

Alan Agresti and Maria Kateri

R-Web-Appendix of Foundations of Statistics for Data Scientists



Contents

0	CHAPTER 0: BASICS OF R	1
A0.1	Starting a Session, Entering Commands, and Quitting	1
A0.2	Installing and Loading R Packages	2
A0.3	R Functions and Data Structures	3
A0.3.1	Vectors and Lists	4
A0.3.2	Factors	6
A0.3.3	Matrix and Array	6
A0.3.4	Data Frames	8
A0.3.5	User-Defined Functions	9
A0.3.5.1	Example of user-defined functions: weighted mean	9
A0.4	Data Input in R	10
A0.5	Control Flows	12
A0.6	Graphs in R	13
1	CHAPTER 1: R FOR DESCRIPTIVE STATISTICS	15
A1.1	Data Handling and Wrangling	15
A1.1.1	<code>summarize</code>	16
A1.1.2	<code>group_by</code>	16
A1.1.3	<code>arrange</code>	17
A1.1.4	<code>slice</code> and <code>filter</code>	17
A1.1.5	<code>select</code>	18
A1.1.6	<code>mutate</code>	18
A1.2	Histograms and Other Graphics	18
A1.2.1	Histograms with Bins of Unequal Size	19
A1.3	Descriptive Statistics	20
A1.3.1	Trimmed Mean	21
A1.3.2	Summarizing Categorical Data	21
A1.3.3	Group-Specific Descriptive Statistics	24
A1.3.3.1	Stacked histograms and side-by-side box plots	25
A1.4	Missing Values in Data Files	27
A1.5	Summarizing Bivariate Quantitative Data	28
A1.5.1	Pairwise Correlations for Quantitative Data	29
A1.6	Summarizing Bivariate Categorical Data	31
2	CHAPTER 2: R FOR PROBABILITY DISTRIBUTIONS	35
A2.1	R Functions for Probability distributions	35
A2.1.0.1	Densities Plots	36
A2.2	Quantiles, $Q-Q$ Plots and the Normal Quantile Plot	37
A2.3	Joint and Conditional Probability Distributions	42
A2.4	The bivariate normal distribution	43

3	CHAPTER 3: R FOR SAMPLING DISTRIBUTIONS	45
	A3.1 Simulating the Sampling Distribution of a Statistic	47
	A3.2 Monte Carlo Simulation	48
	A3.2.1 MC Estimation of a Binomial Success Probability π	50
	A3.2.2 MC Estimation of the Sample Median	51
4	CHAPTER 4: R FOR ESTIMATION	53
	A4.1 Confidence Intervals for Proportions	53
	A4.2 Confidence Intervals for Means of Subgroups and Paired Differences	55
	A4.3 The t and Other Probability Distributions for Statistical Inference	57
	A4.4 Empirical Cumulative Distribution Function	58
	A4.5 Nonparametric and Parametric Bootstrap	59
	A4.5.1 Bootstrap Confidence Intervals (CIs)	60
	A4.5.2 Bootstrap Bias Estimation	61
	A4.5.3 Example: UN Data File	62
	A4.5.4 Parametric Bootstrap	64
	A4.6 Bayesian HPD Intervals Comparing Proportions	66
	A4.7 Example: HPD and EQT Intervals for a Binomial Success Probability	67
	A4.8 Example: EQT Intervals for Normal Means	69
5	CHAPTER 5: R FOR SIGNIFICANCE TESTING	73
	A5.1 Bayes Factors and a Bayesian t Test	73
	A5.2 Simulating the Exact Distribution of the Likelihood-Ratio Statistic	77
	A5.3 Nonparametric Statistics: Permutation Test and Wilcoxon Test	79
6	CHAPTER 6: R FOR LINEAR MODELS	81
	A6.1 Linear Models with the <code>lm</code> Function	81
	A6.2 Diagnostic Plots for Linear Models	82
	A6.2.1 A Linear Model Example: Salaries Data Set	83
	A6.3 Plots for Regression Bands and Posterior Distributions	85
	A6.4 Bayesian Linear Regression for Improper Priors	89
7	CHAPTER 7: R FOR GENERALIZED LINEAR MODELS	93
	A7.1 The <code>glm</code> Function	93
	A7.2 Plotting a Logistic Regression Model Fit	94
	A7.3 Model Selection for GLMs	95
	A7.3.1 Log-Linear Models Selection	97
	A7.4 Correlated Responses: Marginal, Random Effects, and Transitional Models	100
	A7.4.1 Marginal Models and Generalized Linear Mixed Models	101
	A7.5 Modeling Time Series	103
8	CHAPTER 8: R FOR CLASSIFICATION AND CLUSTERING	107
	A8.1 Visualization of Linear Discriminant Analysis Results	107
	A8.2 Cross-Validation in R	109
	A8.3 Classification and Regression Trees	110
	A8.4 A Cluster Analysis with Quantitative Variables	112
	Bibliography	117

0

CHAPTER 0: BASICS OF R

R is a free software for statistical computing and graphics that enjoys increasing popularity among data scientists. It is supported by Windows, Linux and macOS (see <http://www.r-project.org/> for downloading R and for information about installation, help and documentation). Software R uses the S programming language and a similar environment. It is continuously enriched and updated by researchers who develop new statistical methods and supplement their published results with the associated R code. This way, one can find a variety of updated add-on packages for basic or advanced data analysis and visualization, most of them stored in CRAN (Comprehensive R Archive Network). Meanwhile R became the dominating statistical software, supported by a strong voluntary R-community that developed an R-culture. With the occasion of the 25th anniversary of the creation of R, *Significance* published an article on its history and perspectives (Thieme, 2018).

Many R users prefer to work in RStudio, which is an integrated development environment (IDE) for R that includes a console, syntax-highlighting editor supporting direct code execution, as well as tools for plotting, history, debugging and workspace management. RStudio is available also in an open source edition (RStudio (download)).

This Appendix is not to be considered as a kind of R-manual. It supplements the chapters of this book and is an extended version its Appendix A, motivating and enabling the direct application of all the discussed statistical analyses. For starting with R and detailed guidance on R-programming, very helpful are the freely online available R-manuals, especially the ‘An Introduction to R’ and ‘R Data Import/Export’. Furthermore, there exists a vast variety of books and on-line notes; see for example the Swirl tutoring system at <https://swirlstats.com>, books by Hothorn and Everitt (2014), Wickham and Golemund (2017), and Baumer et al. (2017), or Altham’s Notes at <http://www.statlab.cam.ac.uk/pat/redwsheets.pdf>.

Here, we briefly refer to some very basics, needed follow the examples worked-out in R. Essentials to start with R are summarized in the first chapter of this appendix. The sequel chapters provide chapter-wise additional R-examples and highlights.

A0.1 Starting a Session, Entering Commands, and Quitting

Activation of R opens a console awaiting for input at the prompt (`>`). An R session is terminated by typing

```
> q()
```

or by selecting ‘Exit’ in the ‘File’ Menu. Comments (inactive text) can be entered in an R-command script after the symbol `#`, within the same line. Values to variables are assigned by the operator `←` or `=` (`→` is also possible but not recommended since variables definitions are not easy identifiable and thus code checking procedures are more complicated). The assignment operator does not provide any output. In order to see the value of a variable x , you need to type x at the command prompt. or to provide the assignment in parentheses.

You can provide multiple commands in a line by separating them by ‘;’ while a command can expand to more than one lines. In this case, a + appears at the beginning of the additional line(s), indicating that this is not a new command or assignment. Finally note that R is a case-sensitive language.

```
> x <- 7; x; X <- 10; X # equivalent to: (x <- 7); (X <- 10)
[1] 7
[1] 10
```

Output starts with [1], indicating that this line starts with the first value of the results.

The provided R-code examples throughout this book resemble the R-console. Thus, command lines start with a > (or + if a command expands along more lines). These > and + are not part of the command and should not be typed in the console while reproducing the examples.

A0.2 Installing and Loading R Packages

The philosophy of R is based on the built-up of a subjective personal computing environment, loading selected R-packages, additional to the **base**-package and some other basic packages that come default with the R-installation. An R package consists of a set of functions (required for performing the statistical analyses or graphics the package is designed for), their documentation files and data sets.

A package, say **binom**, is available for use in an R-workspace, only if it is installed and then loaded in the workspace by selecting it from the lists of the corresponding options in the ‘Packages’ Menu. Installation of a package requires the selection of a CRAN mirror. Alternatively, you can directly type in the console

```
> install.packages("binom")
> library(binom)
```

We list next the major R packages used in this book. A base R installation along with these packages form a powerful and broad tool-box for performing statistical analysis and tackling data analysis problems of diverge fields. An overview of all contributed packages in CRAN can be found in <http://cran.r-project.org/web/packages/>

Package	Description
BayesFactor	Bayes factors for simple designs (contingency tables, one- and two-sample designs, ANOVA designs, linear regression)
BEST	Bayesian <i>t</i> -test
binom	Binomial confidence intervals for several parameterizations (nine methods)
boot	Bootstrap R (S-Plus) functions
car	Companion to Applied Regression
caret	Classification and regression training
coin	Conditional inference procedures in a permutation test framework
DescTools	Tools for descriptive statistics and exploratory data analysis
factoextra	Functions for visualizing cluster analysis results
fMultivar	Functions for analyzing multivariate data sets of financial returns
foreign	Read data stored by Minitab, SAS, SPSS, Stata, Systat, dBase, ...
gplots	Various R programming tools for plotting data
graphics	R functions for base graphics
LearnBayes	Functions helpful in learning the basic tenets of Bayesian statistical inference
MASS	Functions and datasets supporting 'Modern Applied Statistics with S'
MCMCpack	Markov Chain Monte Carlo (MCMC) Package
mvtnorm	Multivariate normal and <i>t</i> probabilities, quantiles and densities
prop.test	For testing the null hypothesis that binomial success probabilities in several groups are the same, or equal certain given values
proportion	Inference on a single binomial proportion (including Bayesian)
plot3D	Tools for plotting two- and three-dimensional data
rattle	Graphical user interface for data science in R
rpart	Recursive partitioning for classification, regression and survival trees
vcd	Visualizing categorical data
TeachingDemos	Functions that demonstrate statistical concepts and the programming
tidyverse	Collection of R packages for data science (<code>ggplot2</code> , <code>dplyr</code> , <code>tidyr</code> , <code>readr</code> , ...); see https://www.tidyverse.org/
visdat	Exploratory data visualization of an entire dataset for problems identification
xlsx	Reading/writing in excel format

A0.3 R Functions and Data Structures

Tasks are performed in R through functions, which are organized in packages. The simplest function is `c()`, for combine, used to create a vector as shown later in this section. R exhibits very good documentation of its functions and you can get help for an R function by the command `help`, which opens the associated entry of the httpd help server. For instance, information on and examples for the function `plot` used for plotting data are provided by

```
> help(plot) # equivalent to: ?plot
```

Data values are synthesized and appear in a wide variety of data structure forms, the most commonly used being *vectors*, *matrices*, *arrays*, *lists* and *data frames*. Data structures, functions and more complex structures synthesized by such components, are all known as *objects*. Actually all entities created and handled in R are objects. The names of all objects defined so far in your active workspace are listed on the display by typing `objects()` or `ls()`. Another useful information that provides compact information on the structure of an arbitrary R object is `str`.

The most common data value types treated in R are double (numeric, including `-Inf`: $-\infty$, `Inf`: ∞ and `NaN`: not a number), integer (defined by placing an L behind the number),

character (character string, given in ' ' or " ") and logical (TRUE (T) and FALSE (F)). Variables of all types can take the value NA (not available), which by default is logical.

The specific (R internal) type of any object is determined through the `typeof` function. It can be also identified by `class`, which is an attribute of an object and can be assigned by a user to an object, regardless of its internal storage mode. Furthermore, the functions `as.numeric`, `as.integer`, `as.logical` and `as.character` can be used to convert a variable to the stated type. For example consider:

```
> a <- TRUE; typeof(a)      # same output as with: class(a)
[1] "logical"
> as.numeric(a)             # same output as with: as.integer(a)
[1] 1
> b <- "color"; typeof(b)   # same output as with: class(b)
[1] "character"
> c <- 2L; typeof(c); str(c) # same output as with: class(c)
[1] "integer"
int 2
> d <- 2; typeof(d); class(d) # different output for typeof() and class()
[1] "double"
[1] "numeric"
> as.logical(d)
[1] TRUE
```

The data structures used in this book are briefly described below.

A0.3.1 Vectors and Lists

Vector is the most basic data structure and combines *values of the same type* in an one-dimensional vector, through the `c` function. The type of a vector can be verified through the `typeof` function, introduced earlier in this section. The variables used above to illustrate basic data types are all vectors of length one. Examples of vectors of various types follow.

```
> x <- c(4,5,6,7); x      # equivalent to: x <- 4:7
[1] 4 5 6 7
> x1 <- c(1,2.3,-3.5,exp(2)); x1 # numeric vector
[1] 1.000000 2.300000 -3.500000 7.389056
> typeof(x1)
[1] "double"
> x2 <- c(1L,2L,3L,-5L); x2   # integer vector
[1] 1 2 3 -5
> x3 <- c("A","B","C"); x3    # character vector
[1] "A" "B" "C"
> x4 <- c(TRUE,FALSE,F,T); x4 # logical vector
[1] TRUE FALSE FALSE TRUE
> x4.L <- as.integer(x4.L); x4.L
[1] 1 0 0 1
```

Specific components of a vector can be extracted as scalars or subvectors:

```
> x1[2]
[1] 2.3
> x1[2:4]
[1] 2.300000 -3.500000 7.389056
```

Some helpful functions in creating vectors are the functions `seq` and `rep`, standing for sequence and repeat, respectively:

```
> s1 <- seq(from=1, to=4, by=1); s1 # creates a double vector while
[1] 1 2 3 4 # 1:4 creates an integer vector
> s2 <- rep(1:4, 3); s2
```



```
[1] 1 2 3 4 1 2 3 4 1 2 3 4
> s3 <- rep(1:4, each=3); s3
[1] 1 1 1 2 2 2 3 3 3 4 4 4
```

It is straightforward to assign labels to the values of a vector, during its construction or later on using the **names** function:

```
> v1 <- c(a=1, 2:5, f=6); v1
a      f
1 2 3 4 5 6
> v2 <- 1:6; names(v2)=c("a","b","c","d","e","f"); v2
a b c d e f
1 2 3 4 5 6
```

Basic functions for handling vectors are **length**, **min**, **max**, **sum** and **prod**, providing for a vector its length, minimal value, maximal value, sum and product of its elements, respectively. Furthermore, **sort** sorts the elements of a vector in increasing order (or decreasing, specified by the argument **decreasing** = **TRUE**) and **rank** provides the rank values for its elements.

A vector with components of mixed types is not possible. In such a case, the resulting type is the more flexible one. Thus,

```
> x5 <- c(2.1, TRUE, "A","B","C"); x5
[1] "2.1" "TRUE" "A" "B" "C"
typeof(x5)
[1] "character"
```

A *list* is a flexible data structure that allows the combination of *data components of different value types*. Continuing the example above, we define a list with the elements of **x5** and verify that the constructed list has 5 components of different type.

```
> y1 <- list(2.1, TRUE, "A","B","C"); y1
[[1]]
[1] 2.1

[[2]]
[1] TRUE

[[3]]
[1] "A"

[[4]]
[1] "B"

[[5]]
[1] "C"
```

A list can also combine vectors or lists of different lengths. Note the difference of list **y1** to **y2** below.

```
> y2 <- list(2.1, TRUE, c("A","B","C")); y2
[[1]]
[1] 2.1

[[2]]
[1] TRUE

[[3]]
[1] "A" "B" "C"
```

Compare **y1[3]** to **y2[3]**:

```
> y1[3]
[[1]]
[1] "A"
> y2[3]
[[1]]
[1] "A" "B" "C"
```

A0.3.2 Factors

A *factor* is a special type of vector that corresponds to a nominal or ordinal categorical variable and has a relatively small number of pre-specified possible outcomes (numeric or character), known as *levels*. Factors are important in modeling with categorical data while the definition of factors is also required for the creation of some plots, as we shall see later on (e.g. in Section A1.2). The levels can be predefined or specified by the data, as illustrated in the example that follows. Consider the quality category (A, B or C) of a sample of 7 products in the case that none of the sampled products was of the lowest category C.

```
> q <- c("A","B","B","A","C","A","B","E","C","A","A","B"); q
[1] "A" "B" "B" "A" "C" "A" "B" "E" "C" "A" "A" "B"
> q_f <- factor(q); q_f                                     # levels specified by the data
[1] A B B A C A B E C A A B
Levels: A B C E
> q_fc <- factor(q,levels=c("A","B","C","D","E"))          # prespecified levels
> q_fc
[1] A B B A C A B E C A A B
Levels: A B C D E
> table(q_f)        # frequency table for variable quality
q_f
A B C E
5 4 2 1
> table(q_fc)       # frequency table for variable quality
q_fc
A B C D E
5 4 2 0 1
```

Equivalently to `q_fc` the quality factor can be defined through a numeric vector as follows.

```
> q_n <- c(1,2,2,1,3,1,2,5,3,1,1,2)
> q_n_fc <- factor(q_n, levels=c(1,2,3,4,5), labels=c("A","B","C","D","E"))
```

Note that the `forcats` package in `tidyverse` provides tools for simplifying certain tasks for factors, such as reordering the levels of a factor by some criterion (like frequency of another variable, useful for better visualization in graphs) or collapsing levels. For our example, we can easily collapse the categories C to E:

```
> q3 <- fct_collapse(q_fc, CDE=c("C","D","E")); q3
[1] A B B A CDE A B CDE CDE A A B
Levels: A B CDE
# alternatively, levels can be collapsed by duplicating labels:
> qf <- factor(q_fc, levels=c("A","B","C","D","E"),
+             labels=c("A","B","C","C","C"))
```

A0.3.3 Matrix and Array

Vectors and lists are both one-dimensional arrays of data. A multidimensional array is defined in R by the function `array`. For example, a vector of length 12 can be re-organized in a 3×4 or a $2 \times 3 \times 2$ table as illustrated below.

```

> z1 <- 1:12;
> z2 <- array(z1, c(3,4)); # c() in the argument specifies the dimension of the array
> z2
      [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> z3 <- array(z1, c(2,3,2)); z3
, , 1
      [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
, , 2
      [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12

```

Arrays of higher dimension are defined analogously by adjusting the dimension vector in the array argument accordingly.

A two-dimensional array can alternatively be defined by **matrix**. Thus, **z2** of the example above can alternatively be defined as follows.

```

> z2 <- matrix(z1, nrow=4, ncol=3)

```

By default, **matrix** expands the vector by columns. An expansion by rows is also possible.

```

> m2 <- matrix(z1, nrow=4, ncol=3, byrow=T); m2
      [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12

```

Alternatively, a matrix can be constructed by binding vectors of the same length row- or column-wise using the functions **rbind** or **cbind**:

```

> r1 <- 1:5; r2 <- 6:10; r12 <- rbind(r1,r2); r12
      [,1] [,2] [,3] [,4] [,5]
r1      1    2    3    4    5
r2      6    7    8    9   10
# cbind(r1,r2): the transpose of r12

```

Useful functions for working with matrices include **dim**, **nrow**, **ncol** and **t** for providing the dimension, number of rows, number of columns and the transpose of a matrix, respectively. Additionally, the sums and means for each row or column of a matrix are obtained applying functions **rowSums** and **rowMeans** or **colSums** and **colMeans**, respectively.

Computational tasks on vectors and matrices can be simplified and fastened in R, using the provided matrix algebra operations. For instance, two matrices of the same dimension can be multiplied component-wise using the ***** operator or the sum of squares of the components of a vector can easily be computed through matrix multiplication (**%*%**). Furthermore, it is handy for calculation that standard functions and operators for numerical variables apply component-wise to numerical vectors.

```

> a <- c(1,4,9) # computation of sum of squares
> sum_a2 <- t(a)%*%a ; sum_a2 # t: transpose of a matrix
      [,1]
[1,]    98
> sqrt(a)
[1] 1 2 3

```

A0.3.4 Data Frames

The most common form data are stored in R is that of a `data.frame` object. This is the data form required by many R functions as well as the form data are stored when read from an external source by the `readr` package (see Section A0.4).

A data frame stores the data in a two-dimensional format, where columns correspond to variables and rows to items (subjects) in the sample (see for example Figure 1.1 in Chapter 1 of the book. Variables of a data frame are reached using the `$` operator. Consider the following simplified example.

```
> v1 <- 1:5
> v2 <- v1^2
> v3 <- v2-2*v1>3
> df <- data.frame(v1,v2,v3)
> df
  v1 v2  v3
1  1  1 FALSE
2  2  4 FALSE
3  3  9 FALSE
4  4 16  TRUE
5  5 25  TRUE
> df$v1
[1] 1 2 3 4 5
> z<- df$v2
> z
[1] 1 4 9 16 25
> df[4,]
  v1 v2  v3
4  4 16 TRUE
> df[4,3]          # equivalent to df$v3[4]
[1] TRUE
```

The command `cbind` which combines vectors (or matrices) to a matrix, when applied on objects consisting of at least one data frame, yields a data frame; otherwise not.

```
> v4 <- 6:10
> df2 <- cbind(df,v4)  # df2 is a data frame
> df2
  v1 v2  v3 v4
1  1  1 FALSE 6
2  2  4 FALSE 7
3  3  9 FALSE 8
4  4 16  TRUE 9
5  5 25  TRUE 10
> v5 <- df$v1+v4
> v45 <- cbind(v4,v5)  #column bind: v45 is a matrix
%> v45
%      v4 v5
%[1,]  6  7
%[2,]  7  9
%[3,]  8 11
%[4,]  9 13
%[5,] 10 15
```

One can verify whether an object is a data frame or not as follows.

```
> is.data.frame(v45)
[1] FALSE
> is.data.frame(df2)
[1] TRUE
```

A0.3.5 User-Defined Functions

Users can create their own functions but cannot use a name already used by R. For example, you cannot use `mean` for a mean of some values, because an R function exists with this name (try `?mean`).

The construction of user-defined R functions is straightforward. In general, an R function has the following structure:

```
> nameOfFunction <- function(argument1, argument2, ... argumentq){
  # body of statements for calculating variable result
  return(result)
}
# Implementation (assigning variables or values to the arguments):
> nameOfFunction(argument1=v1, argument2=v2, ... argumentq=vq)
# or equivalently:
> nameOfFunction(v1,v2, ... vq)
```

Note that the return statement is not compulsory for constructing a function. If omitted, the value returned is the last executed command in the function. Furthermore, more than one variables can be returned, replacing `return` by the function `list`. Functions are frequently used in this Appendix (e.g., see Section A2.2).

A0.3.5.1 Example of user-defined functions: weighted mean

Consider we have three samples means \bar{x}_1 , \bar{x}_2 and \bar{x}_3 , corresponding to samples $(x_1^{(i)}, \dots, x_{n_i}^{(i)})$ of size n_i , $i = 1, 2, 3$, and we want to construct a function for evaluating the weighted sample mean $\bar{x} = \frac{n_1\bar{x}_1 + n_2\bar{x}_2 + n_3\bar{x}_3}{n_1 + n_2 + n_3}$. A first trial could be:

```
> wght.mean3 <- function(mean1, mean2, mean3, n1, n2, n3){
  mean.all <- (n1*mean1+n2*mean2+n3*mean3)/(n1+n2+n3)
  return(mean.all)
}
# Implementation example (mean1=12, mean2=15, mean3=21, n1=20, n2=30, n3=10):
> wght.mean3(12,15,21,20,30,10)
[1] 15
```

A more elegant and practical way to programize the above problem is through vectorization, which is compacter and simultaneously allows the calculation of the weighted mean of an arbitrary number of samples:

```
> wght.mean <- function(mean.vector, n.vector){
  # mean.vector: the vector of sample means
  # n.vector: a vector of the same length, containing the corresponding
  #           sample sizes
  mean.all <- sum(mean.vector*n.vector)/sum(n.vector)
  return(mean.all)
}
# Implementation example:
> mean.v <- c(12,15,21); n.v <- c(20,30,10)
> wght.mean(mean.v, n.v) # equivalently: wght.mean(c(12,15,21), c(20,30,10))
[1] 15
```

Notice that there exists a corresponding function `weighted.mean` in R-package `stat`. You may also verify that the arguments of this function are the vector of values whose weighted mean is to be computed and the vector of corresponding weights (adding to one). Hence the weighted mean of the example above can alternatively be computed as follows:

```
> wght <- n.v/sum(n.v); weighted.mean(mean.v, wght)
[1] 15
```

A0.4 Data Input in R

Data files are considered to be in a tabular format, with rows corresponding to cases and columns to variables measured. Thus, if we have a data set of sample size n and there are measured k variables on the items of our sample, then our data form a $n \times k$ table and the value in the (i, j) -th cell is the value observed for the i -th item ($i = 1, \dots, n$) on the j -th variable ($j = 1, \dots, k$). This is the most informative form, the form required for statistical procedures to apply and the form required by all statistical software. It corresponds to the form of data in a `data.frame`. However, in the era of data science and information society, data are recorded and reported in many different ways, depending on the source and the objective of the associated presentation. These forms of data representation can be messy and are not appropriate for further statistical analysis and modeling. For this, before proceeding with the data analysis, data have to be brought in the table format described above. Data of this axiomatic format (in a statistical framework), from which all possible representations can be derived, has a special name in the data science terminology, they are called *tidy data*.

For learning and illustrative purposes, many data sets are available in R- packages in form of data frames. They can be attached in our workspace through the function `data` while further useful functions for getting an impression of or viewing the data are `names`, `dim`, `str`, `print` and `View`, the last opening the data in an R spreadsheet. We illustrate on the Iris flower data set of Fisher (1936). It consists of 150 samples of Iris, 50 from each of three species (Iris setosa, Iris virginica and Iris versicolor). For each sample, the width and length of the sepals and petals is reported (in centimeters).

```
> data(iris)
> names(iris)
[1] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" "Species"
> dim(iris)
# dimension of a matrix, array or data frame
[1] 150 5
> str(iris)
# compact display of the structure of an R object
'data.frame': 150 obs. of 5 variables:
 $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 ...
> print(iris[4:6,]) # prints cases 4-6 of the data file
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
4           4.6           3.1           1.5           0.2  setosa
5           5.0           3.6           1.4           0.2  setosa
6           5.4           3.9           1.7           0.4  setosa
```

Simple data can be imported in R manually from the keyboard, using for example the `c` or `scan` function. Thus, the values of a vector `y` can be read as follows:

```
> y <- scan()
1: 2 # start typing the data
2: 5 # one vector element is provided per line
3: 3
4: # enter a blank line to signal the end of data reading
Read 3 items
```

Alternatively, an R spreadsheet can be used to type in the data in form of a list, which is activated by the `edit` function, as shown below:

```
> toy_example <- data.frame(salary=numeric(0), gender=character(0))
> toy_example <- edit(toy_example)
```

Most importantly, R provides functions for importing and exporting data, supporting many data files formats (table formatted data in plain-text files or data files from excel, SAS, SPSS, Stata and Systat).

For plain-text data files, the basic function is `read.table` and `write.table` for importing and exporting data, respectively. For these functions the default separator is ‘white space’ (i.e. one or more spaces, tabs or newlines). Most commonly, data files have comma separated values (csv). Such forms are handled by functions `read.csv` and `write.csv` while `read.csv2` and `write.csv2` are for semicolon separated data. The first argument of these functions is the file name of the data to be read (or written), given by a full path or the relative path of the current working directory. For the latter, the working directory can be get or set by functions `getwd` or `setwd`, respectively. They further have a variety of arguments that allow additional specifications and handling options for the data format. For example, the logical argument `header` can be used to read/write data sets with (`=TRUE`) or without (`=FALSE`) header. By default, character string variables are converted to factor variables. You may change this default setting using the arguments `stringsAsFactors` or `as.is`. For reading contingency tables, there exist features that enable the reading of labels for the categories of the classification variables from the source file or the assignment labels when they are not provided in the source file. For further details you may consult the [RDocumentation](#).

Consider for example the file ‘drugs.dat’ that is allocated in the local folder ‘DS.Data’ and contains the data of a survey on the use of drugs at high schools having values separated by comma. This data set corresponds to a $2 \times 2 \times 2$ contingency table, formulated by cross-classifying 2276 high school students according to whether they consume alcohol (a), cigarettes (c) or marijuana (m). It can be read in a `data.frame` format as follows:

```
> setwd("C:/Users/.../DS.Data")      # provide the full path of the folder
> drugs <- read.csv("drugs.dat", header=TRUE)
> drugs
  a c m count
  a  c  m count
1 yes yes yes   911
2 yes yes no   538
3 yes no yes    44
4 yes no no   456
5 no yes yes     3
6 no yes no    43
7 no no yes     2
8 no no no   279
>typeof(drugs$a)      # character string variable a is converted to a factor
[1] "integer"
```

To read a data file from a website, we simply need to provide the full website path, as illustrated for example in Section 1.4.1 of the book for the carbon dioxide emissions data set, which was read by `read.table`, since data values are separated in file ‘Carbon.dat’ by white spaces.

As it is extensively commented in the ‘R Data Import/Export’ manual, these functions are not to be used for reading large data files (use a lot of memory and are slow). Thus, when reading large data matrices (having many columns) it is preferable to use `scan` instead of `read.table`.

An alternative option for data importing is the `readr` function, which is included in the collection of R-packages `tidyverse`. The functions `read_tsv`, `read_csv`, and `read_csv2` are analogous to the basic R functions `read.table`, `read.csv` and `read.csv2`, respectively (the argument `header` is now replaced by the argument `col.names`). It is claimed that for large data sets they are typically much faster (up to 10 times). Further functions are available, as for example the general `read_delim`, that reads data values delimited with any separator.

These functions, in contrary to the basic ones, do not convert character vectors to factors. They also provide additional handy options, as for example the possibility to skip n lines of meta data at the beginning of the file (argument `skip=n`) or to drop lines with comments, signalized by the specific character they start with, e.g. `#` (argument `comment="#"`) The data file in Section 1.4.1 could equivalently be read by:

```
> library(tidyverse)
-- Attaching packages ----- tidyverse 1.3.0 --
v ggplot2 3.3.2    v purrr  0.3.4
v tibble  3.0.4    v dplyr  1.0.2
v tidyr   1.1.2    v stringr 1.4.0
v readr   1.4.0    v forcats 0.5.0
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()     masks stats::lag()
> Carbon <- read_tsv("http://stat4ds.rwth-aachen.de/data/Carbon.dat")

-- Column specification -----
cols(
  `Nation`      C02` = col_character()
)
```

There are special packages available that facilitate data exchange between R and other statistical software. For example, the `foreign` package can be used to import data from (or export to) a variety of sources, including Minitab, SAS, SPSS, Stata and Systat. Excel data can be read employing the `xlsx` package. A functionality similar to `foreign` offers the `haven` package, which provides the functions `read_sas`, `read_sav` and `read_dta` for reading data files in SAS, SPSS and Stata file formats, respectively, and can be faster than `foreign`. Finally, `readxl` is an analogous to `xlsx` package for importing data from excel while data frames can be exported to excel files by the `writexl` package. Packages `haven`, `readxl` and `writexl` are among the packages installed automatically with `tidyverse`.

A0.5 Control Flows

For programming, it is important to know the basic control flow structures of R. These are the if and if-else statements, the for, while and repeat loops, along with the break and next statements. Some illustrative examples follow:

```
> x <- 2
> if (x>0) {y <- 2*x ; z <- 1} else {y <- -3*x+1 ; z <- 0}; y; z
[1] 4
[1] 1
> f <- numeric(10); f[1] <- 1; f[2] <- 2
> for (i in 3:10) {f[i] <- f[i-1]+2*f[i-2]}; f
[1] 1 2 4 8 16 32 64 128 256 512
> g1 <- 1; g2 <- 2; g <- g1
> while(g2<520){g <- c(g, g2); g2.old <- g2      # equivalent to the 'for' loop above
               g2 <- g2+2*g1; g1 <- g2.old}

> g
[1] 1 2 4 8 16 32 64 128 256 512
> h <- numeric(10); h[1] <- 1; h[2] <- 2; i <- 0
> repeat{i <- i+1                                # equiv. to the 'for'/'while' loops above
  if (i < 3) next                                # if the condition holds all commands in this loop are
    h[i] <- h[i-1]+2*h[i-2]                      # skipped and moves to the next loop
  if (i>9) break }; h
[1] 1 2 4 8 16 32 64 128 256 512
```


For expressing conditions, the usual logical operators employed are `<` (less), `<=` (less or equal), `>` (greater), `>=` (greater or equal), `==` (equal), `!=` (not equal) and the Boolean operators `|` (OR), `&` (AND), `xor` (elementwise exclusive OR)).

A0.6 Graphs in R

For data visualization and presentation of statistical analysis output, for example some diagnostic plots, R provides powerful graphical tools.

The most common plotting function in R is the `plot` function, which may produce a scatterplot, a time series plot, a bar plot or a box plot, depending on its arguments. Other types of plots include qq-plots (`qqnorm`, `qqplot`), histograms (`hist`) and contour plots (`contour`). For multivariate data, the `pairs` function is used for producing a matrix of pairwise plots while `coplot(a~b|c)`, where `c` is a factor, produces a number of (`a`, `b`) plots for every level of `c`.

The `ggplot2` package in `tidyverse` offers many options for elegant and very flexible graphics that can be tailored to meet user's expectations and advanced demands for visualizing complex data sets. It is based on the *grammar of graphics* and the idea that the construction of a graph is based on the same core components (data set, set of geoms and a coordinate system). Thus, a graph is gradually built, starting with `ggplot` (or `qplot`), and specifying (i) the data set, (ii) aesthetic mappings (by `aes`), and (iii) the type of visual representation of the data (geom). Then further levels with a `geom_*` or `stat_*` function (e.g. `geom_histogram`) can be gradually added (after the `+` sign) for setting further specifications as well as controlling coordinate systems and faceting. A very helpful overview of the logic, options and features of `ggplot` is provided in the [RStudio Cheat Sheet](#).

We shall introduce and explore some of the options of R graphical packages in this appendix, motivated by specific examples.



CHAPTER 1: R FOR DESCRIPTIVE STATISTICS

A1.1 Data Handling and Wrangling

Data sets can be messy and usually data from real applications are not captured in the desired tabular form described in Section A0.4. Furthermore, depending on the features of the data we want to demonstrate or model, the same data set may be expressed in different formats. Thus, especially in situations of complex data structures, data reading and handling can become hard. The [tidyverse](#) is a collection of R packages designed for the special needs on data handling and manipulation appearing in data science. Among the packages included in `tidyverse`, we have already referred to `readr`, for reading data (Section A0.4), to `forcats`, for defining and treating factors (Section A0.3.2) and to `ggplot2` for creating graphics (Section A0.6). Here, we shall highlight some powerful aspects of `tidyr` and `dplyr` for organizing the data in a tidy form and manipulating the data. Data tidying and manipulating is known as *data wrangling*. A nice overview of the data wrangling options offered by `tidyr` and `dplyr` is provided in the [data wrangling sheet](#) of the RStudio. For a detailed insight in `tidyverse`, we refer to Baumer et al. (2017) and Wickham and Grolemund (2017), the latter also provided in a free online version ([R for Data Science](#)).

We assume that our data are *tidy*, i.e. in a tabular form where each column corresponds to a characteristic (variable) and each row to a case of the sample (see Section A0.4). For this, we do not discuss the `tidyr` package, which is designed for transforming data to a tidy form. The most basic functions of `tidyr` are the `spread` and `gather` functions, which serve for reshaping the data set to a tidy form. Furthermore, `separate` and `unite` are used for combining or splitting cells of the data table so that finally each cell consists of a single observation.

