6 A mosaic plot for visualizing multiple-way table

package. Here we display the count in the combination of the categorical variables `chd`, `job`, and `energy.grp`:

```
ggplot(diet) +
  geom_bar(aes(chd, fill = job)) +
  facet_grid( ~ energy.grp) # or facet_wrap( ~ energy.grp)
```

The more dimensions, the more difficult for readers to understand. If there are more than four categorical variables, bar plots might be very confusing, and a mosaic plot may be a better choice:

```
mosaicplot(tab3, shade = TRUE)
```

The mosaic plot draws a rectangle for each frequency in the table and arranges all the rectangles in a hierarchical way (Fig. 8.6). The size of the rectangles represents the frequency, and the height and width represent the proportions. Figure 8.6 can be, at the first sight, divided into three columns (i.e. `Driver`, `Conductor`, and `Bank worker`), and their widths indicate that `Bank worker` > `Driver` > `Conductor`. Within each column there are two rows, i.e. the upper one is the frequency of `chd` = 0 and the lower one `chd` = 1. Comparing the heights of the rows, we can see that the frequency of `ch` = 0 is larger than that of `ch` = 1 in every job. Within the rows in every job, we can compare the frequency between the energy groups according to the rectangle widths. For example, within the second row (`ch` = 1) of the first column (`job` = `Driver`), the width of the rectangle of `<=2750KCals` is larger than that of `>2750KCals`, indicating the frequency of the former energy groups is greater than that of the latter. Apart from the comparison of the frequencies, the

mosaic plot indicates the residuals of the fitting models (Pearson's χ^2 by default): a dashed border indicates a negative residual, while a solid one does a positive. The colours in the legend shows the range of the residuals.

8.6 Dot Charts and Strip Charts

Dot charts and strip charts are often used for displaying raw quantitative data in one dimension. In a dot chart, each value of the data, from the first observation to the last, is represented by a dot, mapped to one axis. A strip chart, also called a one-dimensional scatterplot, also displays each value of the data as a dot, but orders them by the magnitude. Here we draw a dot chart with the `dotchart()` function and two strip charts with the `stripchart()` function, all for the tumour thickness data of the *melanoma* dataset in the **boot** package.

```
# Define a colour and a label for plotting
thecol <- rgb(0, 0, 0, alpha = 0.5)
thexlab <- 'Tumour thickness (mm)'
```

```
dotchart(melanoma$thickness, pch = 15, xaxt = 'n', lcolor = "white")
stripchart(melanoma$thickness, col = thecol, pch = 15, xaxt = 'n')
stripchart(melanoma$thickness, col = thecol, pch = 15, xlab = thexlab, method = "stack")
```

Figure 8.7a displays the raw data without any transformation. Figure 8.7b prints all the values at the position mapped to the horizontal axis in one strip, even when some of them are overlapped. We set the transparency of the doc colour as `alpha = 0.2`

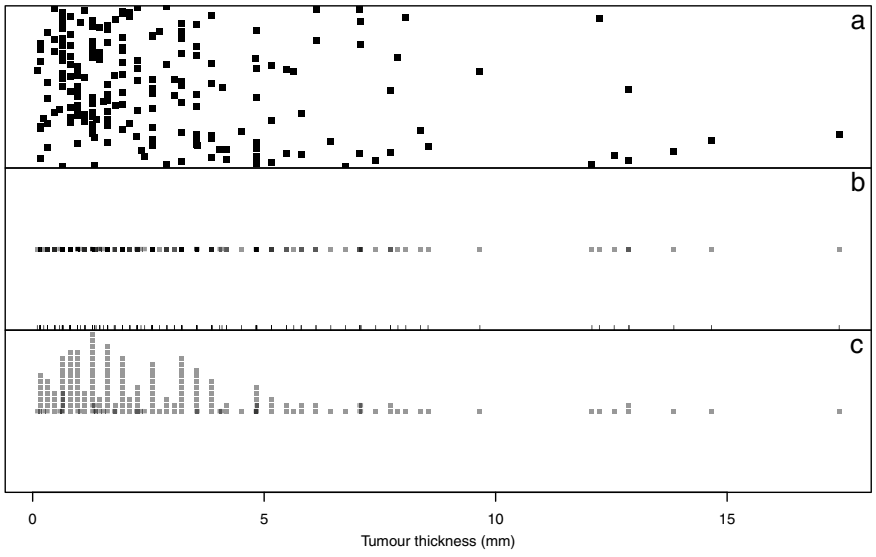


Fig. 8.7 Strip Chart for the tumour thickness data

so as to display the points hidden behind other ones: the darker, the more duplicated or adjacent values. Moreover, we add rugs (i.e. the small vertical lines) on the horizontal axis, each of which indicates a value, so that we could see the overlapping. Figure 8.7c sets the strip chart in the stack mode for displaying overlapped dots, as the duplicated values are shown as stacked dots, while the darker colour indicates only the adjacent dots.

8.7 Histograms, Box Plots, and Violin Charts

Histograms, box plots, and violin charts are used for visualizing the distribution of a numeric variable. Here we visualize the tumour thickness data in a histogram (Fig. 8.8a).

```
# base R:
hist(melanoma$thickness)
# ggplot2:
ggplot(melanoma) +
  geom_histogram(aes(x = thickness)) +
  theme_bw()
# plotly:
plot_ly(x = ~melanoma$thickness, type = "histogram")
```

The histogram is a special form of "bar chart", where the horizontal axis represents the ranges of the variable of interest while the vertical axis represents frequencies. The bars in the histogram indicate the counts of the values that fall in a certain range. There is usually no space between the bars. The ranges of values are called classes or bins. In principle, the histogram converts the numeric variable into a categorical one in bins, counts the frequency of the observations in each bin and display them in bars. The choice of the bins depends on our purpose. The more bins we choose, the

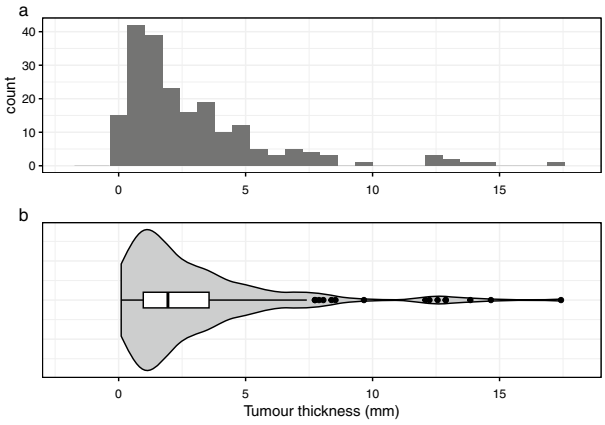


Fig. 8.8 A histogram and a violin plot

more details that can be seen, but the more chance that we might miss the pattern of the data. The Sturges' rule recommends the bin number (n_{class}) as:

$$n_{\text{class}} = 1 + \frac{\log_{10} N}{\log_{10} 2} = 1 + \log_2 N$$

where N is the number of the records. Other algorithms, such as the Scott's rule and the Freedman-Diaconis' rule, are applicable as well:

```
nclass.Sturges(melanoma$thickness)
```

```
## [1] 9
```

```
nclass.scott(melanoma$thickness)
```

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

```
nclass.FD(melanoma$thickness)
```

```
## [1] 20
```

Box plots (also called box-and-whisker plots) and violin charts are more informative tool for visualizing the distribution of one dimensional numerical data. The following code generates box plots and violin charts for the tumour thickness data:

```
# Box plot

## base R:
boxplot(melanoma$thickness, horizontal = TRUE)
points(mean(melanoma$thickness), 1, pch = 4)

## ggplot2:
ggplot(melanoma) +
  geom_boxplot(aes(thickness)) +
  geom_point(aes(x = mean(thickness), y = 0), pch = 4) +
  theme_bw()

## plotly:
plot_ly(x = ~melanoma$thickness, type = "box", boxmean = TRUE)

# Violin chart

## base R:
library(vioplot)
vioplot(melanoma$thickness, horizontal = TRUE)
points(mean(melanoma$thickness), 1, pch = 4, col = "white")

## ggplot2:
ggplot(melanoma, aes(x = thickness, y = 0)) +
  geom_violin(fill = 'grey') +
  geom_boxplot(width = 0.1) +
  geom_point(aes(x = mean(thickness), y = 0), pch = 4)

## plotly:
plot_ly(x = ~thickness, type = "violin", data = melanoma,
        box = list(visible = TRUE), meanline = list(visible = TRUE))
```

The box plot shows the numerical data through their quartiles, with lines extending from the boxes (whiskers) indicating variability outside the lower and upper quartiles (Q_1 and Q_3, see Sect. 6.3). The thick line or the dot in the middle indicates the median, the box indicates the P_{25}/Q_1 and P_{75}/Q_3, and the extended lines indicate 1.5 (by default) times IQR. We add a cross or a short dash line for the mean in order to indicate the skewness.

The violin plot is a combination of the box plot and a density plot that is mirrored and placed on each side to show the distribution shape of the data (Fig. 8.8b). The `vioplot::vioplot()` function (Adler and Kelly 2021) displays the box by default, while the **ggplot2** and **plotly** methods do not: we have to additionally use the `geom_boxplot()` function and the `box` argument, respectively.

Public health science often deals with many kind of qualitative data, and the real-world data are often mixtures of qualitative and quantitative data. Although histograms, box plots, and violin charts can be a single box/violin for detecting outliers and display heterogeneity of distribution for one numerical variable, we often compare the distributions of one numerical variable grouped by one categorical variable, which actually results in the visualization of the mixture of qualitative and quantitative data with multiple histograms/boxes/violins.

```
mean_th <- data.frame(sex = c(0, 1),
                      mean = tapply(melanoma$thickness, melanoma$sex, mean))

# Box plot

## base R:
boxplot(melanoma$thickness ~ melanoma$sex, horizontal = TRUE)
points(mean_th$mean, 1:2, pch = 4)

## ggplot2:
ggplot(melanoma, aes(x = sex, y = thickness, group = sex)) +
  geom_boxplot() +
  stat_summary(fun = "mean", geom = "point", shape = 4) +
  coord_flip() +
  theme_bw()

## plotly:
plot_ly(x = ~melanoma$thickness, y = ~melanoma$sex, orientation = 'h',
        type = "box", boxmean = TRUE)

# Violin chart

## base R:
library(vioplot)
vioplot(thickness ~ sex, horizontal = TRUE, data = melanoma)
points(mean_th$mean, 1:2, pch = 4, col = "white")

## ggplot2:
ggplot(melanoma, aes(x = sex, y = thickness, group = sex)) +
  geom_violin(fill = 'grey') +
  geom_boxplot(width = 0.1) +
  stat_summary(fun = "mean", geom = "point", shape = 4) +
  coord_flip() +
  theme_bw()

## plotly:
plot_ly(x = ~thickness, y = ~sex, type = "violin", data = melanoma,
        orientation = 'h',
        box = list(visible = TRUE), meanline = list(visible = TRUE))
```

8.8 Scatterplots

8.8.1 For Two Variables

Scatterplots, also called x-y plots, explore the relationship between quantitative variables. They are the most popular graphic for visualizing two numerical variables. We use the weight and height data in the *diet* dataset as an example.

```
data(diet, package = 'Epi')
dietlm <- lm(weight~height, data = diet)

# base R:
plot(diet$height, diet$weight,
     xlab = 'Height (cm)', ylab = 'Weight (kg)', las  = 1, pch = 16)
abline(dietlm)

# ggplot2
ggplot(diet, aes(height, weight)) +
  geom_point() +
  geom_smooth(method = 'lm') +
  labs(x = 'Height(cm)', y = 'Weight (km)') +
  theme_bw()

# plotly
diet1 <- na.omit(diet[, c('height', 'weight')])
diet1lm <- lm(weight~height, data = diet1)
plot_ly(diet1, x = ~height) |>
  add_markers(x = ~height, y = ~weight) |>
  add_lines(x = ~height, y = fitted(diet1lm))
```

The base R function `plot()` is a simple and fast way for generating a scatterplot. When we apply regression models to the dataset, fitted curves are often drawn through the data points in scatterplots (Chap. 7), and the `abline()` function adds a linear fitted line to it (Fig. 8.9a). The **ggplot2** package uses the `geom_point()` function for generating the scatterplot and `geom_smooth()` for the fitted line with the

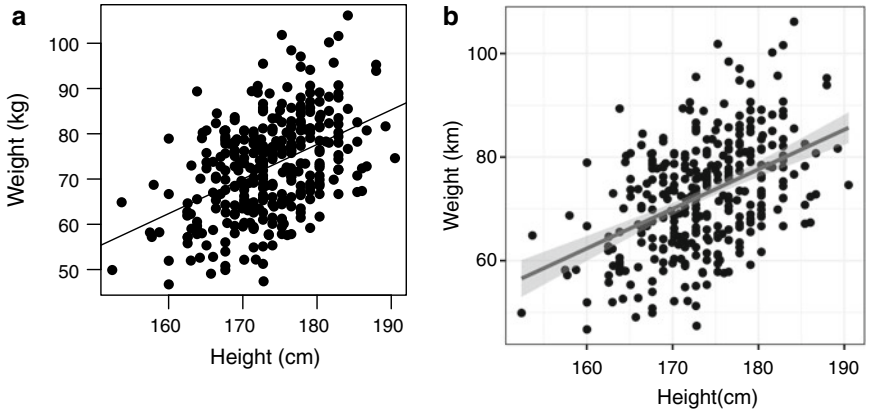


Fig. 8.9 Scatterplots for the body weight against the body height of the *diet* dataset

confidence interval (`se`, Fig. 8.9b). The **plotly** package generates the scatterplot simply with `plot_ly(diet, x = ~height, y = ~weight)`, but it seems a little complicated for the fitted line: we have to remove the NA values first, and use `add_markers()` and `add_lines()` for adding the traces of the scatter points and the fitted line. The advantage of **plotly** for scatterplots is that we could see the observed value and the fitted value for a certain x value as long as we put the cursor over it, which is very helpful for checking the details of the regression.

When the dataset is large and many data points are too close to each other, scatterplots may encounter a problem of overlapping points we cannot visually distinguish. R has multiple solutions for this problem. Here we use the *NHANES* dataset shipped by the **NHANES** package as an example. The influence of the age on the average systolic blood pressure can be visualized as a scatterplot with the age as the independent variable x and the average systolic blood pressure as the dependent variable y:

```
data(NHANES, package = 'NHANES')
```

```
plot(NHANES$Age, NHANES$BPSysAve, pch = 16, col = rgb(0, 0, 0, alpha = 0.2))
rug(NHANES$Age, side = 3, col = rgb(1, 0, 0, alpha = 0.2))
rug(NHANES$BPSysAve, side = 4, col = rgb(1, 0, 0, alpha = 0.2))
```

The scatterplot is shown in Fig. 8.10a. We use the `rgb()` function for setting the transparency of the colour as `alpha = 0.2` just like Fig. 8.7b and c, so that dark areas indicate the overlaps of data points. Moreover, the `rug()` function adds a short red line for each value of the two variables, indicating the overlaps along the two dimensions.

The **ggplot2** package can directly set the transparency of the data points with the argument `alpha`, and it provides a method for a better visualization of the overlapped data points with the `geom_bin2d()` function:

```
ggplot(NHANES, aes(Age, BPSysAve)) +
  geom_point(alpha = 0.2, shape = 15) +
  theme_bw()
ggplot(NHANES, aes(Age, BPSysAve)) +
  geom_bin2d() +
  theme_bw()
```

which generates a heatmap as Fig. 8.10b. It can be interpreted as two-dimensional histograms, i.e. the data are binned with both x and y into grids, and the colours of the grids indicate the frequency of the observations in each bin. The function `geom_bin2d()` can be replaced with `geom_hex()`, which bins the data into regular hexagons rather than squares.

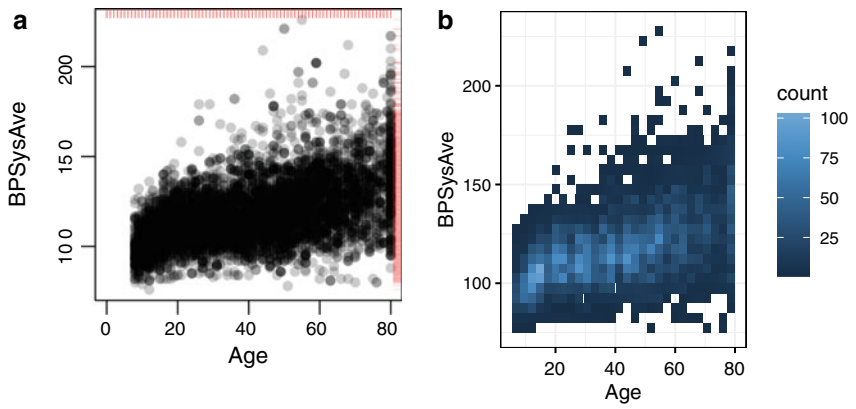


Fig. 8.10 A scatterplot and a two-dimensional histogram

Similarly, **plotly** generates such a two-dimensional histogram with `add_histogram2d()`:

```
plot_ly(x = NHANES$Age, y = NHANES$BPSysAve) |> add_histogram2d()
```

As the graph is interactive, we can see the the age, the average systolic blood pressure, and the data point frequency within a grid as long as we put the cursor over it.

Researches in public health often deal with survey data. Ordinary scatterplots can be misleading for survey data, as observations in survey samples may represent different numbers of units in the population. The **survey** package provides a wrapper function `svyplot()`, which produces scatterplots adjusted in various ways for sampling weights. Here we use the dataset *apistrat* from this package as a demonstration. This dataset contains the Academic Performance Index in 2000 (`api00`) and 1999 (`api99`), computed for all California schools. The column name `stype` is short for school type (Elementary/Middle/High School), `pw` for sampling weights, and `fpc` for finite population corrections to variance:

```
library(survey)
data(api)
dstrat <- svydesign(id = ~ 1, strata = ~ stype, weights = ~ pw, data = apistrat, fpc = ~ fpc)
svyplot(api00 ~ api99, design = dstrat, style = "bubble")
```

This dataset is first converted to an object of class `survey.design`, defined by the **survey** package, and then be visualized with the `svyplot()` function in the `bubble` style.

8.8.2 For Pairwise Variables

If we have more than two numerical variables, we could generate scatterplots for every pair of them (although it does not necessarily make sense). R provides a fast way to do this job. Here we use *dietn*, the numerical subset of *diet* we use in Chap. 6, for demonstration.

All of the base R, **ggplot2**, and **plotly** methods can generate an $n \times n$ scatterplot matrix for a numerical data frame, where n is the number of columns:

```
# base R:
pairs(dietn)
## or
plot(dietn)

# ggplot2:
library(GGally)
ggpairs(dietn)

# plotly:
ggplotly(ggpairs(dietn))
```

The base R function `pairs()` generates a matrix with the column names in the diagonal by default. Each column name indicates the meaning of the horizontal axis and the vertical axis of the sub figures that are in the same column or row, respectively. The `plot()` function does the same job when the input is a data frame. There are around 20 arguments for customizing `pairs()`. The following code defines three panel functions, which are assigned to the relevant arguments in `pairs()`. They put histograms on the diagonal panel, the absolute values of the correlation coefficients on the upper panel, and add linear fitted lines to the scatterplots on the lower panel (Fig. 8.11):

```
## add histograms on the diagonal
panel.hist <- function(x){
  usr <- par("usr"); on.exit(par(usr))
  par(usr = c(usr[1:2], 0, 1.5))
  h <- hist(x, plot = FALSE)
  breaks <- h$breaks
  nB <- length(breaks)
  y <- h$counts
  y <- y/max(y)
  rect(breaks[-nB], 0, breaks[-1], y, col = "grey")
}

## put the absolute values of the correlation coefficients on the upper panels,
panel.cor <- function(x, y, digits = 2, prefix = "", cex.cor){
  usr <- par("usr"); on.exit(par(usr))
  par(usr = c(0, 1, 0, 1))
  r <- abs(cor(x, y, use = 'complete.obs'))
  txt <- format(c(r, 0.123456789), digits = digits)[1]
  txt <- paste0(prefix, txt)
  if(missing(cex.cor)) cex.cor <- 0.8/strwidth(txt)
  text(0.5, 0.5, txt, cex = cex.cor * r)
}

## add linear regression lines
panel.reg <- function(x, y) {
  points(x, y, col = 'grey', pch = 16)
  abline(lm(y~x), col = 'red')
```

```
}

pairs(dietn,
      diag.panel = panel.hist,
      upper.panel = panel.cor,
      lower.panel = panel.reg)
```

The **ggplot2** package itself does not provide a function for scatterplot matrix, while the **GGallay** package, based on **ggplot2**, provides the `ggpairs()` function for this purpose. By default, the diagonal panel displays a distribution density curve for each numerical variable, a scatterplot for every pair of two numerical variables in the lower panel, and the correlation coefficients with the significant level indicated by the asterisks (*). We could add linear fitted lines to the scatterplots:

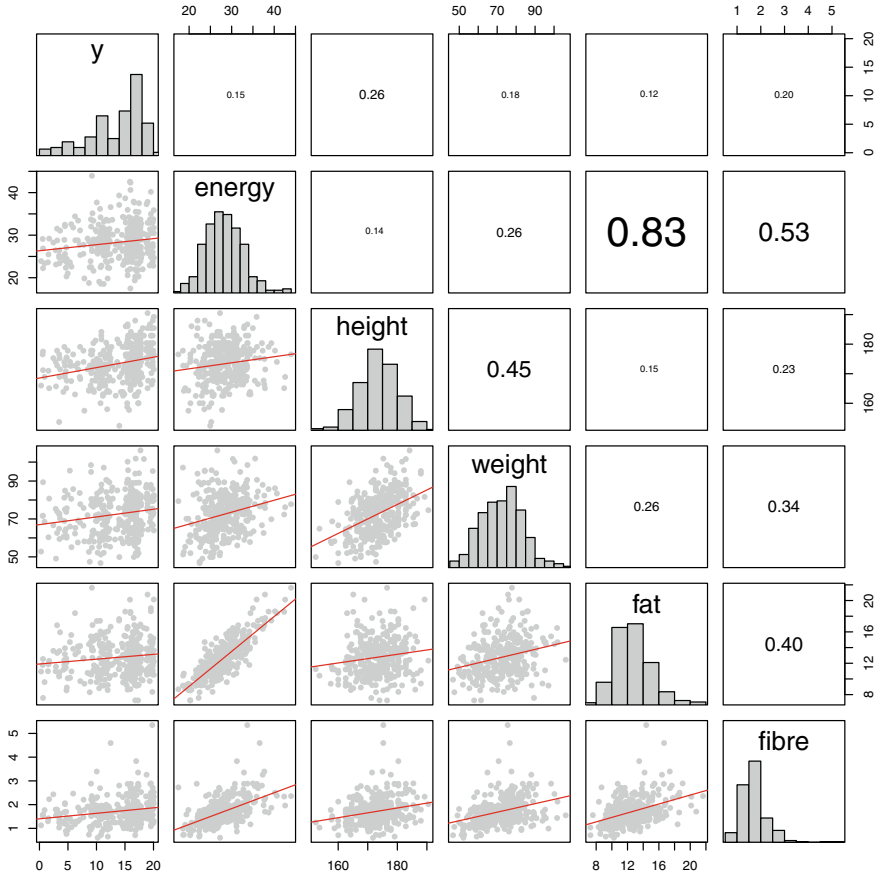


Fig. 8.11 A pair plot with customized panels

```
my_fn <- function(data, mapping){
   ggplot(data = data, mapping = mapping) +
    geom_point() +
    geom_smooth(method='lm')
}
ggpairs(dietn, lower = list(continuous = my_fn))
```

The **plotly** package, as far as we know, cannot directly generate scatter matrices. Fortunately, it provides a universal function `ggplotly()`, which can convert a **ggplot2** graph in to a **plotly** one.

```
ggplotly(ggpairs(dietn))
```

8.8.3 For Multiple Variables

It is possible and sometimes attractive to visualize more than two dimensions of the data in one figure. Three dimensional plots can be produced with the **rgl** package. Here we plot a 3D image for three numerical variables in the subset of the *NHANES* data with the `plot3d()` function.

```
library(rgl)
plot3d(diet$height, diet$weight, diet$fat)
```

Then a window with an interactive 3D image pops up, where we can drag, rotate, and zoom it in a flexible way.

We have to be very careful about plotting 3D graphs, because more dimensions could confuse the reader' mind. 3D graphs are not recommended in publications, as most of the publications are presented in two-dimensional media, and human brains are not good at reading them. Instead, it is a common way to map the third variable as other elements, such as gradient colours, sizes, or shapes of the plotting characters. In fact, Fig. 8.10b is a three-dimensional image, with the coordinates $\{x, y\}$ indicating the age and the average systolic blood pressure, respectively, and the colour indicating the observation counts. We can draw a scatterplot for the weight against the height of the *diet* dataset, and use colours (Fig. 8.12a) or the sizes (Fig. 8.12b) for the fat:

```
diet_sub <- diet[sample(1:nrow(diet), 30), ]
ggplot(diet_sub) +
  geom_point(aes(x = weight, y = height, colour = fat))
ggplot(diet_sub) +
  geom_point(aes(x = weight, y = height, size = fat), shape = 1)
```

Note that the continuous variable is automatically converted into categorical intervals when mapping it to the point size.

Conditional plot is another way for visualizing three dimensional quantitative data by grouping the two-dimensional scatterplots according to the third variable in multiple conditions. Here we draw a conditional plot for the weight against the height in the conditions of the fat (Fig. 8.13):

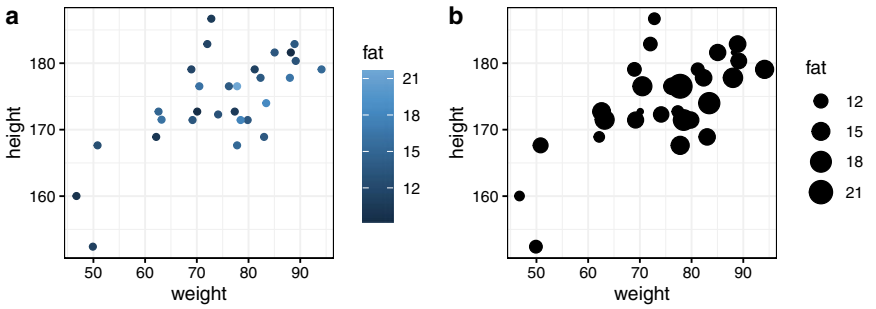


Fig. 8.12 Mapping the third variable as gradient colours and sizes in a scatterplot

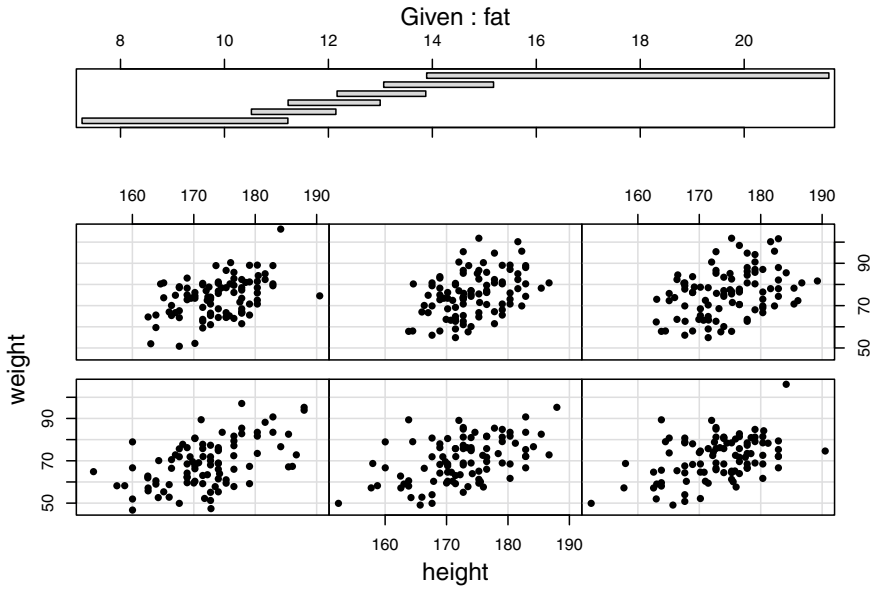


Fig. 8.13 A conditional plot for the *diet* dataset

```
coplot(weight ~ height | fat, data = diet, pch = 16)
```

```
##
##  Missing rows: 28, 71, 73, 112, 282
```

The `coplot()` function separate the graph into multiple sub-figures of scatterplots, based on the range of the fat. Note that there are overlaps of the conditioning variables, because the default value for the argument `overlap` is 0.5.