Options and features of the `tidyverse` package will be illustrated along this Appendix. Basic functions of `dplyr` that apply on tidy data sets include those provided in Table A1.1.

TABLE A1.1: Basic functions of `dplyr`.

<code>summarize</code>	provides user specified descriptive statistics for one or more variables
<code>group_by</code>	groups cases (data in rows) having the same value of a specific variable
<code>arrange</code>	sorts the full data set according to the ordering of specified variables
<code>slice</code>	selects a subset of rows (cases) by position
<code>filter</code>	selects a subset of rows that fulfill a condition
<code>select</code>	selects a subset of columns (variables) based on a condition
<code>mutate</code>	computes and appends new variables (columns)

Important in applying `dplyr` is the “pipe” operator `%>%`, which passes the object on the left hand side as an argument of the function in the right hand side, making thus the

code more handy. In particular, `x %>% f(y)` is the same as `f(x,y)` and `y %>% f(x,.,z)` the same as `f(x,y,z)`.

All actions carried out in `dplyr` can be also obtained in standard R. The advantage is that `dplyr` provides a more handy way, using verbs of common sense that make the code more user friendly (but in some cases not necessarily shorter). Next, we provide some data handling examples, illustrated on the very famous iris data frame.

A1.1.1 summarize

The `summarize` function provides summary statistics for a data frame, providing a single value for the columns (variables) asked. Some characteristic examples follow. The functions used (e.g. `mean`, `sd`) are exchangeable. Every function is eligible as long as its output is a single value.

```
> library(dplyr)
> data(iris)
> names(iris)
[1] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" "Species"
> iris %>% summarise(mean=mean(Sepal.Length)) # equivalent to:
      mean                                     # mean(iris$Sepal.Length)
1 5.843333
> iris %>% summarize(sample_size=n(), mean_SL=mean(Sepal.Length), mean_SW=
  mean(Sepal.Width), mean_PL=mean(Petal.Length), mean_PW=mean(Petal.Width))
  sample_size mean_SL mean_SW mean_PL mean_PW
1          150  5.843333  3.057333   3.758  1.199333
> iris %>% summarize(mean_SL=mean(Sepal.Length), sd_SL=sd(Sepal.Length))
  mean_SL      sd_SL
1 5.843333 0.8280661
```

A1.1.2 group_by

The `group_by` function is a convenient function that groups the sample units (i.e. rows of a tidy data set) according to their values on a variable (column) of the data set. Most data operators applied on grouped data, are then performed group-wise. This is very convenient as a first step, since in most analyses we want to compare responses, profiles, etc. among groups. Grouping is removed by `ungroup`.

Frequently it is applied in combination with `summarize`, to provide summary statistics per group.

```
> iris_spec <- iris %>% group_by(Species)
> iris_spec %>% summarize(n=n(), mean_SL=mean(Sepal.Length), mean_SW=
  mean(Sepal.Width), mean_PL=mean(Petal.Length), mean_PW=mean(Petal.Width))
  Species      n mean_SL mean_SW mean_PL mean_PW
<fct>    <int>    <dbl>    <dbl>    <dbl>    <dbl>
1 setosa      50     5.01     3.43     1.46     0.246
2 versicolor  50     5.94     2.77     4.26     1.33
3 virginica   50     6.59     2.97     5.55     2.03
```

The `%>%` operator is very convenient for wrangling data, since it allows the successive use of nested function in one step. Thus, for deriving the within groups means above we do not need to create a new data frame (`iris_spec`). It is equivalent to:

```
> iris %>% group_by(Species) %>% summarize(n=n(), mean_SL=mean(Sepal.Length),
  mean_SW=mean(Sepal.Width), mean_PL=mean(Petal.Length),
  mean_PW=mean(Petal.Width))
```

A1.1.3 arrange

The `arrange` function rearranges the cases in a data frame in increasing order of a specified variable, while decreasing order is also possible. In case of ties, cases can be ordered according to other specified variables. There is also the option to sort within groups (specified by a grouping variable). Do not run this function applied to a data frame, since it will print on screen the rearranged data set; save the rearranged data frame, instead.

```
> iris.ord <- iris%>%arrange(Sepal.Length)      # Sepal.Length: increasing
> iris.ord[1:3,] # you could also try: head(iris.ord)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          4.3         3.0          1.1          0.1  setosa
2          4.4         2.9          1.4          0.2  setosa
3          4.4         3.0          1.3          0.2  setosa
# If ties, items are arranged by increasing order of Petal.Length, ... :
> iris.ord2 <- iris%>%arrange(Sepal.Length, Petal.Length, Sepal.Width)
> iris.ord2[1:3,]
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          4.3         3.0          1.1          0.1  setosa
2          4.4         3.0          1.3          0.2  setosa
3          4.4         3.2          1.3          0.2  setosa
> iris.ord3 <- iris%>%arrange(desc(Sepal.Length)) # Sepal.Length: decreasing
> iris1 <- iris_spec %>% arrange(desc(Sepal.Length), .by_group = TRUE)
> iris1[97:102,]
# A tibble: 6 x 5
# Groups:   Species [2]
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
    <dbl>         <dbl>         <dbl>         <dbl> <fct>
1         5.1         2.5           3           1.1 versicolor
2         5          2           3.5           1  versicolor
3         5          2.3           3.3           1  versicolor
4         4.9         2.4           3.3           1  versicolor
5         7.9         3.8           6.4           2  virginica
6         7.7         3.8           6.7           2.2 virginica
```

A1.1.4 slice and filter

These functions are used for selecting cases (rows) of a data set, either by the position of the rows (`slice`) or by one or more conditions they have to fulfill (`filter`). See in the examples below the corresponding handling in basic R and note that in basic R the initial positions of the selected rows are kept while this is not the case for the functions of `dplyr`.

```
> slice(iris,1L) # compare to: iris[1,]
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1         5.1         3.5          1.4          0.2  setosa
> slice(iris,n()) # compare to: iris[nrow(iris),]
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1         5.9           3          5.1          1.8  virginica
> slice(iris, 5:8) # compare to: iris[5:8,]
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1         5.0         3.6          1.4          0.2  setosa
2         5.4         3.9          1.7          0.4  setosa
3         4.6         3.4          1.4          0.3  setosa
4         5.0         3.4          1.5          0.2  setosa

> iris%>%filter(Petal.Length==6) # compare to: iris[iris$Petal.Length==6,]
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1         6.3         3.3           6           2.5  virginica
2         7.2         3.2           6           1.8  virginica
> iris%>%filter(Sepal.Length >=6, Sepal.Width < 2.5)
```

```

      Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
1          6.0         2.2         4.0         1.0 versicolor
2          6.2         2.2         4.5         1.5 versicolor
3          6.3         2.3         4.4         1.3 versicolor
4          6.0         2.2         5.0         1.5 virginica
# can equivalently be written as:
> filter(iris, Sepal.Length >=6, Sepal.Width < 2.5)

# compare to:
> iris[iris$Sepal.Length >=6 & iris$Sepal.Width < 2.5, ]
      Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
63          6.0         2.2         4.0         1.0 versicolor
69          6.2         2.2         4.5         1.5 versicolor
88          6.3         2.3         4.4         1.3 versicolor
120         6.0         2.2         5.0         1.5 virginica

```

A1.1.5 select

This functions returns a subset of the columns (variables) of a data frame, as demonstrated below.

```

> iris2 <- select(iris, Sepal.Length Sepal.Width) # equivalent to: iris[,1:2]
> slice(iris2, 2:3)
      Sepal.Length Sepal.Width
1          4.9         3.0
2          4.7         3.2
> iris %>%filter(Sepal.Length==7)%>% select(Sepal.Width)
      Sepal.Width
1          3.2
# equivalent to: iris[iris$Sepal.Length==7,2]

```

A1.1.6 mutate

The `mutate` function creates new variables and adds them as columns to a data frame.

```

iris%>%mutate(above.meanSL=Sepal.Length>mean(Sepal.Length))%>%slice(.,107:109)
      Sepal.Length Sepal.Width Petal.Length Petal.Width   Species above.meanSL
1          4.9         2.5         4.5         1.7 virginica      FALSE
2          7.3         2.9         6.3         1.8 virginica      TRUE
3          6.7         2.5         5.8         1.8 virginica      TRUE
> iris.SLW <- iris%>%mutate(SLW=Sepal.Length/Sepal.Width)
# equivalent to: cbind(iris,iris$Sepal.Length/iris$Sepal.Width)
> iris.SLW[1,]
      Sepal.Length Sepal.Width Petal.Length Petal.Width   Species      SLW
1          5.1         3.5         1.4         0.2 setosa 1.457143

```

A1.2 Histograms and Other Graphics

For some general information on the graphical tools of R, we refer to Section [A0.6](#). This section focuses on creating a histogram for a frequency distribution. With equal length classes (bins) for the values, various rules can determine the number of classes and their width. To graph variable y with k bins and bin width h , the rules specify k or h in $k = \lceil [\max(y) - \min(y)]/h \rceil$ and let k increase and h decrease as n increases. The default rule, called

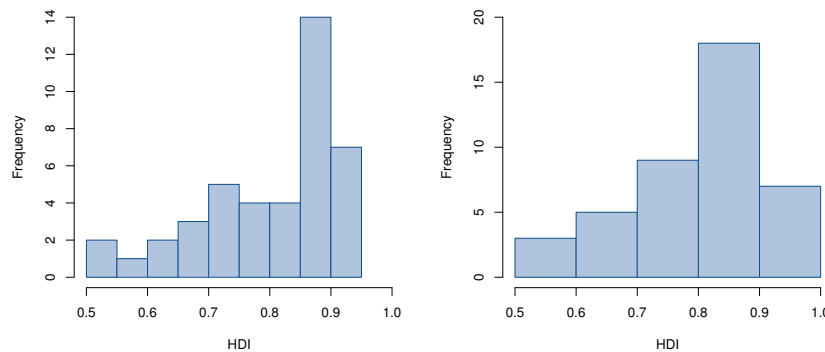


FIGURE A1.1: Histograms for the variable HDI in UN data file with number of classes being specified by the rule of Sturges (left) and Scott (right).

Sturges, takes $k = \log_2(n) + 1$. Its derivation assumes a bell-shaped distribution, and outliers can be problematic. The *Scott* rule takes $h = 3.49s/n^{1/3}$, for sample standard deviation s . For large n , Sturges' rule yields wider bins than Scott's and may oversmooth the histogram.

We explore and illustrate the R facilities and options for histograms using the UN data file from the book's website, consisting of nine variables measured over 42 countries, described in Exercise 1.24:

```
> UN <- read.table("http://stat4ds.rwth-aachen.de/data/UN.dat", header=T)
# use header=T (or header=TRUE) when file has variable names at top
> names(UN) # provides the names of the variables
[1] "Nation" "GDP" "HDI" "GII" "Fertility" "CO2"
[7] "Homicide" "Prison" "Internet"
```

We illustrate histograms with the human development index (HDI), a summary measure with components referring to life expectancy at birth, educational attainment, and income per capita:

```
> hist(UN[,3], xlab="HDI", xlim=c(0.5,1), main=NULL) # breaks="Sturges"
> hist(UN[,3], breaks="Scott", xlab="HDI", ylim=c(0,20), main=NULL)
```

The derived histograms, with equal length classes and for the two considered choices for the determination of the number of classes, are provided in Figure A1.1.

A1.2.1 Histograms with Bins of Unequal Size

Usually histograms have bins of equal width. However, sometimes the classes are prespecified and for some variables (like age or income) these are not of equal length. Then histograms are possible but special caution is needed. We illustrate this issue by constructing such a histogram for the age of the GSS2018 survey participants (GSS2018 data file from the book's website):

```
GSS <- read.table("http://stat4ds.rwth-aachen.de/data/GSS2018.dat", header=T)
```

The histogram of age for prespecified classes of age, provided in Figure A1.2 (left), can easily be produced using a function from the *ggplot2* package.

```
> library(ggplot2)
> ggplot(GSS, aes(x=AGE)) + geom_histogram(breaks=
  c(18,25,35,50,65,75,90), color="dodgerblue4", fill="lightsteelblue")
```

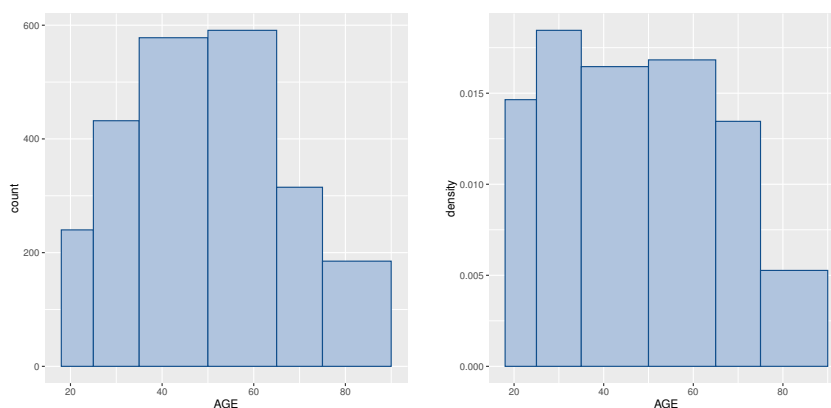


FIGURE A1.2: Histogram for the age of the GSS2018–participants for bins of unequal width (left: incorrect, right: correct).

The default values for the y-axis are the counts of the classes. However, in case the classes are of unequal width (as here), this histogram is incorrect, since the surface of each bin shall be proportional to the relative frequency of the corresponding class. In such cases, the right scale for the y-axis is the density and the right histogram is in Figure A1.2 (right).

```
> ggplot(GSS, aes(x=AGE)) + geom_histogram(breaks=c(18,25,35,50,65,75,90),
+     aes(y = ..density..), color="dodgerblue4", fill="lightsteelblue")
```

A1.3 Descriptive Statistics

We next present R facilities for descriptive statistics. For the UN data file, we form some summary statistics and boxplots:

```
# summary statistics for the 2nd up to the 9th variables in UN:
> summary(UN[,2:9])
      GDP      HDI      GII      Fertility
Min.   : 4.40   Min.   :0.5000   Min.   :0.0300   Min.   :1.200
1st Qu.:13.18   1st Qu.:0.7400   1st Qu.:0.0850   1st Qu.:1.700
Median :27.45   Median :0.8600   Median :0.1850   Median :1.900
Mean   :26.83   Mean   :0.8045   Mean   :0.2414   Mean   :2.038
3rd Qu.:40.33   3rd Qu.:0.8975   3rd Qu.:0.3875   3rd Qu.:2.200
Max.   :62.90   Max.   :0.9400   Max.   :0.5600   Max.   :6.000

      CO2      Homicide      Prison      Internet
Min.   : 0.500   Min.   : 0.300   Min.   : 30.0   Min.   :11.00
1st Qu.: 4.025   1st Qu.: 0.900   1st Qu.: 82.0   1st Qu.:46.00
Median : 6.900   Median : 1.550   Median :119.5   Median :67.00
Mean   : 6.695   Mean   : 4.257   Mean   :153.9   Mean   :63.86
3rd Qu.: 8.975   3rd Qu.: 3.650   3rd Qu.:188.8   3rd Qu.:84.00
Max.   :17.000   Max.   :30.900   Max.   :716.0   Max.   :95.00

> boxplot(UN[,2], xlab="GDP",main="", horizontal=TRUE) # 2nd variable is GDP
> points(mean(UN[,2]),1, col = "blue", pch = 18)      # adds mean to box plot
> arrows(mean(UN[,2])-sd(UN[,2]),1,mean(UN[,2])+sd(UN[,2]),1,
+     code = 3, col = "blue", angle = 75, length = .1) # shows mean +- std dev
```


On the box plot, provided in Figure A1.3 (upper left), we added the sample mean and the interval ‘sample mean \pm standard deviation’, as well. Replacing UN[,2] with UN[,3] to UN[,9], the rest of the box plots in Figure A1.3 are produced.

A1.3.1 Trimmed Mean

All functions, beyond their standard usage modus, provide further options controlled via additional arguments. Consider for example the function `mean` applied in Section 1.4.5. This function provides also the trimmed mean¹, of a sample vector (see Exercise 4.21).

For the data vector $y = (y_1, \dots, y_{10}) = (11, 7, 12, 10, 21, 2, 1, 12, 25, 14)$ with sample mean $\bar{y} = \frac{1}{10} \sum_{i=1}^{10} y_i = 11.5$, the trimmed mean excluding (symmetrically) the 10% of lowest and the 10% highest-valued observations is $\bar{y}_{0.8} = \frac{1}{8} \sum_{i=2}^9 y_{(i)} = 11$, where $y_{(i)}$ denotes the i -th ordered observation, e.g. $y_{(1)} \leq y_{(2)} \leq \dots \leq y_{(10)}$. This calculation can easily be implemented in R:

```
> y <- c(11,6,12,9,68,15,5,12,23,14)
> mean(y)      # mean of elements in vector x
> median(y)    # median of elements in vector x
> mean(y,0.1)  # trimmed mean (mean of 80% ‘central’ observations)
```

The last command is equivalent to:

```
> y <- c(11,6,12,9,15,12,23,14); mean(y) # manually exclude values 68, 5
```

A1.3.2 Summarizing Categorical Data

Caution is needed when treating categorical data. In finding descriptive summaries, we identify a categorical variable as a *factor*, so that the `summary` function provides its frequency distribution rather than statistics for quantitative variables, which are inappropriate. We illustrate with the `Income` data file, which lists annual income (in thousands of dollars) of 80 subjects classified by three categories of racial-ethnic status (black, Hispanic, white).²

```
> Inc <- read.table("http://stat4ds.rwth-aachen.de/data/Income.dat", header=TRUE)
> head(Inc, 3)      # shows first 3 lines of data file
  income education  race
1    16         10    B
2    18          7    B
3    26          9    B
> print(Inc[3:5,]) # prints observations 3-5 of the data file
  income education  race
3    26          9    B
4    16         11    B
5    34         14    B
> Inc$race <- factor(Inc$race)
> summary(Inc) # shows frequency distribution for factors
      income      education      race
Min.   : 16.00   Min.   : 7.00   B:16
1st Qu.: 22.00   1st Qu.:10.00   H:14
Median : 30.00   Median :12.00   W:50
Mean   : 37.52   Mean   :12.69
```

¹ The mean value of a data vector is influenced strongly by *outliers*, i.e. single observations that are much lower or higher than the main body of the data. A more robust description of the central location of the data is provided by their median (see Section 1.4.3). The $p\%$ trimmed mean, is a corrected mean that excludes the $p\%$ lowest and $p\%$ highest data points before computing the mean.

²The data are based on a much larger sample taken by the U.S. Bureau of the Census and are analyzed in Sections 6.5.2 and 6.5.5.

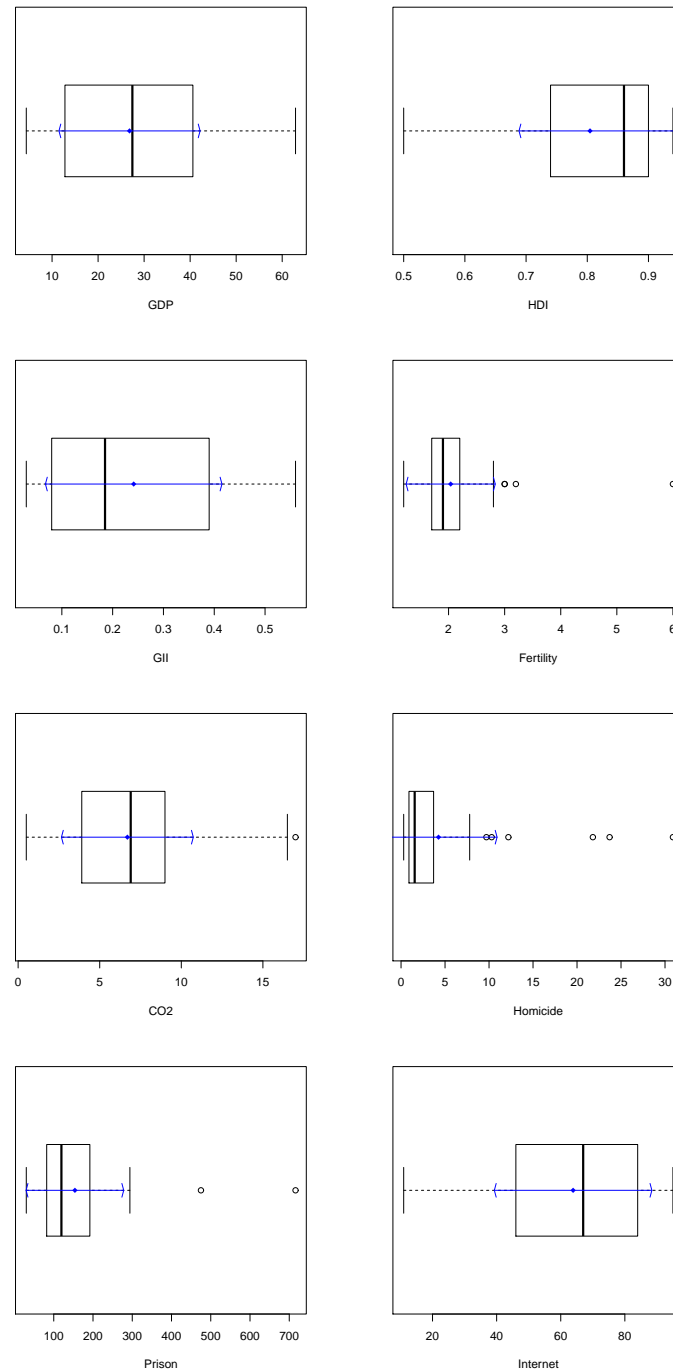


FIGURE A1.3: Box plots for the variables values of the 41 nations with their corresponding sample mean \pm standard deviation (in blue).

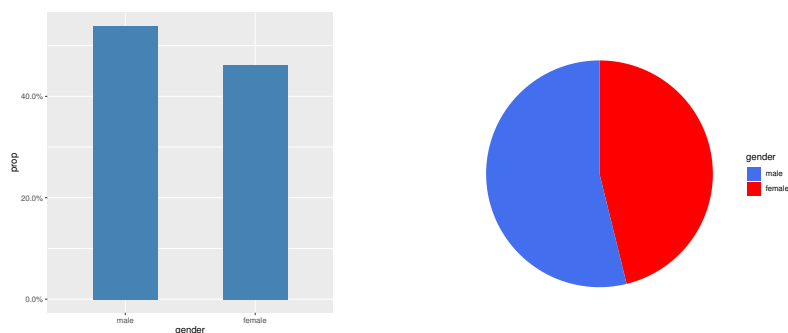


FIGURE A1.4: Barplot and pie chart for the gender of the persons in the income data.

```
3rd Qu.: 46.50    3rd Qu.:15.00
Max.    :120.00    Max.    :20.00
```

Categorical variables can be visualized by a barplot or pie chart, as illustrated next with the employees' gender of a certain sector of a company (`Salaries` data file³ from the book's website). The derived graphs are derived in `ggplot2` and given in Figure A1.4.

```
> library(ggplot2)
> library(scales)
> salaries <- read.table("http://stat4ds.rwth-aachen.de/data/Salaries.dat", header=T)
> salaries$gender <- factor(salaries$gender, levels=c(1,2),
                           labels=c("male", "female"))

# Bar plot of counts:                                (not shown)
> ggplot(data=salaries, aes(x=gender)) +
  geom_bar(width=0.5, fill="steelblue")
# Bar plot of proportions:                            Figure A1.4 (left)
> ggplot(data=salaries, aes(gender)) +
  geom_bar(aes(y=.prop., group = 1), width=0.5, fill="steelblue") +
  scale_y_continuous(labels=percent_format())

# Pie chart based on a factor:                        Figure A1.4 (right)
> pie <- ggplot(data=salaries, aes(x=factor(1), fill=gender)) +
  geom_bar(width = 1) + coord_polar("y")
> pie + scale_fill_manual(values=c("royalblue2", "red")) +
  theme_void() # remove background, grid, numeric labels
```

Notice that for calculating the right descriptive statistics for categorical data in R, it is important to treat categorical variables as factors. Hence, since gender in `Salaries` data file is a numeric variable, the corresponding frequency distribution is derived by:

```
> summary(as.factor(income$gender)) # compare to summary(income$gender)
 1  2
14 12
```

In `dplyr`, a frequency distribution is straight forwardly obtained by the `count` function, as shown next for the vote in the presidential elections 2016 (`PRES16`) of the `GSS2018` data set.

```
> GSS$PRES16 <- factor(GSS$PRES16, levels=c(1:4),
                      labels = c("Clinton", "Trump", "Other", "Not vote"))
```

³Consists of three variables, providing the monthly salaries (in Euro), the seniority (in years) and the gender for the $n = 26$ employees of a certain sector of a company.

```

> GSS %>% count(PRES16) # NA not recognized as missing! (output hidden)
> GSS$PRES16 <- fct_explicit_na(GSS$PRES16)
> GSS %>% count(PRES16)
# A tibble: 5 x 2
  PRES16      n
  <fct>    <int>
1 Clinton    764
2 Trump      577
3 Other       87
4 Not vote    20
5 (Missing)  900

```

A1.3.3 Group-Specific Descriptive Statistics

Often it is of interest to show descriptive statistics by groups, identified by a factor. Consider for the `salaries` data file that the company claims that the reported salaries depend only on the seniority of the employees and men are not better paid than women. We can easily obtain in `dplyr` the mean and standard deviation of the salary for men and women, which do not indicate a gender bias in favor of men:

```

> library(dplyr)
> salaries %>% group_by(gender) %>% summarize(mean=mean(salary), sd=sd(salary))
# A tibble: 2 x 3
  gender mean    sd
  <int> <dbl> <dbl>
1     1 3231.  721.
2     2 3394.  603.

```

In base R, group specific statistics are possible through the `tapply` function (as illustrated in Section 1.4.5 with the `Murder2` data file) and the `aggregate` function. Thus the above results can equivalently be derived as follows:

```

> aggregate(salaries$salary ~ income$gender, data =income, mean)
> aggregate(salaries$salary ~ income$gender, data =income, sd)

```

The `purrr` library facilitates a straightforward way of obtaining descriptive statistics for all variables in a data set and all levels of a categorical covariate:

```

> library(purrr)
> salaries %>% split(.$gender) %>% map(summary)
$'1'
  salary      years      gender
Min.   :1965   Min.   : 8.00   Min.    :1
1st Qu.:2720   1st Qu.:13.75   1st Qu.:1
Median :3172   Median :18.50   Median :1
Mean   :3231   Mean   :19.21   Mean    :1
3rd Qu.:3902   3rd Qu.:25.00   3rd Qu.:1
Max.   :4387   Max.   :32.00   Max.    :1

$'2'
  salary      years      gender
Min.   :2280   Min.   :11.00   Min.    :2
1st Qu.:3044   1st Qu.:19.00   1st Qu.:2
Median :3424   Median :22.50   Median :2
Mean   :3394   Mean   :22.00   Mean    :2
3rd Qu.:3719   3rd Qu.:24.75   3rd Qu.:2
Max.   :4438   Max.   :33.00   Max.    :2

```

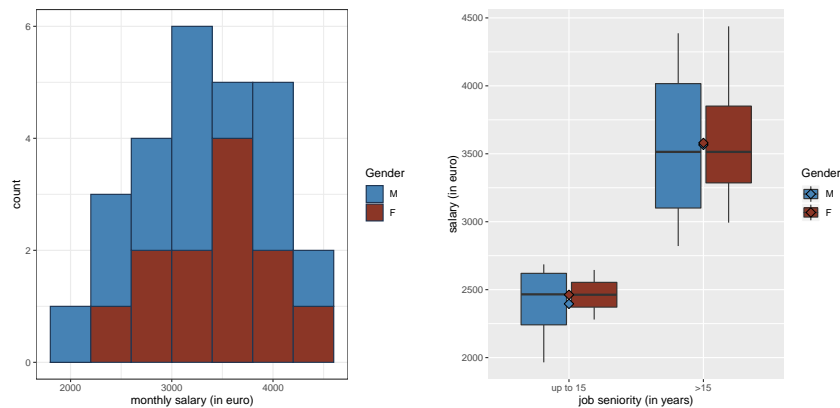


FIGURE A1.5: Stacked histogram by gender (left) and side-by-side box plot (right) for salaries.

A1.3.3.1 Stacked histograms and side-by-side box plots

For the `Salaries` data file, the effect of gender on the salary can be visualized by the stacked histogram, as shown in Figure A1.5 (left). A *stacked histogram* is a sort of bivariate visualization, exhibiting the distribution of the cases within each histogram class with respect to a categorical covariate. Stacked histograms can easily be delivered by the `ggplot` function of the `ggplot2` package. For this, `ggplot` requires the categorical covariate to be defined as a factor.

Furthermore, we provide in Figure A1.5 (right) a side-by-side box plot for salaries that visualizes simultaneously the effect of gender and a job seniority binary variable (years in job ≤ 15 or > 15). The box plots do also not seem to support a gender bias.

The R-code for deriving the stacked histogram and the side-by-side box plot shown in Figure A1.5, follows.

```
library(ggplot2)
library(RColorBrewer)
# ggplot requires the variable used for stacking to be a factor:
Gender <- factor(salaries$gender, labels=c("M","F"))
# stacked histogram:
barlines <- "#1F3552"
ggplot(salaries, aes(x = salary, fill = Gender)) +
  geom_histogram(aes(y = ..count..), binwidth = 400,
    colour = barlines, position="stack") +
  labs(x="monthly salary (in euro)") +
  theme_bw() +
  scale_fill_manual(values=c("steelblue","tomato4"))

# box plot (multiple groups):
years.cat <- factor(salaries$years>15, labels=c("up to 15", ">15"))
ggplot(salaries, aes(x=years.cat, y=salary, fill=Gender)) +
  geom_boxplot() +
  stat_summary(fun.y=mean, geom="point", shape=23, size=3) +
  labs(x="job seniority (in years)", y="salary (in euro)") +
  scale_fill_manual(values=c("steelblue","tomato4","steelblue",
    "tomato4"))
```

Side-by-side box plots can be constructed via the basic function `boxplot` as well, as shown in Section 1.4.5 (Figure 1.6).

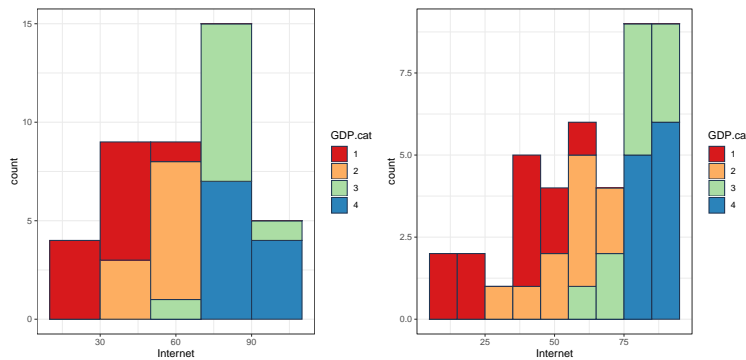


FIGURE A1.6: Stacked histograms for the variable **Internet** with classes of length 20 (left) and 10 (right).

Analogously, for the UN data file, we can derive a histogram for the values of the **Internet** variable stacked by different levels of **GDP**. For this, we categorize **GDP**, to an ordinal variable **GDP.cat** of 4 levels, corresponding to the intervals defined by the quantiles of **GDP** and produce the corresponding stacked histogram shown in Figure A1.6 (left) as follows:

```
> GDP.cat<-as.integer(cut(UN[,2], c(0,quantile(UN[,2]))))
> GDP.cat <- as.factor(ifelse(GDP.cat==1, 2, GDP.cat)-1)

> library(ggplot2)
> library(RColorBrewer)
> barlines <- "#1F3552"      # the color for the barlines
> ggplot(UN, aes(x = Internet, fill = GDP.cat)) +
  geom_histogram(aes(y = ..count..), binwidth = 20,
    colour = barlines, position="stack") +
  theme_bw() +
  scale_fill_brewer(palette="Spectral")
```

The histogram for the same variables but for shorter classes (`binwidth = 10`) is shown in Figure A1.6 (right).

For the **GSS2018** data file, the age can easily be categorized (based on the intervals used for the histogram), as follows:

```
> age.cat <- table(cut(age, c(18,25,35,50,65,75,90), include.lowest=T, right=T))
```

The age histogram stacked by the survey participants vote in the 2016 presidential elections, is provided in Figure A1.7 (left) and produced as follows (the variable **PRES16** needs first to be factorized):

```
> pres16 <- factor(GSS$PRES16, levels=c(1,2,3,4), labels =
  c("Clinton", "Trump", "Other", "Not vote"))
> ggplot(GSS, aes(x=age, fill=pres16)) +
  geom_histogram(breaks=c(18,25,35,50,65,75,90),
    aes(y = ..count../(n*width)), color="black") +
  scale_fill_manual(values=c("royalblue2","red","green4","white"))
```

Note that this histogram (ignoring the **pres16** factor) is the same as that of Table A1.2 (right). A high percentage of participants did not respond to the presidential elections question. The category of missing values (NA) is considered as level of the **pres16** factor and thus included in the histogram constructed above. The histogram for the ages of the responders that did respond this question (i.e. excluding NA), is provided in Figure A1.7 (right) and is derived as given below.