If we have four or more numerical variables to visualize in one graph, the symbol plot is a solution. The `symbols()` function allows drawing up to six symbols,

including circles, squares, rectangles, stars, thermometers, and box plots, at a specified set of x and y coordinates. Specific aspects of the symbols, such as relative size, can be customized by the numerical variables we would like to visualize apart from x and y. Here is an example adapted from the **MSG** package, which ships a dataset with the population, the growth rate, the urbanization level, the life expectancy, and the well-educated population of each of the provinces from the China population statistics in 2005. Firstly, we load the data, convert the Chinese characters in the data into English, and rescale some columns for better visualization.

```
library(MSG)
source(system.file("extdata", "ChinaPop.R", package = "MSG"), encoding = "UTF-8")
library(pinyin)
dimnames(ChinaPop)[[1]] <- toupper(
  py(dimnames(ChinaPop)[[1]], sep = '',
     dic = pydic(method = 'toneless', dic = 'pinyin2')
     )
  )
dimnames(ChinaPop)[[2]] <- c('Growth rate', 'Population', 'Urbanization level',
                             'Life expectancy', 'Well-educated population')
ChinaPop[, 5] <- ChinaPop[, 5]/1000
ChinaPop[, 1:2] <- apply(ChinaPop[, 1:2], 2, function(x)
  20 * (x - min(x)) / (max(x) - min(x)) + 5)
```

If you are a MS Windows OS user, you might have to convert your locale into Chinese for supporting the input of Chinese characters with the following code before loading the data:

```
Sys.setlocale('LC_CTYPE', 'Chinese')
```

We then use the `symbols()` function for plotting a scatterplot of the well-educated population against the average life expectancy (Fig. 8.14). Instead of dots, each province is represented as a thermometer-like symbol, with the width representing the growth rate, the height the population, and the proportion of the height the urbanization level. Thus, five numerical variables are visualized in one graph.

```
symbols(ChinaPop[, 4], ChinaPop[, 5],
        thermometers = ChinaPop[, 1:3], fg = "gray40",
        inches = 0.5, las = 1,
        xlab = "Average Life Expectancy (years)",
        ylab = "Well-educated population (thousand)"
)
library(KernSmooth)
est <- bkde2D(ChinaPop[, 4:5], apply(ChinaPop[, 4:5], 2, dpik))
contour(est$x1, est$x2, est$fhat, add = TRUE, lty = "12")
for (i in 1:nrow(ChinaPop)) {
  text(ChinaPop[i, 4], ChinaPop[i, 5], rownames(ChinaPop)[i],
       cex = 0.75, adj = attr(ChinaPop, "adj")[i, ], col = 'red',
  )
}
rug(ChinaPop[, 4], 0.02, side = 3, col = "blue")
rug(ChinaPop[, 5], 0.02, side = 4, col = "blue")
```

```
boxplot(ChinaPop[, 4],
        horizontal = TRUE, pars = list(
          boxwex = 7,
          staplewex = 0.8, outwex = 0.8
        ), at = -6, add = TRUE, notch = TRUE, col = "skyblue",
        xaxt = "n"
)
boxplot(ChinaPop[, 5],
        at = 63, pars = list(
          boxwex = 1.4,
          staplewex = 0.8, outwex = 0.8
        ), add = TRUE, notch = TRUE, col = "skyblue",
        yaxt = "n"
)
```

Furthermore, we add more elements into the graph, including a contour for the binned Kernel density estimate (`contour()`), the labels for each symbol (`text()`), the rugs (`rug()`) and box plots (`boxplot()`) of the well-educated population and the average life expectancy.

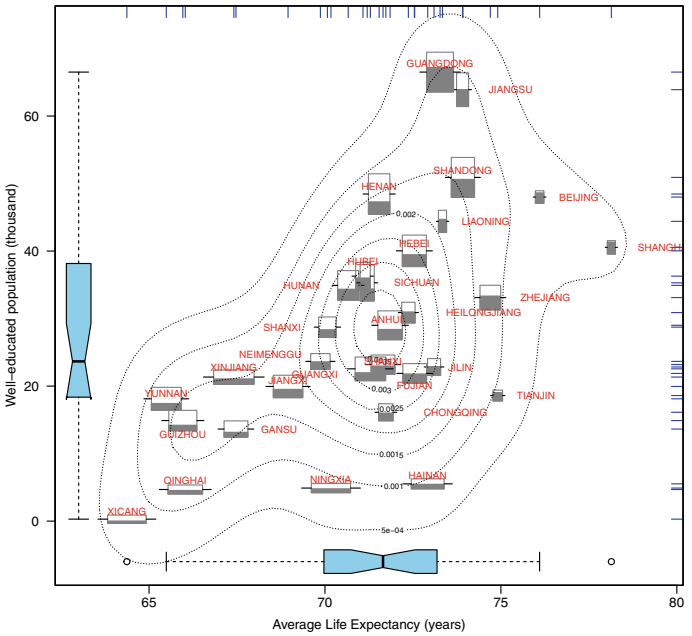


Fig. 8.14 Visualization of multiple variables in one graph. Adapted from Xie and Zhao (2021)

8.8.4 *Line Charts and Area Charts*

As a variant of the scatterplots, line charts are often used to show the trend of a variable over time. Line charts are often useful when the dataset is a time series. Here we use the COVID-19 pandemic data from the **nCov2019** package (Yu and Wu 2021):

```
library(nCov2019)
data_covid <- query()
dtf_covid <- data_covid$historical$table
dtf_usa <- dtf_covid[dtf_covid$country == 'USA', ]
```

```
# base R:
plot(dtf_usa$date, dtf_usa$deaths, type = 'l')

# ggplot2:
ggplot(dtf_usa) +
  geom_line(aes(x = date, y = deaths)) +
  theme_bw()

# plotly:
plot_ly(dtf_usa, x = ~date, y = ~deaths, type = 'scatter', mode = 'lines')
```

The line chart shows the trend in the number of deaths over time for USA.

Another variant of scatterplot, called area plot, can also visualize the time series in a more impressive way:

```
# base R:
plot(dtf_usa$date, dtf_usa$deaths, type = 'n')
polygon(c(dtf_usa$date, dtf_usa$date[nrow(dtf_usa)]),
        c(dtf_usa$deaths, 0), border = NA, col = 'grey')

# ggplot2:
ggplot(dtf_usa) +
  geom_area(aes(x = date, y = deaths), fill = 'grey')

# plotly:
plot_ly(dtf_usa, x = ~date, y = ~deaths,
        mode = 'lines', fill = 'tozeroy')
```

Especially, when we have one more variable which is categorical, we can either stack them in multiple shaded colours in an area plot or plot individual lines in multiple colours in a line chart. The following code gives an example, which plots a line chart (Fig. 8.15a) and an area chart (Fig. 8.15b) for the death cases of COVID-19 in three countries in North America.

```
dtf_na <- dtf_covid[dtf_covid$country %in% c('USA', 'Canada', 'Mexico'), ]
ggplot(dtf_na) +
  geom_line(aes(x = date, y = deaths, colour = country))
ggplot(dtf_na) +
  geom_area(aes(x = date, y = deaths, fill = country))
```

If we would like to display the relative proportions rather than the absolute number of the cases, we can set the argument `position = "fill"` in the `geom_area()` function.

Interactive visualization for such time series data could be performed with the **dygraphs** package:

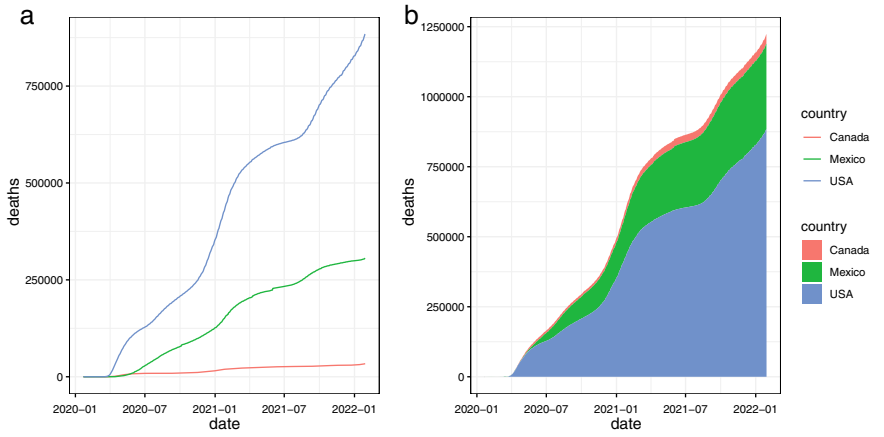


Fig. 8.15 A line chart and an area plot for the COVID-19 confirmed cases in North America

```
dtf_wide <- dtf_na[, c('date', 'country', 'cases')] |>
  tidyr::pivot_wider(names_from = country, values_from = cases)
xts::xts(dtf_wide[, -1], order.by = dtf_wide$date) |>
  dygraphs::dygraph()
```

In this example, we have to convert the data frame into a wide version, and then into an extensible time-series (xts) object as the input of the `dygraph()` function. In the interactive graph, we could zoom in and out, and the number of the cases for each country appears at the top of the graph.

8.9 Export Graphs

The static graphs produced by the base R functions and **ggplot2** are displayed in the *Plots* tab in the bottom-right pane (Pane 2) of Rstudio IDE. It could be exported via clicking the *Export* button in the tool bar of the pane and then "Save as Image" for *png*, *jpeg*, *tiff*, *metafile*, *svg*, and *eps* formats, or "Save as PDF" for pdf format. The interactive graphs generated by **plotly** and **dygraph** are displayed in the *Viewer* tab. Click *Export* and it could be saved as a static image files or a web page file.

Instead of clicking the buttons, we often export the graphics with code. The graphics plotted by the base R functions are exported with a graphics device function, including `pdf()`, `png()`, `jpeg()`, `tiff()`, and so on, paired with the `dev.off()` function:

```
pdf('diet-fat-hist.pdf')
hist(diet$fat)
dev.off()
```

The **ggplot2** graphics can be exported by the `ggsave()` function:

```
p1 <- ggplot(diet) + geom_histogram(aes(fat))
ggsave('diet-fat-hist-gg.pdf', p1)
```

The interactive graphics produced by **plotly** and **dygraph** can be exported by the `htmlwidgets::saveWidget()` function:

```
g1 <- ggplotly(p1)
htmlwidgets::saveWidget(g1, 'diet-fat-hist-plotly.html')
```

8.10 Exercises

1. Download the COVID-19 pandemic data from the **nCov2019** package.
   - Make a scatterplot with the global confirmed cases as x, and the deaths as y. Change the points as closed circles in blue. Add a linear regression line in gray.
   - Make a pie chart showing the confirmed cases for the Top Five countries.
   - Make a bar chart for the confirmed cases and deaths side-by-side for each Top Five country. Add a red horizontal line for the mean of them.
   - Make a boxplot for the confirmed cases, suspected cases and deaths for the Top Five countries. Add the means as blue points to the boxplot. Change the colors of the boxes.
   - Combine these figures into one figure. Save it as a .pdf file.
2. Use the **ggplot2** package or the **plotly** package to present the Hodgkin data in the **pubh** package.
   - Load the *Hodgkin* data set. What does each column mean?
   - Plot a scatterplot of CD4 against CD8 with ggplot2. Add a smooth line.
   - Use different colors to distinguish the Hodgkin Group and non-Hodgkin Group. Add smooth lines to each group.
   - Use multiple facets to distinguish the Hodgkin Group and non-Hodgkin Group. Add smooth lines to each group.
   - Plot the previous graphs with the **plotly** package.
   - Make a bar plot for the count of Hodgkin and non-Hodgkin cases.
   - Make a box plot for CD4 and CD8 values of the whole data set.
   - Make a box plot for CD4 and CD8 values of Hodgkin and non-Hodgkin cases.

References

Adler, Daniel, and Duncan Murdoch. 2022. *rgl: 3D Visualization Using OpenGL*. https://CRAN.R-project.org/package=rgl.

Adler, Daniel, and S. Thomas Kelly. 2021. *vioplot: Violin Plot*. https://github.com/TomKellyGenetics/vioplot.

Kassambara, Alboukadel. 2022. *ggpubr: ggplot2 Based Publication Ready Plots*. https://rpkgs.datanovia.com/ggpubr/.

Sarkar, Deepayan. 2021. *lattice: Trellis Graphics for R*. http://lattice.r-forge.r-project.org/.

Schloerke, Barret, Di Cook, Joseph Larmarange, Francois Briatte, Moritz Marbach, Edwin Thoen, Amos Elberg, and Jason Crowley. 2021. *GGally: Extension to ggplot2*. https://CRAN.R-project.org/package=GGally.

Vanderkam, Dan, JJ Allaire, Jonathan Owen, Daniel Gromer, and Benoit Thieurmel. 2018. *dygraphs: Interface to Dygraphs Interactive Time Series Charting Library*. https://github.com/rstudio/dygraphs.

Xie, Yihui, and Peng Zhao. 2021. *MSG: Data and Functions for the Book Modern Statistical Graphics*. https://github.com/yihui/MSG.

Yu, Guangchuang, and Tianzhi Wu. 2021. *nCov2019: Exploring COVID-19 Statistics*. https://github.com/YuLab-SMU/nCov2019.

Chapter 9
Presenting Data

9.1 Chapter Highlights

- Understand the channels and basic principles of presenting data,
- Set up the R Markdown environment, and
- Present data via multiple forms with the integration of R Markdown in various academic occasions.

9.2 Channels

One of the most important objectives of science research is that the research questions, methods, and results can be understood and reproduced by other researchers or people who are interested in the research area. Thus, presenting research outcomes with related data evidence is the foundation of the development of the study's influence and exchange of different peers' opinions. There are three major channels for presenting data to the community or the world: presenting them in journal papers, talks, and posters (Table 9.1). Other channels, such as dissertations, academic mails, newspapers, blogs, books, etc., might not take place as often for everyone in public health science, although they are important as well.

Each channel of presenting data has its features. Journal papers aims at peers and readers who work in related disciplines. They are more specialized than posters and slides, which often aim at a wider audience. In terms of degree of formality, journal papers are the most formal, while posters are the least formal ones. Journal papers are often presented with long text to analyze and discuss the data, so the accuracy of the data is particularly important. Slides are often presented in talks, which require a clear and aesthetically pleasant presentation of the data. Posters are often displayed in a fixed location and are flexible for the audience to choose, thus a good structure in organizing the results and an elegant visual design would help presenting data.

P. Zhao, *Working with Data in Public Health: A Practical Pathway with R*,
https://doi.org/10.1007/978-981-99-0135-7_9

Table 9.1 Three main channels of presenting data

Channels	Description	Advantages	Limitation
Talks	Presenting data orally, usually with slides	Vivid with animations and universal in various occasions	Time limit. Electronic devices based. Dependence on the presenter's improvisation
Journal papers	Presenting data with a completed study and published it in a academic journal	Professional, detailed, strict, and structured	Requirement changes over publishers. Less interaction with readers
Posters	Presenting data with a designed poster and discuss it with other people	Convenient, instant with interaction	Space limit. Dependence on the presenter's explanation

9.3 General Principles

First of all, the objective of presenting data is to deliver the core results of a study. Only express the relevant data that the researcher most wishes to convey. Do not complicate it. Too much information often prevents the readers from getting a precise idea of the research focus.

Express general ideas first, and then go further to show specific points. For example, start with a description of the study participants to give the reader an idea of how representative and applicable the study is. Then proceed to the presentation and analysis of key findings. It is essential that the data presentation corresponds to the research question and answers the research question. The past tense is usually used when describing the data. Choose the most appropriate format for presenting the data, and avoid duplication of information and multiple repetitions of formats where possible.

For presenting data, three key points of principles are emphasized:

1. Precise: Presenting data should only include the data that are planned to be shown for readers or audiences so as to deliver the information without addressing the data or information that is not relevant for the presentation.
2. Detailed: Presenting data should address the related context and methods applied into the data with clear instructions of the presentation forms which help readers or audience understand the information delivered.
3. Proper-formed: The methods should be correctly applied for presenting data so as to show different characters of data to readers or audiences. For instance, statements and tables are more suitable for presenting qualitative data's details, while graphs are more suitable for quantitative data.

It makes a lot of sense to present our data in a manner that the peers could understand well. If the data are presented to the general public, the presentation may

have to be adapted to the audience, but the principles must be preserved. Generally, presenting data should be based on and serve the view of readers and audiences to provide and generate the understanding which is consistent with the presenters.

9.4 R Markdown

9.4.1 Set Up R Markdown

As a presenting tool, R Markdown is a framework for writing reproducible and dynamic documents for data science. Reproducible research refers to scientific products, such as reports, papers, theses, etc., along with the laboratory notebooks and full computational environment, which new work can be produced based on. As a document format, R Markdown was first introduced in 2012 in the **knitr** package (Xie 2022a), which allows embedding code chunks in Markdown documents. The R Markdown community has been developing rapidly in the past decade. With R Markdown, we could nowadays present data in all kinds of common documents, including journal papers, dissertations, slides, posters, websites, etc., in a flexible way.

R Markdown is built on the shoulder of Markdown, which is a markup language that defines a set of writing rules. Markdown supports a variety of components. For example, a single hash (`#`) or multiple hashes lead the chapter or section title in the level determined by the number of hashes, and a pair of dollar (`$$`) inserts an inline maths expression. Markdown is popular on many online forums, such as GitHub, Stack Overflow, and the Capital of Statistics (a Chinese forum for statistics), for asking and answer questions, discussions, and writing documents.

R Markdown extends the usage of Markdown by integrating R into Markdown. The R Markdown syntax is shown in Table 9.2. Technically, R Markdown is composed of a series of packages in R and external tools. For setting up the R Markdown environment, the R packages **knitr** (Xie 2022a), **rmarkdown** (Allaire et al. 2022), and **tinytex** (Xie 2022b), are required as fundamental packages. The **knitr** package executes the codes in the R Markdown document (`.Rmd`) and converts it into a Markdown document (`.md`), which is then converted to the desired final output document by Pandoc, a program which could either be installed independently or shipped with RStudio IDE. The entire process is wrapped in the R **rmarkdown** package. If the output document is `.pdf`, Pandoc requires a LaTeXdistribution, which could be installed and managed by the R **tinytex** package. As beginners, we do not have to understand every single step in the process. We only have to install some R packages, and write the data presentation document in a .Rmd document, and click one button or a hot key, and Tah-dah! A nice product is there.

The installation of the **knitr**, **rmarkdown**, and **tinytex** packages can be found in Chapter 1. Note that **tinytex** is rather an R package for managing the LaTeXinstallation

Table 9.2 Basic markdown syntax

You type	You get
`*italic*`	*Italic*
`**bold**`	**Bold**
`$E = mc^2$`	Equations: $E = mc^2$
`[link](https://...)`	Link
`![](image)`	Insert an image
`# Chapter 1`	Headings
`1. list...`	Numbered list
`- list...`	Unnumbered list
`(ref:fig1) The caption fig.cap = "(ref:fig1)"`	formatted figure caption
`\@ref(fig:fig1)`	Cross reference for figures/tables
`\#eq:mc2) \@ref(eq:mc2)`	Cross reference for equations
`[@R-rmarkdown]`	Citation

than a LaTeXdistribution. The **tinytex** package can install TinyTeX, which is a lightweight version of LaTeX:

```
tinytex::install_tinytex()
```

It takes seconds to minutes to complete the installation, which depends on the network speed and the computer performance.

After the R Markdown environment is set up, we could create a new simple R Markdown document in RStudio from the menu `File -> New File -> R Markdown`. In the dialogue window which pops out, choose `Document` on the left, and fill in the blanks on the right with the title and the author (which we could change later) of the document. Click the desired output format (which we could also change later), and we get the .Rmd document. Click the `Knit` button on the top bar of the top-left panel, or press the hot key `ctrl+shift+k`. If the .Rmd document is compiled and the output document is displayed, it means the R Markdown environment has been successfully set up. Otherwise, search the internet for the solutions to any installation problem.

9.4.2 Structure of an R Markdown Document

A common .Rmd document is composed of two parts, i.e. a header and a body. For presenting data, the body is the main work space, where we write the statements as well as display tables and graphs. Other information is written in the header.

9.4.2.1 Header

The header contains one or multiple fields, including the title, the author(s), and other metadata, which are written in the syntax of YAML surrounded by a pair of three dashes `---`. A minimal YAML header with one single field `output` looks like:

```
---
output: pdf_document
---
```

Then the output document is a `.pdf` file. What YAML fields are supported depends on the R Markdown template. The default template supports the title, the author, and the date, as typical fields:

```
---
title: "The title of the document"
author: "Wukong Sun"
date: 2022-02-02
output: pdf_document
---
```

Then they will appear in the title of the output `.pdf` document.

Many R packages provide R Markdown templates which support more YAML fields. The usage of the YAML syntax is beyond the scope of this book. In most cases, we do not have to care much about it but only have to replace the metadata with our own. For instance, the **rticles** package (Allaire et al. 2022) provides R Markdown templates for 40 journals or formats:

```
rticles::journals()
```

```
##  [1] "acm"         "acs"             "aea"             "agu"
##  [5] "ajs"         "amq"             "ams"             "arxiv"
##  [9] "asa"         "bioinformatics" "biometrics"     "copernicus"
## [13] "ctex"        "elsevier"        "frontiers"       "glossa"
## [17] "ieee"        "ims"             "informs"         "iop"
## [21] "isba"        "jasa"            "jedm"            "joss"
## [25] "jss"         "lipics"          "lncs"            "mdpi"
## [29] "mnras"       "oup_v0"          "oup_v1"          "peerj"
## [33] "pihph"       "plos"            "pnas"            "rjournal"
## [37] "rsos"        "rss"             "sage"            "sim"
## [41] "springer"    "tf"              "trb"             "wellcomeor"
```

Suppose we are going to submit a manuscript to the Elsevier journal *Public Health*, we can, after the installation of the `rticles` package, create an Rmd document in RStudio from the menu `File -> New File -> R Markdown` and choose `From Template` on the left of the dialogue window and `Elsevier Journal Article` on the right. In the new R Markdown document, we can see the header like:

```
title: Short Paper
author:
  - name: Alice Anonymous
    email: alice@example.com
    affiliation: Some Institute of Technology
    footnote: 1
  - name: Bob Security
    email: bob@example.com
    affiliation: Another University
  - name: Cat Memes
    email: cat@example.com
    affiliation: Another University
    footnote: 2
  - name: Derek Zoolander
    email: derek@example.com
    affiliation: Some Institute of Technology
    footnote: 2
address:
  - code: Some Institute of Technology
    address: Department, Street, City, State, Zip
  - code: Another University
    address: Department, Street, City, State, Zip
footnote:
  - code: 1
    text: "Corresponding Author"
  - code: 2
    text: "Equal contribution"
abstract: |
  This is the abstract.

  It consists of two paragraphs.

journal: "Public Health"
date: "2022-12-13"
bibliography: mybibfile.bib
#linenumbers: true
#numbersections: true
csl: elsevier-harvard.csl
output: rticles::elsevier_article
---
```

Besides the typical fields of `title`, `name`, `date`, and `output`, this template supports more, including `address` and `footnote` with sub-fields, `abstract`, `journal`, `bibliography`, `linenumber`, `numbersections`, `cls`, and `name`, `email`, `affiliation`, as well `footnote` as sub-fields in `author`, which are tailored for an Elsevier journal article. The field `journal: "An awesome journal"` is replaced with `journal: "Public Health"`. The value of the `output` field is assigned as `rticles::elsevier_article`, according to which the **knitr** package will compile this Rmd document on the basis of the Elsevier article template in the **rticles** package. When this document is *knitted* into a .pdf document, a LaTeX file (`.tex`) with supportive documents is created simultaneously, which is ready for the submission to the journal.

Many R packages provide user-friendly R Markdown templates which create a variety of documents for presenting data. Here is a list of some interesting packages, including the **rticles** package introduced previously:

- For data reports and manuscripts:

– **distill** (Dervieux et al. 2022): Scientific and technical writing, native to the web.
– **pinp** (Eddelbuettel and Balamuta 2020): Create articles in PNAS (Proceedings of the National Academy of Sciences of the United States of America)-alike style.
– **prettydoc** (Qiu 2021): Creating tiny yet beautiful documents and vignettes with various themes and syntax highlight styles in `.html`.
– **rmdformats** (Barnier 2022): Templates for creating .html documents with some extra features such as automatic table of contents, lightboxed figures, dynamic crosstab helper.
– **rmdTemplates** (Rodriguez-Sanchez 2020): Templates for scientific manuscripts and reviews with support for citations and different bibliography styles. It also clues functions to embed data and Rmarkdown source files within HTML files.
– **rticles** (Allaire et al. 2022): Templates for authoring journal articles and conference submissions.
– **tint** (Eddelbuettel and Gilligan 2022) and **tufte** (Xie and Allaire 2022): Create .pdf and .html documents in 'Tufte' style.

- For slides:

– **binb** (Eddelbuettel et al. 2020): Create slides in `.pdf` based on LaTeXBeamer.
– **rmdshower** (Makeev et al. 2018): Create slides in `.html` for 'shower' presentations.
– **xaringan** (Xie 2022c): Create `.html` slides based on the JavaScript library 'remark.js' .

- For posters:

– **drposter** (Bucior 2020): Generate posters in `.html`, inspired by 'reveal.js' presentations.
– **pagedown** (Xie et al. 2022): Generate posters in `.html`.

- For books and dissertations:

– **bookdownplus** (Zhao 2020): Templates for books or dissertations in `.pdf` based on the **bookdown** package.
– **pagedown**: Creates paginated books or dissertations in `.html`.
– **thesisdown** (Ismay and Solomon 2022): Thesis template based on the **bookdown** package.
– **tint** and **tufte**: Create books in .pdf in 'Tufte' style.

Note that some packages appear multiple times in the list, because they provide templates for different channels of presenting data, and some of their rarely-used features are not listed. For example, the **pagedown** package provides R Markdown

templates not only for posters, books, and dissertations, but also for data reports, letters, resumes, and even business cards. Besides, other packages might also be of interest. For example, the **linl** package (Eddelbuettel and Wolen 2019) creates letters in `.pdf` based on LaTeX'letter' class. The **blogdown** package (Xie et al. 2022) can easily build websites and create blogs in R Markdown. The **uiucthemes** package (Balamuta 2020) provides a set of customized templates for documents and presentations with the colour scheme and identity standards of University of Illinois at Urbana-Champaign (UIUC). The website *R Markdown Gallery* exhibits how these output documents look like in portfolios with helpful information for each package[1].

The template for the .Rmd document is automatically assigned in the `output` field in the YAML header when it is created. Multiple templates are allowed, so that the same Rmd document can be compiled into different outputs. An example is as follows:

```
output:
  bookdown::pdf_book: default
  bookdown::word_document2: default
  bookdown::html_document2:
    toc: TRUE
    number_sections: FALSE
    code_folding: "hide"
```

It shows that this Rmd document can be compiled into three outputs: a `.pdf` book with the default settings (`pdf_book: default`), a Microsoft Word `.docx` document with the default settings (`word_document2: default`), and a `.html` document with the customized settings (`html_document2`), all of which are dependent on the the **bookdown** package (`bookdown`). The customized settings of the `.html` document include the table of contents (`toc: TRUE`), section numbers (`number_sections: FALSE`), and folding the code (`code_folding: "hide"`). Again, we do not have to master everything about the YAML header, but only have to fill in the fields with our own information.

9.4.2.2 Body

The basic Markdown syntax could be used in the body. Here we focus on presenting data with inline R code in statements, tables, and graphs, which are described in subsequent sections. The usage of other R Markdown syntax is out of the scope of this book, but we would like to mention that a convenient visual editor has been available in RStudio IDE since version 1.4 for writing Rmd documents[2]. Here we list some supportive packages for writing an .Rmd document:

[1] https://rmarkdown.rstudio.com/gallery.html.

[2] https://rstudio.github.io/visual-markdown-editing/.