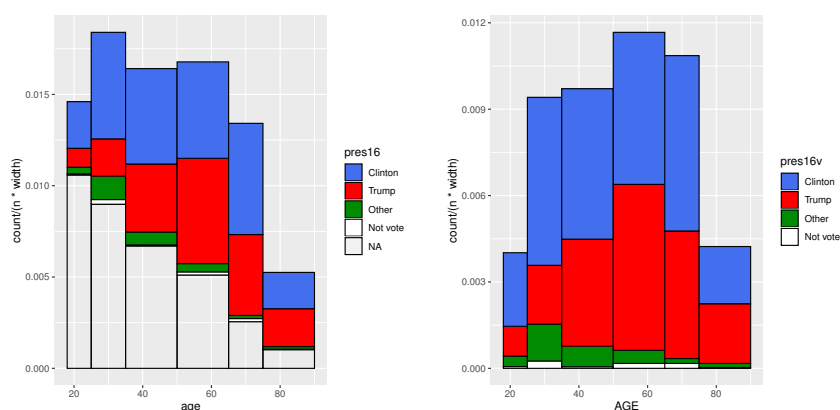


FIGURE A1.7: Histogram for the age of the GSS2018 participants, stacked by their vote in the presidential elections 2016 with (left) and without (right) the NA category.

```
# indicator for NA in PRES16:
> ind.vote <- complete.cases(GSS[,12])
> GSS2018vot <- GSS[ind.vote,] # subsample of complete PRES16
> pres16v <- factor(GSS2018vot$PRES16, levels=c(1,2,3,4), labels =
  c("Clinton", "Trump", "Other", "Not vote"))
> ggplot(GSS2018vot, aes(x=AGE, fill=pres16v)) +
  geom_histogram(breaks=c(18,25,35,50,65,75,90),
    aes(y = ..count../(n*width)), color="black") +
  scale_fill_manual(values=c("royalblue2","red","green4","white"))
```

A1.4 Missing Values in Data Files

Many data files have missing observations on some or all variables. We illustrate with the GSS2018 data file. The names of the variables in GSS and the sample size of this survey are given by:

```
> names(GSS)
[1] "subject" "AGE" "SEX" "RACE" "EDUC" "WRKSTAT" "MARITAL"
[8] "EARNRS" "INCOME" "RINCOME" "PARTYID" "GUNLAW" "PRES16"
[14] "SMALLGAP" "TRCOURTS"
> n <- nrow(GSS); n
[1] 2348
```

Starting with the summary statistics for all variables in GSS

```
> summary(GSS) # (output not shown)
```

beyond the basic descriptive statistics for all values in the data set, we get also information about the number of non available values (NA) per variable. NA's may cause problems in the analysis; for example function `mean` results to NA. A way out is to apply the function removing the NA-values (`na.rm=T`):

```
> mean(GSS$AGE, na.rm=T)
[1] 48.97138
```

One could restrict the analysis only to the subsample of complete cases but this can be very small and leads to loss of valuable information. In our case the complete cases are just 224 out of 2348:

```
> GSS2018full <- na.omit(GSS)    # subsample of fully complete cases
> ind.df <- complete.cases(GSS)  # indicator (logical: F for NA)
> sum(ind.df)                    # number of complete cases
[1] 224
```

For this it is better to remove cases with NA values selectively (only for the variables analyzed). For example, we can verify that there are only 7 missing age values in the sample. If we want to follow in the sequel analysis only the cases with known age, then we can restrict our sample, as shown below.

```
> sum(is.na(GSS$AGE)) # number of NA cases for variable age
# [1] 7
# indicator for NA-values in AGE (1st variable):
> ind.age <- complete.cases(GSS[,1])
# subsample without missing age values:
> GSS2018 <- GSS[ind.age,]
> mean(GSS2018$AGE)    # equivalent to: mean(GSS$AGE, na.rm=T)
```

In large surveys data, it is important to have an overview of the extent and distribution of the missing values. A graphical visualization is provided in the **visdat** package by the **vis_miss** function which applies on data frames and provides a heatmap of missingness, colouring cells according to whether they are missing or not. There is the additional option to cluster the NA values and the columns (variables) by the rows (sample units) to identify easier missing patterns, if present. For the **GSS** dataframe, this missingness heatmap, given in Figure A1.8, is produced by:

```
> GSS2 <- data.frame(GSS$AGE, GSS$INCOME, GSS$RINCOME, GSS$PARTYID, GSS$GUNLAW)
> library(visdat)
> vis_miss(GSS2)
> vis_dat(GSS2) # produces a variant of vis_miss(GSS2) with columns colored by
                # the type of the corresponding variable (not shown)
> vis_miss(GSS2, cluster=TRUE)
```

The heatmap shows that AGE and PARTYID have little missing data whereas the other variables (e.g. RINCOME, the respondent's income) are missing a lot. Overall 19.1% of the observations are missing for these five variables.

Identifying missing data is important in the *data wrangling* phase of a data analysis. Missingness can be due to different causes and has to be treated accordingly. Advanced methods of statistics, such as *data imputation*, analyze the data by estimating the missing observations.

A1.5 Summarizing Bivariate Quantitative Data

Bivariate quantitative data can be visualized by scatterplots while the strength of their linear correlation is measured by the correlation coefficient of Pearson, implemented in R by **plot** and **cor** functions, respectively, as illustrated in Section 1.5.1.

Next, we illustrate for the **Salaries** data file how to additionally visualize the effect of a categorical covariate in a scatterplot. In particular, the relation between salary and seniority in job is shown in the scatterplot of Figure A1.9 (left), along with the fitted linear

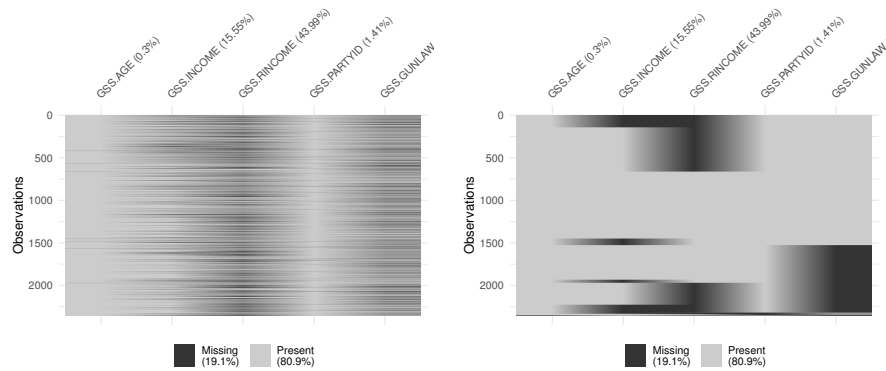


FIGURE A1.8: Missing data heatmaps for the GSS2018 data file, without and with clustering according to missingness.

regression line. The data points are differently marked per gender, indicating that for males and females of the same job seniority, the men are slightly better paid. The regression lines fitted on the subsamples of males and females separately are illustrated in Figure A1.9 (right) as well. The R-code for producing these scatterplots is given below.

```
Gender <- factor(salaries$gender, labels=c("M", "F"))
# scatterplot (with regression lines):
plot(salaries$salary~salaries$years, pch=as.character(Gender),
     ylab="Monthly Salary (euro)", xlab="Job Seniority (years)")

# adds the fitted linear regression:
abline(lm(salaries$salary~salaries$years))

# regression lines fitted on subsamples defined by gender:
abline(lm(salaries$salary~salaries$years, subset=Gender=="M"),
      lty=1, col="blue")
abline(lm(salaries$salary~salaries$years, subset=Gender=="F"),
      lty=2, col="red")
# add a legend to the plot:
legend("topleft", legend=c("Male", "Female"), lty=1:2, col=
      c("blue", "red"))
```

A1.5.1 Pairwise Correlations for Quantitative Data

In case of more than two quantitative variables, the matrix of all pairwise correlations among all quantitative variables is considered. For the UN data file, the correlation matrix, with entries rounded to 3 decimal points, is provided as follows:

```
> cm <- round(cor(UN[,2:9], method='p'), 3); cm
```

	GDP	HDI	GII	Fertility	C02	Homicide	Prison	Internet
GDP	1.000	0.903	-0.851	-0.486	0.674	-0.407	-0.003	0.877
HDI	0.903	1.000	-0.882	-0.669	0.681	-0.428	0.037	0.868
GII	-0.851	-0.882	1.000	0.598	-0.555	0.511	0.216	-0.866
Fertility	-0.486	-0.669	0.598	1.000	-0.448	0.306	-0.086	-0.480
C02	0.674	0.681	-0.555	-0.448	1.000	-0.165	0.369	0.641
Homicide	-0.407	-0.428	0.511	0.306	-0.165	1.000	0.331	-0.337
Prison	-0.003	0.037	0.216	-0.086	0.369	0.331	1.000	-0.014
Internet	0.877	0.868	-0.866	-0.480	0.641	-0.337	-0.014	1.000

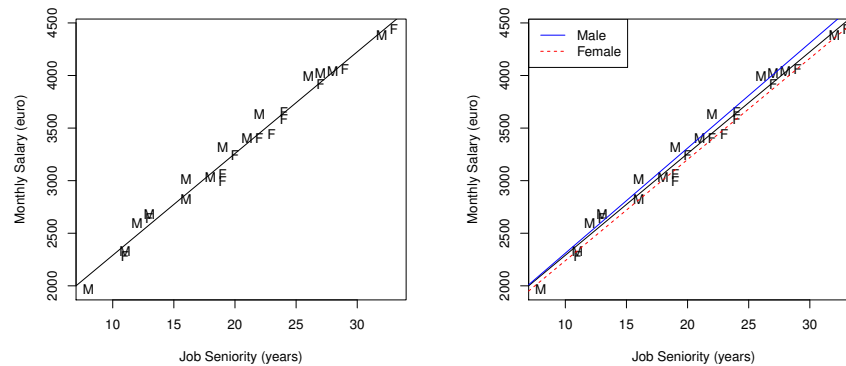


FIGURE A1.9: Scatterplots of salaries by gender with the fitted linear regression line. The regression lines fitted on the subsamples of males and females are shown on the right scatterplot in blue and red, respectively.

The correlation matrix can be visualized by a *heatmap*, where the cells of the correlation matrix are colored according to the strength and direction of the corresponding pairwise correlations. Heatmaps can be produced by the function `heatmap`. Alternatively, it can be constructed in `ggplot2`, as shown next. In order to apply the `ggplot` function, the correlation matrix has to be expanded (melted) in a vector. This is achieved by function `melt` of the `reshape2` package, which is installed with `tidyverse`.

```
> library(reshape2)      # needed for melt()
> cm_melt <- melt(cm, na.rm = TRUE)

> ggheatmap <- ggplot(cm_melt, aes(Var2, Var1, fill = value)) +
  geom_tile(color = "white") +
  scale_fill_gradient2(low = "blue", high = "red", mid = "white",
    midpoint = 0, limit = c(-1,1), space = "Lab",
    name="Pearson\nCorrelation") +
  theme(axis.text.x = element_text(angle = 45, vjust = 1,
    size = 10, hjust = 1),
    axis.title.x = element_blank(),
    axis.title.y = element_blank(),
    panel.grid.major = element_blank(),
    panel.border = element_blank(),
    panel.background = element_blank(),
    axis.ticks = element_blank() ) +
  coord_fixed()

# Print the heatmap (not shown here)
> print(ggheatmap)
```

A nice and very convenient feature of the `ggplot2` package is that it allows to build a graph gradually. Thus, once the heatmap is printed (not shown here), we can update the graph and add the values of the pairwise correlations in its cells by simply adding the corresponding argument. The output of this command is shown in Figure A1.10.

```
ggheatmap +
  geom_text(aes(Var2, Var1, label = value), color = "black", size = 2.5)
```

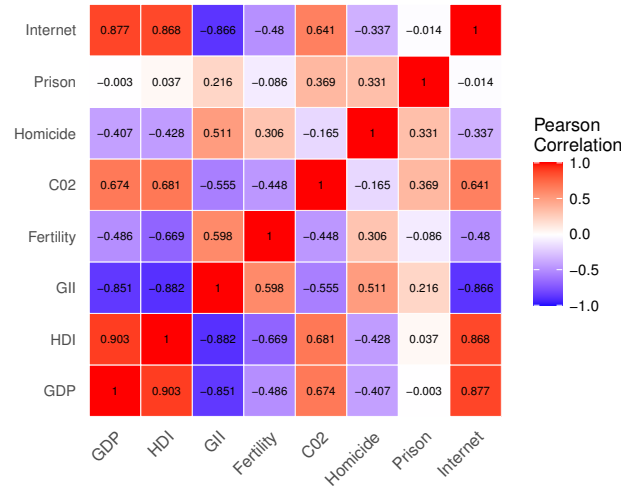


FIGURE A1.10: Heatmap of the Pearson's correlation matrix for the UN data.

For alternative variants of heatmaps for the correlation matrix with `ggplot2`, also in triangular form, we refer to the corresponding entry of [STHDA](#) (statistical tools for high-throughput data analysis).

Furthermore, a visualization of all possible pairwise linear relationships among the variables is provided by all paired scatterplots. In the scatterplots given in Figure [A1.11](#), nations can be identified with respect to the level of their GDP, i.e. the level of `GDP.cat`. This set of scatterplots is derived as follows.

```
pairs(UN[2:9], main = "UN Data", pch = 21, bg = c("red",
+ "orange", "darkseagreen3", "steelblue")[unclass(GDP.cat)])
```

A1.6 Summarizing Bivariate Categorical Data

Furthermore, one can combine the `count` and `group_by` functions to get the frequency distributions of a qualitative variable within the values of another variable (qualitative or quantitative with small number of distinct observed values). For example, the frequency distribution of `PRES16` for males and females is obtained as shown below.

```
> GSS$SEX <- factor(GSS$SEX, levels=c(1:2),
+ labels = c("male", "female"))
> GSS %>% group_by(SEX) %>% count(PRES16)
```

As shown in Section 1.5.2, a two-way contingency table can be created by cross-classifying two categorical variables of a data frame using the `table` function. Furthermore, associated proportion tables can be derived by the `prop.table` function.

Alternatively, a contingency table can be typed directly in R as a matrix. Let us consider

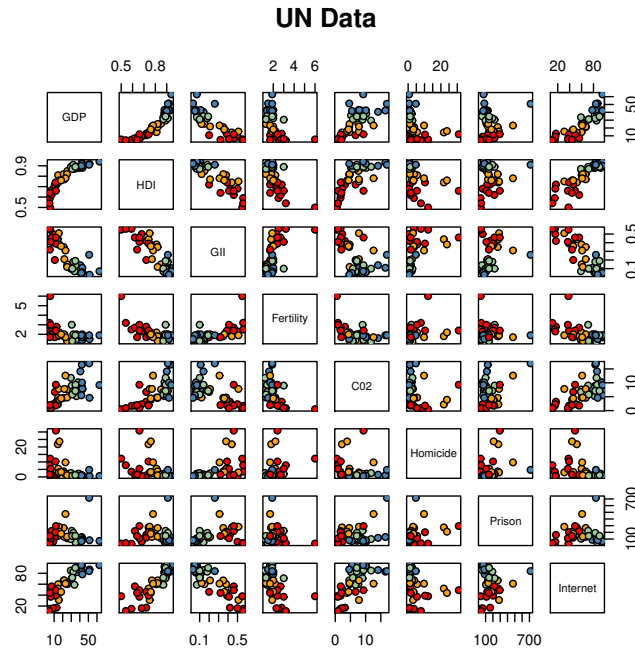


FIGURE A1.11: Pairwise scatter plots for the variables values of UN data, where the nations are identifiable with respect to the quantiles of the GDP, i.e. nations with $GDP \leq 13.18$, in $(13.18, 27.45]$, $(27.45, 40.33]$ or > 40.33 , are marked in red, orange, blue or green, respectively.

the prayer study discussed in Sections 4.5.5 and 5.4.2. The associated 2×2 contingency table, provided in Table 4.3, can be constructed as follows:

```
> heart.surg <- matrix(c(315, 304, 289, 293), nrow=2, ncol=2)
> dimnames(heart.surg) <- list(Prayer=c("Yes", "No"),
                               Complications=c("Yes", "No"))

> addmargins(heart.surg) # the row and column sums are added
      Complications
Prayer Yes  No  Sum
Yes    315 289  604
No     304 293  597
Sum    619 582 1201
```

The `prop.table` function applies on matrices and thus corresponding sample proportions tables are produced as follows.

```
> hs.p <- prop.table(heart.surg) # equivalent to: heart.surg/sum(heart.surg)
> addmargins(hs.p)
      Complications # proportions of the table sum to 1
Prayer Yes  No  Sum
Yes    0.2622814 0.2406328 0.5029142
No     0.2531224 0.2439634 0.4970858
Sum    0.5154038 0.4845962 1.0000000

> hs.col <- prop.table(heart.surg, margin=2) # within column proportions
> addmargins(hs.col, margin=1)
```

	Complications		
Prayer	Yes	No	
Yes	0.5088853	0.4965636	
No	0.4911147	0.5034364	
Sum	1.0000000	1.0000000	# proportions sum to 1 within columns

However, though the contingency table is a convenient form of data representation, modeling functions usually require that of a data frame. Furthermore, the data frame representation is more handy for multi-dimensional contingency tables, while it is the only option for data of mixed type (continuous and categorical), which is mostly the case in data science applications. The data frame corresponding to the example above would be

```
> freq <- c(315,289,304,293)      # vector of cell frequencies (by row)
> row <- rep(1:2, each=2)         # vector of row categories
> col <- rep(1:2,2)               # vector of column categories
> heart.df <- data.frame(freq,row,col)
> heart.df
  freq row col
1  315   1   1
2  289   1   2
3  304   2   1
4  293   2   2
```

For our GSS2018 data file, the relationship between political views and opinion towards the gun law can be visualized by stacked bar plots. The bar plots of Figure A1.12 show the distribution of law support within the voters of each candidate and the distribution of the voting behavior within the supporters and opponents of the law. These bar plots are derived by the `barplot` function of basic R as shown below.

```
> election16 <- prop.table(xtabs(~ gender + pres16, data = GSS))
> GunLaw <- factor(GSS$GUNLAW, levels=c(1,2), labels =
+                  c("favor","oppose"))
> gun_elect16 <- prop.table(xtabs(~ pres16+ GunLaw, data = GSS))

> barplot(gun_elect16, col=c("royalblue2","red","green4","white"),
+         legend.text=T,main="Gun Law (by Vote in Presidential Elections 2016)",
+         xlab="Law requiring police permit for gun purchase (GSS2018)",
+         ylab="Proportions", ylim=c(0,0.8))

> barplot(t(gun_elect16), col=c("green3", "red2"), legend.text=T,
+         main="Vote in Presidential Elections 2016 (by Gun Law)",
+         xlab="", ylab="Proportions", ylim=c(0,0.6))
```

Note that the barplot in Figure A1.12 (right) is fitted on `t(gun_elect16)`, the transpose of the data matrix `gun_elect16`.

The most popular graphical display for contingency tables is the *mosaic plot*. For a two-way contingency table, a mosaic plot displays the cells of the table as rectangular areas of size proportional to the corresponding frequencies. Additionally, the alignment of these areas is indicative for the underlying association between the classification variables. Worse alignment signals stronger association. Mosaic plots can be constructed by the `mosaicplot` function of the `graphics` package or in the `vcd` package by `mosaic`.

The mosaic plot for the contingency table cross-classifying GSS2018 participants' gender and their vote in the presidential elections 2016, constructed as shown below, is provided in Figure A1.13.

```
> library(vcd)
> vote16 <- factor(GSS$PRES16, levels=c(1,2,3,4),
+                 labels = c("Clinton", "Trump", "Oth", "Nv"))
> ct <- table(gender,vote16)
> mosaic(ct)
```

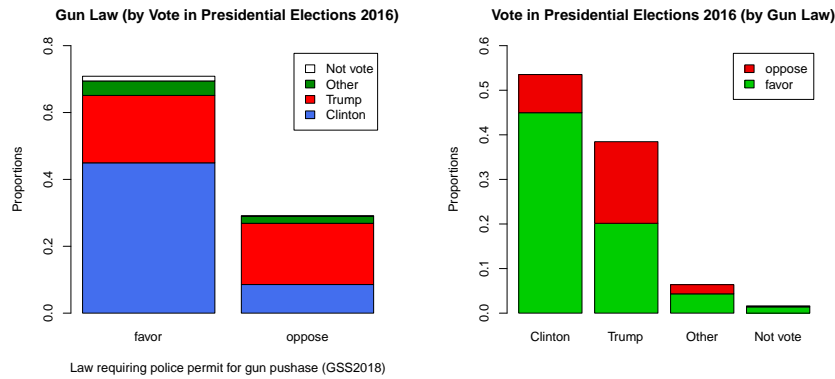


FIGURE A1.12: Stacked barplots of the GSS2018 participants' vote in presidential elections in 2016 vs their opinion with respect to a gun law.

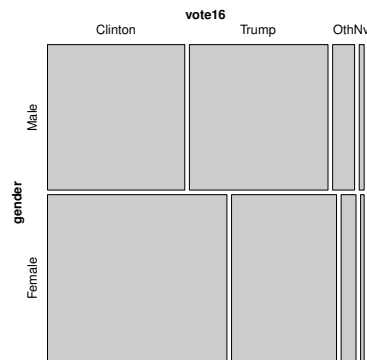


FIGURE A1.13: Mosaic plot for the two-way table cross-classifying GSS2018-participants' gender and their vote in the presidential elections 2016.

A higher dimensional contingency table cannot be formulated by the `table` function. It can be defined as an `array`. For example, consider the data of a study on substances use in high schools, available in the `Substance` data file (see Exercise 7.59). The data can be converted to a three-way contingency table.

```
> Drugs <- read.table("http://stat4ds.rwth-aachen.de/data/Substance.dat", header=TRUE)
> table.3w <- array(Drugs$count, c(2,2,2), dimnames=names); table.3w
, , alcohol = yes
      cigarette
marijuana yes no
      yes 911 44
      no 538 456

, , alcohol = no
      cigarette
marijuana yes no
      yes   3  2
      no  43 279
```

2

CHAPTER 2: R FOR PROBABILITY DISTRIBUTIONS

A2.1 R Functions for Probability distributions

R handles directly major continuous and discrete distributions; the univariate distributions directly treated in R are provided in Table A2.1.

TABLE A2.1: Functions in R for common univariate probability distributions.

Distribution	Base name	Parameters	<i>pdf</i> or <i>pmf</i>
Beta	beta	shape1, shape2	Equation (2.20)
Binomial	binom	size, prob	Equation (2.6)
Cauchy	cauchy	location, scale	Section 4.4.5
Chi-squared	chisq	df	Section 4.4.5
Exponential	exp	rate	Equation (2.12)
F	f	df1, df2	Section 6.4.1
Gamma	gamma	shape, rate	Equation (2.10)
Geometric	geom	p	Section 2.2.2
Hypergeometric	hyper	m, n, k	Equation (2.18)
Log-normal	lnorm	meanlog, sdlog	Exercise 2.71
Logistic	logis	location, scale	Section 7.2.1
Negative binomial	nbinom	size, prob	Equation (2.19)
Normal	norm	mean, sd	Equation (2.8)
Poisson	pois	lambda	Equation (2.7)
t	t	df	Section 4.4.1
Uniform	unif	min, max	Section 2.2.4
Weibull	weibull	shape, scale	Exercise 2.72

For each of these distributions, R provides four functions for computing values of its cumulative density (*cdf*), its inverse *cdf* (used for confidence interval constructions and for hypothesis testing), its *pdf* or *pmf*, as well as for generating random numbers from this distribution. In particular, each distribution has a ‘base name’ (for example **norm** for the normal) and the names of the four basic functions mentioned above are derived by imposing the corresponding prefix-letter, as given below.

- **p** for ‘probability’: *cdf*
- **q** for ‘quantile’: *inverse cdf*
- **d** for ‘density’: *pdf* or *pmf*
- **r** for ‘random’: random sample generation

For example, for the normal distribution these are `pnorm`, `qnorm`, `dnorm`, and `rnorm`. In Sections 2.5.2 and 2.5.3 we found cumulative probabilities and quantiles for normal distributions using `pnorm` and `qnorm`. We used `rnorm` for simulations in Section 1.5.3.

The multinomial distribution (see equation (2.14)) is also treated directly in R, with base function `multinom`. However, for the multinomial distribution, which is multivariate, only the `dmultinom` and `rmultinom` functions are available.

Some probability distributions have alternative parameterizations. For example, the standard *pdf* parameterization in R of a gamma distribution is

$$f(y; \theta, k) = \frac{1}{\theta^k \Gamma(k)} e^{-y/\theta} y^{k-1}, \quad y \geq 0; \quad f(y; \theta, k) = 0, \quad y < 0,$$

for *shape parameter* k and *scale parameter* θ . It has $\mu = k\theta$ and $\sigma = \sqrt{k\theta}$. The scale parameter relates to the *rate parameter* λ in equation (2.10) and the exponential special case (2.12) by $\theta = 1/\lambda$. The R functions for the gamma distribution provide the ability to choose parameterization in terms of scale or rate. However only one of these parameters should be used. For instance, `rgamma(1000, shape=10, scale=2)` is equivalent to `rgamma(1000, shape=10, rate=1/2)`.

Thus, when applying the R functions for probability distributions, attention has to be paid in the parameterization of the corresponding function.

A2.1.0.1 Densities Plots

Probability functions (*pdf* and *pmf*) can be plotted using the *d* versions of the base functions. For example, the graph of the *pmf* of a binomial distribution with sample size $n = 3$ and success probability $\pi = 0.5$, shown in Figure 2.5, is derived as follows:

```
> y <- 0:3
> plot(y, dbinom(y, 3, 0.5), type="h", xaxt="n", lwd=8, lend=2,
+       col="dodgerblue4", ylim=c(0, 0.4), xlab="y", ylab="P(y)")
> axis(1, at = seq(0, 3, by = 1))
```

The plots of the *pdf* and *cdf* of the uniform distribution in interval $[0,1]$, s. Figures 2.6 and 2.7, respectively, have been produced in R by the code given below:

```
# pdf of U(0,1) with highlighted P(a<Y<b):
> x<- seq(0,1,length=2)
> plot(x,dunif(x), type="l", xlim=c(-0.4,1.4), col="blue", lwd=2, xaxt="n",
+       ylim=c(0, 1), xlab="y", ylab="f(y)")
> x <- seq(0.2,0.6, length=2)      # the borders of the interval [a, b]
> y <- dunif(x)
> polygon(c(0.2,x,0.6), c(0,y,0), col="gray")
> axis(1, at=c(0,0.2,0.6,1), labels=c(0,"a", "b", 1))
> text(0.4,0.5,paste("P(a<Y<b)"),cex=1,col="white")
> segments(-0.4, 0, 0, 0, col="blue", lwd=2)
> segments(1, 0, 1.4, 0, col="blue", lwd=2,lend=0.1)

# cdf of U(0,1) :
> x<- seq(0,1,length=2)
> plot(x,punif(x), type="l", xlim=c(-0.4,1.4), col="blue", lwd=2, xaxt="n",
+       ylim=c(0, 1), xlab="y", ylab="f(y)")
> axis(1, at = seq(-0.4, 1.4, by = 0.2), las=2)
> segments(-0.4, 0, 0, 0, col="blue", lwd=2,lend=0.1)
> segments(1, 1, 1.4, 1, col="blue", lwd=2,lend=0.1)
> segments(-0.45,0.8,0.8,0.8,lty=2)
> segments(0.8,-0.04,0.8,0.8,lty=2)
> text(-0.2,0.85, paste("P(Y<0.8)=0.8"), cex=1)
```

The *pdf* in Figure 2.8 is an example of an exponential distribution with parameter $\lambda = 1$ and is derived as follows:


```
> curve(dexp(x,1), xlim=c(0,14), col="blue", lwd=2, xlab="y", ylab="f(y)")
```

The plot in Figure 2.8 is given over the support (i.e. for $x \geq 0$). Alternatively, the *pdf* $f(y) = \begin{cases} e^{-y}, & \text{if } y \geq 0 \\ 0, & \text{if } y < 0 \end{cases}$ can be plotted over \mathbb{R} . The associated R-code follows. Replacing in the code the **dexp** function through **pexp**, the graph of the *cdf* is derived. Both plots are given in Figure A2.1.

```
> x<- seq(0,14,length=300)
> plot(x,dexp(x,1), xlim=c(-2,14), xaxt="n", type="l", col="blue", lwd=2,
      xlab="y", ylab="f(y)")
> axis(1, at = seq(-2, 14, by = 2))
# x-axis values are listed vertically if axis() is replaced as follows:
# axis(1, at = seq(-2, 14, by = 2),las=2)
> segments(-2, 0, 0, 0, col="blue", lwd=2)
```

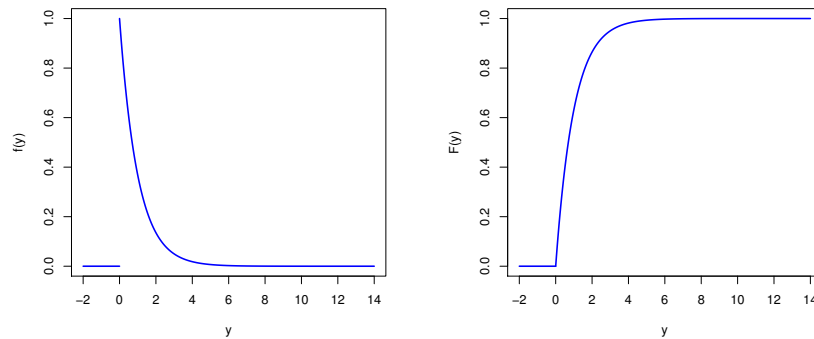


FIGURE A2.1: The *pdf* (left) and *cdf* (right) of an exponential distribution with parameter $\lambda = 1$.

The R-code for producing the graph of gamma *pdfs* of different shape parameter values but all having expected value equal to 10, provided in Figure 2.12 (right), follows below. Replacing function **dgamma** by **pgamma**, the corresponding graph of the *cdfs* is derived.

```
> y = seq(0, 40, 0.0001)
> plot(y, dgamma(y, shape=10, scale=1), ylab="f(y)", type="l", col="blue")
> lines(y, dgamma(y, shape=2, scale=5), col="green") # scale = 1/lambda
> lines(y, dgamma(y, shape=1, scale=10), col="red")
> legend(20,0.12, c("k=10","k=2","k=1"),lty=c(1,1,1),col=c("black","green","red"))
```

A2.2 Quantiles, $Q-Q$ Plots and the Normal Quantile Plot

In Sections 2.5.3 and 4.4.1 examples of R-code for calculating the probability that a random variable Y with known *cdf* F takes values in a prespecified interval have been provided. The code can be organized in a function for convenient repeated use. Next we provide the function for calculating the probability that a normal random variable Y takes values in the interval (a, b) and constructing the associated graph, given in Figure A2.2 (left).

```

> probNormal <- function(a,b,mu,sigma){
  prob <- pnorm(b,mu,sigma)-pnorm(a,mu,sigma)
  low <- min(mu-4*sigma,a); up <- max(mu+4*sigma,b)
  curve(dnorm(x,mu,sigma), xlim=c(low,up), main=" ", xlab="y",
        ylab=expression(phi(y)), col="blue", lwd=2)
  x=seq(a,b,length=200)
  y=dnorm(x,mu,sigma)
  polygon(c(a,x,b),c(0,y,0),col="coral2")
  result <- paste("N(",mu,"",sigma^2,"): ",
                 "P(",a,"< Y <",b,") =",signif(prob, digits=3))
  mtext(result,3, col="coral2")
  # text(low,0.15,paste(prob),cex=1,col="grey60")
  curve(dnorm(x,mu,sigma),col="blue", lwd=2,add=T)
  return(prob)}

#-----
> probNormal(2,5,3,3)      # example
[1] 0.3780661

```

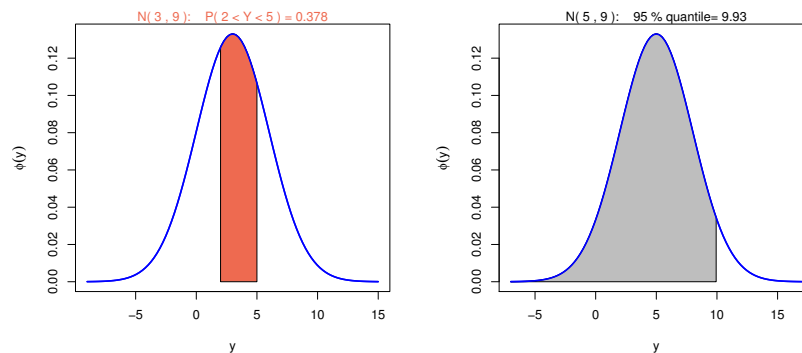


FIGURE A2.2: The probability that a normal distribution with $\mu = 3$ and $\sigma^2 = 9$ takes values in the interval $[2, 5]$ (right) and the 90% quantile of a normal distribution with $\mu = 5$ and $\sigma^2 = 9$.

Please notice the use of the `expression()` function for adding mathematical expressions on an R plot. Here, we applied `expression()` for using Greek letters in the y-axis label.

The function above can be adjusted for other distributions or for calculating probabilities of lower or upper tails. Caution is required when calculating probabilities of discrete random variables. Since though for a continuous random variable Y it holds $P(Y \in (a, b]) = P(Y \in [a, b])$, this is not true if Y is discrete, since in this case in general $P(Y = a) \neq 0$.

For a continuous random variable, Section 2.5.6 defined the p th quantile (100 p percentile) as the point q at which the *cdf* satisfies $F(q) = p$. For a discrete random variable, the *cdf* is a step function, and the p th quantile is defined as the minimum q such that $F(q) \geq p$. For instance, for the binomial distribution in Table 2.3 with $n = 12$ and $\pi = 0.50$, the *cdf* has $F(5) = 0.3872$ and $F(6) = 0.6128$, so the p th quantile is 6 for any $0.3872 < p \leq 0.6128$, such as $p = 0.40$ and 0.60 as shown in the following R code:

```

> cbind(5:6, pbinom(5:6, 12, 0.50))
      [,1] [,2]
[1,]   5 0.3872
[2,]   6 0.6128
> qbinom(0.40, 12, 0.50); qbinom(0.60, 12, 0.50)
[1] 6
[1] 6

```

Figure A2.3 illustrates. It shows, for instance, that 0.60 on the vertical cdf probability scale maps to the 0.60 quantile of 6 on the horizontal scale of binomial random variable values.

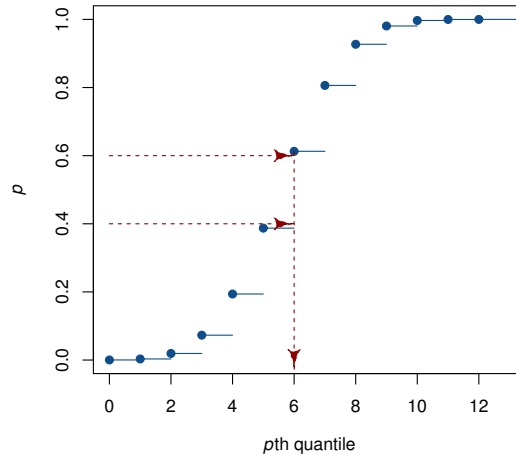


FIGURE A2.3: The cdf of a binomial distribution with $n = 12$ and $\pi = 0.50$ with the 0.40th and 0.60th quantiles, both equal to 6.

The code that follows produces a plot allocating the p th quantile of a normal distribution on the horizontal axis (y) and shading gray the probability p , as illustrated in the produced graph in Figure A2.2 (right).

```
quantNormal <- function(a,mu,sigma){
  quant <- qnorm(a,mu,sigma)
  low <- min(mu-4*sigma,a)
  up <- max(mu+4*sigma,a)
  curve(dnorm(x,mu,sigma), xlim=c(low,up), main=" ", xlab="y",
        ylab=expression(phi(y)), col="blue", lwd=2)
  x=seq(low,quant,length=200)
  y=dnorm(x,mu,sigma)
  polygon(c(low,x,quant),c(0,y,0),col="gray")

  result <- paste("N(",mu,",",sigma^2,"):",
                  a*100,"% quantile=",signif(quant, digits=3))
  mtext(result,3)
  curve(dnorm(x,mu,sigma),col="blue", lwd=2,add=T)
  return(quant)}

#-----
# example:
> quantNormal(0.95,5,3)      # equivalent to qnorm(0.95,5,3)
[1] 9.934561
```

Some inferential statistical methods assume that the data come from a particular distribution, often the normal. The $Q-Q$ plot (*quantile-quantile plot*) is a graphical comparison of the observed sample data distribution with a theoretical distribution. As explained in Exercise 2.67, it plots the *order statistics* $y_{(1)} \leq y_{(2)} \leq \dots \leq y_{(n)}$ of the data against the ordered quantiles $q_{\frac{1}{n+1}} \leq q_{\frac{2}{n+1}} \leq \dots \leq q_{\frac{n}{n+1}}$ of the reference distribution. If $\{q_{\frac{i}{n+1}}\}$ and $\{y_i\}$ come from the same distribution, the points $\{(q_{\frac{1}{n+1}}, y_{(1)}), (q_{\frac{2}{n+1}}, y_{(2)}), \dots, (q_{\frac{n}{n+1}}, y_{(n)})\}$ should approximately follow a straight line, more closely so when n is large. With the standard normal distribution for $\{q_{\frac{i}{n+1}}\}$ and a normal distribution assumed for $\{y_i\}$, the $Q-Q$ plot is called

a *normal quantile plot*. With a standard normal distribution assumed for $\{y_i\}$, the points should approximately follow the straight line $y = x$ having intercept 0 and slope 1, which R plots with the command `abline(0,1)`. When the points deviate greatly from a straight line, this gives a visual indication of how the sample data distribution differs from the reference distribution.