- **caseconverter** (Yazici 2022, GitHub strboul/caseconverter): Convert text cases to lower, upper, snake, camel cases
- **citr** (Aust 2019, GitHub crsh/citr): Search a BibTeX-file and insert formatted Markdown citations
- **remedy** (Fay et al. 2018): RStudio addins to simplify Markdown writing
- **splitChunk** (Olsen 2017, GitHub LudvigOlsen/splitChunk): Split code chunk in R Markdown
- **strcode** (Walthert 2022, GitHub lorenzwalthert/strcode): Insert code block separators and section titles
- **uniscape** (Korpela 2022, GitHub mvkorpel/uniscape): Convert non-ASCII strings to a portable format
- **wordcountaddin** (Marwick 2022): Count non-code words in Rmd documents
- **wrapRmd** (Mahr 2022, GitHub tjmahr/WrapRmd): Wrap selected R Markdown text but don't insert lines breaks into inline R code

9.5 Presenting Data via Statements

Statement is the most common way of presenting data. Comparing with table and graph, statement is the most suitable to describe simple observations and the results of statistical analyses. When the data we are presenting don't need special emphasis, or when showing data in a table would make it a very small table, statement is a better choice.

In an .Rmd document, inline R code is a handy tool for presenting data via statements. Inline R code is written as a piece of R code which is led by "`r`", surrounded by a pair of back sticks ("`` ` ``"), and embedded in a literal sentence. The inline R code will be replaced by the returned value when the .Rmd document is knitted into the desired output document.

In the subsequent sections, we give some recommendations with examples about how to present summaries, hypothesis tests, and regression models with inline R code in statements.

9.5.1 Summaries

When we present a summary of numerical data, we often describe the measure of the center, spread, and the sample size or the degree of the freedom. Section 6.3 summarizes the *diet* data. We could write the summary in the .Rmd document like this:

```
  The mean height was `r mean(diet$height, na.rm = TRUE)`
cm (sd = `r sd(diet$height, na.rm = TRUE)`,
n = `r sum(!is.na(diet$height))`).
```

which will be compiled into:

> The mean height was 173.3579072 cm (sd = 6.4106339, n = 332).

The inline R code `` `r mean(diet$height, na.rm = TRUE)` `` is replaced by its returned value 173.3579072. So are the other two pieces of R code. The advantage of using the inline R code is that the integration of the data in the statement is reproducible: we can see exactly where these numbers come from. Any mistakes could be traced back. If we type them manually or copy and paste them from somewhere else, mistakes might occur and it is hard to track them.

The inline R code is, however, not friendly to read in this example. There are at least two annoying problems: (1) the inline code is too long, and (2) the returned values have too many digits. Fortunately, we have the flexibility to improve it. For example, we can save these values as objects before embed them in the statements. It is a good practice to organize them in a list:

```
z <- list(h_mean = round(mean(diet$height, na.rm = TRUE)),
          h_sd = round(sd(diet$height, na.rm = TRUE)),
          h_n = sum(!is.na(diet$height)))
```

Then the statement can be written literally as:

```
The mean height was `r z$h_mean` cm (sd = `r z$h_sd`,
n = `r z$h_n`).
```

which will be knitted into the same text as before.

9.5.2 Hypothesis Tests

Hypothesis test results are often reported in journal papers with the key information, usually including:

- The test statistic (e.g. the t value in a t-test, the F valued in ANOVA),
- The sample size (n) or the degree of freedom ($d.f.$),
- The significance (i.e. the p-value), and
- The critical values of the test statistic (sometimes).

Here we give some examples of reporting hypothesis tests in statements with the insertion of R code in a .Rmd document.

Suppose we have a sample of the body height with 10 random subjects, and we would like to use the t-test to find whether the sample comes from a population with the mean body height of 170 cm. The sample data set is has follows:

```
x <- c(176.8, 181.4, 168.7, 187.8, 167.6, 180.3, 172.5,
165.4, 175.9, 165.9)
mu <- 170
xt <- t.test(x, mu = mu)
```

Most, if not all, of the hypothesis test functions in R return a list, such as `t.test()` for the Student's t-test, `var.test()` for the F-test, `wilcox.test()` for the Wilcoxon test, `chisq.test()` for the χ^2 test, and `aov()` for ANOVA. The returned list contains the key information of the test. We could extract and re-organize it in a new list for future use. For example:

```
z1 <- list(n = length(x),                    # sample size
           m = round(xt$estimate, 1),        # sample mean
           s = round(sd(x), 1 ),             # sample standard deviation
           ci = round(xt$conf.int, 1),       # confidence interval
           t = round(xt$statistic, 3),       # t score
           df = xt$parameter,                # degree of freedom
           p = round(xt$p.value, 3))               # p-value
```

Thus, the statement for the t-test can be written literally as:

```
A sample of `r z1$n` random subjects shows a mean of `r
z1$m`     cm ($sd = `r z1$s`$ cm, 95% CI `r z1$ci[1]` to
`r   z1$ci   [2]` cm). A comparison was made between the
sample mean and the previous study which possesses a
population  mean of `r mu` cm. A single sample $t$ test
was performed ($t = `r z1$t`$,$df = `r z1$df`$, $p$-value
= `r z1$p`)    leading to the conclusion that the sample
comes from  the same population.
```

in a .Rmd document, which is compiled into:

> A sample of 10 random subjects shows a mean of 174.2 cm (sd = 7.5 cm, 95% CI 168.9 to 179.6 cm). A comparison was made between the sample mean and the previous study which possesses a population mean of 170 cm. A single sample t test was performed (t = 1.783, df = 9, p-value = 0.108) leading to the conclusion that the sample comes from the same population.

in the output document.

Similarly, the result of an unpaired sample t-test could be reported like:

```
We randomly allocated `r z2$n` subjects to either
receive treatment with a particular bronchodilater
(mean = `r z2$m1`, $sd = `r z2$s1`$) or receive a
placebo (mean = `r z2$m2`, $sd = `r z2$s2`$). An F
test ($F = `r z2$F`$, $p$-value = `r z2$Fp`) indicated
that both samples did not have significantly different
variances. A two sample independent t test produced a
statistically significant result at α = 0.05 ($t =
`r z2$p`$  $df = `r z2$df`$ , $p$-value = `r z2$p`)
leading to the conclusion that the two samples did not
come from the sample population.
```

as long as we prepare the t-test result in a list `z2` in advance.

9.5.3 Regressions

R generates extensive outputs for SLR models (see Section 7.4), of which some results are actually redundant and can be ignored, because the `lm()` function deals with MLR models as well, and the SLR models are treated as a special case. For instance, the SLR model gives the following output:

```
summary(lm_ba)
```

```
##
## Call:
## lm(formula = BPSysAve ~ Age, data = dtf_sub)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -21.667  -9.815  -0.652  10.168  41.612
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 102.3386     5.1322   19.94  < 2e-16 ***
## Age           0.4385     0.1105    3.97 0.000221 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 14.25 on 52 degrees of freedom
##   (6 observations deleted due to missingness)
## Multiple R-squared:  0.2326, Adjusted R-squared:  0.2179
## F-statistic: 15.76 on 1 and 52 DF,  p-value: 0.0002214
```

Not all the output information has to be presented for the SLR model. The t-test for the slope has the same meaning here as the F-test, which is indicated in the identical p-values (2.21×10^{-4}), and the equality of the F value (15.764) and the square of t-value (3.97). The adjusted R^2 is only used for the MLR models. It is usually sufficient to only present the t-test for the significance of the regression, the estimates of the slope and the intercept, and R^2 for how well the model fits the data.

After deciding what information to be presented, now we must find how to extract them and embed them into the statement. The results of the regression model can be assigned to an object as a list:

```
lm_bas <- summary(lm_ba)
```

```
str(lm_bas)
```

```
## List of 12
##  $ call         : language lm(formula = BPSysAve ~ Age, data = dtf_sub)
##  $ terms        :Classes 'terms', 'formula'  language BPSysAve ~ Age
##   .. ..- attr(*, "variables")= language list(BPSysAve, Age)
##   .. ..- attr(*, "factors")= int [1:2, 1] 0 1
##   .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. ..$ : chr [1:2] "BPSysAve" "Age"
##   .. .. .. ..$ : chr "Age"
##   .. ..- attr(*, "term.labels")= chr "Age"
##   .. ..- attr(*, "order")= int 1
##   .. ..- attr(*, "intercept")= int 1
##   .. ..- attr(*, "response")= int 1
##   .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   .. ..- attr(*, "predvars")= language list(BPSysAve, Age)
##   .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
##   .. .. ..- attr(*, "names")= chr [1:2] "BPSysAve" "Age"
##  $ residuals    : Named num [1:54] -15.897 0.452 9.575 17.579 -14.372 ...
##   ..- attr(*, "names")= chr [1:54] "1" "2" "3" "4" ...
##  $ coefficients : num [1:2, 1:4] 102.339 0.439 5.132 0.11 19.94 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:2] "(Intercept)" "Age"
```

```
##   .. ..$ : chr [1:4] "Estimate" "Std. Error" "t value" "Pr(>|t|)"
##  $ aliased      : Named logi [1:2] FALSE FALSE
##   ..- attr(*, "names")= chr [1:2] "(Intercept)" "Age"
##  $ sigma        : num 14.3
##  $ df           : int [1:3] 2 52 2
##  $ r.squared    : num 0.233
##  $ adj.r.squared: num 0.218
##  $ fstatistic   : Named num [1:3] 15.8 1 52
##   ..- attr(*, "names")= chr [1:3] "value" "numdf" "dendf"
##  $ cov.unscaled : num [1:2, 1:2] 0.12963 -0.00258 -0.00258 0.00006
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:2] "(Intercept)" "Age"
##   .. ..$ : chr [1:2] "(Intercept)" "Age"
##  $ na.action    : 'omit' Named int [1:6] 14 17 21 27 28 59
##   ..- attr(*, "names")= chr [1:6] "14" "17" "21" "27" ...
##  - attr(*, "class")= chr "summary.lm"
```

The structure of the object `lm_bas` shows that it is a lists with 12 elements (`List of 12`). The names of the list elements are shown after `$`. For example, we could find the R^2 in `$ r.squared`, where we could extract it as:

```
z$lm_r2 <- round(lm_bas$r.squared, 3)
```

We can extract the p-value, the F-value, the t-value, and other information in a similar way:

```
z$lm_p <- round(lm_bas$coefficients[2, 4], 6)
z$lm_f <- round(lm_bas$fstatistic[1], 3)
z$lm_t <- round(lm_bas$coefficients[2, 3], 2)
z$lm_n <- length(lm_bas$residuals)
z$lm_df <- lm_bas$df[1:2]
z$lm_b0 <- round(lm_bas$coefficients[1, 1])
z$lm_b1 <- signif(lm_bas$coefficients[2, 1], digits = 3)
```

Thus, the SLR model result could be presented in a statement like:

```
We obtained `r z$lm_n` subjects with the aim to produce a regression model predicting
the the average systolic blood pressure given the age. The regression equation
indicates that only `r z$lm_r2 * 100` % of the variability in the average systolic
blood pressure can be accounted for by variation in the age (Slope: `r z$lm_b0`,
Intercept: `r z$lm_b1`, $F_2, 52 = `r z$lm_f`$, $p = `r z$lm_p`$,
$R^2 = `r z$lm_r2`$).
```

which will be knitted into:

> We obtained 54 subjects with the aim to produce a regression model predicting the the average systolic blood pressure given the age. The regression equation indicates that only 23.3% of the variability in the average systolic blood pressure can be accounted for by variation in the age (Slope: 102, Intercept: 0.439, $F_{2,52} = 15.764$, $p = 2.21 \times 10^{-4}$, $R^2 = 0.233$).

The subscripts in $F_{2,52}$ indicate the F-test with numerator $d.f. = 2$ and the denominator $d.f. = 52$.

The R package **equatiomatic** (Anderson et al. 2022) provides a more straightforward and convenient way for inserting a regressed model as a maths expression in .Rmd. For example, we could write the following chunk:

```r
equatiomatic::extract_eq(lm_ba, use_coefs = TRUE)
```

and we get in the output document an equation with the fitted coefficients:

$$\widehat{\text{BPSysAve}} = 102.34 + 0.44(\text{Age}) \tag{9.1}$$

This method could be used for other regression models as well, such as `stats::glm()`, `MASS::polr()`, and `forecast::Arima()`.

9.6 Presenting Data via Tables

Tables, consisting of rows and columns, are often used to compare variables and organize data in a structured way. When the data are too much to be presented in statements, tables might be more preferred. For example, either a statement (preferably) or a simple table can present a one-way ANOVA table, while a table might be the only way for presenting multiple-way ANOVA results.

Tables are especially useful for presenting multiple dimensional results, which are challenging to be described in statements. The key principle of choosing a table is that the table makes it easy for the readers to understand the data. Thus, tables are often used to show precise values or specific data in a small space, to compare data values across shared characteristics, and to show the presence or absence of specific characteristics.

As a basic rule for tables in journal papers, data reports, and dissertations, every table is identified by a number and a caption. This rule does not have to be inherited in slides and posters. In public health science, observations in experimental groups or treatments are placed by rows, and statistics are placed by columns. Units of measurements must be clearly clarified, usually in the header (column names).

Here are some tips on constructing effective tables:

1. The caption should clearly describe what the table is about.
2. The header must be carefully designed. It must be descriptive and clearly indicate the nature of the data.
3. The information displayed in the table should be independent from the paragraphs. The readers could understand the table even without reading the entire paper.
4. Table footnotes could offer additional information.
5. A clean layout with sufficient spacing between columns and rows and legible fonts could help readers.

Table 9.3 Summary of the *NHANES* dataset

Variable	Mean	Standard deviation	Sample size
Age	37	22	10,000
Height	162	20	9647
Weight	71.0	29.1	9922
BPSysAve	118	17	8551

In an .Rmd document, a table can be manually created with the basic Markdown syntax:

```
| Variable | Mean | Standard deviation | Sample Size |
|----------|------|--------------------|-------------|
| Age      | 37   | 22                 | 10000       |
| Height   | 162  | 20                 | 9647        |
| Weight   | 71.0 | 29.1               | 9922        |
| BPSysAve | 118  | 17                 | 8551        |

: (\#tab:summary1) Summary of the NHANES dataset
```

It is knitted into Table 9.3. The last line beginning with `: (#tab:summary1)` gives the caption `Summary of the NHANES dataset` and assigns an automatic number as well as a unique label (`(\#tab:summary1)`) to this table.

More frequently, we use the `knitr::kable()` function for presenting a table in a .Rmd document, as long as the table has been prepared as a matrix or a data frame. For example, Table 9.3 can also be created with the code chunk:

```
```r summary1
library(tibble)
dtf_summary1 <- tribble(
  ~ Variable, ~ Mean, ~ `Standard deviation`, ~ `Sample Size`,
  'Age'      ,  37   , 22                   , 10000       ,
  'Height'   ,  162  , 20                   , 9647        ,
  'Weight'   ,  71.0 , 29.1                 , 9922        ,
  'BPSysAve' ,  118  , 17                   , 8551        ,
)
knitr::kable(dtf_summary1, format = 'latex', caption =
'Summary of the NHANES dataset')
```
```

In this example, the table label is given in the chunk option `{r summary1}`, and the table caption is specified as an argument of the `knitr::kable()` function.

Some functions create ready-for-print tables directly. For example, the `psych::describe()` function creates a summary table, and we could insert it into the Rmd document like this:

```
```r summary2
dtf_NHANES<- psych::describe(NHANES[, c('Age', 'Height', 'Weight', 'BPSysAve')])
knitr::kable(dtf_NHANES[, 1:5],
             format = 'latex',
             caption = 'Summary for NHANES dataset by the psych package.')
```
```

Even if the table has to be created manually, we could firstly make it in spreadsheet software (e.g. Microsoft Excel, LibreOffice) and then import it into R as a data frame.

A list of useful R package for tables in Rmd documents is as follows:

- **beautifyR** (Weigand 2020, GitHub mwip/beautifyR): RStudio addin for formatting R Markdown tables and comments
- **flextable** (Gohel and Skintzos 2022): Easily create tables for reporting and publications
- **gt** (Iannone et al. 2022): Easily generate information-rich, publication-quality tables from R
- **inserttable** (Busetto 2022, GitHub lbusett/insert_table): RStudio add-in facilitating insertion of nicely formatted tables in R markdown documents or plain R scripts.
- **kableExtra** (Zhu 2021): Construct complex tables with 'kable' and pipe syntax
- **stargazer** (Hlavac 2022): Well-formatted regression and summary statistics tables

For example, the `stargazer::stargazer()` creates a well organized table for the linear regression result:

```
stargazer::stargazer(lm_ba,
                     header = FALSE,
                     label = 'summary3',
                     title = 'Summary of the linear regression.')
```

A table can deliver more information than a statement with the same space. Sometimes need an additional paragraph to describe the context and content of the table as well as the methods applied on data to produce the table.

9.7 Presenting Data via Graphs

Graphs are a common way of presenting data in science. A graph can express a lot of information in limited space. In many cases, a reader's brief view on the graphs determines whether to complete the reading. Compared with statements and tables, graphs are often used to summarize large data sets, to present a visual explanation of events or features, and to show trends, patterns or relationships when the general pattern is more important than exact values. Using graphs to visualize large amounts of complex data is easier than poring over spreadsheets or reports. Nowadays, interactive graphs allow us to take the concept a step further by using technology to drill down into charts and graphs for more details, interactively changing what data we see and how it's processed.

Like a table, a graph is also identified with a number and a caption in journal papers, data reports, and dissertations, because they are cross-referred in the statements. Although the graph number and caption are not mandatory in slides and posters, they would be helpful in explaining them in the talks. A graph caption contains not

only a title, but also the explanations of the elements in the graph. For example, if there are sub-figures, each figure should be labelled with a letter, which should be clarified in the caption. If there are multiple colours, line types, or plotting characters, they should be explained in the caption unless they are self-explanatory. Most of the graphs in Chap. 8 follow these rules.

Here are some tips for presenting data via graphs:

- It is crucial to choose an appropriate graph type. Consider what kind of information we want to show to the readers before starting making a graph. Is it a comparison of categories between data, or is there a certain data relationship, or do you want to see the distribution of data, or is the data distributed over time, space, and geographical information? When the answer is clear, it becomes easy to choose a graph type.
- Give sufficient information in the caption, legends, and axis labels. The axes in a graph must be labelled with the variable names or meanings and the units, if any. All the texts in the graph must be readable. Avoid obscure abbreviations.
- All the parts of the graph must be legible and clear. Never use a distracting background. Use appropriate colours. Principles and methods for choosing colours involve a lot of knowledge in art and design, which is out of the scope of this book, but some ideas are introduced in Chapter 8.
- Do not blindly pursue unnecessarily complex graphs, which might break the original aim for presenting data. For example, beginners may think that 3D graphs are more trendy than 2D ones. There are times when a simple table may present data more clearly and accurately than an eye-catching graph.
- Reduce redundant information and highlight the key points. Redundant information results in losing the focus and disturbing the readers' mind in reflecting the key points.

R Markdown supports three major methods for presenting data in graphs.

When there is an existing image file, we can insert it using the basic Markdown syntax. Suppose we have an image file named `img1.png` located in the `image/` folder in the project path. The following Markdown syntax can insert the image:

```
![](image/img1.png)
```

This method is straightforward. However, we can hardly control the appearance, e.g. the size, the alignment, and the location, of the image.

The second method is the `knitr::include_graphics()` function, which provides sufficient setting for controlling the appearance:

````
```{r img1, out.width='60%', fig.cap='Image 1.', fig.align='center'}
knitr::include_graphics('image/img1.png')
```
````

The chunk options determines the label (`img1`), the width (`'60%'` means 60% of the page width), the caption (`Image 1.`), and the alignment (`'center'`) of the figure. Multiple images could be inserted side by side if we customize the

`out.width` and `results`. The following chunk inserts four images with the same width:

````
```{r img4, out.width='25%', results='hold', fig.cap='Windrose.', fig.align='center'}
knitr::include_graphics(c('img1.png', 'img2.png', 'img3.png', 'img4.png'))
```
````

When the image is a hyperlink, `knitr::include_graphics('url_of_the_image')` works fine for the `.html` output. If the output is `.pdf`, it does not work. In such a case, we can use the `bookdownplus::include_image('url_of_the_image')` function, which firstly downloads the remote image to the local disk and then inserts it into the output document.

These two methods, i.e. the basic Markdown method and the `knitr::include_graphics()` function, are only applicable for existing image files. We can, of course, plot graphs and save them as image files by R commands before inserting them, but more conveniently, we can directly run them in the code chunk within the .Rmd document, and they will be knitted as graphs which are inserted in the output document.

For example, we could write in R Markdown:

````
```r
plot(women)
```
````

and a graph is automatically inserted into the output document. The chunk options can be utilized as well as the previous examples.

In publications, we often have to combine multiple plots into one, either to better express our ideas, or to save space. Figure 8.15 shows an example. It is a graph including two sub-figures, which are labelled as A and B. The three colours of the filled areas and the lines indicate three countries. Thus, we insert it into this book by using the following code chunk:

````
```{r, fig.cap='Line chart and area plot.'}
p1 + p2 + plot_annotation(tag_levels = 'A')
```
````

Here we use the + operator, supported by the **patchwork** package (Pedersen 2022), for placing two ggplot2 graphs side by side horizontally. This package also provides the function `plot_annotation()`, which adds labels to the sub-figures automatically. Note that **patchwork** only works for graphs created by **ggplot2**. Other packages, such as **Rmisc** (Hope 2022), **ggpubr** (Kassambara 2022), **cowplot** (Wilke 2020), **grid**, and **gridExtra** (Auguie 2017), also provide solutions for combining multiple ggplot2 graphs and fulfil the complicated requirements of graph layouts.

The graphs plotted with base R functions should use other functions for combining sub-figures with labels. The most often used function is `par(mfrow)` or `par(mfcol)` for a simple layout, which displays grids in an equal size, and `layout()` for complicated one, which displays grids in customized sizes. Here are two examples:

```
# Organize 6 figures with par()
par(mfrow = c(2, 3)) # 2 rows, 3 columns
hist(women$height)
boxplot(women$height)
plot(women$height, women$weight)
plot(0, type = 'n', axes = FALSE, xlab = '', ylab = '')
barplot(VADeaths)
pie(VADeaths[, 1])

# Organize 6 figures with layout()
layout(matrix(1:6, nrow = 2), widths = c(3, 2, 1))
hist(women$height)
boxplot(women$height)
plot(women$height, women$weight)
plot(0, type = 'n', axes = FALSE, xlab = '', ylab = '')
barplot(VADeaths)
pie(VADeaths[, 1])
```

9.8 Integration of Statements, Tables, and Graphs

Statements, tables, and graphs are often mixed and integrated into each other in data reports, journal papers, and dissertations, so that they could complement each other and present data in a thorough way.

Cross-reference is the most common way of integrating tables and graphs into statements. In a .Rmd document, if a graph or a table is produced by a code chunk with the ID `fig-id` or `tab-id`, then we could use the syntax `\@ref(fig:fig-id)` or `\@ref(tab:tab-id)` for inserting the cross-reference into a statement, from where the readers not only know the values of the statistics, but also visually see the data in the referred graph or table.

For example, in Sect. 6.3 we summarizes the *diet* data. We could write:

````
```r fig-summary, fig.height=6, fig.cap='Visualization of the raw diet data.'
data(diet, package = "Epi")
diet[, c('height', 'weight', 'job', 'energy.grp')] |>
  fecitr::plot_summary(if_box = TRUE)
```

The mean height was 173.4 cm (sd = 6.4 cm, n = 332, Figure \@ref(fig:fig-summary)).
````

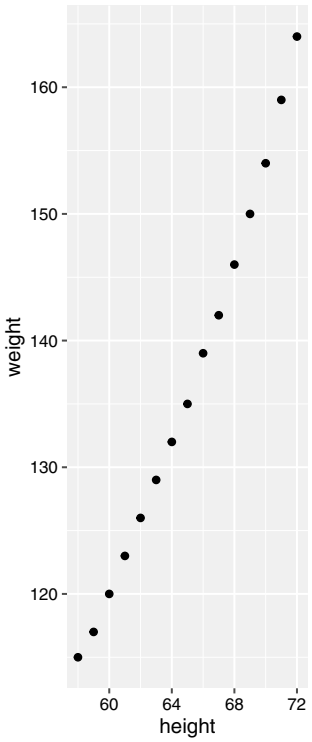
where `fig-summary` is the chunk ID. In the output document, `\@ref(fig:fig-summary)` is automatically replaced with the correct figure number, such as.

> The mean height was 173.4 cm (sd = 6.4 cm, n = 332, Figure 1).

Occasionally, we have to insert a statement, usually as a footnote, into a table or a graph apart from the caption, or combine a table with a graph. This task could be accomplished by the **grid** and **gridExtra** packages. The following chunk in Rmd could display a table beside a graph with additional statements at the top and the bottom (Fig. 9.1).

Title of the page

	height	weight
1	58	115
2	59	117
3	60	120
4	61	123
5	62	126
6	63	129
7	64	132
8	65	135
9	66	139
10	67	142
11	68	146
12	69	150
13	70	154
14	71	159
15	72	164



this footnote is right–justified

Fig. 9.1 Display statements, a table, and a graph together

```
library(gridExtra)
library(grid)
grid.arrange(
  tableGrob(women),
  ggplot(women) + geom_point(aes(height, weight)),
  ncol = 2,
  widths = c(1.5, 1),
  top = "Title of the page",
  bottom = textGrob(
    "this footnote is right-justified",
    gp = gpar(fontface = 3, fontsize = 9),
    hjust = 1,
    x = 1))
```

References

Allaire, J.J, Yihui Xie, Christophe Dervieux, R Foundation, Hadley Wickham, Ramnath Vaidyanathan, et al. 2022. rticles: Article Formats for R Markdown. *Journal of Statistical Software*. https://github.com/rstudio/rticles.

Allaire, J.J., Yihui Xie, Jonathan McPherson, Javier Luraschi, Kevin Ushey, Aron Atkins, Hadley Wickham, Joe Cheng, Winston Chang, and Richard Iannone. 2022. *rmarkdown: Dynamic Documents for R*. https://CRAN.R-project.org/package=rmarkdown.

Anderson, Daniel, Andrew Heiss, and Jay Sumners. 2022. *equatiomatic: Transform Models into LaTeX Equations*. https://CRAN.R-project.org/package=equatiomatic.

Auguie, Baptiste. 2017. *gridExtra: Miscellaneous Functions for "Grid" Graphics*. https://CRAN.R-project.org/package=gridExtra.

Aust, Frederik. 2019. *citr: RStudio Add-in to Insert Markdown Citations*. https://github.com/crsh/citr.

Balamuta, James. 2020. *uiucthemes: R 'Markdown' Themes for UIUC Documents and Presentations*. https://CRAN.R-project.org/package=uiucthemes.

Barnier, Julien. 2022. *rmdformats: HTML Output Formats and Templates for Rmarkdown Documents*. https://github.com/juba/rmdformats.

Rodriguez-Sanchez, Francisco,Jeffrey W Hollister, and Carl Boettiger. 2020. *rmdTemplates: A Collection of Rmarkdown Templates*. https://github.com/Pakillo/rmdTemplates.

Bucior, Ben. 2020. *drposter: Generate Academic Posters in R Markdown and CSS*. https://github.com/bbucior/drposter.

Busetto, Lorenzo. 2022. *inserttable: Automatically Add a Table to a RMarkdown Document*. https://github.com/lbusett/insert_table.

Dervieux, Christophe, J.J. Allaire, Rich Iannone, Alison Presmanes Hill, and Yihui Xie. 2022. *distill: RMarkdown Format for Scientific and Technical Writing*. https://CRAN.R-project.org/package=distill.

Eddelbuettel, Dirk, and Aaron Wolen. 2019. *linl: Linl Is Not Letter*. http://dirk.eddelbuettel.com/code/linl.html.

Eddelbuettel, Dirk, and James Balamuta. 2020. *pinp: Pinp Is Not PNAS*. http://dirk.eddelbuettel.com/code/pinp.html.

Eddelbuettel, Dirk, and Jonathan Gilligan. 2022. *tint: Tint Is Not Tufte*. https://CRAN.R-project.org/package=tint.

Eddelbuettel, Dirk, Ista Zahn, and Rob Hyndman. 2020. *binb: Binb Is Not Beamer*. https://github.com/eddelbuettel/binb.

Fay, Colin, Jonathan Sidi, and Luke Smith. 2018. *remedy: RStudio Addins to Simplify Markdown Writing*. https://github.com/ThinkR-open/remedy.