We illustrate by generating random samples from a standard normal distribution, a t distribution (introduced in Section 4.4.1, symmetric around 0 like the standard normal but with thicker tails), an exponential distribution (2.12) with $\lambda = 1$, and a uniform distribution over $(0, 1)$. The `qqnorm` function creates normal quantile plots:

```
> Y1 <- rnorm(1000); Y2 <- rt(1000, df=3) # generating random samples
> Y3 <- rexp(1000); Y4 <- runif(1000)      # from four distributions
> par(mfrow=c(2, 2))                      # plots 4 graphs in a 2x2 matrix format
> qqnorm(Y1, col='blue', main='Y1 ~ N(0,1)'); abline(0,1)
> qqnorm(Y2, col='blue', main='Y2 ~ t(3)'); abline(0,1)
> qqnorm(Y3, col='blue', main='Y3 ~ exp(1)')
> qqnorm(Y4, col='blue', main='Y4 ~ uniform(0,1)')
```

Figure A2.4 shows the normal quantile plots. The first plot shows excellent agreement, as expected, between the normal sample and the normal quantiles, with the points falling close to the straight line $y = x$. The plot for the sample from the t distribution indicates that more observations occur well out in the tails (i.e., larger $|t|$ values) than expected with a standard normal distribution. The plot for the uniform distribution indicates the opposite, fewer observations in the tails than expected with the normal distribution. The plot for the sample from the exponential distribution reflects the right skew of that distribution, with some quite large observations but no very small observations, reflecting its lower boundary of 0 for possible values.

To illustrate the normal quantile plot for actual data, we construct if for the ages in the GSS data frame created in Section A1.2.1:

```
> ind.age <- complete.cases(GSS[,2]) # subsample without missing ages (variable 2)
> GSSsub <- GSS[ind.age,]           # new data frame without missing ages
> qqnorm(GSSsub$AGE, col="dodgerblue4"); qqline(GSSsub$AGE)
```

Figure A2.5 (left) shows the plot. The `qqline` function adds the straight line corresponding to the trend in points if the sample distribution were normal. It suggests that the distribution has fewer observations in the tails than expected with the normal, reflecting subjects under 18 not being sampled in the GSS and very old subjects being in a smaller cohort and also dropping out because of deaths. (A histogram also shows evidence of non-normality.) The `ggplot2` package provides options for controlling the format of the plot according to groups defined by levels of a factor. The following shows the code for the Q - Q plots for the ages of females and of males, which is shown in Figure A2.5 (right).

```
> library(ggplot2)
> GSS$SEX <- factor(GSS$SEX, labels = c("male","female"))
> p <- qplot(sample=AGE, data=GSS, color=SEX, shape=SEX); # no output
# one qq-plot for males and females:
> p + scale_color_manual(values=c("blue", "red")) + scale_shape_manual(values=c(2,20)) +
  labs(x="quantiles of N(0,1)", y = "Age")
# separate qq-plot for males and females (facet_wrap):
> p + scale_color_manual(values=c("blue", "red")) + scale_shape_manual(values= c(2,20)) +
  labs(x="quantiles of N(0,1)", y="Age") + facet_wrap(~ SEX)
```

The `qqPlot` function in the `EnvStats` library can construct Q - Q plots for reference distributions other than the normal.

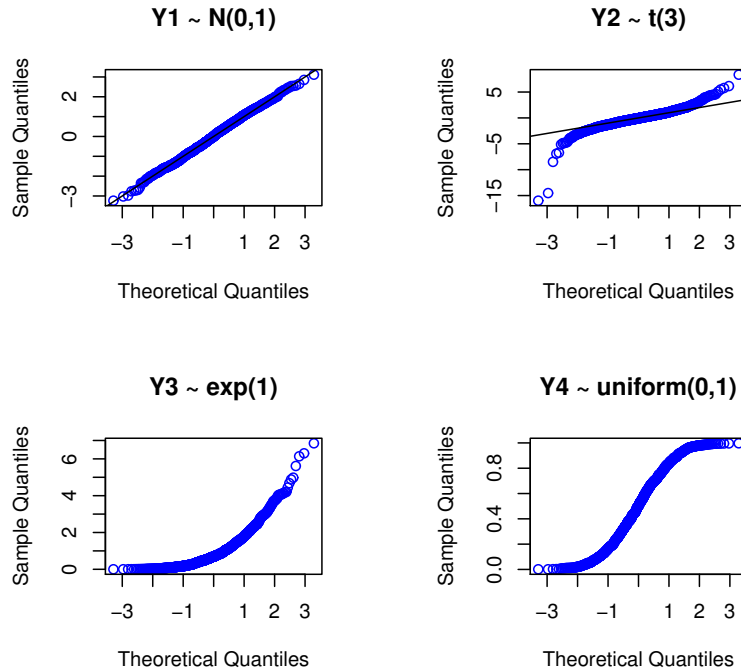


FIGURE A2.4: Normal quantile plots, plotting quantiles of the standard normal distribution against quantiles of random samples from a $N(0,1)$ distribution, a t distribution with $df = 3$, an exponential distribution with $\lambda = 1$, and a uniform distribution over $[0, 1]$.

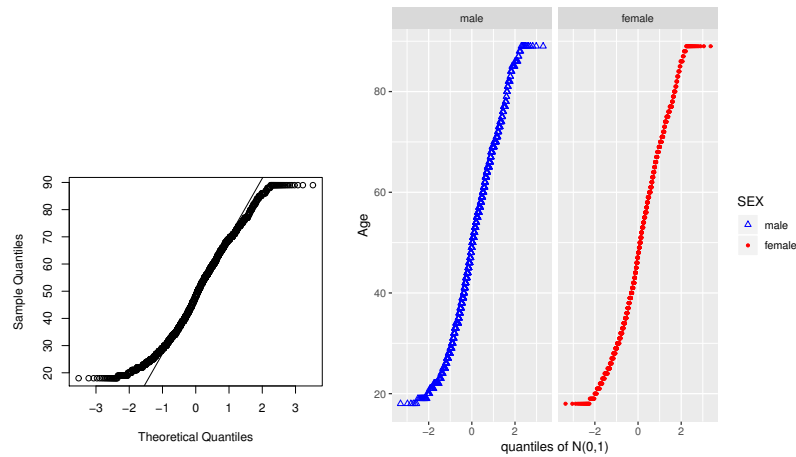


FIGURE A2.5: Normal quantile ($Q-Q$) plots for the ages of respondents in the GSS2018 data file, overall and grouped by gender.

A2.3 Joint and Conditional Probability Distributions

Let X and Y be two categorical random variables with r and c number of categories, respectively. Then, the joint probabilities form an $r \times c$ probability table, formed by cross-classifying the categories of X (in rows) and Y (in columns). Such tables are known as *contingency tables*. The cell entries of a contingency table can be the joint probability $\pi_{ij} = P(X = i, Y = j)$, $i = 1, \dots, r$, $j = 1, \dots, c$, with $\sum_{i,j} \pi_{ij} = 1$, or the corresponding expected cell frequency $\mu_{ij} = n\pi_{ij}$, where n is the total sample size. The realized sample has observed cell frequency n_{ij} , with $\sum_{i,j} n_{ij} = n$. In order to construct contingency tables in R, the cross-classifying variables have to be defined as **factors**. Let us focus for the GSS data frame on the joint probability of gender (SEX) and the responders attitude about a fair society (SMALLGAP). The corresponding barplot diagram of the observed proportions (stacked by gender) is provided in Figure A2.6 (left).

```
> GSS <- read.table("http://stat4ds.rwth-aachen.de/data/GSS2018.dat", header=T)
> gender <- factor(GSS$SEX, levels=c(1,2), labels = c("Male",
  "Female"))
> smallgap <- factor(GSS$SMALLGAP, levels=c(1:5),
  labels = c("strongly agree", "agree","neutral","disagree",
  "strongly disagree"))
> fairsociety <- table(gender,smallgap)           # frequency table
> fairsociety
      smallgap
gender  strongly agree agree neutral disagree strongly disagree
Male           47    140    149    183           38
Female          58    156    192    159           26

> joint.prob <- fairsociety/sum(fairsociety)      # joint cell proportions (total = 1)

## add to joint prob. the row and column marginal probabilities:
> round(addmargins(joint.prob),3)
      smallgap
gender  strongly agree agree neutral disagree strongly disagree Sum
Male           0.041 0.122  0.130  0.159           0.033 0.485
Female          0.051 0.136  0.167  0.139           0.023 0.515
Sum             0.091 0.258  0.297  0.298           0.056 1.000

> barplot(joint.prob, density=40, main="In a fair society, differences
  in people's standard of living should be small",xlab="",
  ylab="proportions",ylim=c(0,0.30),legend=rownames(joint.prob))
```

The joint probability table above is derived manually, dividing all cell entries of the frequency table by the total sample size n . Alternatively, it is convenient to use the function `prop.table`, since it allows also to derive the conditional probabilities within rows or columns. Thus we could produce `joint.prob` through the following command.

```
> joint.prob <- prop.table(fairsociety)          # derives proportion table
  # from a frequency table
> cond.prob1 <- prop.table(fairsociety, 1)      # cond. prop. within rows
> cond.prob2 <- prop.table(fairsociety, 2)      # cond. prop. within columns
      smallgap
gender  strongly agree agree neutral disagree strongly disagree
Male           0.448 0.473  0.437  0.535           0.594
Female          0.552 0.527  0.563  0.465           0.406

> barplot(cond.prob2, density=40, main="In a fair society, differences
  in people's standard of living should be small",xlab="",
  ylab="conditional proportions",ylim=c(0,1))
> abline(h=0.5, col="blue")
```

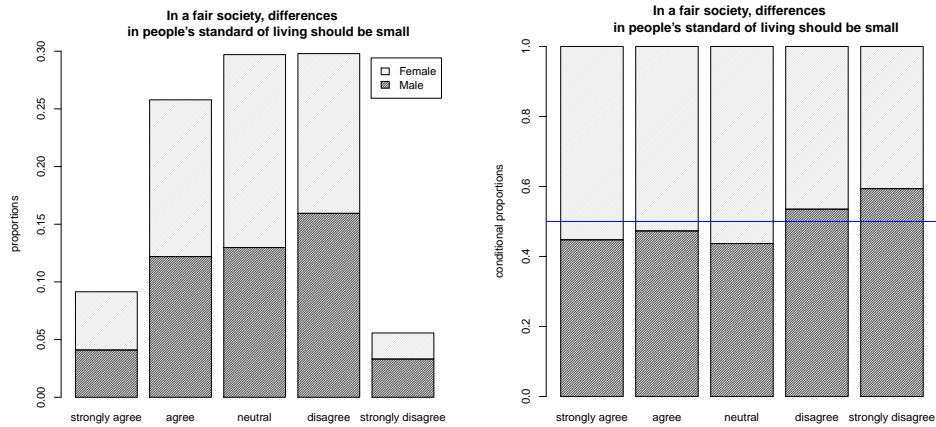


FIGURE A2.6: Barplots for the observed proportions (left) and conditional proportions (right) of the attitude towards a fair society, stacked by gender.

Is the responders' opinion of fair society (Y) independent of gender (X)? If so, then (see Section 2.6.6) it should hold

$$P(X = x|Y = y) = P(X = x), \quad x = 1, 2, \quad y = 1, \dots, 5.$$

We now from the GSS set-up that $P(X = 1) = P(X = 2) = 0.5$. We can observe that the observed conditional gender proportions within each level of response on the small social gap attitude are close to 50% (s. R results above, or the barplot in Figure A2.6, right).

Here, we illustrated a bivariate joint probability. In practical applications, quite often occur joint probabilities of higher dimensions (and high dimensional contingency tables). For a detailed discussion on contingency table analysis we refer to Agresti (2019) and Kateri (2014).

A2.4 The bivariate normal distribution

Multivariate normal (and t) probabilities, densities and quantiles can be computed in R using the package `mvtnorm`. A fuusing nction for random number generation from a multivariate normal with given mean vector and covariance matrix is also provided in this package. A plot of a bivariate normal density can be constructed using the R-package for three-dimensional plots `plot3D`. For an illustration, the R-code for deriving Figure 2.15 is given below:

```
> library(mvtnorm); library(plot3D)

> x <- (-40:40)/10; y <- x
> xy <- mesh(x,y)           # function that generates a full 2-D or 3-D grid
> x0 <- as.vector(xy$x); y0 <- as.vector(xy$y); z0 <- cbind(x0, y0)

# parameter-values of the 2-dim. normal distribution:
> mean <- c(0,0)             # mean vector
> sigma <- matrix(c(1,0.7,0.7,1), ncol=2)    # covariance matrix

# pdf of a multivariate normal distribution:
```

```

> z1 <- dmnorm(z0, mean, sigma)      # pdf of multivariate normal
> z = matrix(z1, ncol = length(x))
> persp3D(z = z, x = x, y = y, theta = -30, phi = 30, colvar = z,
          col="steelblue",shade = 0.5, main="cor=0.7")

```

Alternatively, function `dnorm2d` from the package `fMultivar` can be used in combination with `persp` of package `graphics`, as follows, which produces the plot given in Figure A2.7.

```

> library(fMultivar); library(graphics)
> x <- (-40:40)/10; X <- grid2d(x)
> z <- dnorm2d(X$x, X$y, rho = -0.8)
> X <- list(x = x, y = x, z = matrix(z, ncol = length(x)))
# Perspective Density Plot:
> persp(X, theta = -30, phi = 25, expand = 0.5, col = "lightblue",
        ltheta = 120, shade = 0.15, ticktype = "detailed",
        xlab = "x", ylab = "y", zlab = " ")

```

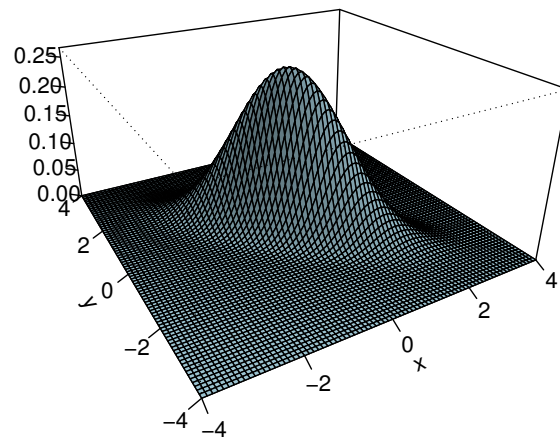


FIGURE A2.7: The *pdf* of a bivariate normal distribution with $\mu_X = \mu_Y = 0$, $\sigma_X = \sigma_Y = 1$ and $\rho_{XY} = -0.8$.

3

CHAPTER 3: R FOR SAMPLING DISTRIBUTIONS

In Section 1.3.1, the function `sample` is applied to generate a random sample of 5 integers from a discrete uniform distribution on $\{1, 2, \dots, 60\}$. This sampling is without replacement, i.e. the same value cannot be considered more than once. Sampling with replacement is carried out as shown next and is the basis of bootstrapping, a very important method in computational statistics (Section 4.6 and A4.5).

```
> sample(1:60, 5, replace = TRUE)
[1] 56 27 54 24 27      # value 27 is observed twice
```

Randomization is also used in *data jittering*, which is the action of adding noise to the data (see Section 7.2.4). Most commonly jittering is applied on scatterplots to reduce overplotting, especially when the data are rounded. It further helps to visualize the data and relationships among variables. However, caution is needed with jittering, since there is no obvious way how to generate the added random noise, while jittering can weaken the visual impact of an underlying relationship between two variables. A scatterplot with jittering is given Figure A3.1, derived as shown below. The `jitter` function of base R provides this option. You can try:

```
> jitter(5) # equivalent to: 5 + runif(1,-1,1)
[1] 4.945889
> jitter(5,2) # equivalent to: 5 + runif(1,-2,2)
[1] 5.08814
x <- sample(1:10, 50, TRUE)
y <- 2 * x + rnorm(50)
plot(y~x, pch=1,col="blue")      # presented in the figure
plot(y~jitter(x,2), pch=1,col="blue")
```

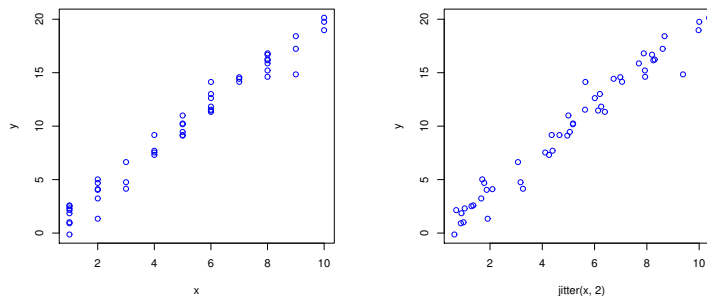


FIGURE A3.1: Scatterplot without (left) and with (right) jittering on the x-values.

Random number generation

In Section 2.5.7, the probability integral transformation is discussed for generating continuous random variables based on the uniform in $[0,1]$ distribution. The uniform distribution can also be used to generate discrete random variables. The discrete version of the inverse transformation method is briefly described next. Consider a nonnegative discrete random variable Y with *pmf*

$$P(Y = k) = \pi_k, \quad k \geq 0, \quad \pi_k \in [0, 1] \quad \text{with} \quad \sum_{k=1}^{\infty} \pi_k = 1.$$

To randomly generate a realization from Y , we first generate randomly a uniform random variable X over $[0, 1]$ and then set $Y = 0$, if $X \leq \pi_0$ and $Y = k$, $k \geq 1$, if

$$\sum_{i=0}^{k-1} \pi_i < X \leq \sum_{i=0}^k \pi_i.$$

For example, for generating randomly from a Bernoulli random variable with success probability π , we generate a uniform X and set $Y = 0$ if $X \leq 1 - \pi (= \pi_0)$ or $Y = 1$ if $X > 1 - \pi$. Consequently, to simulate from a binomial rv Y with parameters n and π , we simulate n *iid* Bernoulli (π) random variables Y_1, \dots, Y_n and then set $Y = \sum_{i=1}^n Y_i$. The corresponding R-function is given below, along with an R function for simulating from a multinomial random variable.

```
# random number generation from Bernoulli (p):
> randbern <- function(s, p) { # p: success probability,
# s: number of values to be simulated

  randnumber1 <- function(){
    x <- runif(1)
    Y <- as.numeric(x>1-p) ; Y } # assignment "x>1-p" produces logical data
  replicate(s, randnumber1()) }

# application for s=15, p=0.3
> Y <- randbern(15,0.3) # vector of length 15 with values 0,1
> table(Y)             # table of frequencies
Y
 0  1
11  4

# random number generation from binomial distribution (n,p):
randbin <- function(s,n,p) { # n, p: parameters of distribution
# s: number of values to be simulated
# requires function randbern

  randnumber2 <- function(){
    X <- randbern(n,p)
    Y <- sum(X) ; Y } # assignment "x>1-p" produces logical data
  replicate(s, randnumber2()) }

# application for s=50, n=12, p=0.15
> Y <- randbin(50,12,0.15) # vector of length 40 with values 0,1,...,10
> table(Y)
Y
 0  1  2  3  4
 7 17 12 10  4

# random number generation from multinomial distribution (n, p)
randmult <- function(n, prob) { # prob: probability vector
  cumprob <- cumsum(prob)      # cumulative probabilities vector
  randnumber <- function(){
    x <- runif(1)
    N <- sum(x>cumprob) ; N }
```

```

replicate(n, randnumber()) }

# application for n=45, p=(0.10, 0.25, 0.45, 0.15, 0.05)
> n <- 45
> prob <- c(0.10, 0.25, 0.45, 0.15, 0.05)
> Y <- randmult(n,prob) # vector of length 45 with values 0,1,...,4
> table(Y)
Y
 0  1  2  3  4
4 12 19  9  1

```

More than one multinomial random samples can be simulated using the command **replicate** as done above for the binomial distribution.

The discrete inverse transformation can be applied also for simulating from a Poisson distribution. An alternative method is based on the connection between the Poisson and the exponential distributions (see Section 2.5.5). Let Y be a random variable counting the number of occurrences of an event over time that has a Poisson distribution with expected number of occurrences in a time interval of length 1 equal to λ . Then the random times T_1, T_2, \dots between successive events are independent and exponential distributed with parameter λ . Based on this, we simulate times between events, t_1, t_2, \dots , from independent exponential distributions with mean 1 and set $Y = i^*$, where i^* is the smallest index such that $\sum_{i=1}^{i^*+1} t_i > \lambda$.

```

# random number generation from Poisson distribution (lambda)

> randpois <- function(s, lambda){
  # lambda: Poisson's parameter
  # s: number of random numbers simulated
  randnumber3 <- function(){
    count <- 0; sumT <- 0
    repeat{T <- rexp(1); sumT <- sumT + T
      if (sumT>lambda) {break}
      else {count <- count+1}}
    count}
  replicate(s, randnumber3()) }

# application for s=100, lambda=2
> Y <- randpois(100,2)
> table(Y)
Y
 0  1  2  3  4  5  6  7
11 23 28 23  9  4  1  1

```

A3.1 Simulating the Sampling Distribution of a Statistic

Suppose we would like to simulate the sampling distribution of some statistic, assuming a simple random sample from a particular probability distribution. We could write an R function to simulate a very large number of random samples from that distribution and then construct a histogram of the values of the statistic.

We illustrate for approximating the sampling distribution of \bar{Y} for a random sample of size n from a Poisson distribution with μ . In the R function created in the following code, B is the number of random samples of size n simulated from the Poisson. We next discuss the steps in the function for $n = 10$, using the value $B = 100,000$ selected at the end of the code. Letting Y be a numeric vector of length $nB = 10(100,000) = 1,000,000$, we randomly

generate 1,000,000 Poisson random variables with mean denoted by μ . We then organize these in a matrix with 10 columns and 100,000 rows. Each row of the matrix contains a simulated random sample of size 10 from the Poisson distribution. The `apply` function then finds the mean within each row. (In the second argument of the `apply` function, 1 indicates rows, 2 indicates columns, and `c(1, 2)` indicates rows and columns.) At this stage, the vector `Ymean` is a vector of 100,000 means. The remaining code creates plots, showing the sample data distribution for the first sample and the empirical sampling distribution of the 100,000 simulated values of \bar{y} .

```
> pois_CLT <- function(n, mu, B) {
  # n: vector of 2 sample sizes [e.g. n <- c(10, 100)]
  # mu: mean parameter of Poisson distribution
  # B: number of simulated random samples from the Poisson
  par(mfrow = c(2, 2))
  for (i in 1:2){
    Y <- numeric(length=n[i]*B)
    Y <- matrix(rpois(n[i]*B, mu), ncol=n[i])
    Ymean <- apply(Y, 1, mean) # or, can do this with rowMeans(Y)
    barplot(table(Y[1,]), main=paste("n=", n[i]), xlab="y",
      col="lightsteelblue") # sample data dist. for first sample
    hist(Ymean, main=paste("n=", n[i]), xlab=expression(bar(y)),
      col="lightsteelblue") # histogram of B sample mean values
  }
  # implement: with 100000 random sample sizes of 10 and 100, mean = 0.7
  > n <- c(10, 100)
  > pois_CLT(n, 0.7, 100000)
```

Figure A3.2 shows the results with $\mu = 0.7$, which we used to find the sampling distribution of the sample median in Section 3.4.3.

We use both $n = 10$ and $n = 100$ to show the impact of the *Central Limit Theorem* (see Section 3.3) as n increases. The figures on the left are bar graphs of the sample data distribution for the first of the 100,000 simulated random samples of size n . With $\mu = 0.7$, a typical sample has a mode of 0, few if any observations above 3, and severe skew to the right. The sampling distributions are shown on the right. With random samples of size $n = 10$, the sampling distribution has somewhat of a bell shape but is still skewed to the right. (It is a re-scaling of a Poisson with mean 7, since adding 10 independent Poissons with mean 0.7 gives a Poisson with mean 7.) With $n = 100$, the sampling distribution is bell-shaped, has a more symmetric appearance, and is narrower because the standard error decreases as n increases.

By changing the function from the mean in the `apply` command, you can simulate sampling distributions of other statistics, such as we did for the sample median in Section 3.4.3. By changing the `rpois` argument, you can simulate sampling distributions for other probability distributions as well as other statistics.

A3.2 Monte Carlo Simulation

The *Monte Carlo (MC) method* is a way to use simulation to investigate characteristics of random variables, such as means and variances of their probability distributions.

Technically, MC methods are employed to approximate integrals (also high-dimensional) via simulations by expressing them as expected values of random variables. MC simulation plays an important role in statistics, especially in Bayesian statistics. Methods of MC sim-

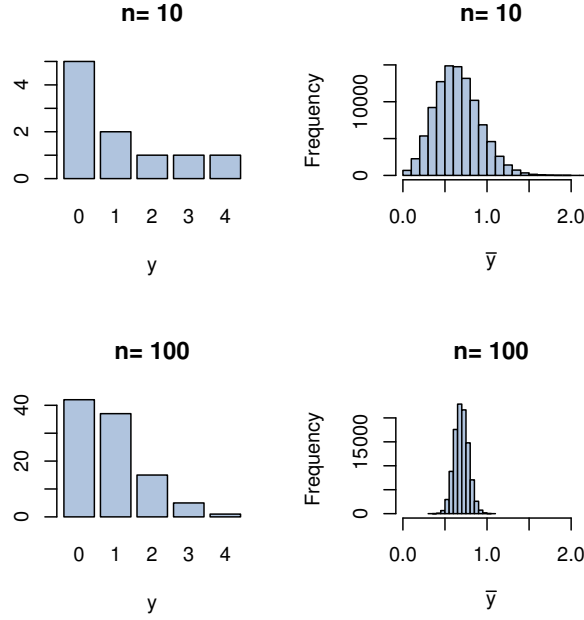


FIGURE A3.2: Bar plot of sample data distribution (left) and histogram of empirical sampling distribution of sample mean (right), for random samples of size $n = 10$ (upper figures) and $n = 100$ (lower figures) from a Poisson distribution with $\mu = 0.7$.

ulations and their properties is an extended field, which is not discussed here.¹ We restrict on the presentation of the very basic idea.

Monte Carlo (MC) method

For a function g and a random variable Y with probability function f , consider

$$E[g(Y)] = \begin{cases} \sum_y g(y)f(y), & Y \text{ discrete} \\ \int_y g(y)f(y)dy, & Y \text{ continuous.} \end{cases}$$

If Y_1, Y_2, \dots, Y_B are independent random variables sampled from f , then $\frac{1}{B} \sum_{i=1}^B g(Y_i)$ is the **Monte Carlo (MC) estimator** of $E[g(Y)]$.

For a parameter θ , estimated by $\hat{\theta}$, we can use the Monte Carlo method to estimate the variance of the sampling distribution of $\hat{\theta}$ for a random sample (Y_1, \dots, Y_n) . Here are the steps for a basic MC algorithm:

- (1) For $j = 1, \dots, B$, draw $\mathbf{y}^{(j)} = (y_{j1}, \dots, y_{jn})$ independently from the distribution of interest.
- (2) Use $\mathbf{y}^{(j)}$ to find $\hat{\theta}_j$ for sample j of size n , $j = 1, \dots, B$.
- (3) The Monte Carlo estimate of $E(\hat{\theta})$ is $\bar{\theta} = \frac{1}{B} \sum_{j=1}^B \hat{\theta}_j$.

¹see for example Rubinstein RY and Kroese TP (2017), *Simulation and the Monte Carlo Method*, 3rd ed., Wiley.

- (4) The Monte Carlo estimate of $\text{var}(\hat{\theta})$ is $\frac{1}{B} \sum_{j=1}^B (\hat{\theta}_j - \bar{\theta})^2$.

A3.2.1 MC Estimation of a Binomial Success Probability π

The MC algorithm will be illustrated for the estimation of the success probability π of a binomial experiment of fixed number n of Bernoulli trials (Y_1, \dots, Y_n) , based on a realized sample (y_1, \dots, y_n) of binary (0, 1) data. In this we know that the MLE of π is $\hat{\pi} = \frac{Y}{n}$, with $Y = \sum_{i=1}^n Y_i$ (see Section 4.2.2) while $\text{var}(\hat{\pi}) = \frac{\pi(1-\pi)}{n}$. The R function below computes the MC estimates of π and $\text{var}(\hat{\pi})$. It also visualizes the central limit theorem, since it produces the histogram of the $\hat{\pi}$ values computed over the B simulated samples along with the normal distribution $\mathcal{N}(\pi, \frac{\pi(1-\pi)}{n})$, which is coming closer to the histogram as n increases.

```
> CLT_binom <- function(B,n,p) {
  # B: number of iterations used to approximate the distribution of Xmean
  # n: sample size
  # p: success probability pi
  Y <- rbinom(B,n,p)
  Ymean <- Y/n # vector (length B) with the p-estimates: Algorithm 1 (2)
  var.mean <- p*(1-p)/n # variance of the estimator of p
  p.MC <- mean(Ymean) # Monte Carlo estimate of p
  varp.MC <- var(Ymean) # MC variance estimate of var.mean
  h <- hist(Ymean, col = "gray", probability=TRUE, main=paste("n=",n))
  xfit<-seq(0, 1,length=5000)
  yfit<-dnorm(xfit,mean=p,sd=sqrt(p*(1-p)/n))
  gr <- lines(xfit, yfit, col="blue",lwd=2)
  list(var.mean=var.mean, p.MC=p.MC, varp.MC=varp.MC) }
```

An implementation of this algorithm for the case of binomial distributions with $\pi = 0.3$ and sample sizes $n = 10, 30, 100$ and 1000 , based on 100000 replications, is shown next while the derived plots are given in Figure A3.3.

```
> par(mfrow=c(2,2)) # multiple graphs layout in a 2x2 table format
> CLT_binom(100000, 10, 0.3)
$var.mean
[1] 0.021
$p.MC
[1] 0.299555
$varp.MC
[1] 0.02093151
> CLT_binom(100000, 30, 0.3)
$var.mean
[1] 0.007
$p.MC
[1] 0.3002177
$varp.MC
[1] 0.0070047
> CLT_binom(100000, 100, 0.3)
$var.mean
[1] 0.0021
$p.MC
[1] 0.2998553
$varp.MC
[1] 0.002109033
> CLT_binom(100000, 1000, 0.3)
$var.mean
[1] 0.00021
$p.MC
[1] 0.3000596
$varp.MC
[1] 0.0002109015
```

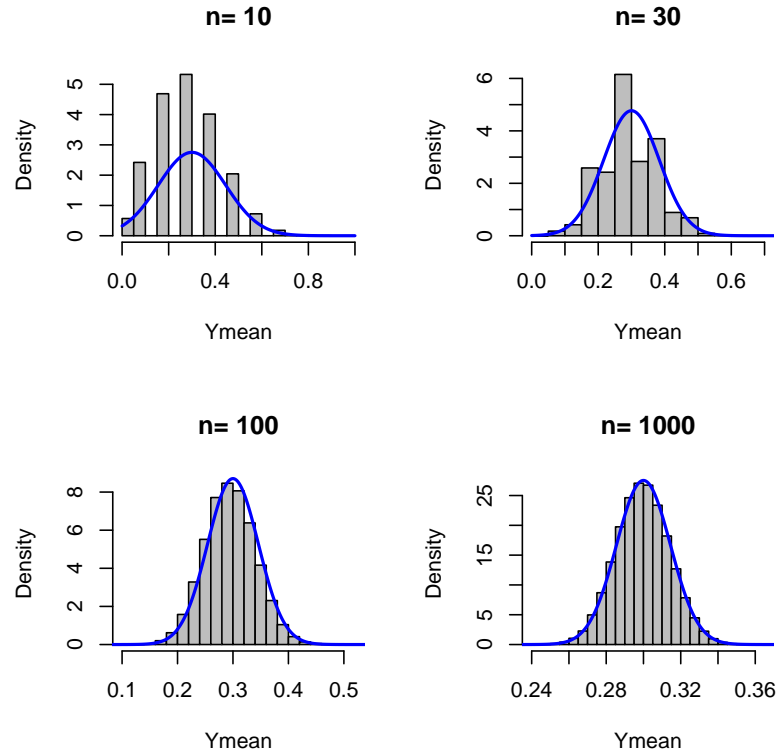


FIGURE A3.3: Histograms of estimates of π based on 100000 replicates simulated from binomial distributions with $\pi = 0.3$ and n equal to 10, 30, 100 and 10000, along with the pdf of $\mathcal{N}(0.3, 0.21/n)$.

We illustrated here the MC approach, though unnecessary in this case since closed form expressions exist for the estimator and its variance. However, although we have a standard error formula for a sample mean, most probability distributions do not have a simple standard error formula for a sample median, so Monte Carlo approximation is useful to approximate it.

A3.2.2 MC Estimation of the Sample Median

Let's investigate how the standard error of a sample median compares to that of a sample mean in sampling from a normal distribution. We create a function to take a large number B of simple random samples of size n from a $N(\mu, \sigma^2)$ distribution, calculate the median for each sample, and then find the standard deviation of those sample medians:

```
> sdmed <- function(B, n, mu, sigma){
  medians <- rep(0, B)
  for(i in 1:B){
    y <- rnorm(n, mu, sigma)
    medians[i] <- median(y) }
  sd(medians)
}
```

If IQ scores have a $N(100, 16^2)$ distribution, then for a simple random sample of size $n = 100$, \bar{Y} has a standard error of $\sigma/\sqrt{n} = 16/\sqrt{100} = 1.60$. Let's approximate the standard error of the sample median by simulating $B = 1,000,000$ random samples of size $n = 100$ each, finding the 1,000,000 sample medians, and then finding the standard deviation of their values:

```
> sdmed(1000000, 100, 100, 16) # B=1000000, n=100, mu=100, sigma=16
[1] 1.990409
```

We approximate that the standard error is 1.99. In fact, for sampling from a normal population, the standard error of the sample median is 25% larger than the standard error of a sample mean. The sample mean tends to be closer than the sample median to the joint population mean and median of a normal distribution. Apparently the sample mean is a better estimator than the sample median of the center of a normal distribution. Constructing good estimators is the subject of Chapter 4.

We provide next a function for estimating via MC the median of a random sample from a gamma distribution with shape and rate parameters equal to s and r , respectively. We illustrate the algorithm for $s = 2$ and $r = 0.5$. If r and s are unknown, we replace them with their estimates.

```
> median_gamma <- function(B,s,r) {
  # B: number of iterations used in the MC procedure
  # s: shape parameter
  # r: rate parameter

  Y <- rgamma(B,shape=s, rate=r)
  Ymedian <- median(Y)          # vector (length=B) with the median-estimates
  median.MC <- mean(Ymedian)    # Monte Carlo estimate of median
  return(median.MC) }

# Illustration:
> median_gamma(100000,2,0.5)
[1] 3.35453
```

In case the *cdf* F is unknown, the MC algorithm does not apply and an alternative way to estimate $\text{var}(\hat{\theta}_n)$ is through the bootstrap (see Section 4.6 and A4.5).

CHAPTER 4: R FOR ESTIMATION

The `maxLik` package has a function for ML estimation when you supply the log-likelihood function to be maximized. Output includes the standard errors of the ML estimates.

A4.1 Confidence Intervals for Proportions

As shown in Section 4.3.4, several confidence intervals (CIs) for a proportion are available in R with the function `ciAllx` in the `proportion` package. This is a comprehensive package for constructing confidence intervals for a binomial probability π . Beyond the approximate CIs illustrated for the example in Section 4.3.4, `proportion` facilitates the construction of further approximate CIs along with exact CIs (not addressed in this book) and Bayesian credible intervals.