Gohel, David, and Panagiotis Skintzos. 2022. *flextable: Functions for Tabular Reporting*. https://CRAN.R-project.org/package=flextable.

Hlavac, Marek. 2022. *stargazer: Well-Formatted Regression and Summary Statistics Tables*. https://CRAN.R-project.org/package=stargazer.

Hope, Ryan M. 2022. *Rmisc: Ryan Miscellaneous*. https://CRAN.R-project.org/package=Rmisc.

Iannone, Richard, Joe Cheng, Barret Schloerke, Ellis Hughes, and JooYoung Seo. 2022. *gt: Easily Create Presentation-Ready Display Tables*. https://CRAN.R-project.org/package=gt.

Ismay, Chester, and Nick Solomon. 2022. *thesisdown: An Updated R Markdown Thesis Template Using the bookdown Package*.

Kassambara, Alboukadel. 2022. *ggpubr: ggplot2 Based Publication Ready Plots*. https://rpkgs.datanovia.com/ggpubr/.

Korpela, Mikko. 2022. *uniscape: Create Portable Strings*. https://github.com/mvkorpel/uniscape.

Mahr, Tristan. 2022. *WrapRmd: RStudio Addin for Wrapping RMarkdown Paragraphs*.

Makeev, Vadim, Oleg Jahson, Slava Oliyanchuk, Roman Komarov, Artem Polikarpov, Tony Ganch, Denis Hananein, Gábor Csárdi, Doug Ashton, and J.J. Allaire. 2018. *rmdshower: R 'Markdown' Format for Shower Presentations*. https://github.com/mangothecat/rmdshower.

Marwick. 2022. *wordcountaddin: Word Counts and Readability Statistics in R Markdown Documents*.
Olsen, Ludvig Renbo. 2017. *splitChunk: RStudio Addin for Splitting a Code Chunk in R Markdown*.
Pedersen, Thomas Lin. 2022. *patchwork: The Composer of Plots*. https://CRAN.R-project.org/package=patchwork.
Qiu, Yixuan. 2021. *prettydoc: Creating Pretty Documents from R Markdown*. https://github.com/yixuan/prettydoc.
Walthert, Lorenz. 2022. *strcode: Structure and Abstract Your Code*.
Weigand, Matthias. 2020. *beautifyR: RStudio Addin to Format Markdown Tables and R Comments*.
Wilke, Claus O. 2020. *cowplot: Streamlined Plot Theme and Plot Annotations for ggplot2*. https://wilkelab.org/cowplot/.
Xie, Yihui, and J.J. Allaire. 2022. *tufte: Tufte's Styles for R Markdown Documents*. https://github.com/rstudio/tufte.
Xie, Yihui, Christophe Dervieux, and Alison Presmanes Hill. 2022. *blogdown: Create Blogs and Websites with R Markdown*. https://CRAN.R-project.org/package=blogdown.
Xie, Yihui, Romain Lesur, Brent Thorne, and Xianying Tan. 2022. *pagedown: Paginate the HTML Output of R Markdown with CSS for Print*. https://github.com/rstudio/pagedown.
Xie, Yihui. 2022a. *knitr: A General-Purpose Package for Dynamic Report Generation in R*. https://yihui.org/knitr/.
Xie. 2022b. *tinytex: Helper Functions to Install and Maintain TeX Live, and Compile LaTeX Documents*. https://github.com/rstudio/tinytex.
Xie. 2022c. *xaringan: Presentation Ninja*. https://github.com/yihui/xaringan.
Yazici, Metin. 2022. *caseconverter: RStudio Addin to Convert Text Cases*.
Zhao, Peng. 2020. *bookdownplus: Generate Assorted Books and Documents with R bookdown Package*. https://github.com/pzhaonet/bookdownplus.
Zhu, Hao. 2021. *kableExtra: Construct Complex Table with kable and Pipe Syntax*. https://CRAN.R-project.org/package=kableExtra.

Chapter 10
Managing Data

10.1 Chapter Highlights

- Illustrate the basic flow and key elements of data management,
- Make sense of the importance and the means of recording metadata, and
- Utilize R packages for project management.

10.2 Introduction

Managing data is the process of organizing, managing and utilizing data at an optimal way to complete a project within the required scope, time and costs (Basu 2016). It is an integral component of a research or any other project, which is carried out during the whole life cycle of the data from planning, collection, analysis, dissemination, and depositing.

Managing data in an appropriate and timely way is significantly crucial for public health researches and related work. It enables our colleagues, sponsors, or sometimes the publishers to understand our research findings and values. For instance, it is common that public health researchers might have some funding and need the collaboration of people from various fields, such as epidemiologists, health economists, and health policy-makers. Consequently, the researchers need to demonstrate stage results and achievements for other people, and a project with a logical structure as well as efficient and secure access is required. Managing data also helps to ensure the safety and security of the data and increase the data processing efficiency. For example, some qualitative studies may include interviews involving sensitive topics that need to ensure confidentiality. We often get datasets that are collected and saved in various formats during different programs and periods by different people. Some variables are labelled with abbreviations or cryptic codes without documentations, which nobody can understand after a long time. Moreover, some research projects might last for several years, such as the cohort study needing a long follow-up period.

P. Zhao, *Working with Data in Public Health: A Practical Pathway with R*,
https://doi.org/10.1007/978-981-99-0135-7_10

Thus, data categorized and documented well could reduce mistakes and improve efficiency during the follow-up studies. A good data management, especially in a public health research project, enables researchers to precisely achieve the research objectives, control the process and quality of the research with the limited financial, human and other various types of resources.

Managing data plays a key role throughout the project stages, including the project development, implementation, and closeout (Basu 2016). The project development refers to the stage that conducts some preparation and planning for the whole project, which often contains drafting data management plans. During the project implementation stage, the efforts should be put into improving the communication efficiency, controlling the data quality, and implementing the data management plan, as well as managing crisis that might occur. Finally, when the project comes into the closeout stage, all the data should be stored or disseminated.

10.3 Data Management Framework

A clear logical framework is a tool to improve the data management and ensure the project implementation. It breaks down the blue picture of the project into step-by-step practical action plans. Basically, the logical framework includes the input data as well as the metadata, the processing code, the output documents, the means to obtain the required resources, the assumption of the project.

R packages are available for helping managing data and projects in a logical way, such as **ProjectTemplate** (Auguie 2017), **makeProject** (Silverman 2012), **rrtools** (Marwick 2019), and **workflowr** (Blischak et al. 2021). The packages **devtools** (Wickham et al. 2022b) and **usethis** (Wickham et al. 2022a) suggest using the R package structure for organizing projects. In this chapter we introduce two packages: **prodigenr** (Johnston 2022) and **rosr** (Zhao 2021).

10.3.1 The prodigenr Package

The **prodigenr** package is named as an abbreviation of the ***Pro***ject ***di***rectory ***gen***erato***r***. We could create a **prodigenr** project either from the RStudio menu `RStudio - File - New Project - New Directory - Scientific Analysis Project using prodigenr` or with the command:

```
prodigenr::setup_project("dph")
```

Then a structured directory `dph/` is created. The sub-directories and files in it could be viewed either by the file browser or by the R package **data.tree** (Glur 2020):

```
mydir <- list.files('dph', full.names = TRUE, recursive = TRUE)
mytree <- data.tree::as.Node(data.frame(pathString = mydir))
print(mytree)
```

```
dph
 ¦--data-raw
 ¦   |--README.md
 ¦--data
 ¦   |--README.md
 ¦--DESCRIPTION
 ¦--doc
 ¦   |--README.md
 ¦--dph.Rproj
 ¦--R
 ¦   |--README.md
 ¦--README.md
 |--TODO.md
```

A recommended work flow of **prodigenr** is as follows:

1. Write up the analysis and associated written explanations of the results in the abstract, poster, slides, or manuscript `.Rmd` files in the `doc/` folder.
2. Convert any piece of code which we use more than once or is fairly complex into a function. Put this new function into a file (or a `functions.R` file) in the `R/` directory. Load that function using `devtools::load_all()` (hotkey: ctrl+shift+l).
3. Fetch and wrangle the data in the `R/fetch_data.R`. Access the data using the `devtools::load_all()` function.
4. Create more .Rmd files in the `doc/` folder to add analyses that will supplement the main document or that are exploratory.
5. Knit the `.Rmd` files in `doc/`. Now we have our final abstract, poster, slides, or manuscript to use for our research.

10.3.2 The rosr Package

The **rosr** package is named as the abbreviation of *Create Rep**ro**ducible Re**s**earch Projects*. It is an R package for creating reproducible academic project with integrated various academic elements, including data, bibliography, codes, images, manuscripts, dissertations, slides and so on. These elements are well connected so that they can be easily synchronized and updated. Users don't have to repeat copying and pasting their results and figures from time to time. It will be easy for the scientific researchers to use, even if they are R beginners, or even non-R-users.

A **rosr** project could be created either via the RStudio tool bar `Addins - Create a rosr project` or the command:

```
rosr::create_rosr('dph2')
```

Then a structured directory `dph2/` is created with demonstrative files including some Rmd documents. The sub-directories and files look like:

```
dph2
 |---dph2.Rproj
 |--bib
 |   °--rosr.bib
 |--data
 |   °--rosr.csv
 |--equation
 |   °--rosr-eq.Rmd
 |--manuscript
 |   °--copernicus
...
 |--poster
 |   °--drposter
...
 |--R
 |   °--rosr.R
 |--rpkg
...
 °--slide
     °--xaringan
```

By default, the demonstration Rmd documents are only created rather than knitted. If we want to see the output demonstrative journal manuscript, slides, etc., we could set the argument `if_render = TRUE`:

```
rosr::create_rosr('dph2', if_render = TRUE)
```

It might take some minutes to finish rendering the documents if it is the first time you use the package.

A recommended work flow of **rosr** is as follows:

1. Prepare data in the `data/` folder. Create meta data.
2. Write R scripts in the `R/` folder.
3. Prepare reference libraries in the `bib/` folder.
4. Prepare equations in the `equation/` folder.
5. Prepare external images in the `image/` folder.
6. Use these materials in the manuscript in the `manuscript/` folder.

10.4 Raw Data

During the project development stage, only the data planning is needed. A data management plan basically includes what to do with the data to achieve the aims better. The plan's components include the type and form of the data people plan to collect, the file formats that are convenient for the data analysis, and the size of the data, which all have been discussed in the previous chapters. Additionally, project and data documentation, which are also called metadata, write down all the above explanatory information in texts and are also critical in the data management plan. The project documentation describes the whole project and data set, while the data documentation is the detailed explanation of the data set. Both documentations make the data have context and be meaningful all the time.

10.5 Metadata

10.5.1 Introduction

Metadata are the data that describe other data. R provides good examples for showing the metadata for built-in datasets. For example, we use the `help()` function to view the metadata for the *Vanderpump* dataset shipped by the **pubh** package:

```
help(Vanderpump, package = 'pubh')
```

The page for the metadata is displayed with multiple sections in the help tab, including:

- Title, which briefly describes what data set it is ("Smoking and mortality in Whickham, England");
- Description, which describes the dataset in a statement;
- Usage, which shows how to load it in R;
- Format, which describes in detail the meaning, the unit, and the factor levels of each column;
- Source, which lists the references from which the dataset comes, and
- Examples, which demonstrates how to use this dataset in R.

In the subsequent sections, we demonstrate a variety ways of documentating metadata with R. Here we prepare the dataset for example. We can download a data frame of life expectancy at birth with the **WHO** package.

```
LEB <- WHO::get_data("WHOSIS_000001")
```

For simplicity, we remove unnecessary columns and convert the character columns into factors:

```
LEB <- as.data.frame(unclass(LEB[, -c(5, 6)]),
                     stringsAsFactors = TRUE)
```

10.5.2 Metadata in Variable Names

The simplest way for saving metadata is including them in the variables. For example, the original column names of 'LEB' are:

```
names(LEB)
```

```
## [1] "region"                "year"                  "country"
## [4] "sex"                   "value"                 "worldbankincomegroup"
```

The column `value`, which is not clear to the users, needs more explanation. Thus, this column could be renamed as:

```
LEB2 <- LEB
names(LEB2)[5] <- 'Life expectancy (years)'
```

The new data frame `LEB2` could be exported either as a .csv file or a .xlsx file, in which the metadata are shipped in the column names. This method is easy and straightforward, but the usage is very limited. When there is more information in the metadata, the variable names could be too long for further data analysis and visualization.

10.5.3 Metadata in Labels

Metadata could be saved in a variable label, which serves as an alias of variable name. The **sjlabelled** package provides a function `var_label()`, which adds a label as attribute (named "label") to the variable. For example, we could add a label "Life expectancy (years)" to the variable `value`:

```
require(sjlabelled)
LEB3 <- var_labels(LEB,
                   value = "Life expectancy (years)")
str(LEB3$value)
```

```
##  num [1:2329] 82.8 54.6 55.4 55 46.8 52.1 49.3 74.2 78.3 76.2 ...
##  - attr(*, "label")= chr "Life expectancy (years)"
```

```
get_label(LEB3)
```

```
##                    region                      year                   country
##                        ""                        ""                        ""
##                       sex                     value      worldbankincomegroup
##                        "" "Life expectancy (years)"                        ""
```

LEB3 could be exported as a `.rda` or `.rds` file, in which the metadata are shipped as column labels. The labels could contain a long plain string as the metadata, and the labels do not affect the usage of the column names. It is very convenient for R users. The limitation is that `.rda` and `.rds` files are only supported by R; if we share them with non-R users, it might a problem for them to reuse the data.

10.5.4 Metadata in Headers or Independent Files

A common way for recording metadata is saving them in the header of a plain text file which includes the data. For example, we could save LEB as a plain text file:

```
write.csv(LEB, 'LEB.csv', row.names = FALSE, quote = FALSE)
```

Then, open `LEB.csv` in a text editor (e.g. NotePad in Windows OS), and insert the metadata at the beginning of the file:

```
The numbers of deaths estimated from life table
and population by age groups  are aggregated by
relevant region in order to compute regional life tables.
Variables:
value: Value of the life expectancy at birth, in years.
Other variables are self-explanary.
Data source: \url{https://www.who.int/data/gho/indicator-metadata-registry/imr-
details/65}

region,year,country,sex,value,worldbankincomegroup
Americas,1920,Canada,Both sexes,82.8,NA
Eastern Mediterranean,2000,Afghanistan,Male,54.6,NA
Eastern Mediterranean,2000,Afghanistan,Female,55.4,NA
```

Now LEB.csv ships both the data and the metadata. As plain text files can be opened with any text editor, and this method is not limited to R users.