An alternative package, also offering several methods for constructing CIs for binomial data, as well as options for determining their *actual coverage probability*, is the `binom` package. The example in Section 4.3.4 can be implemented in `binom` as shown below, providing also the sample proportion (i.e. the MLE of π), followed by the determination of their coverage probabilities.

```
> library(binom)
> binom.confint(778, 1497, conf.level=0.95, method="asymptotic") # Wald CI
  method x    n    mean   lower   upper
1 asymptotic 778 1497 0.5197061 0.4943974 0.5450148
> binom.coverage(0.5, 1500, conf.level = 0.95, method = "asymptotic")
  method p    n coverage
1 asymptotic 0.5 1500 0.9472287

> binom.confint(778, 1497, conf.level=0.95, method="wilson") # Score CI
  method x    n    mean   lower   upper
1 wilson 778 1497 0.5197061 0.4943793 0.544932
> binom.coverage(0.5, 1500, conf.level = 0.95, method = "wilson")
  method p    n coverage
1 wilson 0.5 1500 0.9472287
```

Notice that the score CI is called Wilson after the name of the statistician who originally proposed it.

Of course the sample size of 1497 is quite large and the two types of CIs do not differ in terms of their actual coverage probability which achieves the nominal level of 95%. Trying with a relatively small sample size ($n = 25$) and a probability $\pi = 0.25$, we can verify their poor performance (s. Section 4.3.2 and the fact that the score CI is preferable to the Wald CI (i.e. closer to 95%), confirmed by a simulation study. The conduction of a simulation study to compare their performance in terms of their actual coverage probability is straight forward in the `binom` package, as shown next. The output provided is the mean, lower and upper coverage probabilities based on M replications (here $M = 1000$).

```

> binom.coverage(0.25, 20, conf.level=0.95, method="asymptotic")
      method    p    n coverage
1 asymptotic 0.25  20 0.8948752
> binom.coverage(0.25, 20, .level=0.95, method="wilson")
      method    p    n coverage
1 wilson 0.25  20 0.9347622

> binom.sim(M=1000, n=20, p=0.25, conf.level=0.95, methods="asymptotic")
      mean    lower    upper    width
asymptotic 0.904 0.8857414 0.9222586 0.3680756
> binom.sim(M=1000, n=20, p=0.25, conf.level=0.95, methods="wilson")
      mean    lower    upper    width
wilson 0.947 0.9313254 0.9592535 0.3477152

```

One could further try the function `BinomCI` of the `DescTools` package, which provides a variety of methods for constructing CIs beyond the score and Wald CIs, like for example the Clopper–Pearson and the Agresti–Coull CIs. The score CI can also be obtained applying the very basic `prop.test` function (of the `stats` package in base R). Both these functions offer also the option of *one-sided* CIs, as demonstrated below. For example, the 99% score CI for the success probability of a therapy, based on observing 38 successes in a random sample of 45 patients is

```

> BinomCI(38,45, conf.level = 0.99, method = c("wilson", "wald"))
      est    lwr.ci    upr.ci
wilson 0.8444444 0.6629331 0.9374360 # score CI
wald   0.8444444 0.7052765 0.9836124

> prop.test(38,45,conf.level=0.99,correct=FALSE)$conf.int # score CI
[1] 0.6629331 0.9374360
attr(,"conf.level")
[1] 0.99
# one-sided CI:
> BinomCI(38,45, conf.level = 0.99, sides = "left", method = "wilson")
      est    lwr.ci    upr.ci
[1,] 0.8444444 0.6830924      1
> prop.test(38,45,conf.level=0.99,alternative ="greater",correct=F)$conf.int
[1] 0.6830924 1.0000000
attr(,"conf.level")
[1] 0.99

```

As discussed in Section 4.3.7, the sample size of a survey can be determined by controlling the error. The sample size n derivation described there can be easily calculated in R by a simple user-defined function, as shown next. Note the use of the `ceiling` function to set the sample size equal to the least integer greater than the resulting real value for n .

```

> nBinom <- function(error, p=0.5, alpha=0.05){
  # error: margin of error allowed
  # alpha: significance level, default set equal to 5%
  # p: guess for p, default set equal to 0.5
  # returns: the sample size n
  n <- ceiling(qnorm(alpha/2)^2*p*(1-p)/(error^2))
  return(n)}

```

The example in Section 4.3.7 is then implemented as follows.

```

> nBinom(0.04)          # p=0.50, 95% CI (alpha=0.05)
[1] 601
> nBinom(0.04,0.25)     # p=0.25, 95% CI (alpha=0.05)
[1] 451We have derived in Section 4.8.2
> nBinom(0.04,0.25,0.1) # p=0.25, 90% CI (alpha=0.10)
[1] 318

```

A4.2 Confidence Intervals for Means of Subgroups and Paired Differences

The confidence intervals for the mean considered here are based on the t distribution and are applicable if the data is a random sample from a normal distributed population (s. Section 4.4.2). However, they are robust also under violation of the normality assumption (s. Section 4.4.4). In the sequel the t CI are also referred as classical CI.

Furthermore if the sample is from a non-normal population (even discrete) but of sufficiently large sample size, then, due to the central limit theorem, the CI for the population mean is based on the standard normal distribution (s. (4.6) and Section 4.3.2). Thus the asymptotic CI for the mean is

$$\left[\bar{y} - z_{\alpha/2} \left(s / \sqrt{n} \right), \bar{y} + z_{\alpha/2} \left(s / \sqrt{n} \right) \right],$$

where \bar{y} and s are the sample mean and sample standard deviation, respectively. This CI can be easily derived in R by the following function.

```
> zCI<- function(x, conf.level=0.95){
  # x: vector of the sample values
  mean(x)+c(-1,1)*qnorm((1+conf.level)/2)*sqrt(var(x)/length(x)) }
```

We revisit the UN data file and construct a 95% CI for the mean GDP of these 42 nations, based (i) on the t distribution with 41 degrees of freedom and (ii) on the standard normal distribution, applying the above defined function `zCI`. As expected, since the degrees of freedom are relatively high and thus the t distribution approximates the standard normal, these two CIs are very close. The usual t -CI, along with the sample mean can also be derived by the `MeanCI` function of the `DescTools` package, which gives also the option of constructing bootstrap CIs as well as one-sided CIs.

```
> t.test(UN$GDP, conf.level=0.95)$conf.int
[1] 22.05311 31.60879
> zCI(UN$GDP)
[1] 22.19406 31.46784
> library(DescTools)
> MeanCI(UN$GDP, conf.level=0.95) # provides also the sampling mean
      mean   lwr.ci   upr.ci
26.83095 22.05311 31.60879
```

Let us construct 95% CI for the mean GDP for nations with low (≤ 2) and high (> 2) homicide rate. In this case, the sample sizes of the two groups are 25 and 17, respectively and thus we shall not consider the standard normal based CI. We demonstrate this derivation below, using the `group_by` function of `tidyverse` and summarizing the sample sizes and the CIs for the mean GDP. Note that since the `summarize` function returns just one value, we need to ask separately the lower and upper bound of the CI.

```
> library(tidyverse)
> UN %>% group_by(Homicide>2) %>% summarize(n=n(),
+      GDB_L=t.test(GDP)$conf.int[1],GDB_U=t.test(GDP)$conf.int[2])
# A tibble: 2 x 4
  'Homicide > 2'      n GDB_L GDB_U
  <lgl>          <int> <dbl> <dbl>
1 FALSE          25  26.8  38.7
2 TRUE           17  11.8  24.5

# alternative derivations:
> length(UN$GDP[UN$Homicide<=2]);length(UN$GDP[UN$Homicide>2]) # sample sizes
> t.test(UN$GDP[UN$Homicide<=2], conf.level=0.95)$conf.int
> t.test(UN$GDP[UN$Homicide>2], conf.level=0.95)$conf.int
```

In Section 4.5.3, a CI for the difference of the means of two independent populations is discussed and demonstrated on the Anorexia study, comparing the weight gain for the cognitive behavioral therapy group to that of the control group. The sample weight differences were saved under the vectors `cogbehav`, and `control`, respectively. Alternatively, a CI for the difference of two means can be constructed by the `MeanDiffCI` function of the `DescTools` package. This provides the t CI without assuming equality of the standard deviations of the two populations. Furthermore, it provides the estimate of the mean difference, the option of constructing one-sided CIs, as well as bootstrap CIs.

```
> library(DescTools)
> MeanDiffCI(cogbehav, control)
      meandiff      lwr.ci      upr.ci
3.9344828 -0.7044632  7.6182563
```

Asymptotic CIs for the difference of two proportions corresponding to two independent populations are derived in Section 4.5.5 for the difference in the after surgery complications proportions for two groups of patients, one having prayers and the other not. The CI provided by the classical `prop.test` function is a Wald CI while a score CI can be obtained in the `PropCIs` package by the `diffscoreci` function. The `DescTools` package has the `BinomDiffCI` function that provides Wald and score CI, along with a variety of other methods, having analogous options with the corresponding `BinomCI` function for one proportion.

```
> BinomDiffCI(315,604,304,597, method="wald") # (x1,n1,x2,n3)
      est      lwr.ci      upr.ci
[1,] 0.01231045 -0.04421536 0.06883625
> BinomDiffCI(315,604,304,597, method="score")
[1,] 0.01231045 -0.04398169 0.06871075
```

We have derived in Section 4.4.3 a 95% CI for the mean weight change of young girls suffering from anorexia after their therapy, for the group of cognitive behavioral (cb) therapy, by calculating the differences of weight (change = weight after - weight before) and considering then for the weight differences the classical t CI for a mean. This type of "before - after" studies are the so called *paired design* studies and the CI for this `MeanDiffCI` function of `DescTools`, both with the argument `paired=TRUE`. The latter has the option of constructing the CI based on the standard normal distribution (`method = "norm"`). Verify that the CI in Section 4.8.2 can equivalently be derived as given next while the normal CI is narrower than the t CI.

```
> t.test(Anor$after[Anor$therapy=="cb"], Anor$before[Anor$therapy=="cb"],
+        paired=TRUE)$conf.int
[1] 0.2268902 5.7869029
attr(,"conf.level")
[1] 0.95

> MeanDiffCI(Anor$after[Anor$therapy=="cb"], Anor$before[Anor$therapy=="cb"],
+            paired=TRUE, conf.level = 0.95, sides = "two.sided")
      meandiff      lwr.ci      upr.ci
3.0068966 0.2268902 5.7869029 # t CI

> MeanDiffCI(Anor$after[Anor$therapy=="cb"], Anor$before[Anor$therapy=="cb"],
+            paired=TRUE, method = "norm", conf.level = 0.95, sides = "two.sided")
      meandiff      lwr.ci      upr.ci
3.0068966 0.3319241 5.5965078 # normal CI
```

A4.3 The t and Other Probability Distributions for Statistical Inference

The most basic distributions used in parametric statistical inference, beyond the normal, are the t , χ^2 and F distributions, while for Bayesian approaches also the beta and gamma distribution have a special role as prior distributions. In order to understand the shape and properties of a particular family of distributions and the influence of their parameters values, the graphical visualization of their *pdfs* and *cdfs* is important. In Figure 4.5 for example, the *pdfs* of t distributions for a selection of degrees of freedoms, along with the standard normal pdf is provided. The R-code for deriving this plot is provided next.

```
# pdf of t distributions:

> x <- seq(-4, 4, length=100)
> phi <- dnorm(x)
> df <- c(1, 3, 8, 30)
> colors <- c("orchid2", "seagreen4", "blue", "orange", "red")
> labels <- c("df=1", "df=3", "df=8", "df=30", "normal")
> plot(x, phi, type="l", lty=2, lwd=2, xlab="y value", col="red",
       ylab="Probability density function", main="pdf of t distributions")
> for (i in 1:4){
  lines(x, dt(x,df[i]), lwd=2, col=colors[i]) }
> legend("topright", inset=.05, title="Distributions",
       labels, lwd=2, lty=c(1, 1, 1, 1, 2), col=colors)
```

The plot of the *cdfs* for these t distributions is derived by replacing in the code above *dnorm* and *dt* by *pnorm* and *pt*, respectively.

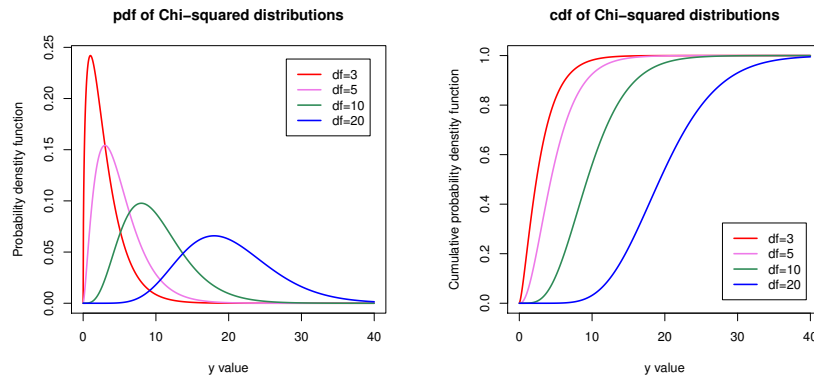
Furthermore, let us compare quantiles (0.01, 0.05, ..., 0.95, 0.99) of a t distributions with 30 and 90 degrees of freedom to the corresponding ones of the standard normal. We can verify that the quantiles of a t distribution approach those of a standard normal, as the degrees of freedom increase. For degrees of freedom higher than 30 they are quite close having the highest differences in the tails.

```
quant <- c(0.01,0.05,0.10,0.25,0.5,0.75,0.9,0.95,0.99)
qt(quant,30)-qnorm(quant)
[1] -0.130913668 -0.052407260 -0.028863460 -0.008265943 0.000000000
[6] 0.008265943 0.028863460 0.052407260 0.130913668
qt(quant,90)-qnorm(quant)
[1] -0.042149602 -0.017107457 -0.009477333 -0.002735750 0.000000000
[6] 0.002735750 0.009477333 0.017107457 0.042149602
```

The *pdf* and *cdf* of chi squared distributions are provided in Figure A4.1 and are derived adjusting the code provided above for the t distribution.

The R-code for producing Figure 4.7 is

```
> y = seq(0,70)
> plot(dchisq(y,10), type="l", xlab="Chi-squared", ylab="Probability
       density function")
> lines(y, dchisq(y, 20), col="red")
> lines(y, dchisq(y, 40), col="green")
> legend(40, 0.09, c("df=10","df=20","df=40"),lty=c(1,1,1),
       col=c("black","red","green"))
```

FIGURE A4.1: *pdfs* (left) and *cdfs* (left) of χ^2 distributions.

A4.4 Empirical Cumulative Distribution Function

Some inferential methods assume that the data come from a family of distributions, such as the normal assumption when we form t confidence intervals. Other inferential methods, such as the *bootstrap* method (Section 4.6) and *nonparametric* methods (Section 5.8), do not make such an assumption. When the *cdf* is completely unknown, the natural estimator is its *empirical cumulative distribution function*.

Empirical cumulative distribution function

For independent random variables Y_1, \dots, Y_n from a particular distribution, the *empirical cumulative distribution function* (*ecdf*) is

$$F_n(y) = \frac{1}{n} \sum_{i=1}^n I(Y_i \leq y),$$

where $I(\cdot)$ is the indicator function, $I(Y_i \leq y) = \begin{cases} 1, & \text{if } Y_i \leq y \\ 0, & \text{otherwise} \end{cases}$.

That is, $F_n(y)$ is the sample proportion of the n observations that fall at or below y .

We illustrate by generating a random sample of size $n = 10$ from the $N(100, 16^2)$ distribution of IQ values and constructing the empirical *cdf*:

```
> y <- rnorm(10, 100, 16)
> plot(ecdf(y), xlab="y", ylab="Empirical CDF", col="dodgerblue4") # ecdf = empirical cdf
> lines(seq(50, 150, by=.1), pnorm(seq(50,150,by=.1), 100, 16), col="red4", lwd=2)
```

Figure A4.2 shows the empirical *cdf* and the *cdf* of the normal distribution from which the data were simulated. The empirical *cdf* is the *cdf* of the discrete distribution having probability $1/n$ at each observation, so it is a step function. The figure also shows an empirical *cdf* for a random sample of size $n = 50$. As n increases, the empirical *cdf* converges uniformly over y to the true underlying *cdf*.¹

In medical applications that focus on the survival time of patients following some

¹This result is called the *Glivenko–Cantelli Theorem*, named after the probabilists who proved it in 1933.

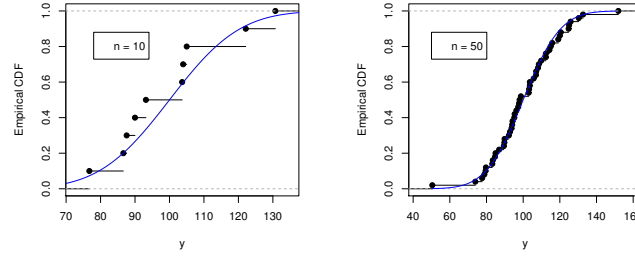


FIGURE A4.2: Empirical cumulative distribution functions for random samples of sizes $n = 10$ and $n = 50$ from the $N(100, 16^2)$ distribution, showing also the $N(100, 16^2)$ cdf.

procedure of diagnosis, it is common to focus on the empirical *survival function*, $S_n(y) = 1 - F_n(y)$. At a particular value y , it shows the proportion of the sample that had survival time greater than y . It can be informative to compare two treatments by plotting their survival functions next to each other (see for example the Kaplan-Meier estimators in Section 5.8).

A4.5 Nonparametric and Parametric Bootstrap

Bootstrap is one of the most important techniques in computational statistics, introduced by Efron² and belongs to the wider class of *resampling methods*. It is a simulation-based method for estimating *biases* and *variances* of statistical estimates as well as for obtaining *approximate confidence intervals*. Bootstrap applies to both parametric and nonparametric settings. We discuss the latter case first, which is based on the concept of the empirical cumulative density function *ecdf*.

In order to follow easier the bootstrap procedure and the associated algorithms, we summarize next basic notion in a compact form.

- Y_1, \dots, Y_n are *iid* random variables with *unknown* cdf F ,
- F_n is the empirical cdf (*ecdf*) of Y_1, \dots, Y_n ,
- $\mathbf{y} = (y_1, \dots, y_n)$ is an observed sample and \hat{F}_n the estimated *ecdf* based on \mathbf{y} ,
- of interest is a parameter θ , which can be any function of F ,
e.g., the mean $\mu = E(X)$, the variance $\sigma^2 = \text{var}(X)$ or the median $m = F^{-1}(0.5)$,
- $\hat{\theta}_n = T(Y_1, \dots, Y_n)$ is an estimator of θ ,
- $\hat{\theta}_{obs} = T(y_1, \dots, y_n)$ is the observed estimate of θ ,
- $\mathbf{Y}^* = (Y_1^*, \dots, Y_n^*)$ denotes a *bootstrap sample*, i.e. $Y_i^* \stackrel{iid}{\sim} F_n$, $i = 1, \dots, n$,
- $\hat{\theta} = T(Y_1^*, \dots, Y_n^*)$ is a bootstrap estimator of θ ,

²Efron B. (1979). Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, **7**, 1–26.

- B bootstrap samples are observed and $\mathbf{y}^{*(j)} = (y_1^{*(j)}, \dots, y_n^{*(j)})$ denotes the j -th observed bootstrap sample, $j = 1, \dots, B$,
- $\hat{\theta}^{(j)} = T(y_1^{*(j)}, \dots, y_n^{*(j)})$ is a bootstrap estimate, based on $\mathbf{y}^{*(j)}$, $j = 1, \dots, B$,
- the *cdf* of $\hat{\theta}_n$ is approximated by the *ecdf* of $\hat{\theta}$.

In practice, *bootstrap sampling* is implemented by sampling from $\mathbf{y} = (y_1, \dots, y_n)$ with *replacement*, which in R is straightforward:

```
# derivation of a sample of size n from the elements of x:
> xstar <- sample(x, n, replace=T)
```

For the bootstrap variance estimation, the MC Algorithm (see Section A3.2) is applied, replacing the unknown *cdf* F by its *ecdf* F_n . Hence, the variance $\text{var}(\hat{\theta}_n)$, based on F , is approximated by $\text{var}(\hat{\theta})$, based on F_n . The bootstrap estimate of $\text{var}(\hat{\theta})$ is

$$\hat{\sigma}_B^2 = \frac{1}{B} \sum_{j=1}^B \left(\hat{\theta}^{(j)} - \frac{1}{B} \sum_{j=1}^B \hat{\theta}^{(j)} \right)^2.$$

A4.5.1 Bootstrap Confidence Intervals (CIs)

Recall that if the estimator $\hat{\theta}_n$ is normally distributed (or the CLT holds for it), then

$$\frac{\hat{\theta}_n - \theta}{\sqrt{\text{var}(\hat{\theta}_n)}} \xrightarrow{d} \mathcal{N}(0, 1)$$

and an approximate $(1 - \alpha)100\%$ CI for θ is

$$\left[\hat{\theta}_n - z_{\alpha/2} \sqrt{\text{var}(\hat{\theta}_n)}, \hat{\theta}_n + z_{\alpha/2} \sqrt{\text{var}(\hat{\theta}_n)} \right].$$

Replacing $\text{var}_F(\hat{\theta}_n)$ by its bootstrap estimate, the bootstrap approximate normal CI is derived.

$(1 - \alpha)100\%$ Bootstrap Approximate Normal CI

The $(1 - \alpha)100\%$ **bootstrap approximate normal CI** is derived by estimating $\text{var}(\hat{\theta}_n)$ by the bootstrap variance estimate:

$$\left[\hat{\theta}_n - z_{\alpha/2} \hat{\sigma}_B, \hat{\theta}_n + z_{\alpha/2} \hat{\sigma}_B \right].$$

If $\hat{\theta}_n$ is not normally distributed and the CLT does not apply, then a bootstrap approximate normal CI is not adequate. There exist alternative bootstrap CIs, the simplest being a ‘percentile’ or a ‘pivotal’ bootstrap CI.

$(1 - \alpha)100\%$ Percentile Bootstrap CI

The $(1 - \alpha)100\%$ *percentile bootstrap CI* is given by

$$[\hat{\theta}_{(\alpha/2)}, \hat{\theta}_{(1-\alpha/2)}],$$

where $\hat{\theta}_{(\beta)}$ denotes the β -th empirical quantile of the bootstrap samples $\{\hat{\theta}^{(j)} : j = 1, \dots, B\}$.

The *pivotal* bootstrap CI assumes that the distribution of the random variable $\delta_n = \hat{\theta}_n - \theta$ is a distribution not involving θ and that it is approximated by the bootstrap distribution of $\delta = \hat{\theta} - \hat{\theta}_{obs}$. Then, the quantiles of the *cdf* of δ_n are approximated by the quantiles of the *ecdf* of δ .

 $(1 - \alpha)100\%$ Pivotal Bootstrap CI

The $(1 - \alpha)100\%$ *pivotal bootstrap CI*, also known as *empirical* or *basic* bootstrap CI, is given by

$$[2\hat{\theta}_{obs} - \delta_{(1-\alpha/2)}, 2\hat{\theta}_{obs} - \delta_{(\alpha/2)}],$$

where $\delta_{(\beta)}$ is the β -th empirical quantile of $\{\delta^{(j)} = \hat{\theta}^{(j)} - \hat{\theta}_{obs} : j = 1, \dots, B\}$.

Other variants of bootstrap CIs include ‘bias-corrected (BC)’ and ‘bias-corrected and accelerated (BCa)’ bootstrap CIs, having coverage probabilities closer to the nominal level than the percentile-based method.

The bootstrap method for variance estimation and derivation of confidence intervals can be easily implemented in R using the function `boot`, as illustrated for the example in Section 4.6.2, where bootstrap CIs for the median and the standard deviation of the books’ shelf time. Alternatively, it is straight forward to program manually the simulations required and construct a bootstrap CI. Thus, for the median shelf time of books, the 95% bootstrap CI is derived as follows.

```
> Books <- read.table("http://stat4ds.rwth-aachen.de/data/Library.dat", header=TRUE)
> n <- 54; nboot <- 100000
> Psample <- matrix(0, nboot, n) # matrix of nboot rows and n columns
> for (i in 1:nboot) Psample[i,] <- sample(Books$P, n, replace=TRUE)
> MedianBoot <- apply(Psample, 1, median) # finds median in each row
> quantile(MedianBoot, c(0.025, 0.975))
 2.5%   97.5%                # 95% percentile CI for median
 11    18.5
> sd(MedianBoot)                # standard deviation of the bootstrapped medians
[1] 2.196998
> SDBoot <- apply(Psample, 1, sd)
> quantile(SDBoot, c(0.025, 0.975)) # 95% percentile CI for standard deviation
 2.5%   97.5%
13.45806 35.79818
```

A4.5.2 Bootstrap Bias Estimation

The *unbiasedness* property of an estimator, introduced in Section 4.1.1 and discussed further in Section 4.1.2, is an important property. In some cases unbiased estimators do not exist. Quite often, though an unbiased estimator exists, some bias may be allowed in order to get an estimator with smaller variance. The bias–variance decomposition of the mean square error

of an estimator (4.1), shows that one can find a minimum mean squared biased estimator having smaller variance than the unbiased one. In machine learning and prediction modeling, the bias (variance) of prediction is decreasing (increasing) in model complexity and the problem of optimal decision for the model complexity is known as the *bias-variance tradeoff*. Thus, the bias of an estimator $\hat{\theta}_n$, $b(\hat{\theta}_n) = E(\hat{\theta}_n) - \theta$, is commonly of interest and quite often difficult to estimate. It can be estimated via bootstrap by

$$\hat{b}_n(\hat{\theta}_n) =_{F_n} (\hat{\theta}_n^*) - \hat{\theta}_{obs} ,$$

approximating $F_n(\hat{\theta}_n)$ using bootstrap samples $\hat{\theta}^{*(j)}$, $j = 1, \dots, B$, analogously to the bootstrap estimation of the $\text{var}_{F_n}(\hat{\theta}_n^*)$.

Bootstrap Bias Estimation

The bootstrap estimate of the bias $b(\hat{\theta}_n) = E(\hat{\theta}_n) - \theta$ is given by

$$\hat{b}(\hat{\theta}_n) = \left(\frac{1}{B} \sum_{j=1}^B \hat{\theta}^{(j)} \right) - \hat{\theta}_{obs} .$$

A4.5.3 Example: UN Data File

We shall implement the bootstrap method on the UN data set, applying the function `boot` in R, which provides as standard output the bootstrap estimates of the parameter of interest, its bias and standard error. The first argument of function `boot` specifies the sample data, on which the bootstrap should be based. They can be in a vector, matrix or data frame form. The second argument is the statistic function producing $\hat{\theta}_n$ that will be bootstrapped. This statistic is of dimension $p \geq 1$. The associated statistic function should have at least two arguments; the first is the data and the second is a vector of indices, frequencies or weights, used for selecting cases for each replication, i.e. for specifying the bootstrap sample.

We provide next examples of such (univariate) functions for estimation of the mean, median and variance, using indices, which is the default value of the parameter `stype` of `boot`. Notice that for these specific statistics, one can use directly the corresponding R functions `mean`, `median` and `sd`, as done in Section A.4.5 of the book's appendix. We provide these functions here to illustrate an example of user specified statistic functions.

The function `boot.ci`, applied on the bootstrapped sample of the parameter estimates provides the asymptotic confidence interval for the parameter of interest along with the basic, percentile and BCa bootstrap confidence intervals for the specified confidence level. Furthermore, the function `plot` can be used to derive the histogram and $Q-Q$ plot of $\hat{\theta}^{(1)}, \dots, \hat{\theta}^{(B)}$, i.e., the bootstrapped estimates of θ .

Here, we illustrate function `boot` for the median value of HDI, thus the first argument of `boot` is `UN[,3]` and the second argument is the function `y.median` (s. below).

```
> library(boot)
> UN <- read.table("http://stat4ds.rwth-aachen.de/data/UN.dat", header=T)
#..... examples of functions .....
> y.mean <- function(y, i){ # y: data vector; i: indices vector
  return(mean(y[i]))}
> y.median <- function(y, i){ # y: data vector; i: indices vector
  return(median(y[i]))}
> y.var <- function(y, i){ # y: data vector; i: indices vector
  return(var(y[i]))}
```

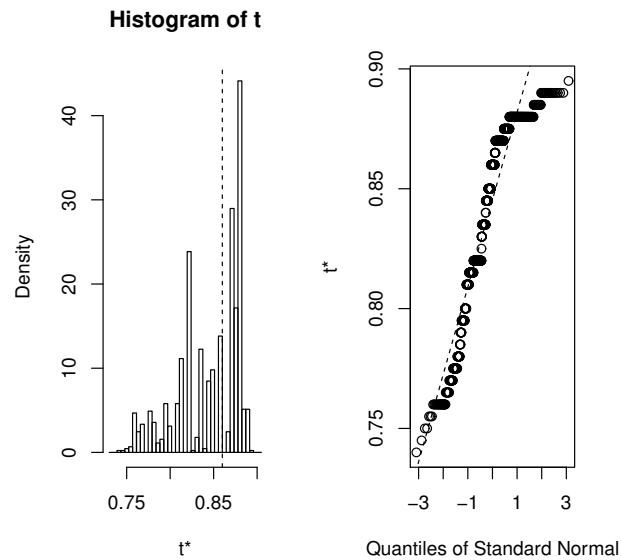


FIGURE A4.3: Bootstrap plots for the median of HDI of UN data.

```
#.....--.....
> set.seed(54321)                # for reproducibility of the example
> b_median = boot(UN[,3], y.median, R=1000); b_median
#..... output .....
ORDINARY NONPARAMETRIC BOOTSTRAP

Call:
boot(data = UN[, 3], statistic = y.median, R = 1000)

Bootstrap Statistics :
      original    bias      std. error   # t1*: bootstrapped estimate of the median
t1*      0.86 -0.014815  0.03493223
#.....
> boot.ci(b_median, conf=0.90); plot(b_median)
#..... output .....
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = b_median, conf = 0.9)

Intervals :
Level      Normal              Basic
90%   ( 0.8146,  0.9345 )   ( 0.8400,  0.9500 )

Level      Percentile          BCa
90%   ( 0.770,  0.880 )   ( 0.765,  0.880 )
Calculations and Intervals on Original Scale
Warning message:
In boot.ci(b_median, conf = 0.9) :
  bootstrap variances needed for studentized intervals
```

The produced plots are given in Figure A4.3.

Next, we provide an example of a statistic function defined over a bivariate random

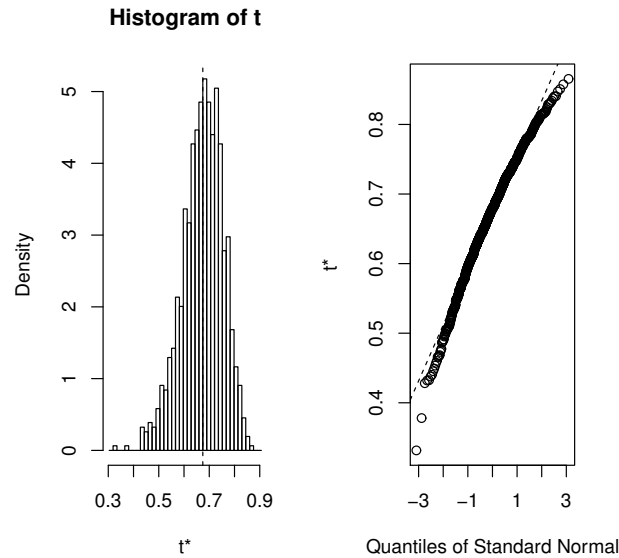


FIGURE A4.4: Bootstrap plots for the Pearson correlation between GDP and CO2 of UN data.

variable, namely the Pearson correlation, defined by the function `xy.cor` that follows. We illustrate it for the correlation between GDP and CO2 of the UN data.

```
> xy.cor <- function(y, i){
  v <- y[i,]
  return(cor(v[,1], v[,2], method='p'))}
> b_cor = boot(cbind(UN[,2], UN[,6]), xy.cor, R=1000)
> boot.ci(b_cor, conf=0.90); plot(b_cor)
#..... output .....
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = b_cor, conf = 0.9)

Intervals :
Level      Normal          Basic
90%   ( 0.5436, 0.8084 ) ( 0.5559, 0.8216 )

Level      Percentile      BCa
90%   ( 0.5274, 0.7931 ) ( 0.5042, 0.7811 )
Calculations and Intervals on Original Scale
Warning message:
In boot.ci(b_cor, conf = 0.9) :
  bootstrap variances needed for studentized intervals
```

The produced plots are to be seen in Figure A4.4.

A4.5.4 Parametric Bootstrap

The bootstrap can also be applied in a parametric set-up. For this, consider Y_1, \dots, Y_n be iid random variables with *cdf* F_θ , of known form, depending on an unknown parameter θ .

Let $\hat{\theta} = T(Y_1, \dots, Y_n)$ be an estimator of θ . The parametric bootstrap procedure is analogue to the nonparametric; the only difference lies on the fact that instead of obtaining the bootstrap samples from the *ecdf*, the estimated *cdf* $F_{\hat{\theta}_{obs}}$ is used instead. Note that if the parametric model is correctly specified, it will lead to better performances compared to the nonparametric bootstrap. For understanding the structure and functioning of a bootstrap algorithm, a ‘manual’ function for calculating a bootstrap CI is provided in the example below.

Consider a random sample Y_1, \dots, Y_6 of binomial distributed random variables with $Y_i \stackrel{iid}{\sim} \text{bin}(10, \theta)$, $i = 1, \dots, 6$, where $\theta \in [0, 1]$ unknown. Consider further an observed sample $\mathbf{y} = (5, 2, 3, 1, 4, 2)$, giving $\hat{\theta} = 0.283$. A parametric 95% percentile bootstrap CI can be derived as follows:

```
> alpha <- 0.05
> y = c(5, 2, 3, 1, 4, 2)
> m = 10 # known size of binomial
> n = length(y) # sample size (here: n=6)
> thetahat = mean(y)/m # MLE for theta=$\theta$
> nboot = 1000 # number of bootstrap samples to use
# nboot parametric samples of size n:
> tmpdata = rbinom(n*nboot, m, thetahat)
# bootstrap samples (organized in a matrix):
> bootstrapsample = matrix(tmpdata, nrow=n, ncol=nboot)
# computation of bootstrap estimates thetahat.b:
> thetahat.b = colMeans(bootstrapsample)/m
# derivation of percentile bootstrap CI:
> q.CI = quantile(thetahat.b, c(alpha/2, 1-alpha/2)); q.CI
      2.5%      97.5%
0.1833333 0.3837500
```

Parametric bootstrap is also implemented in the package `boot` by setting the parameter `sim="parametric"` (the default value is "ordinary"). In this case, the function used to generate the data is given in the `ran.gen` function. The second argument of `ran.gen` is set in the argument `mle=`, which usually will be the vector of maximum likelihood estimates of the parameters computed on the original data.

Revisiting the example on the median of a gamma distribution in Section A3.2.2, assume that \mathbf{y} is a realization of a random sample $Y = (Y_1, \dots, Y_{25})$ of 25 *iid* random variables from a gamma distribution with shape and rate parameters $s = 2$ and $r = 0.5$, respectively. Then, based on \mathbf{y} , the bootstrap estimate of the median of a gamma (2,0.5) distribution and its standard error can be computed as follows. In this case, since we assume the parameters to be known, we provide these values in argument `mle=` and the second argument of function `ran.gen`.

```
> library(boot)
# ..... required functions .....
> y.median <- function(y, i){ # y: data vector; i: indices vector
  return(median(y[i]))}
> gamma.rg <- function(y,p){ # function to generate random gamma variates
  rgamma(length(y), shape=p[1], rate=p[2])}
#.....
> y <- c(5.88,5.55,5.40,1.83,2.31,1.32,1.52,6.79,4.99,3.87,1.21,10.44,
  3.71,1.68,2.53,5.40,0.17,9.00,1.41,3.37,2.99,1.68,1.73,6.43,4.16)
> s <- 2; r <- 0.5
> p <- c(s,r)
> gamma_bootmed = boot(y, y.median, R=10000, sim = "parametric",
  ran.gen=gamma.rg, mle=p)
> gamma_bootmed
#..... output .....
PARAMETRIC BOOTSTRAP
```

```

Call:
boot(data = x, statistic = x.median, R = 10000, sim = "parametric",
      ran.gen = gamma.rg, mle = p)

Bootstrap Statistics :
      original      bias    std. error
t1*      3.37 0.02498518   0.6300741
#.....
# Bootstrap estimate of the median of gamma(2,0.5):
> 3.37 - 0.02      # =sample median - bias
[1] 3.35

```

The B sample medians of the simulated bootstrap samples are saved under the vector `gamma_bootmed$t`. Thus the bootstrap estimate of the median in the example above can equivalently be computed as follows. You may also verify the bootstrap estimate of the standard error of the median estimator.

```

> round(median(gamma_bootmed$t),2)
[1] 3.35
> round(sd(gamma_bootmed$t),2)
[1] 0.63

```

A4.6 Bayesian HPD Intervals Comparing Proportions

Section 4.7.6 introduced the highest posterior density (HPD) interval as an alternative to using posterior quantiles with equal tail (EQT) areas. The latter are very easy to derive but one situation in which we recommend using the HPD interval instead of the EQT interval is when the posterior *pdf* is monotone increasing or decreasing from the boundary of the parameter space, or if it is highly skewed. If the posterior distribution is symmetric, then these two types of intervals coincide.

For example, in estimating a proportion π , suppose that all n trials are successes or all n trials are failures. When $y = n$ and we use a Jeffreys or a uniform prior for π , $f(\pi | y)$ is monotone increasing from 0 to 1. It is not then sensible to exclude 1.0 and nearby values from the posterior interval. Rather than forming an interval from the 2.5 to 97.5 percentiles of the posterior distribution, we use the interval from the 5.0 percentile to 1.0. When $y = 0$, $f(\pi | y)$ is monotone decreasing from 1 to 0, and a sensible posterior interval goes from 1.0 to the 95.0 percentile. For estimating a difference $\pi_1 - \pi_2$ with Jeffreys or more diffuse priors, the posterior density is monotone when one of the samples has all successes and the other sample has all failures.

A caution: Unlike EQT intervals, HPD intervals are not invariant under nonlinear transformations. We illustrate with π_1/π_2 , sometimes called the *risk ratio* or *relative risk*. If a 95% HPD CI for π_1/π_2 is (L, U), the 95% HPD CI for π_2/π_1 is not (1/U, 1/L). Furthermore note that when the posterior density has two or more modes, then the HPD region (credible set) may not be a coherent interval but a union of disjoint intervals. EQT interval are computationally more convenient and keep the ‘invariance under transformation’ but may have worse coverage probability than the HPD intervals.

Consider the clinical trial example in Section 4.7.5 in which the 11 patients allocated to the experimental treatment were all successes and the only patient allocated to the control treatment was a failure. With uniform prior distributions, the posterior distributions were

beta(12.0, 1.0) for π_1 and beta(12.0, 1.0) for π_2 . For inference about the risk ratio, we obtain with simulation:

```
> library(PropCIs) # EQT intervals
> rrci.bayes(11, 11, 0, 1, 1.0, 1.0, 1.0, 0.95, nsim = 1000000)
[1] 1.078 73.379 # EQT interval for pi1/pi2
> rrci.bayes(0, 1, 11, 11, 1.0, 1.0, 1.0, 0.95, nsim = 1000000)
[1] 0.01363 0.92771 # EQT for pi2/pi1; endpoints (1/73.379, 1/1.078)
> library(HDInterval) # HPD interval for ratio of probabilities
> pi1 <- rbeta(1000000, 12.0, 1.0) # random sample from beta posterior
> pi2 <- rbeta(1000000, 1.0, 2.0)
> hdi(pi1/pi2, credMass=0.95)
      lower      upper
0.6729 36.6303
> hdi(pi2/pi1, credMass=0.95)
      lower      upper
7.820e-07 8.506e-01 # quite different from (1/36.63, 1/0.67)
```

The HPD 95% interval for π_1/π_2 is (0.673, 36.630). Taking reciprocals, this would suggest (0.027, 1.486) as plausible values for π_2/π_1 , but the HPD 95% interval is (0.000, 0.851). The inference then depends on which group we identify as Group 1 and which we identify as Group 2! Because of this, we prefer EQT intervals for nonlinear functions of parameters.

When you can specify the posterior distribution, the HPD interval is also available in R with the `hpd` function of the `TeachingDemos` package. The `LearnBayes` package is a collection of functions helpful in learning the Bayesian approach to statistical inference. Albert (2009) is a related excellent introduction to applied Bayesian analysis and computations.

A4.7 Example: HPD and EQT Intervals for a Binomial Success Probability

Bayesian point and interval estimation for a binomial π are illustrated on an example in Section 4.7.3. There, the `proportions` package was used for deriving the HDP intervals while it was demonstrated that the EQT intervals are easily delivered by the `qbeta` function. It was also verified that for large sample size n and a vague prior the Bayesian credible intervals are practically identical to the corresponding frequentist t based CI.

Here, we shall consider an example of small sample size and use the `hpd` function of the `TeachingDemos` package for computing the HPD interval, which is not restricted only to the beta distribution while it works also for empirical posterior distributions. This package, along with `LearnBayes`, are implemented and illustrated in Albert(2009), which is an excellent first reading in applied Bayesian analysis and computations. `LearnBayes` provides further flexible options for the choice of priors.

Consider a binomial random variable Y with parameters $n = 10$ and unknown success probability $\pi \in (0, 1)$. We have interest in the 95% EQT and the 95% HPD credible intervals in case we observed (i) $y = 0$, (ii) $y = 10$ and (iii) $y = 3$ successes, using as a prior a beta with $\alpha = \beta = 1/2$ (Jeffreys).

We shall deal first with the extreme cases (i) $y = 0$ and (ii) $y = 10$. The pdf of the corresponding prior and posterior beta densities are provided in Figure A4.5. The 95% HPD credible intervals are marked on the horizontal axis. Note that in this case the posterior is monotone decreasing in case (i) and increasing in (ii). Hence the HPD interval is on the left or right border of the parameter space for (i) and (ii), respectively. In these cases, the variance is estimated equal to zero and the classical CIs do not exist.

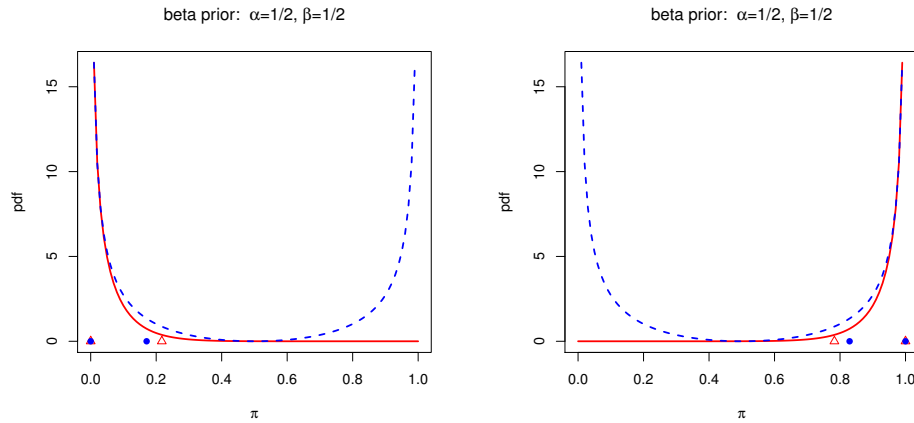


FIGURE A4.5: The beta (1/2,1/2) prior (dotted line) and the beta posterior (solid line) for parameter π , when $n = 10$ and $y = 0$ (left) or $y = 10$ (right). The 95% EQT and the 95% HPD intervals are marked on the x-axis by red triangles and blue bullets, respectively.

For case (iii) $y = 3$, the posterior is initially increasing and then decreasing; the corresponding graph is provided in Figure A4.6.

The code specifying the posterior, deriving the HPD credible intervals and producing these plots follows for (i). For the other cases, we simply need to change $x \leftarrow$ accordingly and repeat all the other commands.

```
> library(TeachingDemos)
> alpha <- 0.05 # significance level
> a0 <- 1/2; b0 <- 1/2 # beta prior parameters
> n <- 10 # n: number of trials
> x <- 0 # x: no. of successes in case (i)

> a <- a0+x; b <- b0+n-x # parameters of the posterior
# (1-alpha) Credible Intervals:
> q <- qbeta(c(alpha/2, 1-alpha/2), a, b); q # EQT
[1] 4.789043e-05 2.171963e-01
> h <- hpd(qbeta, shape1=a, shape2=b, conf=1-alpha); h # HPD
[1] 3.083137e-18 1.707731e-01

# Plots:
> prior <- function(p,a,b) { # beta prior pdf
  dens<-function(p) dbeta(p,a,b)
  return(dens)}
> post<- function(p,a,b,x,n) { # beta posterior pdf
  dens<-function(p) dbeta(p,x+a,n-x+b)
  return(dens)}
> y0=c(0,0) # (needed in plots)
> p1<-plot(post(p,a0,b0,x,10), col="red", lwd=2, xlim=c(0,1),type="l",
  xlab=expression(paste(pi)), ylab="pdf", main=expression(paste(
    "beta prior: ", alpha,"=1/2, ", beta,"=1/2")))
> points(q,y0, pch=2, col="red");
## changing q by h, you get on the plot the HPD bounds:
> points(h,y0, pch=16, col="blue");
> par(new=T)
> plot(prior(p,a0,b0),axes=F,xlab="",ylab="", lwd=2, lty=2, col="blue");
> par(new=F)
```

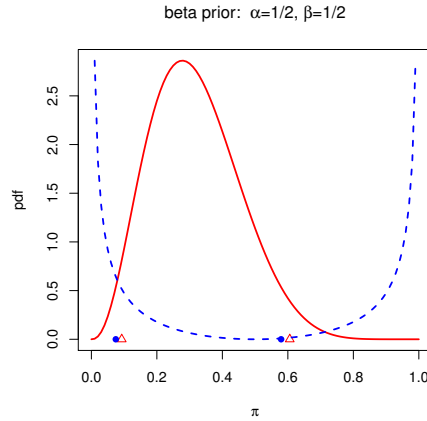



FIGURE A4.6: The beta (1/2,1/2) prior (dotted line) and the beta posterior (solid line) for parameter π , when $n = 10$ and $y = 3$. The 95% EQT and the 95% HPD intervals are marked on the x-axis by red triangles and blue bullets, respectively.

Some times the prior information is given in terms of the expected value and the standard deviation of the parameter of interest. This information can be transformed to information in terms of the parameters of the beta prior via the following functions.

```
> priorpar <- function(Ep,SDp) {
  # Prior knowledge for expected mean (Ep) and SD (SDp) of pi
  a <- (Ep/SDp^2)*(Ep*(1-Ep)-SDp^2); b <- a*(1-Ep)/Ep
  par <- c(round(a,3),round(b,3))
  return(par)}

# Example:
> priorpar(0.5, 0.3536) # corresponds to a=b=1/2
[1] 0.5 0.5
```

A4.8 Example: EQT Intervals for Normal Means

Let us revisit the anorexia study example, for which the posterior distribution for the mean change in weight for the girls taking part in the cognitive behavioral therapy was derived, assuming that the variance was unknown and imposing a non-informative conjugate normal/gamma prior on the parameters (μ, σ^2) . Now we assume for the same problem an improper prior, i.e. $f(\sigma^2) \propto 1/\sigma^2$. In this case, the joint posterior is of a normal/inverse chi-square form where the posterior of μ , conditional on σ^2 is distributed $N(\bar{y}, \frac{\sigma^2}{n})$ and the marginal posterior of σ^2 is distributed $(n-1)s^2\chi_{n-1}^{-2}$, where χ_{n-1}^{-2} is the inverse chi-square distribution with $n-1$ degrees of freedom and s^2 is the sample variance. For details and implementation in R we refer to Albert (2009, p. 57–58). Following his algorithm, we provide next the R-code for our example. As expected, the results are very close to those based on a non-informative conjugate prior. In Figure A4.8 we provide the plot of the simulated posterior pdf of μ along with the theoretical posterior, with σ^2 estimated by $\hat{\sigma}^2 = (n-1)s^2/(n-3)$, since $E(\chi_{df}^{-2}) = 1/(df-2)$, for $df > 2$. Note that for our sample

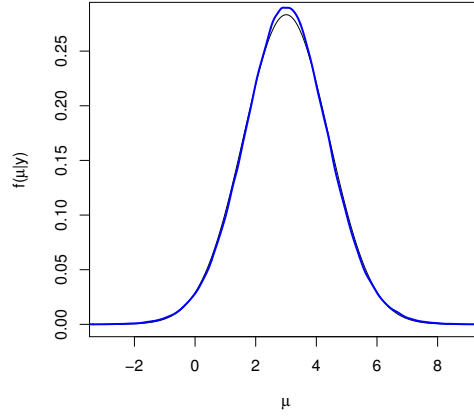


FIGURE A4.7: The simulated posterior density function (blue) for μ , the expected weight change for the girls of the "cb" therapy group, conditional on σ^2 , along with the pdf of a $N(\bar{y}, \frac{\sigma^2}{n})$ (black), i.e. the theoretical posterior density conditional on σ^2 , with estimated σ^2 .

$\hat{\sigma}^2 = 57.523$ (compare to 7.511, the mean of the simulated values from the marginal posterior of σ^2).

```
> change <- Anor$after - Anor$before
> y <- change[Anor$therapy=="cb"]; n=length(y)
> S = sum((y-mean(y))^2)
> sigma2 <- S/rchisq(500000,n-1)
> mu <- rnorm(500000, mean=mean(y), sd=sqrt(sigma2)/sqrt(n))
> Anor%>%summarize(n=n, mu.post=mean(mu), sd.mu=sd(mu),
+   sd.post=mean(sqrt(sigma2)), sd.sd=sd(sqrt(sigma2)))
  n mu.post sd.mu sd.post sd.sd
1 29 2.997451 1.412274 7.511078 1.047082
> quantile(mu, c(0.025,0.975))
  2.5%    97.5%
0.2059197 5.7869275
> quantile(sqrt(sigma2), c(0.025,0.975))
  2.5%    97.5%
5.795127 9.895656

# posterior density plot:
> b <- sqrt(S/(n-3))/sqrt(n) # simulated:b1 <- sqrt(mean(sigma2))/sqrt(n)
> x <- seq(-3, 9, length=1000)
> fx <- dnorm(x, mean=mean(y), sd=b)
> plot(x, fx, type="l", lwd=1, xlab=expression(paste(mu)),
+   ylab=expression(paste("f(",mu,"|y)")))
> lines(density(mu, n=2^14), col="blue", lwd=2)
```

Analogously, the posterior distribution for the difference $\mu_1 - \mu_2$ of the means of two independent normal populations with common variance σ^2 and improper priors can be derived as follows. Assume for prior $f(\mu_i) \propto 1$, $i = 1, 2$ and $f(\sigma^2) \propto 1/\sigma^2$. Let further \bar{y}_i , $i = 1, 2$, be the sample means and $s^2 = \frac{(n_1-1)s_1^2 + (n_2-1)s_2^2}{n_1+n_2-2}$ the pooled estimate of σ^2 , where s_i^2 and n_i denote the corresponding sample variance and sample size, respectively. Then, the posterior of μ_i , conditional on σ^2 , is distributed $N(\bar{y}_i, \sigma^2/n_i)$ and the marginal posterior of σ^2 is distributed $(n_1 + n_2 - 2)s^2 \sim \chi_{n_1+n_2-2}^{-2}$.

Reconsidering the example in Section 4.5.3, we shall derive the EQT interval for the difference of the mean weight gains for the cognitive behavioral therapy and the control groups. Recall that the 95% t CI is (-0.68, 7.59).

```
> cogbehav <- Anor$after[Anor$therapy=="cb"]-Anor$before[Anor$therapy=="cb"]
> control <- Anor$after[Anor$therapy=="c"]-Anor$before[Anor$therapy=="c"]
> n1 <- length(cogbehav); n2 <- length(control)

> S = sum((cogbehav-mean(cogbehav))^2) + sum((control-mean(control))^2)
> sigma2 <- S/rchisq(500000,n1+n2-2)
> mu1 <- rnorm(500000, mean=mean(cogbehav), sd=sqrt(sigma2)/sqrt(n1))
> mu2 <- rnorm(500000, mean=mean(control), sd=sqrt(sigma2)/sqrt(n2))

> Anor%>%summarize(n1=n1, n2=n2, difmu.post=mean(mu1-mu2),
  sd.difmu=sd(mu1-mu2),sd.post=mean(sqrt(sigma2)),sd.sd=sd(sqrt(sigma2)))

  n1 n2 difmu.post sd.difmu sd.post sd.sd
1 29 26 3.460271 2.105025 7.747604 0.7681589

> quantile(mu1-mu2, c(0.025,0.975))
      2.5%      97.5%
-0.6865898  7.6012047

> quantile(sqrt(sigma2), c(0.025,0.975))
      2.5%      97.5%
6.420452  9.426159

# posterior density plot:
> b <- sqrt(S/(n1+n2-4))*sqrt(1/n1+1/n2) # based on E(sigma^2) of posterior
> x <- seq(-3, 11, length=1000)
> fx <- dnorm(x, mean=mean(cogbehav)-mean(control), sd=b)
> plot(x, fx, type="l", lwd=1, xlab=expression(paste(mu)),
  ylab=expression(paste("f(",mu,"|y)")))
> lines(density(mu1-mu2, n=2^14), col="blue", lwd=2)
```

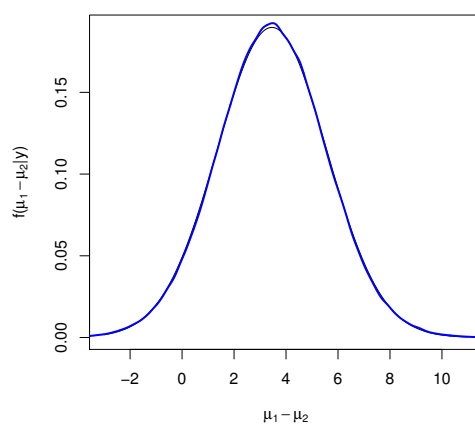


FIGURE A4.8: The simulated posterior density function (blue) for $\mu_1 - \mu_2$, the expected mean difference in weight gains for the "cb" and "control" groups, conditional on σ^2 , along with the pdf of $N(\bar{y}_1 - \bar{y}_2, \sigma^2(\frac{1}{n_1} + \frac{1}{n_2}))$ (black), i.e. the theoretical posterior density conditional on σ^2 , with estimated σ^2 .

5

CHAPTER 5: R FOR SIGNIFICANCE TESTING

A5.1 Bayes Factors and a Bayesian t Test

The book mainly presents classical ("frequentist") approaches of statistical inference. This appendix discusses more about Bayesian approaches.

In a Bayesian framework, the *Bayes factor* (BF) expresses the change in our relative belief about two hypotheses after we have observed the data. Let $P(H)$ and $P(H \mid \mathbf{y})$ denote the prior and posterior probabilities for a hypothesis H . The BF in favor of the alternative hypothesis is

$$BF_{10}(\mathbf{y}) = \frac{P(H_a \mid \mathbf{y})/P(H_0 \mid \mathbf{y})}{P(H_a)/P(H_0)} .$$

A very nice feature of the BF is that interchanging the role of the null and alternative hypothesis is straightforward, which is not the case in frequentist approaches. In particular, the BF in favor of the null hypothesis is simply given by

$$BF_{01}(\mathbf{y}) = 1/BF_{10}(\mathbf{y}) .$$

It can easily be verified that $BF_{10}(\mathbf{y})$ equals the ratio of the marginal distributions of the data under H_1 and H_0 , which are derived by integrating the likelihoods under H_1 and H_0 over the parameter θ . There is a kind of correspondence to the likelihood ratio test, since the likelihood ratio is computed by maximizing the likelihoods in terms of θ while the BF integrating over θ .

The BF was introduced by Jeffreys in 1961. The evaluation scale mostly used is an adjustment of the initial evaluation scale of Jeffreys, proposed by Kass and Raftery ¹:

$2 \ln B_{10}(\mathbf{y})$	$B_{10}(\mathbf{y})$	Evidence against H_0
0 - 2	1 - 3	negligible
2 - 6	3 - 20	positive
6 - 10	20 - 150	strong
> 10	> 150	very strong

We shall consider next Bayesian t tests for comparing the means of two independent normal distributed random variables with common variance, i.e. the Bayesian analogue to the test presented in Section 5.3.4. Various versions of Bayesian t tests have been proposed in the literature, offering options from very vague non-informative prior to strongly informative. These tests reparameterize the problem by expressing the means as deviations from the common mean μ under the null hypothesis, i.e. they consider that $\mu_1 = \mu + \frac{\sigma\delta}{2}$ and $\mu_2 = \mu - \frac{\sigma\delta}{2}$, where $\delta = \frac{\mu_1 - \mu_2}{\sigma}$ is the standardized effect size. Then the hypothesis testing problem becomes $H_0 : \delta = 0$ versus $H_0 : \delta \neq 0$, having a common nuisance parameter vector

¹Kass and Raftery (1995). Bayes factors, *Journal of the American Statistical Association*, 90, 773–795

(μ, σ) under both hypotheses, which facilitates the priors consideration and the Bayesian calculations. Under this set-up, the considered prior for the parameter vector under H_1 is $\pi_1(\mu, \sigma, \delta) = \pi_0(\mu, \sigma)\pi(\delta)$, where $\pi_0(\mu, \sigma)$ is the prior of the parameter vector under H_0 . Then a non-informative prior is used for the nuisance parameters $\pi_0(\mu, \sigma) \propto \sigma^{-1}$ while the decision for the prior on δ depends on the availability of prior information and differs among various considered tests. The predominant options is a Cauchy non-informative prior ² or a normal ³ that, depending on the choice of parameters may be informative or not.

Such Bayesian t tests can easily be performed in the **BayesFactor** package applying the `ttest.tstat` or the `ttestBF` function. The first requires just the value of the t -test statistic and the sample sizes n_1 and n_2 , while the second applies on the data and offers also the possibility to sample from the posterior and hence have options for further deliveries, like for example credible intervals or simulated posterior densities plots.

We reanalyze next the example of Sections 5.3.2 and 5.3.5, comparing cognitive behavioral and control therapies for anorexia, using for δ a non-informative Cauchy prior $\pi(\delta) = \text{Cauchy}(\delta; 0, \gamma)$ with scale $\gamma = \sqrt{2}/2$, which is the default option in **BayesFactor**. The R-code is provided below:

```
> y1 <- Anor$after[Anor$therapy=="cb"] - Anor$before[Anor$therapy=="cb"]
> y2 <- Anor$after[Anor$therapy=="c"] - Anor$before[Anor$therapy=="c"]
> t.anor <- t.test(y1, y2, var.equal=TRUE)
> t.anor
t = 1.676, df = 53, p-value = 0.09963 # classical t test
> library(BayesFactor)
%# requires the vectors "cogbehav" and "control" computed in Section 5.3.4
%> t.anor <- t.test(cogbehav, control, var.equal=TRUE)
> ttest.tstat(t = t.anor$statistic, n1=length(cogbehav), n2=length(control),
             simple = TRUE)

      B10
0.8630774
> ttestBF(x = cogbehav, y = control)
Bayes factor analysis
-----
[1] Alt., r=0.707 : 0.8630774 ±0.01% # r=sqrt(2)/2 (scale of Cauchy prior)

Against denominator:
  Null, mu1-mu2 = 0
---
Bayes factor type: BFindepSample, JZS

# Sample from the corresponding posterior distribution:
samples = ttestBF(x = cogbehav, y = control, paired=FALSE,
                  posterior = TRUE, iterations = 5000)

densplot(samples[, "mu"]) # Figure A.5.1(left)
quantile(samples[, "mu"], c(0.025, 0.975)) # EQT 95% CI
      2.5%      97.5%
-0.7615298  3.3525433

densplot(samples[, "delta"]) # Figure A.5.1(right)
quantile(samples[, "delta"], c(0.025, 0.975)) # EQT 95% CI
      2.5%      97.5%
-0.1116675  0.8850163
```

The estimated Bayes factor of $BF_{10}(\mathbf{y}) = 0.86$ shows only weak evidence against H_0 . The

²Rouder, Speckman, Sun, Morey, Iverson (2009). Bayesian t -tests for accepting and rejecting the null hypothesis, *Psychonomic Bulletin & Review*, 16, 225–237.

³Gönen, Johnson, Lu, Westfall (2005). The Bayesian two-sample t -test, *The American Statistician*, 59, 252–257.

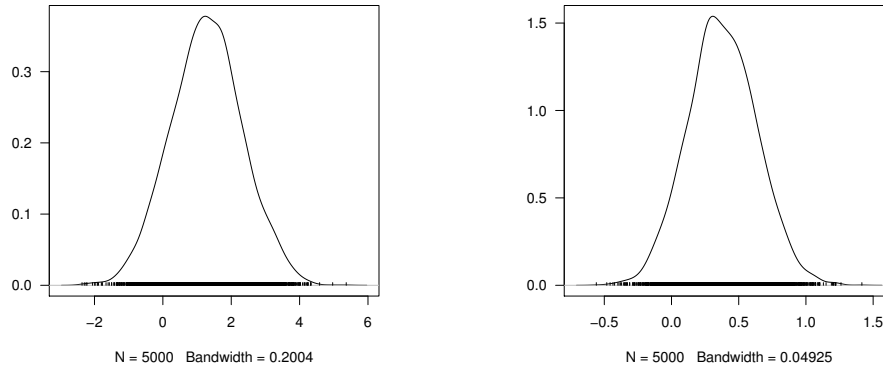


FIGURE A5.1: Simulated posterior density function for (i) the expected mean difference in weight gains for the ‘cb’ and ‘control’ groups $\mu_1 - \mu_2$ (left) and (ii) the associated standardized effect size δ (right), based on 5000 replications.

95% posterior interval for the standardized effect difference δ is $(-0.11, 0.89)$. Although some plausible values for δ are negative, the posterior probability of a negative value is only 0.064. (The classical one-sided P -value is $0.0996/2 = 0.05$.) Figure A5.1 plots the posterior densities of the difference $\mu_1 - \mu_2$ (left) and the effect size δ (right).

In an alternative recent approach⁴, the BF for a Bayesian test, corresponding to a t test with observed statistic value t_{obs} , under any proper prior for δ can be written as

$$BF_{10}(t_{obs}) = \frac{\int T_{df}(t_{obs}|\sqrt{n_\delta}\delta)\pi(\delta)d\delta}{T_{df}(t_{obs})},$$

where $T_{df}(t|c)$ denotes the density function of a t distribution with non-centrality parameter c and $T_{df}(t) = T_{df}(t|0)$. $BF_{10}(t_{obs})$ can easily be evaluated by numerical integration. Using the corresponding R functions provided in `informedTest_functions.R` and adjusting the associated example code (<https://osf.io/37vch/>) for the anorexia example, we can verify the value of BF_{10} and plot the posterior of the effect size along its prior, as shown in Figure A5.2.

```
> library(BayesFactor)
# requires the vectors "cogbehav" and "control" computed in Section 5.3.4

> source("informedTest_functions.R")
> rscale_def <- 1/sqrt(2)
> t <- t.anor$statistic

# BF for default prior Cauchy with scale=\sqrt(2)/2:
bf10_t(t = t, n1 = length(cogbehav), n2 = length(control),
      independentSamples = TRUE, prior.location = 0,
      prior.scale = rscale_def, prior.df = 1)$BF10

      t
0.8630774
#----- Figure A.5.2 (plot - default prior) -----
> xlim <- c(-0.8, 0.8)
```

⁴Gronau, Ly, Wagenmakers (2020). Informed Bayesian t -tests, *The American Statistician*, 74, 137–143.

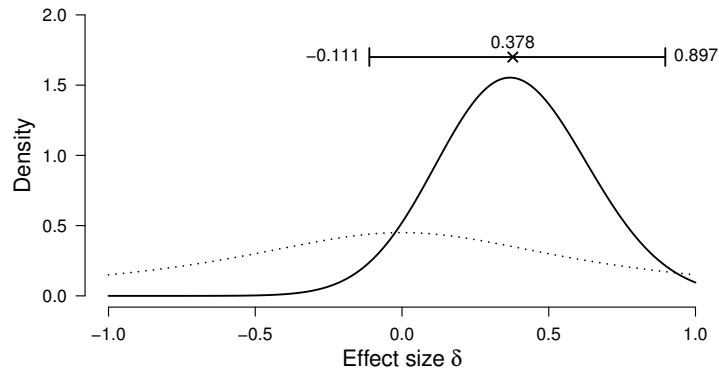


FIGURE A5.2: Prior (Cauchy with scale = $\sqrt{2}/2$) and posterior density functions for the effect size δ corresponding to the difference in weight gains for the 'cb' and 'control' groups.

```
> xticks <- pretty(xlim)
> xlim <- range(xticks)
> xx <- seq(xlim[1], xlim[2], length.out = 400)
> yy <- posterior_t(delta=xx, t=t, n1=length(cogbehav),
  n2=length(control), independentSamples=TRUE, prior.location=0,
  prior.scale=rscale_def, prior.df=1)
> priorLine <- dcauchy(x = xx, scale = rscale_def)
> ci_default <- ciPlusMedian_t(t=t, length(cogbehav), n2=length(control),
  independentSamples=TRUE, prior.location=0,
  prior.scale=rscale_def, prior.df=1)

> lwd <- 2; cexAxis <- 1.2; lwdAxis <- 1.2;
> cexYlab <- 1.5; cexXlab <- 1.5; cex.label <- 1.3; cex.ci <- 1.2
> ylim <- c(0, 2); yticks <- pretty(ylim); ylim <- range(yticks)
> yCI <- 1.7 # controls the position of the CI on the plot

> par(mar = c(5.6, 5, 4, 4) + 0.1, las = 1)
> plot(1, 1, xlim = xlim, ylim = ylim, ylab = "", xlab = "", type = "n",
  axes = FALSE)
> lines(xx, yy, lwd = lwd)
> lines(xx, priorLine, lwd = lwd, lty = 3)

> axis(1, at = xticks, cex.axis = cexAxis, lwd = lwdAxis)
> axis(2, at = yticks, cex.axis = cexAxis, lwd = lwdAxis)

> mtext(text = "Density", side = 2, las = 0, cex = cexYlab, line = 2.85)
> mtext(expression(paste("Effect size", ~delta)), side = 1, cex = cexXlab,
  line = 2.5)

> arrows(ci_default$ciLower, yCI, ci_default$ciUpper, yCI, angle=90, code=3,
  length = 0.1, lwd = lwd)
> points(ci_default$median, yCI, pch = 4, cex = 1.4, lwd = lwd)
> text(ci_default$ciLower, yCI, round(ci_default$ciLower, 3), cex=cex.ci, pos=2)
> text(ci_default$ciUpper, yCI, sprintf("%.3f", round(ci_default$ciUpper, 3)),
  cex=cex.ci, pos=4)
> text(ci_default$median, yCI, round(ci_default$median, 3), cex=cex.ci, pos=3)
```

Another Bayesian alternative to the t test, not based on the Bayes Factor, is provided in

the BEST package, which simulates from the posterior and provides HPD intervals (denoted in the package as HDI) for means, standard deviations and their differences, as well as the effective size. Its implementation is straightforward and provides handy output that is an `mcmc` object and can be thus analyzed further in `coda` (for a brief discussion on `mcmc` objects and `coda`, we refer to Section A6.3). Furthermore, the simulated draws from the posterior of each of the parameters in the model can be used for drawing inference for functions of our parameters, as for example the ratio σ_1/σ_2 , as illustrated below for our anorexia example.

```
install.packages("jags")
install.packages("rjags")
install.packages("coda")
install.packages("BEST")
library(BEST)
# requires the vectors "cogbehav" and "control" computed in Section 5.3.4

# all priors are the default:
BESTout <- BESTmcmc(cogbehav, control, parallel=FALSE)
plot(BESTout) # Figure A.5.3(left)
plot(BESTout, which="sd") # Figure A.5.3(right)
summary(BESTout)
```

	mean	median	mode	HDI%	HDIlo	HDIup	compVal	%>compVal
mu1	2.693	2.684	2.753	95	-0.112	5.53		
mu2	-0.588	-0.590	-0.615	95	-3.867	2.88		
muDiff	3.281	3.283	3.422	95	-1.089	7.61	0	93.2
sigma1	7.203	7.132	6.893	95	4.931	9.74		
sigma2	8.157	8.016	7.733	95	5.824	10.89		
sigmaDiff	-0.954	-0.922	-0.924	95	-4.422	2.38	0	28.4
effSz	0.428	0.429	0.445	95	-0.137	0.99	0	93.2

```
> varRatio <- BESTout$sigma1^2 / BESTout$sigma2^2 # ratio of variances
> median(varRatio)
[1] 0.7808525
> hdi(varRatio) # 95% HPD Interval for the ratio of variances
      lower      upper
0.2407684 1.6386263
```

A5.2 Simulating the Exact Distribution of the Likelihood-Ratio Statistic

Section 5.7 introduced likelihood-ratio tests, for which the test statistic $2\log(\ell_1/\ell_0) = 2(L_1 - L_0)$ has an approximate chi-squared distribution with $df = 1$, for large n . Under H_0 , it is often possible to simulate the true sampling distribution of the test statistic, so the chi-squared approximation is not needed.