R users can import this `.csv` file with skipping the metadata in the first six lines:

```
read.csv('LEB.csv', skip = 6)
```

Alternatively, the metadata can be saved independently in another document, which must always be shipped with the data file.

10.5.5 Metadata in R Packages

Datasets shipped in packages are friendly to R users. Thus, we could develop an R package that ships our own datasets (see Wickham 2015). Here we give an example showing how to do it in an R package.

A new R package can be created from the RStudio menu `File - New Project - New Directory - R Package`. Fill in the blanks with the package name `phdata` and the path `D:/temp` and confirm creating it. A project with the package name `phdata` is opened with a demonstration `hello.R` file. The subsequent work is performed in this project.

Firstly, we save the dataset as R built-in data format in the folder `data/`:

```
save(LEB, file = 'data/LEB.rda')
```

Then, delete everything in `hello.R`, and type metadata into it. A minimal template looks like:

```
#' Title
#'
#' Short description.
#'
#' @format
#'
#' \describe{
#'   \item{item name}{explanation}
#'   \item{item name}{explanation}
#' }
#' @source \url{}
"dataset_name"
```

Let's fill in the fields with the metadata of `LEB`. The format can be found via:

```
class(LEB)
```

```
## [1] "data.frame"
```

```
dim(LEB)
```

```
## [1] 2329    6
```

It is a data frame with 2329 rows and 6 variables. The description for each variable could be found via:

```
str(LEB)
```

```
## 'data.frame':    2329 obs. of  6 variables:
##  $ region             : Factor w/ 7 levels "Africa","Americas",..: 2 3 3 3 1 1 1 4 4 4 ...
##  $ year               : num  1920 2000 2000 2000 2000 2000 2000 2010 2010 2010 ...
##  $ country            : Factor w/ 183 levels "Afghanistan",..: 31 1 1 1 4 4 4 2 2 2 ...
##  $ sex                : Factor w/ 3 levels "Both sexes","Female",..: 1 3 2 1 3 2 1 3 2 1 ...
##  $ value              : num  82.8 54.6 55.4 55 46.8 52.1 49.3 74.2 78.3 76.2 ...
##  $ worldbankincomegroup: Factor w/ 5 levels "Global","High-income",..: NA NA NA NA NA NA NA NA NA NA ...
```

The variables `year` and `value` are numerical and other ones are categorical. The levels of a factor can be obtained either by the `levels()` function or the `unique()` function. It might be boring to fill in the `\item{}{}` with these pieces of information. Here we create two self-defined functions to simplify this step:

```
roxygen_colname <- function(x) {
  y <- paste0("#'   \\item{", names(x), "}{}")
  writeLines(y, 'clipboard')
}
roxygen_fct <- function(x){
  y <- paste("factor with", nlevels(x), "levels:",
             paste(levels(x), collapse = ', '))
  writeLines(y, 'clipboard')
}
```

The `roxygen_colname()` function generates a character vector of all the items with the variable names in the description and writes the vector into the clipboard:

```
roxygen_colname(LEB)
```

The following text is now in the clipboard:

```
#'   \item{region}{}
#'   \item{year}{}
#'   \item{country}{}
#'   \item{sex}{}
#'   \item{value}{}
#'   \item{worldbankincomegroup}{}
```

Then simply paste it into the `\describe{}` field of `hello.R`.

The `roxygen_fct()` function creates a description for a factor. For example, `roxygen_fct(LEB$sex)` generates the following text in the clipboard:

```
factor with 3 levels: Both sexes, Female, Male
```

which can be simply pasted into the field `\item{sex}{}` as `\item{sex}{factor with 3 levels: Both sexes, Female, Male}`. The des-

criptions for all the categorical variables can be generated in the similar way. Those for the numerical variables must be filled manually.

Finally, `hello.R` looks like this:

```
#' Life expectancy at birth
#'
#' The numbers of deaths estimated from life table and population by age groups  are aggregated by relevant region in order to compute regional life tables.
#'
#' @format A data frame with 2329 rows and 6 variables:
#'
#' \describe{
#'   \item{region}{Region name, factor with 7 levels: "Africa", "Americas", "Eastern Mediterranean", "Europe", "Global", "South-East Asia", "Western Pacific".}
#'   \item{year}{Year.}
#'   \item{country}{Country name, factor with 183 levels of the county names in the world.}
#'   \item{sex}{Sex, factor with 3 levels: "Both sexes", "Female", "Male".}
#'   \item{value}{Value of the life expectancy at birth, in years.}
#'   \item{worldbankincomegroup}{World Bank income group, factor with 5 levels: Global, High-income, Low-income, Lower-middle-income, Upper-middle-income}
#' }
#' @source \url{https://www.who.int/data/gho/indicator-metadata-registry/imr-details/65}
"LEB"
```

Save the change, and document the package with:

```
devtools::document()
```

The package is completed.

Now the folder of the package project `D:/temp/phdata` can now be shared and reused. The function `devtools::load_all()` can load the package without installation, the `data()` function can load the datasets in the package, and the `help()` function can display the metadata in a friendly way:

```
devtools::load_all('D:/temp/phdata')
data(LEB)
help(LEB)
```

10.6 Scripts

The R commands for working with data are saved in R script files. An R script file, with the extension name of `.r` or `.R`, contains a set of R commands in one single file. As it is technically a plain text file, it could be viewed and edited with any plain text editor, such as NotePad in Windows OS. Thus, an R script file is cross platform and software-independent: we even can read it without installing R. Not only can it process our data, but also does it tell us how to process the data, if only there are sufficient comments in it.

10.7 Version Control

Multiple versions of data often exist in the process of research. Although a good practice is suggested to preserve the raw data in the original format, it is not surprising that we accidentally change them due to carelessness or the unawareness of their

importance, especially for junior researchers. When cleaning and analyzing data, we create step-by-step intermediate data files and update them from time to time. More usually, multiple versions of the documents for presenting data are produced during the collaboration with the co-authors, mainly the supervisors or the supervisees and reviewers.

The idea of version control came from software development, where tracking and managing changes to software code is very necessary. Version control software tools have nowadays extended their usage from programming to academic data management and scientific writing. For R users, version control helps researchers manage the changes in data, R scripts, LaTeXequations, reference libraries, and manuscripts not only in individual work but also in collaboration. The version control tool, consisting of RStudio IDE and Git, can keep track of every modification to each link in the chain of data work in a special kind of database. Just like a time machine, version control could turn back the clock, compare earlier versions of the documents, and, if necessary, recover them to a desired version, which is extremely useful for avoiding catastrophic mistakes. Furthermore, version control provides a smart way for merging collaborated work from multiple contributors, such as collaborative writing of a manuscript.

The open-source version control program Git could be downloaded from the Git website[1]. It is cross-platform and available for Windows, macOS, and Linux/Unix. After the installation of Git, follow these steps (Fig. 10.1):

1. Open RStudio IDE.
2. Find in the menu bar `Tools - Global Options - Git/SVN`.
3. Click `Enable version control interface for RStudio projects`.
4. Find the path to the Git executable file by clicking the `Browse` button.
5. You may be required to restart RStudio IDE before it is ready for use.
6. Find in the RStudio menu bar `Tools - Project Options - Git/SVN`.
7. Choose `Git` in the `Version control system`.
8. Click `Yes` for initializing a new git repository for this project.
9. Confirm restarting RStudio IDE.

When we see a new tab named `Git` in the top-right pane (Pane 4), it means the version control tool is ready for use. We could see all the files in the working directory now listed in Pane 4, with two question marks before each file name (Fig. 10.2left). Choose the one(s) (`death_rate.txt` in this example) for which we want to track the version, and the question marks turn into `A` (which means *added*) with a different background colour. Click the `Commit` button.

A window for committing pops up. We can see the change of the file(s) with a light-green background in the lower panel. Now we write some text, like "Git starts", in the top-right `Commit message` panel, and click the button `Commit` (Fig. 10.2right). The current version of the chosen file(s) are saved as a ring in the history chain.

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

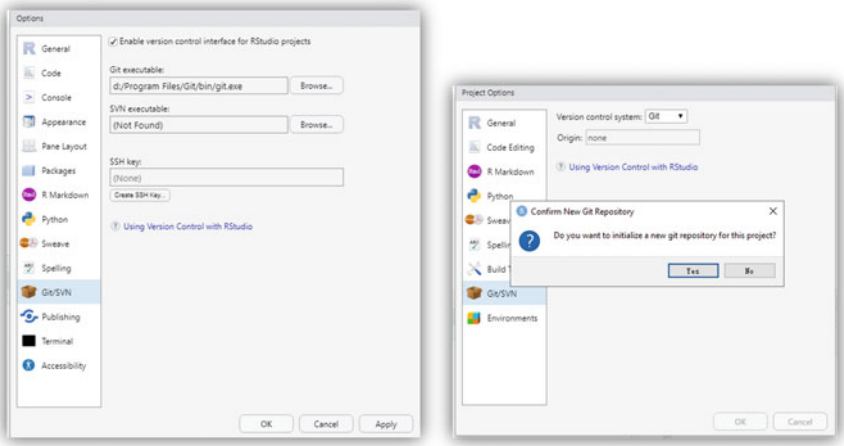


Fig. 10.1 Set up Git in RStudio IDE. Left: The window for enabling Git for RStudio IDE. Right: The window for initialzing Git for a RStudio project

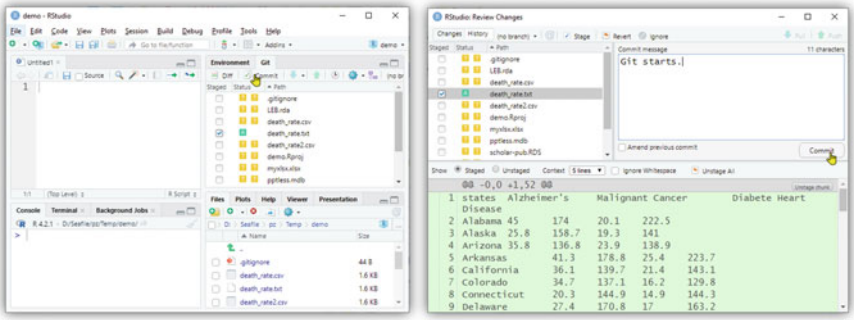


Fig. 10.2 Commit an R project with Git. Left: Selece the file with changes for commit. Right: Check the change in a file and commit it

Let's modify the "death_rate.txt" file with an inserted line, and save it. A label `M` appears before the file name "death_rate.txt", indicating that this file has been modified. Repeat the previous procedure, and the modified version is saved as a new ring in the version chain. We could see the entire version chain of a certain file in the `History` tab, and see the details in the difference between every two neighboured rings/versions.

One of the greatest advantages of the version control is that we could roll back to any tracked version, which is extremely useful in scientific research. There are two situations of rolling back.

1. If the current version of the document is saved but not committed (i.e. the modified version has not been saved in the version history yet), we see it with a label

`M` in the file list in the `Git` panel. Right click the file name, and click `Revert` in the popped-up menu. Then the document rolls back to the latest version saved in the version chain (Caution! The modification is gone.)

2. If the current version is committed, click the terminal tab in bottom-left pane (Pane 2) shown in Fig. 10.2 (left), and run the following code:

```
git reset --hard HEAD~1
```

Then the document rolls back to the previous version saved in the version chain (Caution! The latest version is gone.)

The previous description of the usage of Git only demonstrates the very fundamental usage of Git, which is just a tip of the Git iceberg. We could use online Git service such as GitHub for hosting our projects and inviting colleagues for collaboration with a more comprehensive version control, which is beyond the scope of this book. Note that the version control tool is friendly and not limited to plain text documents. Anyway, if we manage data, programming scripts, equations, literature libraries, manuscripts, and presentation documents in plain text format such as R Markdown (Chap. 9), Git could display each step in the version history in a visual way.

10.8 Exercises

The **COVID19** package is an interface for downloading the COVID-19 data from https://covid19datahub.io. We could get the daily summary of COVID-19 cases, deaths, recovered, tests, vaccinations, hospitalizations, policy measures, mobility data, geospatial identifiers, and so on, for >230 countries, >760 regions, and >12,000 administrative divisions of lower level.

- Download the dataset with the package default settings.
- Insert the metadata as labels of the columns by using the **sjlabelled** package, and export it as the R default format.
- Create an R package, which ships the dataset with a complete documentation.

References

Auguie, Baptiste. 2017. *gridExtra: Miscellaneous Functions for "Grid" Graphics*. https://CRAN.R-project.org/package=gridExtra.

Basu, R. 2016. *Managing Projects in Research and Development. Book*. Milton Park, Abingdon-on-Thames, Oxfordshire, England, UK: Routledge.

Blischak, John, Peter Carbonetto, and Matthew Stephens. 2021. *workflowr: A Framework for Reproducible and Collaborative Data Science*. https://github.com/workflowr/workflowr.

Glur, Christoph. 2020. *data.tree: General Purpose Hierarchical Data Structure*. http://github.com/gluc/data.tree.

Johnston, Luke. 2022. *prodigenr: Research Project Directory Generator*. https://CRAN.R-project.org/package=prodigenr.

Marwick, Ben. 2019. *rrtools: Creates a Reproducible Research Compendium*. https://github.com/benmarwick/rrtools.

Silverman, Noah. 2012. *makeProject: Creates an Empty Package Framework for the LCFD Format*. https://CRAN.R-project.org/package=makeProject.

Wickham, Hadley, Jim Hester, Winston Chang, and Jennifer Bryan. 2022a. *devtools: Tools to Make Developing R Packages Easier*. https://CRAN.R-project.org/package=devtools.

Wickham, Hadley, Jennifer Bryan, and Malcolm Barrett. 2022b. *usethis: Automate Package and Project Setup*. https://CRAN.R-project.org/package=usethis.

Wickham, Hadley. 2015. *R Packages: Organize, Test, Document, and Share Your Code*. "O'Reilly Media, Inc.".

Zhao, Peng. 2021. *rosr: Create Reproducible Research Projects*. https://github.com/pzhaonet/rosr.