We illustrate for the likelihood-ratio test of $H_0: \mu = \mu_0$ for the Poisson distribution. You can verify that the test statistic equals $2n[(\mu_0 - \hat{\mu}) - \hat{\mu} \log(\mu_0/\hat{\mu})]$, where $\hat{\mu} = \bar{y}$ is the ML estimate of μ under H_a . The following code simulates the exact distribution and compares it to the χ_1^2 distribution. We use a relatively small sample size of $n = 25$, when $\mu_0 = 5$, with $B = 100,000$ Monte Carlo simulations:

```
> LRT <- function(n, mu0, mu.hat){ # Poisson likelihood-ratio (LR) test statistic
  2*n*((mu0 - mu.hat) - mu.hat*log(mu0/mu.hat))
# Function returning vector of B values of LR test statistic
# for the B simulated Poisson(mu0) samples of size n:
> simstat <- function(B, n, mu0){ y <- rep(-1,B) # simulating Poisson
```

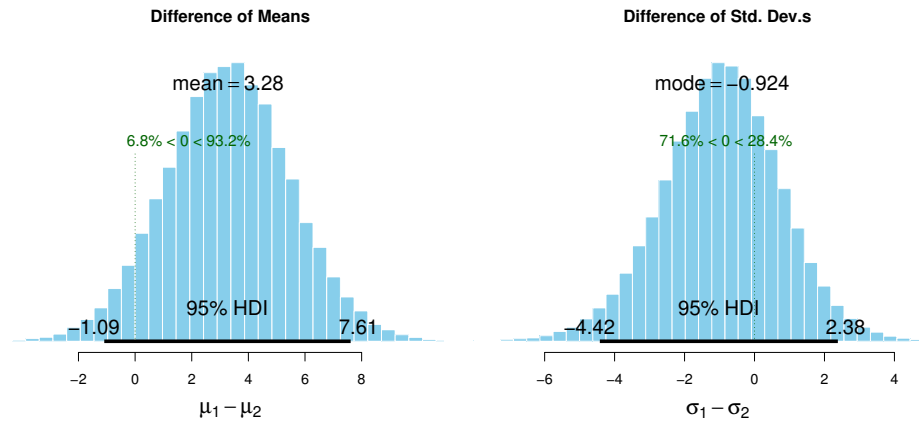


FIGURE A5.3: Simulated posterior density function for (i) the expected mean difference in weight gains for the ‘cb’ and ‘control’ groups $\mu_1 - \mu_2$ (left) and (ii) the difference of their standard deviations (right), based on 100000 replications.

```

      for (i in 1:B){x <- rpois(n, mu0)    # samples and applying
                        ML <- mean(x)      # LRT function to each
                        y[i] <- LRT(n, mu0, ML)}
      return(y) }
> n <- 25; mu0 <- 5; B <- 100000    # B = number of Monte Carlo samples
> stat <- simstat(B, n, mu0)
> hist(stat, prob=TRUE, border="blue", breaks="Scott")
> fchi2 <- function(x) {dchisq(x, 1)}
> curve(fchi2, from=0, to=max(stat), add=TRUE, col="red4",lwd=2)
> quantile(stat, probs=c(0.8,0.9,0.95,0.99)) # simulated exact quantiles
      80%   90%   95%   99%
      1.6301 2.7262 3.7440 6.6845
> qchisq(c(0.8, 0.9, 0.95, 0.99), 1)    # chi-squared quantiles for df=1
[1] 1.6424 2.7055 3.8415 6.6349

```

The simulated quantiles of the exact sampling distribution are close to the χ_1^2 quantiles. Figure A5.4 is a histogram of the 100,000 values of the likelihood-ratio test statistic and the χ_1^2 pdf. The approximation seems fairly good, even though n is relatively small.

```

# Function for computing the Poisson log(LRT):
#-----#
> logLRT <- function(n,lamb0,lamb.hat){
      2*n*((lamb0-lamb.hat)-lamb.hat*log(lamb0/lamb.hat))}
#-----#
# Function returning a vector of length R with the logLRT-values
# for the R simulated Poisson(lambda_0) samples of size n:
#-----#
> testat <- function(R,n,lamb0){ y <- rep(-1,R)
      for (i in 1:R){x <- rpois(n,5)
                        MLE <- mean(x)
                        y[i] <- logLRT(n,lamb0,MLE)}
      return(y) }
#-----#
# Application of the testat function for n=25 and lambda0=5:
> n<- 25; lamb0 <- 5
> R <- 10000    # number of replicates
> T25 <- testat(R,25,lamb0); stat<- T25

```

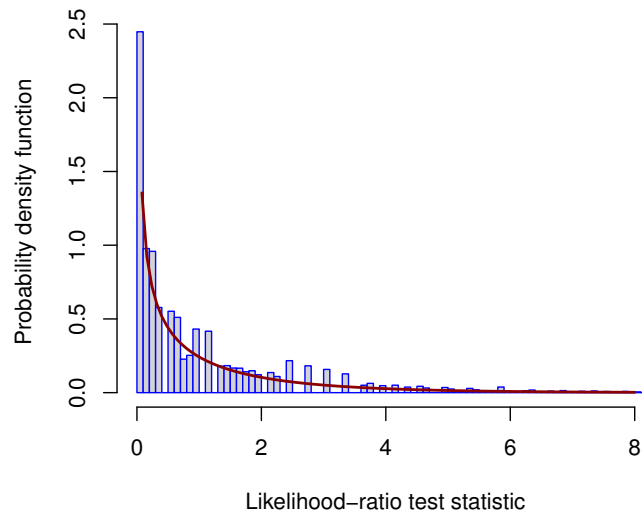


FIGURE A5.4: Histogram of 100,000 values of likelihood-ratio test statistic and the χ_1^2 pdf, for random sampling from a Poisson distribution with $\mu_0 = 5$ and $n = 25$.

```
# Histogram of simulated logLRT values (lambda_0) for n=25:
> n1<-max(stat)
> hist(stat,prob=TRUE,border="blue", main=" ",xlab="-2log(LRT)")
> fchi2 <- function(x) {dchisq(x,1)}
> curve(fchi2, from=0, to=n1, add=TRUE)
#-----#
# Simualted Quantiles (for n=25):
> quantile(stat, probs = c(0.8, 0.9, 0.95, 0.99))
      80%      90%      95%      99%
1.630055 2.751874 3.744050 6.801065
> qchisq(c(0.8, 0.9, 0.95, 0.99), 1) # chi^2(1) Quantiles
[1] 1.642374 2.705543 3.841459 6.634897
```

A5.3 Nonparametric Statistics: Permutation Test and Wilcoxon Test

You can conduct the permutation test comparing means or medians of two groups in R using a function in the **EnvStats** package, as shown in Section 5.8.2. However, it uses the exact distribution only when $n_1 + n_2 \leq 10$, and otherwise uses simulation. Here is an R package and function that can test the hypothesis of identical distributions against an alternative in which the means differ or one is larger than the other, using the exact permutation distribution, illustrated on the dog petting versus praise example of Section 5.8.2:

```
> library(coin) # "coin = conditional inference" uses survival package
> time <- c(114, 203, 217, 254, 256, 284, 296, 4, 7, 24, 25, 48, 71, 294)
```

```
> group <- factor(rep(c(1,2), each=7))
> oneway_test(time ~ group, alt="greater", distribution="exact")
      Exact Two-Sample Fisher-PitmanPermutation Test
      p-value = 0.003788
```

The nonparametric Wilcoxon test for comparing the mean ranks of two independent groups, discussed in Section 5.8.3, can easily be implemented using the `wilcox.test` function of base R, as illustrated below on the same example as above. Notice that for this function the sample values of the two groups need to be given in different vectors:

```
> x <- c(114, 203, 217, 254, 256, 284, 296)
> y <- c(4, 7, 24, 25, 48, 71, 294)
> wilcox.test(x,y, alternative = "greater", exact = TRUE)

      Wilcoxon rank sum exact test

data:  x and y
W = 43, p-value = 0.008741
alternative hypothesis: true location shift is greater than 0
```

An issue in nonparametric statistical inference based on ranks is the treatment of observations of equal rank (*ties*). The `wilcox.test` function does not provide options for this. By default, it computes an exact *p*-value if the samples contain less than 50 finite values and there are no ties. Otherwise, it uses a normal approximation.

For a more insightful non-parametric treatment, one can use the `coin` package that handles ties and offers a greater variety of non-parametric statistical procedures (asymptotic, exact and Monte-Carlo approximated). The associated function for the Wilcoxon test is `independence_test()`, which performs a general test of independence of two sets of variables measured on arbitrary scales. The underlying theory has been developed by Strasser and Weber⁵, implemented by Hothorn et al.⁶ The data need to be a `data.frame` and thus the responses of both groups form one variable and the groups are defined through a factor, as shown below for our example. Note that the `wilcox.test` function offers also this option (replacing the data vectors in the first two arguments of the function by the model formula and the data).

```
> expl <- data.frame(time,group)
> independence_test(time ~ group, data=expl, distribution = exact(),
      alternative="greater")

      Exact General Independence Test

data:  time by group (1, 2)
Z = 2.6102, p-value = 0.003788
alternative hypothesis: greater
```

⁵Strasser H, Weber C (1999). On the asymptotic theory of permutation statistics. *Mathematical Methods of Statistics*, 8(2), 220—250.

⁶Hothorn T, Hornik K, van de Wiel MA, Zeileis A (2006). A lego system for conditional inference. *The American Statistician*, 60(3), 257—263

6

CHAPTER 6: R FOR LINEAR MODELS

A6.1 Linear Models with the `lm` Function

This function is called as

```
lm(formula, data, ...)
```

where the argument `formula` defines the model to be fitted and `data` specifies the data frame on which the model will be applied. There are further optional arguments that can be specified in a call of the `lm` function that can for example weight (`weights`) observations or use a subset (`subset`) of observations in the fitting process.

The `formula` argument specifies the model formula as, e.g. $y \sim x1 + x2$, where `y` contains the data of the response variable while `x1` and `x2` of the explanatory variables that can be numeric continuous or categorical (factors). All variables appearing in this argument must be in the workspace or in the data frame specified in the (optional) `data` argument. Other symbols that can be used in the `formula` argument are

- `x1:x2`, for an interaction between `x1` and `x2`
- `x1*x2`, which expands to `x1+x2+x1:x2`
(e.g. `y~x1*x2` is equivalent to `y~x1+x2+x1:x2`)

Simply calling `lm` does not save any results but simply provides a summary output on the screen. The results of the model fit are saved and can be used for further analysis under an `lm`, say named `model` (e.g. `fit <- lm(...)`). In this case, there is no output provided automatically but can be controlled by the user.

Among the functions that are available for displaying (or saving) components of an `lm` (or `glm`) object, are the following.

- `summary(fit)`: displays the results
- `coef(fit)`: the vector of the model parameters' estimates (coefficients)
- `confint(fit, level=0.95)`: $(1-\alpha)$ t approximate CI for the coefficients of the specified model (by default is provided a 95% CI).
- `confint.default(fit, level=0.95)`: $(1-\alpha)$ CI for the coefficients of the specified model based on asymptotic normality (by default $\alpha = 0.05$).
- `fitted.values(fit)`: the fitted mean response values
- `residuals(fit)`: the residuals (observed response - fitted value)
(standardized and studentized residuals are obtained by the `rstandard` and `rstudent` functions, respectively)
- `df.residual(fit)`: the residual degrees of freedom

- `predict(fit, newdata=)`: the predicted expected responses for new data points, based on the estimated model
- `plot(fit)`: diagnostic plots (discussed next)
A selection of up to six plots of residuals; (1) *residuals* against *fittedvalues*, (2) Scale-Location plot of $\sqrt{|residuals|}$ against *fittedvalues*, (3) normal qq-plot, (4) plot of *Cook's distances* versus *rowlabels*, (5) plot of *residuals* against *leverages*, and (6) plot of *Cook's distances* against *leverage/(1 - leverage)*. By default, plots (1), (2), (3) and (5) are provided. This function requires the **graphics** package.

A6.2 Diagnostic Plots for Linear Models

Section 6.4.2 modeled Y = mental impairment, which has $\bar{y} = 27.30$ and $s_y = 5.46$, with explanatory variables life events and SES. For a linear model fit, the `plot` function can display several diagnostics:

```
> Mental <- read.table("http://stat4ds.rwth-aachen.de/data/Mental.dat", header=TRUE)
> fit <- lm(impair ~ life + ses, data=Mental)
> summary(fit)
```

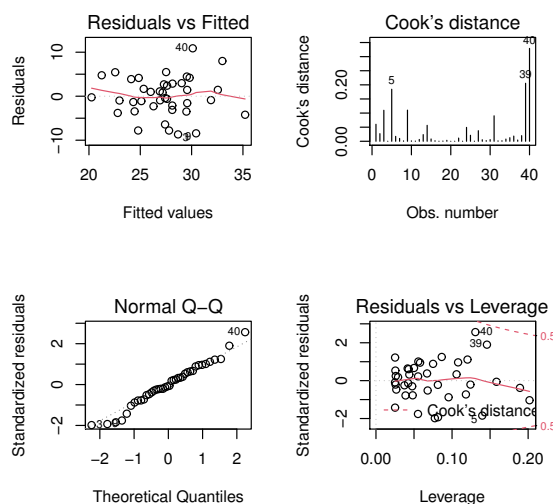
	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	28.2298	2.1742	12.984	2.38e-15
life	0.1033	0.0325	3.177	0.00300
ses	-0.0975	0.0291	-3.351	0.00186

```
-
> layout(matrix(1:4, 2, 2)) # multiple plots in 2x2 matrix layout
> plot(fit, which=c(1,2,4,5)) # diagnostic plots shown in Figure A6.1
```

Figure A6.1 shows four of the available diagnostic plots. These help us check the model assumptions that $E(Y)$ follows the linear model form and that Y has a normal distribution about $E(Y)$ with constant variance σ^2 . The first display plots the residuals $\{y_i - \hat{\mu}_i\}$ against the fitted values $\{\hat{\mu}_i\}$. If the linear trend holds for $E(Y)$ and the conditional variance of Y is truly constant, these should fluctuate in a random manner, with similar variability throughout. With only 40 observations, the danger is over-interpreting, but this plot does not show any obvious abnormality. Observation 40 stands out as a relatively large residual, with observed y more than 10 higher than the fitted value. The residuals can also be plotted against each explanatory variable. Such a plot can reveal possible nonlinearity in an effect, such as when the residuals exhibit a U-shaped pattern. They can also highlight nonconstant variance, such as a fan-shaped pattern in which the residuals are markedly more spread out as the values of an explanatory variable increase.

Figure A6.1 also shows Cook's distance values. From Section 6.2.8, a large Cook's distance highlights an observation that may be influential, because of having a relatively large residual and leverage. In this display, cases 5, 39 and 40 stand out. The plot of the residuals versus the leverage highlight these observations. In this plot, observations fall outside red dashed lines if their Cook's distance exceeds 0.5, which identifies them as potentially influential. Here, no observation has that large a Cook's distance, but when observation 40 is removed from the data file, the estimated life events effect weakens somewhat.

The figure also shows the normal quantile (Q - Q) plot. This enables us to check the normal assumption for the conditional distribution of mental impairment that is the basis for using the t distribution in statistical inference, including prediction intervals. The assumption seems reasonable, as the trend in this Q - Q plot is not far from the straight line expected with normality.

FIGURE A6.1: Diagnostic plots for the linear model fitted to the `Mental` data file.

A6.2.1 A Linear Model Example: Salaries Data Set

Let us revisit the salaries data set of Section A1.3. We have seen in Section A1.5 the job seniority - salary scatterplots, marked by gender along with the fitted regression lines, targeting to conclude whether there is a gender bias present or not. For this data set, we shall (i) fit the simple regression model, first ignoring gender and then in the subsets of males and females, and (ii) the linear model with job seniority and gender as explanatory variables, as shown below.

```
# model (i):
> model.all <- lm(salary~years, data=income) # for all cases, ignoring gender
> summary(model.all)

Call:
lm(formula = salary ~ years, data = income)

Residuals:
    Min       1Q   Median       3Q      Max
-168.02  -68.41  -24.68   82.95  176.79

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 1323.160    63.962   20.69  <2e-16 ***
years        96.730     2.969   32.58  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 100.3 on 24 degrees of freedom
Multiple R-squared:  0.9779,    Adjusted R-squared:  0.977
F-statistic: 1062 on 1 and 24 DF,  p-value: < 2.2e-16
# (i)a ----- model fitted only on males (n1=14) -----
> model.m <- lm(salary~years, data=income, subset=gender==1)
> summary(model.m)

Call:
lm(formula = salary ~ years, data = income, subset = gender == 1)

Residuals:
    Min       1Q   Median       3Q      Max
```

```

-145.131 -79.772    2.566  86.945 118.478

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) 1310.480    78.646   16.66 1.16e-09 ***
years        99.956     3.853   25.94 6.57e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 99.31 on 12 degrees of freedom
Multiple R-squared:  0.9825,    Adjusted R-squared:  0.981
F-statistic: 673 on 1 and 12 DF,  p-value: 6.568e-12
# (i)b ----- model fitted only on females (n2=12) -----
> model.f <- lm(salary~years, data=income, subset=gender==2)
> summary(model.f)

Call:
lm(formula = salary ~ years, data = income, subset = gender == 2)

Residuals:
    Min       1Q   Median       3Q      Max
-112.022  -45.487   -4.634   42.635  117.268

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) 1276.938    71.829   17.78 6.77e-09 ***
years        96.215     3.151   30.54 3.33e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 65.19 on 10 degrees of freedom
Multiple R-squared:  0.9894,    Adjusted R-squared:  0.9883
F-statistic: 932.4 on 1 and 10 DF,  p-value: 3.325e-11
# (ii) ----- analysis of covariance model -----
> model.ancova <- lm(salary~years+gender, data=income)
> summary(model.ancova)

Call:
lm(formula = salary ~ years + gender, data = income)

Residuals:
    Min       1Q   Median       3Q      Max
-161.571  -55.325   -4.797   65.821  137.748

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) 1450.418    66.380   21.850 < 2e-16 ***
years        98.490     2.559   38.492 < 2e-16 ***
gender     -111.771    34.024   -3.285  0.00324 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 84.57 on 23 degrees of freedom
Multiple R-squared:  0.985,    Adjusted R-squared:  0.9836
F-statistic: 752.8 on 2 and 23 DF,  p-value: < 2.2e-16

> layout(matrix(1:6,3,2))      # multiple plots in a 3x2 matrix layout
> plot(model.ancova, which=1:6) # residual plots, shown in Figure A6.2

```

Observe that although model (i) provides an excellent fit, it does not take gender into account and thus it is not informative with respect to our bias question. A first approach would be to fit separate regression lines per gender (models (i)a and (i)b). However, though this approach gives as an indication it does not provide a direct way to test the significance

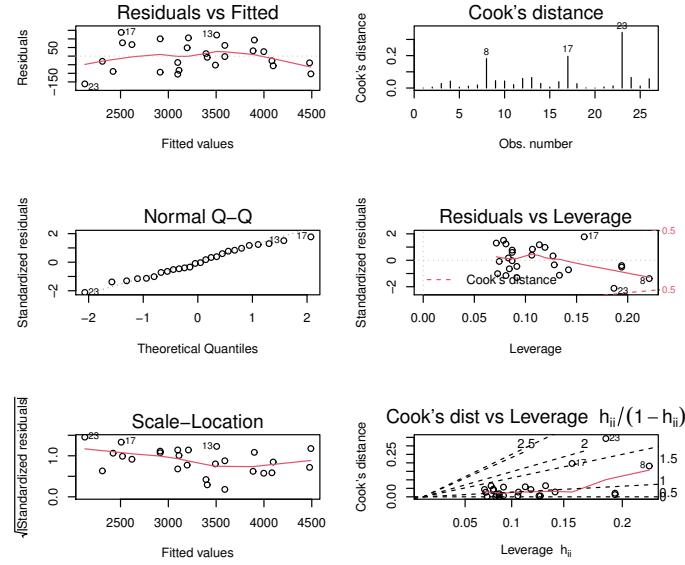
of the gender effect and furthermore fitting two models using half of the available data each time, reduces the accuracy of the parameter estimates based on these models. On the other hand, model (ii) includes the gender as additional explanatory variable and thus estimates a common job seniority effect, based on the total sample size and explains the average difference of the salaries between males and females by the coefficient of the gender effect, which is highly significant and negative, indicating that females earn on average 111.8 euro less than their male colleagues with the same seniority.

Residual diagnostics is a necessary step before accepting our model and using it for relational interpretation of the effect of the explanatory variables on the response or prediction. Observing the plots in Figure A6.2 for our covariance model (ii), we verify that there is no distinctive pattern of unexplained response variation in the residuals (upper, left) and no indication for deviation from the normality assumption (middle, left). There is also no indication for non-constant variance, since the residuals appear randomly spread and the red line is almost horizontal (lower, left). For large sample sizes, a thumb guideline for interpreting the Cook's distance is to consider Cook's distance values higher than 1 as indicative for highly influential observations. The plots on the right column are for detecting influential points. The Cook's distances plot (upper, right) shows that the Cook's distances of cases 8, 17 and especially 23 are higher compared to the rest of the observations and might be influential. These observations are indicated in the residuals vs. leverage plot (middle, right), which serves for the identification of influential observations. Since all our points, even cases 8,17 and 23, are within the Cook's distance lines (red dashed lines; the upper 0.5 line is hardly to be seen in the upper right corner of the plot), it seems that there are no influential points. In case of influential points, the red solid line would cross the Cook's distance lines. Finally, on the Cook's distance vs. Leverage plots (lower, right), the dashed line contours are standardized residuals (of the labeled value). We verify that only case 23 is above the value 2 and none case exceeds the standardized residual value of 2.5. Thus, our model passes all residual diagnostics successfully. The plots that the `plot` function provides by default are the three on the left and the middle one on the right side.

A6.3 Plots for Regression Bands and Posterior Distributions

Confidence intervals for the expected response (6.8) and prediction intervals for the response based on a linear model and at a particular data point of the explanatory variables, given in a vector x_0 , can be obtained by the `predict` function, setting the argument `interval` accordingly, as illustrated in Section 6.4.5 for the mental impairment study and a regression model with two explanatory variables and one particular data point. The new data point has to be defined as a data frame in the argument `newdata`. In case we want to predict the response at more than one data points, then we have to define the new data as a data frame before applying `predict`, as illustrated below for the salary of persons with 12, 20 and 30 years on job, based on model (i) in Section A6.2.1. Observe that though the point estimation is the same at each particular x_0 , the associated confidence intervals for the mean salary are narrower than the corresponding prediction intervals.

```
> new <- data.frame(years = c(12,20,30))
> CI_x0 <- cbind(new, predict(model.all, newdata=new, interval="confidence",
                             level = 0.95))
> CI_x0
  years      fit      lwr      upr
1    12 2483.914 2417.867 2549.961
2    20 3257.751 3217.018 3298.483
```

FIGURE A6.2: Residual plots for the linear model (ii) fitted on the `Salaries` data.

```

3  30 4225.046 4154.067 4296.025
> PI_x0 <- cbind(new, predict(model.all, newdata=new, interval="prediction",
                           level = 0.95))
> PI_x0
  years      fit      lwr      upr
1   12 2483.914 2266.532 2701.297
2   20 3257.751 3046.677 3468.824
3   30 4225.046 4006.115 4443.977

```

For the same model, the scatter plot of the data along with the regression line and the associated 99% confidence intervals and 99% prediction intervals are provided in Figure A6.3 (left). Figure A6.3 (right) pictures the scatter plot of the same data along with the regression lines per gender and the associated 95% confidence intervals.

Please note that although `ggplot2` and other software visualize the confidence and prediction intervals through lower and upper continuous curves, these are pointwise intervals (as illustrated in Figures 6.8) and are not to be considered as confidence intervals for the entire regression line. Providing confidence bands for the regression line, we need to take into account also the distribution of s_p^2 , where p is the dimension of the parameter vector. For example, the regression line confidence band for the simple linear regression model ($p = 1$), is

$$\hat{\mu}(x_0) \pm s\sqrt{2F_{1-\alpha;2,n-2}}\sqrt{\frac{1}{n} + \frac{(x_0 - \bar{x})^2}{\sum_{i=1}^n (x_i - \bar{x})^2}}, \quad (6.1)$$

where $F_{1-\alpha;2,n-2}$ is the $(1 - \alpha)$ quantile of an $F_{2,n-2}$ distribution. These bands can be generalized for multiple predictors ($p > 2$) but is beyond our scope.

The 99% confidence band of the entire regression line for our income example is also provided in Figure A6.3 (left). Note that it becomes brighter than the corresponding CI for the mean response as we move away from the average years of seniority. The R-code for deriving both plots of Figure A6.3 in the `ggplot2` package follows.

```

> library(tidyverse)
> alpha <- 0.01
> pred.dat <- predict(model.all, interval="prediction", level=1-alpha)
#-----
# calculation of the entire regression line CI:
x <- income$years; n <- nrow(income)
cf <- sqrt(sum(model.all$residuals^2)/(n-2))*
      sqrt(2*qf(1-alpha, 2, n-2))*sqrt((1/n)+(mean(x)-x)^2/(n*var(x)))
ci.regr1 <- model.all$fitted.values - cf
ci.regr2 <- model.all$fitted.values + cf
names(ci.regr1) <- "lwr.regr"; names(ci.regr2) <- "upr.regr"
#-----
newdata <- cbind(income, pred.dat, ci.regr1, ci.regr2)
> ggplot(newdata, aes(years, salary)) +
  geom_point() +
  geom_smooth(method=lm, se=TRUE, level=1-alpha)+
  geom_line(aes(y=lwr), color = "red", linetype = "dashed")+
  geom_line(aes(y=upr), color = "red", linetype = "dashed")+
  geom_line(aes(y=ci.regr1), color = "red")+
  geom_line(aes(y=ci.regr2), color = "red")

> # ggplot needs categorical variables used for shape/color to be factors
> Gender <- factor(income$gender, labels=c("M","F"))
ggplot(income, aes(x=years, y=salary, color=Gender, shape=Gender)) +
  geom_point() +
  geom_smooth(method=lm, se=TRUE, aes(fill=Gender))+
  scale_colour_manual(values = c("Blue", "Red"))+
  scale_fill_manual(values=c("blue","red"))

```

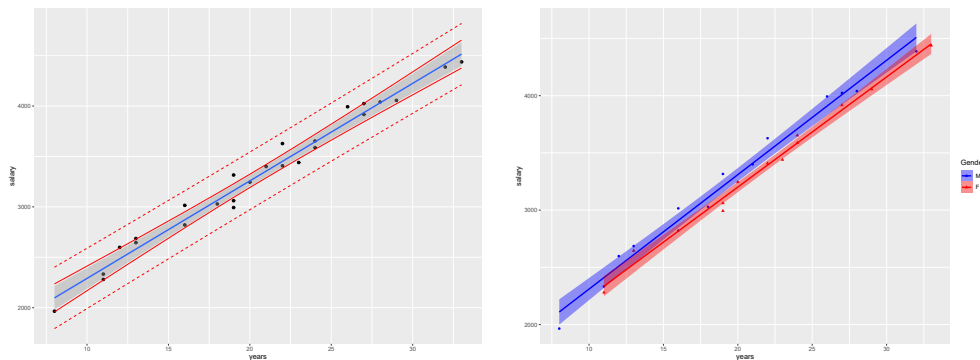


FIGURE A6.3: Scatter plots of the salary (in euro) versus the seniority (in years) for the income data along with (1) the regression fit, and 99% CIs (gray shadowed), 99% confidence band for the entire regression line (red lines) and 99% prediction intervals (dashed red lines) based on model (i) fitted on the income data (left) and (2) regression lines per gender (males: blue, females: red) and the associated 95% confidence interval bands (right).

The linear regression model in Section 6.6.2 is fitted by the `MCMCregress` function of the `MCMCpack` package. Its syntax is similar to `lm`, and, as all model function of `MCMCpack`, returns an `mcmc` object that contains the generated Markov chains for all parameters in the model. Data sets being in a vector or matrix form with one column per variable can be transformed to `mcmc` objects by the `mcmc` functions. `Mcmc` objects resemble time series objects and have optional arguments that allow to control the start, the end and the thinning of the chain (e.g. keep for example one of every 10 observations).

With `MCMCpack`, we can plot a posterior *pdf* using the `densplot` command. We illustrate

this next for the life events effect β_1 (the 2nd model parameter) in the Bayesian fitting of the linear model for mental impairment that used improper prior distributions for $\{\beta_1\}$ (Section 6.6.2):

```
> library(MCMCpack)
> fit.bayes <- MCMCregress(impair ~ life + ses, mcmc=5000000,
+                          b0=0, B0=10^(-10), c0=10^(-10), d0=10^(-10), data=Mental)
> fit.bayes[1:3,] # first three rows of the derived mcmc object
      (Intercept)      life      ses      sigma2
[1,]    26.66740  0.13160292 -0.09251524  23.22411
[2,]    30.17502  0.10860963 -0.12223697  20.57522
[3,]    27.04690  0.06959574 -0.06056731  17.20253
> densplot(fit.bayes[,2], xlab="Life events effect", ylab="Posterior pdf") # Figure A6.4
# Plot of the posterior pdf for the variance (not shown):
> densplot(fit.bayes[,4], xlab=expression(sigma^2))
```

Figure A6.4 shows the plot. For this effect, the posterior mean was 0.103, with a standard error of 0.033.

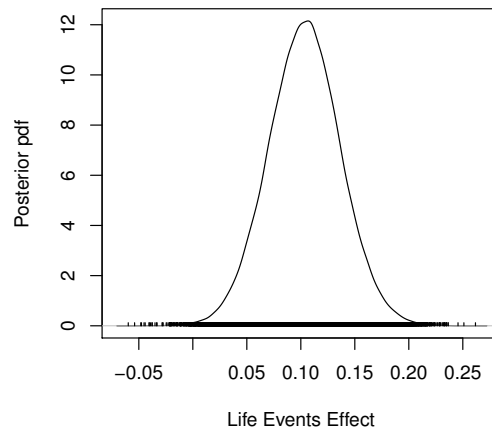


FIGURE A6.4: Posterior *pdf* for the life events effect β_1 on mental impairment, in the linear model fitted to the **Mental** data file

A crucial task in Bayesian inference through MCMC methods is the convergence of the chain to the posterior. For this, the consideration of diagnostics is necessary. Such diagnostics are provided in the **coda** package that applies on **mcmc** objects. Thus, the results of a model function of **MCMCpack** can directly be further exploited in **coda**. The discussion of MCMC methods is beyond the scope of this book and for this we refer to specialized literature. A typical **coda** graphical display is the so called trace plot, obtained by the **traceplot** function and used to monitor the terms in a Markov chain and verify whether the chain mixes well. The traceplot of long chains will be too messy and at the end non-informative, thus we usually plot just a fraction of the chain, commonly the last part. A part of a column of an **mcmc** object is no more an **mcmc** object, thus to apply **traceplot** on a subchain, we need to define the selected part of the chain as an **mcmc** object before. Thus, continuing our example, the trace plot of the last 500 values of the posterior of β_1 can be derived as given next (the figure is not shown).

```

> library(coda)
> U <- nrow(fit.bayes); L <- U-499
> betal.post <- mcmc(fit.bayes[,2][L:U])
> traceplot(betal.post, main="Trace of life events effect")

```

Applying the `plot` function on an `mcmc` object in `coda` delivers the trace plots and posterior density plots for all chains in the columns of the object.

A6.4 Bayesian Linear Regression for Improper Priors

We shall next consider the Bayesian fit of a linear regression model for normal distributed random responses $Y_i \sim N(\mu_i, \sigma^2)$, $i = 1, \dots, n$, with

$$\mu_i = E(Y_i) = \beta_0 + \beta_1 x_{1i} + \dots, \beta_p x_{ip},$$

and unknown error variance σ^2 , under the typical noninformative prior assumption

$$g(\beta, \sigma^2) \sim \frac{1}{\sigma^2},$$

for the parameter vector $\beta = (\beta_0, \beta_1, \dots, \beta_p)$ and σ^2 . The posterior distribution of β conditional on σ^2 , $g(\beta|\mathbf{y}, \sigma^2)$, is multivariate normal distributed with mean the MLE $\hat{\beta}$ and covariance matrix $\text{var}(\hat{\beta})$ (s. (6.1) and (6.13)). The marginal posterior distribution of σ^2 is inverse gamma($\frac{n-(p+1)}{2}, \frac{1}{2}\mathbf{S}$), with $\mathbf{S} = (\mathbf{y} - \mathbf{X}\hat{\beta})^T(\mathbf{y} - \mathbf{X}\hat{\beta})$. For this set-up, we refer to Albert(2009, Section 9.2), where the analysis is based on the functions `blinreg`, `blinregexpected` and `blinregpred` of the `LearnBayes` package, which allow direct derivation of results and insightful visualizations.

We implement next this approach on the Scottish hill races data, introduced in Section 6.1.4, for the linear regression model fitted with least squares in Section 6.2.2. Notice the way to simulate from (i) the posterior distribution and (ii) the prediction distribution of the response variable for a particular set of values for the explanatory variables. These simulated draws can then be used to derive summary statistics and credible or prediction intervals.

```

> fit.dc2 <- lm(timeW ~ distance + climb, data=Scots2, x=TRUE, y=TRUE)
> library(LearnBayes)
# basic function for Bayesian linear regression:
> theta.sample=blinreg(fit.dc2$y, fit.dc2$x, 5000)
#-----
# histograms of posterior densities:
> par(mfrow=c(1,2)) # see Figure A6.5
> hist(theta.sample$beta[,2], main="climb", xlab=expression(beta[1]))
> hist(theta.sample$beta[,3], main="distance", xlab=expression(beta[2]))
#-----
# posterior means/sd:
> b.mean <- apply(theta.sample$beta, 2, mean); b.mean
X(Intercept)  Xdistance  Xclimb
-8.950305     4.173835    43.847641
> b.sd <- apply(theta.sample$beta, 2, sd); b.sd
X(Intercept)  Xdistance  Xclimb
3.3197336     0.2435103    3.7584219
> sigma.mean <- mean(theta.sample$sigma); sigma.mean
[1] 12.37126
> sigma.sd <- sd(theta.sample$sigma); sigma.sd

```

```

[1] 1.116089
# posterior quantiles:
> apply(theta.sample$beta,2,quantile,c(.025,.25,.5,.75,.975))
      X(Intercept) Xdistance  Xclimb
2.5%    -15.424176   3.693711 36.40000
25%     -11.185153   4.012498 41.35451
50%      -8.973437   4.173592 43.83825
75%      -6.721816   4.335581 46.35683
97.5%    -2.368920   4.655774 51.25069
> quantile(theta.sample$sigma,c(.025,.25,.5,.75,.975))
      2.5%    25%    50%    75%    97.5%
10.42304 11.59083 12.29223 13.06214 14.79655
#-----
# (i) Sampling from the posterior expected response at a particular X0:
#-----
> cov1=c(1,min(Scots2$distance),min(Scots2$climb))
> cov2=c(1,quantile(Scots2$distance, 0.25),quantile(Scots2$climb,0.25))
> cov3=c(1,mean(Scots2$distance),mean(Scots2$climb))
> cov4=c(1,max(Scots2$distance),max(Scots2$climb))
> X0=rbind(cov1,cov2,cov3,cov4)
> p <- ncol(X0)
> mean.draws=blinregexpected(X0,theta.sample)
> par(mfrow=c(2,2)) # see Figure A6.6
> hist(mean.draws[,1],main="Covariate set at min",xlab="timeW")
> hist(mean.draws[,2],main="Covariate set at 0.25 quantile",xlab="timeW")
> hist(mean.draws[,3],main="Covariate set at mean",xlab="timeW")
> hist(mean.draws[,4],main="Covariate set at max",xlab="timeW")
#-----
# (1-a) EQT Credible Intervals:
#-----
> alpha <- 0.05
> t(apply(theta.sample$beta,2,quantile,c(alpha/2,1-alpha/2)))
      2.5%    97.5%
X(Intercept) -15.424176 -2.368920
Xdistance      3.693711  4.655774
Xclimb         36.399998 51.250687
> quantile(theta.sample$sigma,c(alpha/2,1-alpha/2))
      2.5%    97.5%
10.42304 14.79655
# EQT intervals for the expected response at new data points in X0:
> X0 <- data.frame(X0)
> names(X0) <- c("Intercept", "distance", "climb")
> eqt.EY <- apply(mean.draws,2,quantile,c(alpha/2,1-alpha/2))
> eqt.EY <- cbind(X0[,2:p], t(eqt.EY)); eqt.EY
      distance    climb    2.5%    97.5%
cov1  3.20000 0.1850000  7.087614 18.03510
cov2 10.10000 0.4300000 48.142879 55.97406
cov3 15.61343 0.8825672 91.929115 97.90148
cov4 43.00000 2.4000000 265.272248 286.23025
#-----
# (ii) Sampling from the posterior predictive distribution of {Y_f}
# ( s. future observation Y_f and compare to prediction interval in Section 6.4.5):
#-----
# function blinregpred is here evaluated at fit.dc2$x (can be replaced
# by new data points)
> pred.draws=blinregpred(fit.dc2$x,theta.sample)
> pred.sum=apply(pred.draws,2,quantile,c(.025,.975))
> par(mfrow=c(1,1)) # see Figure A6.7
> ind=1:nrow(Scots2)
> matplot(rbind(ind,ind),pred.sum,type="l",lty=1,col=1, xlab="INDEX",
          ylab="timeW")
> points(ind,Scots2$timeW,pch=19) # observed responses {yi} with solid points

```

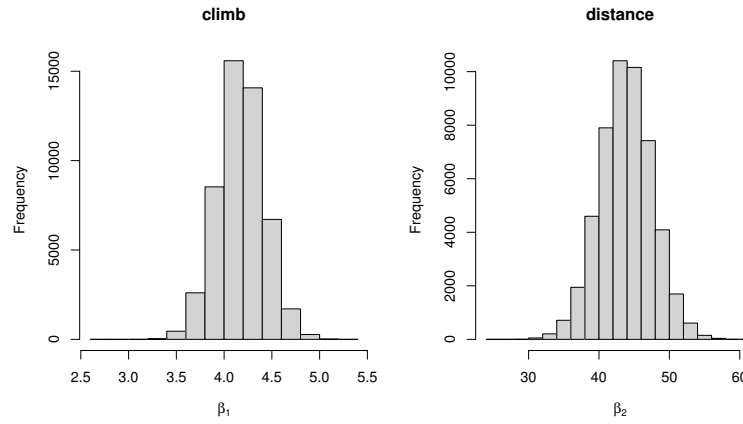


FIGURE A6.5: Histograms of simulated draws from the marginal posterior distributions of β_1 (distance) and β_2 (climb).

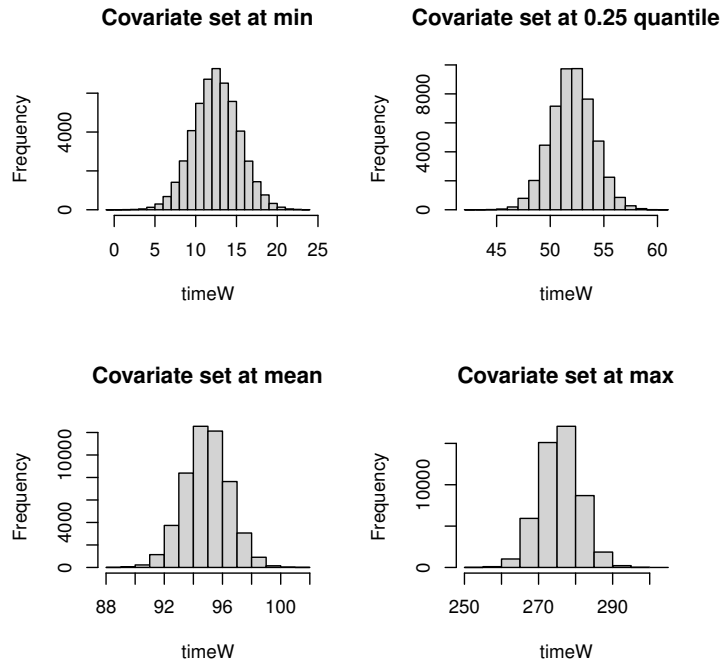


FIGURE A6.6: Histograms of simulated draws of the posterior of mean timeW for four particular sets of values for the explanatory variables (distance, climb).

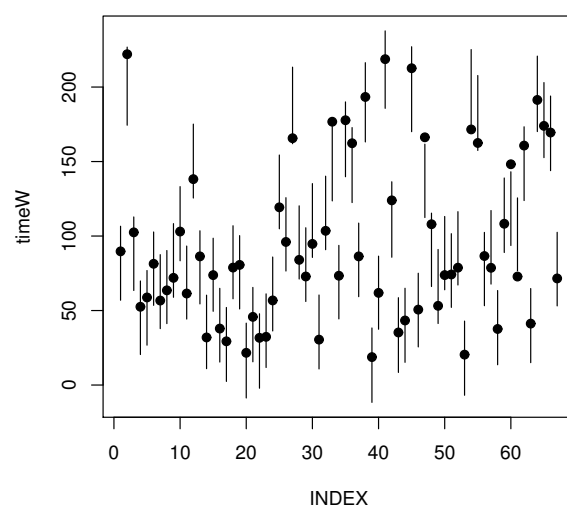


FIGURE A6.7: Posterior 95% prediction intervals of $\{y_i^*\}$ with actual observed values of Wtime indicated by points.

CHAPTER 7: R FOR GENERALIZED LINEAR MODELS

A7.1 The glm Function

This function is called as

```
glm(formula, family = ..., data, ...)
```

where the `formula` and `data` argument, along with some further optional arguments, are analogous to those of the `lm` function, (see Section A6.1). An implementation of the argument `weight` in a `glm` set-up is provided for modeling grouped binary data in Section 7.2.3. Of special interest for modeling rates is the (`offset`) argument which can be used to specify a component to be definitely included in the linear predictor (see Section 7.4.3).

The additional `family` argument determines the error distribution and link function of the model. The exponential dispersion family functions available in R, along with their canonical link, are:

- `binomial(link = "logit")`
- `gaussian(link = "identity")`
- `Gamma(link = "inverse")`
- `inverse.gaussian(link = "1/mu2")`
- `poisson(link = "log")`

However, other links beyond the canonical are possible. Thus, for the Covid-19 example in Section 7.1.8, we used gaussian and gamma GLMs with a log link.

Similarly to the `lm` function, the results of the model fit can be saved as a `glm` object and used for further analysis in the same manner as discussed for the `lm` function in Section A6.1. For a saved object from fitting a model, say called `fit`, typing `names(fit)` yields an overview of what is saved for the object, including characteristics such as the deviance, AIC, coefficients, fitted values, converged, and residuals. For instance, the command `fit$converged` asks whether the Fisher scoring fitting algorithm converged. Useful follow-up functions include `confint` for profile likelihood confidence intervals (the `MASS` package is also required). Furthermore note that in the GLM case not all residual plots make sense. For example, the $Q-Q$ plot does not make sense for non-normally distributed responses.

Included in the output is the number of iterations needed for the Fisher scoring algorithm to converge, with a default maximum of 25. Normally this is small, but it may be large (e.g., 17 for the endometrial cancer example in Section 7.3.2) or not even converge when some ML estimates are infinite or do not exist. You can increase the maximum number of iterations, such as with the argument `maxit = 50` in the `glm` function, but convergence may still fail. In that case, you should not trust estimates shown in the output.

A7.2 Plotting a Logistic Regression Model Fit

Saving as glm object, say `fit`, a logistic regression model fitted in `glm`, verify that the estimated probabilities for the cases of the sample are saved under `fit$fitted.values`. Furthermore, beyond the standard output viewed under `summary`, we have a great flexibility to create our own way of presenting and communicating the results. In the R code that follows we give the `prob.fit` function to estimate the expected success probability at an arbitrary level of the explanatory variable and not only for its observed values, i.e. we estimate the probabilities given by (7.3). Note that in this function we do not type the estimated coefficients of the logistic regression model but access the vector `fit$coefficients`. Thus, the function is applicable for any logistic regression model. This function is then plotted. We implemented it on the example of Section 7.2.3 (see Figure A7.1).

```
> Beetles <- read.table("http://stat4ds.rwth-aachen.de/data/Beetles_ungrouped.dat",
+ header=TRUE)
> names(fit)
[1] "coefficients"      "residuals"        "fitted.values"
[4] "effects"           "R"                 "rank"
[7] "qr"                "family"            "linear.predictors"
[10] "deviance"          "aic"               "null.deviance"
[13] "iter"              "weights"           "prior.weights"
[16] "df.residual"       "df.null"           "y"
[19] "converged"         "boundary"          "model"
[22] "call"              "formula"           "terms"
[25] "data"              "offset"            "control"
[28] "method"            "contrasts"         "xlevels"

prob.fit <- function(z){
  exp(fit$coefficients[1]+ fit$coefficients[2]*z)/
  (1+exp(fit$coefficients[1]+ fit$coefficients[2]*z))
}
# it holds: prob.fit(Beetles$x) = fit$fitted.values
# through the function we can estimate the expected probability at any point:
prob.fit(mean(Beetles$x))
(Intercept)
0.68183
low <- min(Beetles$x); up <- max(Beetles$x)
plot(prob.fit, col='blue', from=low, to=up, xlab='x', ylab='prob.fit(x)')
x0 <- mean(Beetles$x); y0 <- prob.fit(mean(Beetles$x))
points(x0, y0, pch=15, col='red')
```

The goodness of fit (GOF) of a logistic regression model can be fitted by the Hosmer-Lemeshow GOF Test ¹, which is an approximate χ^2 -test. In R it can be implemented in the `ResourceSelection` library, as shown below for our example. The test requires the specification of the number of bins used to calculate the quantiles, which are default set to 10.

```
> library(ResourceSelection)
> hoslem.test(Beetles$y, fit$fitted.values, g = 10)

Hosmer and Lemeshow goodness of fit (GOF) test

data: Beetles$y, fit$fitted.values
X-squared = 9.5074, df = 8, p-value = 0.3013
```

¹Hosmer, Lemeshow (2000). *Applied Logistic Regression*. New York, USA: John Wiley and Sons.

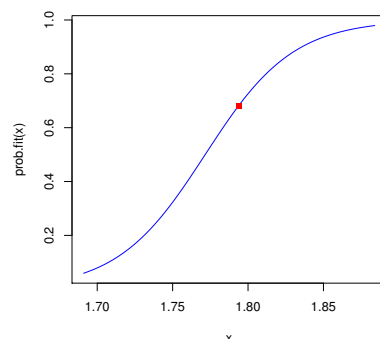


FIGURE A7.1: Estimated expected death probability for the flour beetles study as a function of x (disage of exposure to gaseous carbon disulfide, in mg per liter).

A7.3 Model Selection for GLMs

For a particular data set, how do we select a model? For inference, we can compare two nested GLMs using their deviance difference (Section 7.1.5). To approximate well the relation in the population, we can use a model having relatively small value of AIC or BIC (Section 7.1.6). With many explanatory variables, it is not feasible to fit all possible models. It is standard to consider only *hierarchical models*, which means that if a model includes an interaction term, it should also contain the main effect terms that go into that interaction. One approach finds the best model with a certain number of explanatory variables according to a criterion such as adjusted R -squared. In R, the `regsubsets` function in the `leaps` package can do this.

Alternatively, algorithms exist that select explanatory variables, or delete them, in a stepwise manner. *Backward elimination* begins with a complex model and sequentially removes terms. One version uses AIC, at a particular stage removing the term for which AIC decreases the most, and stopping the process when it no longer decreases. An alternative, *forward selection*, starts with the null model and adds terms sequentially until further additions do not improve the fit. Some statisticians prefer backward elimination, feeling it safer to delete terms from an overly complex model than to add terms to an overly simple one. In exploratory studies, such a process can be informative when used cautiously, but it need not yield a meaningful model and has no theoretical basis. You should regard its results with skepticism. For the final model suggested by a particular selection procedure, any inferences conducted with it are highly approximate. In particular, P -values are likely to appear smaller than they should be and confidence intervals are likely to be too narrow, because the model was chosen that most closely reflects the data, in some sense. The inferences are more believable if performed for that model with a new set of data or using cross-validation. Also, with large n , the process may yield an overly complex model, such as containing interaction terms that are statistically significant but not practically significant. In R, the `step` function provides stepwise model selection procedures that apply on `lm` and `glm` objects. By default it compares AIC values in working forwards and backwards. The first argument in `step` is the saved model (`lm` or `glm` object) where the stepwise algorithm will start from.

We illustrate for the data on house selling prices introduced in Exercise 6.15 and analyzed with a linear model and a gamma GLM in Section 7.1.3. There we observed that the variability in selling prices increases noticeably as the mean selling price increases, so we'll use gamma GLMs. To predict selling prices (in thousands of dollars), we begin with all five explanatory variables (size of house in square feet, annual property tax bill in dollars, number of bedrooms, number of bathrooms, and whether the house is new) and their ten two-way interaction terms as potential explanatory variables. The initial model has $AIC = 1087.85$. After several backward steps this reduces to 1077.31:

```
> Houses <- read.table("http://stat4ds.rwth-aachen.de/data/Houses.dat", header=TRUE)
> step(glm(price ~ (size+taxes+new+bedrooms+baths)^2, family=Gamma(link=identity),
+         data=Houses))
Start: AIC=1087.85
... # several steps not shown here
Step: AIC=1077.31 # final model chosen with backward elimination
price ~ size+taxes+new+bedrooms+baths+size:new+size:baths+taxes:new+bedrooms:baths
> fit <- glm(formula = price ~ size+taxes+new+bedrooms+bath + size:new+size:baths
+         + taxes:new+bedrooms:baths, family=Gamma(link=identity), data=Houses)
> summary(fit)
```

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	8.486e+01	5.202e+01	1.631	0.106311
size	1.422e-01	4.153e-02	3.424	0.000932
taxes	6.030e-02	7.369e-03	8.183	1.71e-12
new	-1.397e+02	7.491e+01	-1.865	0.065465
bedrooms	-7.381e+01	2.737e+01	-2.697	0.008358
baths	-1.673e+01	3.178e+01	-0.526	0.600004
size:new	2.520e-01	8.321e-02	3.028	0.003209
size:baths	-3.601e-02	1.863e-02	-1.933	0.056394
taxes:new	-1.130e-01	5.332e-02	-2.118	0.036902
bedrooms:baths	2.800e+01	1.478e+01	1.894	0.061392

```
---
(Dispersion parameter for Gamma family taken to be 0.0618078)
Null deviance: 31.9401 on 99 degrees of freedom
Residual deviance: 5.6313 on 90 degrees of freedom
AIC: 1077.3
```

The model chosen by backward elimination is still quite complex, having four interaction terms. Only a slightly higher AIC value (1080.7) results from taking out all interactions except the one between size and whether the house is new, which is by far the most significant among the interaction terms. Removing also the baths term gives the model selected based on the BIC criterion, implemented in the `step` function by the additional argument `k=log(n)`:

```
> n <- nrow(Houses) # sample size
> step(glm(price ~ (size+taxes+new+bedrooms+baths)^2, family=Gamma(link=identity),
+         data=Houses), k=log(n)) # output not shown
```

The interpretation is much simpler, as the effects of taxes and bedrooms are main effects solely. Here is the R code:

```
> fit2 <- glm(price ~ size + taxes + new + bedrooms + size:new,
+             family = Gamma(link=identity), data=Houses)
> summary(fit2)
```

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	3.923e+01	2.024e+01	1.938	0.0556
size	8.588e-02	1.837e-02	4.675	9.80e-06 # effect 0.086 for old homes
taxes	5.776e-02	7.563e-03	7.638	1.82e-11
new	-1.273e+02	7.557e+01	-1.685	0.0954
bedrooms	-2.135e+01	9.432e+00	-2.263	0.0259
size:new	8.956e-02	4.310e-02	2.078	0.0404 # size effect 0.086 + 0.090 for

```

---                                     # new homes
(Dispersion parameter for Gamma family taken to be 0.06509439)
Null deviance: 31.9401 on 99 degrees of freedom
Residual deviance: 6.3478 on 94 degrees of freedom
AIC: 1081.4

```

The estimated effect on selling price of a square-foot increase in size, adjusting for the other predictors, is 0.086 (i.e., \$86) for older homes and $(0.086 + 0.090) = 0.176$ (i.e., \$176) for newer homes.

Apart from the AIC not being much higher, how do we know that the fit of the simpler model is essentially as good in practical terms? A measure of how well the model predicts is given by the *correlation* between the observed response variable and the fitted values for the model. For a linear model, this is the *multiple correlation* (Section 6.3.3). For these data, the multiple correlation is 0.918 for the more complex model and 0.904 for the simpler one, very nearly as high, and this is without adjusting for the more complex model having many more terms.

```

> cor(Houses$price, fitted(fit)); cor(Houses$price, fitted(fit2))
[1] 0.9179869 # multiple correlation analog for model chosen by backward elimination
[1] 0.9037955 # multiple correlation analog for model with only size:new interaction

```

You can check that further simplification is also possible without much change in AIC or the multiple correlation. For instance, the simpler model yet that removes the bedrooms predictor has a multiple correlation of 0.903.

A7.3.1 Log-Linear Models Selection

We shall illustrate next a model selection procedure on a three-way contingency table, generated by cross-classifying the variables GUNLAW (in rows), SMALLGAP (in columns) and SEX (in layers) of the GSS2018 survey data introduced in Section A1.2.1. The corresponding frequency table is provided in Table A7.1, having adjusted the names of the variables. This table is produced in R from the GSS2018 data file, as shown next.

```

> GSS <- read.table("http://stat4ds.rwth-aachen.de/data/GSS2018.dat", header=T)
> gender <- factor(GSS$SEX, levels=c(1,2), labels=c("Male", "Female"))
> GunLaw <- factor(GUNLAW, levels=c(1,2),
                  labels=c("favor", "oppose"))
> SmallGap <- factor(SMALLGAP, levels=c(1,2,3,4,5), labels=
                  c("strongly agr.", "agree", "neutral", "disagree", "strongly dis.))
> tab1 <- table(GunLaw, SmallGap, gender); tab1 # output in Table A7.1

```

In general, for an $I \times J \times K$ contingency table, with classification variables L , S and G , for rows, columns and layers, respectively, we denote for the (i, j, k) cell, by n_{ijk} and μ_{ijk} the corresponding observed and expected frequencies, $i = 1, \dots, I$, $j = 1, \dots, J$, $k = 1, \dots, K$. The saturated log-linear model is given by

$$\log \mu_{ijk} = \lambda + \lambda_i^L + \lambda_j^S + \lambda_k^G + \lambda_{ij}^{LS} + \lambda_{ik}^{LG} + \lambda_{jk}^{SG} + \lambda_{ijk}^{LSG}, \quad i, k = 1, 2, \quad j = 1, \dots, 5,$$

and provides the perfect fit ($df = 0$, $\mu_{ijk} = n_{ijk}$, for all cells). Applying a backward stepwise model selection procedure, we search among hierarchical models to find the one that best fits to our data. The next model to consider would be the model with no three-factor interaction, followed by models having only two of the three two-factor interactions, up to the model of complete independence of the three classification variables, i.e. the model consisting only of the main effects ($\lambda^L, \lambda^S, \lambda^G$).

Note that in order to apply a log-linear model through the `glm` function, we need

TABLE A7.1: Cross-classification of the GSS2018 respondents in a $2 \times 5 \times 2$ contingency table by their gender, their position on the statement that For a society to be fair, differences in people's standard of living should be small and on whether they favor or oppose gun permits.

Male		SmallGap				
GunLaw		strongly agr.	agree	neutral	disagree	strongly dis.
favor		29	63	60	66	10
oppose		4	31	34	43	9
Female		SmallGap				
GunLaw		strongly agr.	agree	neutral	disagree	strongly dis.
favor		36	94	97	73	11
oppose		6	16	33	38	4

to transform the contingency table in a data frame class, where the cell frequencies are expanded in a vector form and are in one column (variable) of the data frame while the corresponding classification variables categories levels are given in further columns. It is important, before applying `glm`, to define the classification variables as factors. The process of transforming a contingency table to the appropriate data frame is illustrated for our example.

```
> freq <- as.vector(tab1)
> row <- rep(1:2, 10)
> col <- rep(rep(1:5, each=2), 2)
> lay <- rep(1:2, each=10)
> Glaw <- factor(row, levels=c(1,2), labels = c("favor", "oppose"))
> Sgap <- factor(col, levels=c(1,2,3,4,5), labels = c("strongly agr.",
  "agree", "neutral", "disagree", "strongly dis.))
> G <- factor(lay, levels=c(1,2), labels = c("Male", "Female"))
> table.df <- data.frame(freq, Glaw, Sgap, G)
```

For our example the model selection and model fit procedure is given below. When fitting any GLM by the `glm` function, you need to check in the summary output of the model fit that the number of Fisher Scoring iterations is below 20. Otherwise, the algorithm has not converged² and you cannot trust the output since by default the maximum number of iterations is set to 20. You may fit the model again, increasing the number of iterations (set for example in the `glm` function call the argument `iter=50`).

```
> saturated <- glm(freq~L*S*G, poisson, data=table.df)
> step(saturated, direction="backward", k=2) # output not given
# k=2 for using AIC (default), k=log(n) for BIC
# further direction options: "both", "forward")
> fit <- glm(freq ~ L*S + L*G, poisson)
> fit
Call:
glm(formula = freq ~ L * S + L * G, family = poisson)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.12111  -0.57115   0.01851   0.59029   0.93383
```

²see also Marschner (2011). `glm2`: Fitting generalized linear models with convergence problems, *The R Journal*, 3, 12–15.

```

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)      3.31402    0.13385  24.760 < 2e-16 ***
Loppose          -1.60014    0.34870  -4.589 4.46e-06 ***
Sagree           0.88186    0.14749   5.979 2.25e-09 ***
Sneutral         0.88186    0.14749   5.979 2.25e-09 ***
Sdisagree        0.76009    0.15026   5.058 4.23e-07 ***
Sstrongly dis.   -1.12986    0.25101  -4.501 6.75e-06 ***
GFemale          0.31045    0.08719   3.561 0.000370 ***
Loppose:Sagree   0.66570    0.37819   1.760 0.078371 .
Loppose:Sneutral 1.02025    0.36970   2.760 0.005786 **
Loppose:Sdisagree 1.33178    0.36732   3.626 0.000288 ***
Loppose:Sstrongly dis. 1.39223    0.48982   2.842 0.004479 **
Loppose:GFemale  -0.53153    0.16179  -3.285 0.001019 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

Null deviance: 440.5795  on 19  degrees of freedom
Residual deviance:  8.2117  on  8  degrees of freedom
AIC: 134.07

Number of Fisher Scoring iterations: 4      # check that this is <20!!!

> p.value <- 1-pchisq(fit$deviance, fit$df.residual); pvalue
[1] 0.4130657
# present MLEs and residuals in a table format:
> MLEs <- xtabs(fit$fitted.values ~ L+S+G)
> stdres <- xtabs(rstandard(fit) ~ L+S+G)
> library(car)
> Anova(fit)
Analysis of Deviance Table (Type II tests)

Response: freq
      LR Chisq Df Pr(>Chisq)
L      140.522  1 < 2.2e-16 ***
S      254.655  4 < 2.2e-16 ***
G       4.603   1 0.0319145 *
L:S     21.711  4 0.0002288 ***
L:G     10.877  1 0.0009739 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

> library(vcd)
> mosaic(tab1, gp=shading_Friendly, residuals=stdres,      # Figure A7.2
          residuals_type="Std\nresiduals")

```

The proposed model by the `step` function is

$$\log \mu_{ijk} = \lambda + \lambda_i^L + \lambda_j^S + \lambda_k^G + \lambda_{ij}^{LS} + \lambda_{ik}^{LG}, \quad i, k = 1, 2, \quad j = 1, \dots, 5,$$

with L , S and G , corresponding to the classification variables `GunLaw`, `SmallGap` and `Gender`, respectively. This model is of very good fit (p -value=0.413) having all standardized residuals in absolute value smaller than 1.7. They further do not exhibiting any pattern of unexplained association, as can be verified in the mosaic plot (Figure A7.2), where cells with positive (negative) residuals are framed with a blue (dashed red) boarder. Under this model, the variables `Gender` and `SmallGap` are *conditionally independent* given the level of the variable `GunLaw`.

Log-linear models can be fitted by the `loglin` function in base R or the `loglm` in MASS. Log-linear models for contingency tables are treated in detail in Agresti (2019) and Kateri (2014).

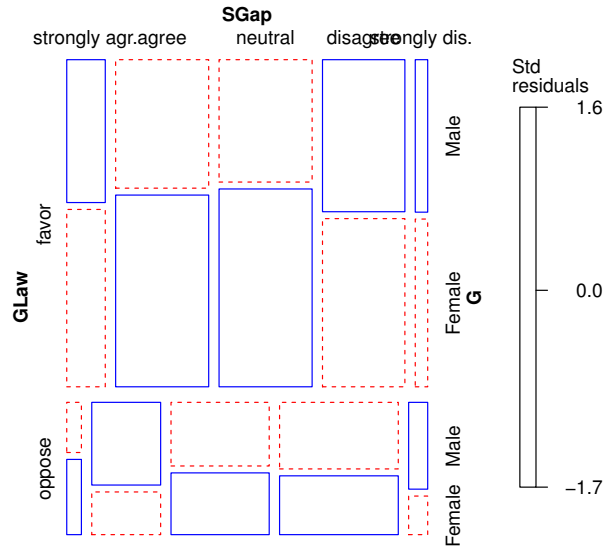


FIGURE A7.2: Mosaic plot for the contingency table of Table A7.1, with tiles shaded by the standardized residuals for model (LS, LG).

A7.4 Correlated Responses: Marginal, Random Effects, and Transitional Models

Fitting of generalized linear models, like other methods presented in this book, assumes *independent* observations from the population of interest. Many studies observe the response variable for each subject repeatedly, at several times or under various conditions. Then, the repeated observations on a subject are typically positively correlated. A study that repeatedly observes the same variables for the same subjects over time is called a **longitudinal study**. Such studies are common in health-related applications, such as when a physician evaluates patients at regular time intervals regarding whether a drug treatment is successful. In longitudinal studies, statistical analyses should take correlation of repeated observations into account. Analyses that ignore the correlation can have badly biased standard error estimators. GLMs can be generalized so that, while maintaining the capability to assume various probability distributions for the response variable and employ various link functions, model-fitting permits observations to be correlated. We next briefly describe the three basic types of models for doing this.

For subject i , let y_{it} be the observation at time $t = 1, \dots, T$, which is a realization of a random variable Y_{it} with $\mu_{it} = E(Y_{it})$. Then, $\mathbf{Y}_i = (Y_{i1}, Y_{i2}, \dots, Y_{iT})$ is a multivariate random variable. The first type of model specifies a GLM for each marginal distribution,

$$g(\mu_{it}) = \beta_{0t} + \beta_{1t}x_{i1t} + \dots + \beta_{pt}x_{ipt}, \quad i = 1, 2, \dots, n.$$

In special cases, values of explanatory variables and some effects may be the same for each t . For continuous responses, it is common to assume a multivariate normal distribution for \mathbf{Y}_i to obtain a likelihood function that also has correlation parameters to account for within-subject responses being correlated. The model is fitted simultaneously for all t . For discrete

responses, simple multivariate parametric families containing correlation parameters do not exist. A popular approach fits the model by solving *generalized estimating equations* (GEE) that resemble likelihood equations but without constructing a likelihood function. The equations utilize the analyst's naive guess about the form of the actual correlation structure for the repeated responses, such as the *exchangeable* structure by which responses for each pair of times have the same correlation, while using the empirical covariances for the sample data to help generate standard errors that are robust to violations of that naive guess. In this so-called **marginal modeling** approach, some software refers to the standard errors of the model parameter estimates as *sandwich* standard errors, because the covariance matrix on which they are based uses a formula that sandwiches the empirical information between two matrices that relate to the naive guess about the correlation structure.

The second type of model uses **random effects** as a mechanism for generating positive correlations for the pairs of repeated responses. In one simple case, the model for Y_{it} has form

$$g(\mu_{it}) = u_i + \beta_{0t} + \beta_{1t}x_{i1t} + \cdots + \beta_{pt}x_{ipt}$$

The random effect u_i , called a *random intercept*, is an unobserved random variable that is typically assumed to have a $N(0, \sigma^2)$ distribution for unknown σ . A subject i with large positive (negative) u_i tends to have relatively high (low) values for each of $(y_{i1}, y_{i2}, \dots, y_{iT})$, relative to other subjects with the same values of the explanatory variables. This tendency accentuates as σ increases, generating stronger correlations. The model is called a **generalized linear mixed model** because it has a mixture of random effects (the $\{u_i\}$ terms) and fixed effects (the β parameter coefficients of the explanatory variables). Such models can contain more than one random effect, such as **multilevel models** that have a hierarchical structure of random effects. For instance, a study of factors that affect student performance might measure each student's exam scores $(y_{i1}, y_{i2}, \dots, y_{iT})$ on a battery of T exams. Students are nested within schools, and the model could incorporate a random effect for variability among students and another random effect for variability among schools.

In some applications, it can be informative to use previously observed responses as explanatory variables in modeling response t . The model can focus on the dependence of Y_{it} on $\{y_{i1}, y_{i2}, \dots, y_{i,t-1}\}$ as well as the ordinary explanatory variables. Models that include past observations as explanatory variables are called **transitional models**.

The book by Fitzmaurice et al. (2011) presents a good overview of these three main methods for modeling longitudinal data. An example showing how to use R to fit a marginal model and a generalized linear mixed model with random effects follows.

A7.4.1 Marginal Models and Generalized Linear Mixed Models

We shall analyze the `abortion` data file from the book's website, reporting responses of 1850 subjects on whether they supported legalized abortion in three situations: (1) if the family has a low income, (2) when the woman is not married and does not want to marry the man, and (3) when the woman wants it for any reason. Gender of the respondents was also recorded (0: male, 1: female). Thus, the data set consists of two binary variables (gender, response (1: support, 0: do not support)), a variable with 3 levels (question) and a variable indicating the subject (case). For the analysis in the `gge` and `ttfamily` `glmmML` packages, we need to transform the data and create binary responses for each question, as shown next:

```
> abortion <- read.table("http://stat4ds.rwth-aachen.de/data/abortion.dat", header=TRUE)
> abortion[1:6,]
  gender response question case
1      1         1         1     1
2      1         1         2     1
```

```

3      1      1      3      1
4      1      1      1      2
5      1      1      2      2
6      1      1      3      2
> abortion[5545:5550,]
      gender response question case
5545      0      0      1 1849
5546      0      0      2 1849
5547      0      0      3 1849
5548      0      0      1 1850
5549      0      0      2 1850
5550      0      0      3 1850
> question1 <- ifelse(abortion$question==1,1,0)
> question2 <- ifelse(abortion$question==2,1,0)
> question3 <- ifelse(abortion$question==3,1,0) # reference category

```

The marginal model is fitted on these data under independent and exchangeable correlation structures:

```

> library(gee)
> fit.gee <- gee(response ~ gender + question1 + question2, id=case, family=binomial,
+               corstr="independence", data=abortion)
> summary(fit.gee)
Model:
Link:                               Logit
Variance to Mean Relation: Binomial
Correlation Structure:               Independent

Summary of Residuals:
      Min      1Q      Median      3Q      Max
-0.5068800 -0.4825552 -0.4686891  0.5174448  0.5313109

Coefficients:
              Estimate Naive S.E.      Naive z Robust S.E.      Robust z
(Intercept) -0.125407576 0.05562131 -2.25466795  0.06758236 -1.85562596
gender       0.003582051 0.05415761  0.06614123  0.08784012  0.04077921
question1    0.149347113 0.06584875  2.26803253  0.02973865  5.02198759
question2    0.052017989 0.06586692  0.78974374  0.02704704  1.92324166

Estimated Scale Parameter: 1.000721
Number of Iterations: 1

Working Correlation
      [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
>
> fit.gee2 <- gee(response ~ gender + question1 + question2, id=case, family=binomial,
+               corstr="exchangeable", data=abortion)
> summary(fit.gee2)
Model:
Link:                               Logit
Variance to Mean Relation: Binomial
Correlation Structure:               Exchangeable

Summary of Residuals:
      Min      1Q      Median      3Q      Max
-0.5068644 -0.4825396 -0.4687095  0.5174604  0.5312905

Coefficients:
              Estimate Naive S.E.      Naive z Robust S.E.      Robust z
(Intercept) -0.125325730 0.06782579 -1.84775925  0.06758212 -1.85442135
gender       0.003437873 0.08790630  0.03910838  0.08784072  0.03913758

```

```
question1    0.149347107 0.02814374 5.30658404 0.02973865 5.02198729
question2    0.052017986 0.02815145 1.84779075 0.02704703 1.92324179
```

```
Estimated Scale Parameter: 1.000721
Number of Iterations: 2
```

```
Working Correlation
      [,1] [,2] [,3]
[1,] 1.0000000 0.8173308 0.8173308
[2,] 0.8173308 1.0000000 0.8173308
[3,] 0.8173308 0.8173308 1.0000000
```

The second type of models, the GLMM is fitted as follows:

```
> library(glmmML)
> fit.glmm <- glmmML(response ~ gender + question1 + question2, cluster=abortion$case,
+                   family=binomial, data=abortion, method = "ghq", n.points=70, start.sigma=9)
> summary(fit.glmm)
              coef se(coef)          z Pr(>|z|)
(Intercept) -0.61874   0.3777 -1.63839 1.01e-01
gender       0.01259   0.4888  0.02575 9.79e-01
question1    0.83470   0.1601  5.21347 1.85e-07
question2    0.29240   0.1567  1.86622 6.20e-02

Scale parameter in mixing distribution: 8.736 gaussian
Std. Error:                             0.5421

LR p-value for H_0: sigma = 0: 0

Residual deviance: 4579 on 5545 degrees of freedom      AIC: 4589
```

Notice that the item effects are much stronger with the GLMM ($\hat{\beta}_1 = 0.835$, $\hat{\beta}_2 = 0.292$) than with the GEE model assuming an "exchangeable" working correlation structure ($\hat{\beta}_1 = 0.149$, $\hat{\beta}_2 = 0.052$). This is expected, due to the strong underlying correlation (large random effects variability), stronger than assumed under the exchangeable correlation structure. Thus, based on the GLMM, regardless of gender, the estimated odds of supporting legalized abortion for situation (1) equals $\exp(0.83) = 2.3$ times the estimated odds for situation (3). The estimated standard deviation of the random effects distribution is 8.736, indicating high heterogeneity among subjects.

A7.5 Modeling Time Series

When observations occur for a single entity over time, such as recording daily sales of a business over some period of time weekly, the observed data form a sequence $\{y_t\}$ that is a realization of a random process $\{Y_t\}$ called a *time series*. In such applications, prediction of future observations is sometimes more important than estimating and interpreting effects of explanatory variables. A *time series model* is a transitional model that often has as its primary goal predicting future responses based on responses observed so far, such as in economic projections.

Here, we briefly discuss a simple time series model for a quantitative response, to show a mechanism that yields observations that are correlated but in which those farther apart in time are less strongly correlated. The model assumes that $\{Y_1, Y_2, \dots\}$ are *stationary*, which means that $E(Y_t)$, $\text{var}(Y_t)$, and $\rho_s = \text{corr}(Y_t, Y_{t+s})$ for $s = 1, 2, \dots$ do not vary over time (t). The correlation ρ_s between observations s units apart in time is called the s -order

autocorrelation, and the values $(\rho_1, \rho_2, \rho_3, \dots)$ are called the *autocorrelation function*. The sequence of observations

$$Y_t - \mu = \phi(y_{t-1} - \mu) + \epsilon_t, \quad t = 1, 2, 3, \dots,$$

where $|\phi| < 1$ and $\epsilon_t \sim N(0, \sigma^2)$ is independent of $\{Y_1, \dots, Y_{t-1}\}$, is called an **autoregressive process**. The expected deviation from the mean at time t is proportional to the previous deviation. This model satisfies $\rho_s = \phi^s$, with autocorrelation exponentially decreasing as the distance s between times increases. The process is a *Markov chain* (Section 2), because the distribution of $(Y_t | y_1, \dots, y_{t-1})$ is the same as the distribution of $(Y_t | y_{t-1})$. In particular, $\text{corr}(Y_t, Y_{t+1}) = \phi$ but $\text{corr}(Y_t, Y_{t+2} | Y_{t+1}) = 0$.

To illustrate the use of R to generate an autoregressive process and to fit an autoregressive model and predict future observations, we start at $y_1 = 100$ and generate 200 observations with $\mu = 100$, $\phi = 0.90$, and $\sigma = 10$:

```
> y <- rep(0, 200)
> y[1] <- 100
> for (t in 2:200){
+   y[t] <- 100 + 0.90*(y[t-1] - 100) + rnorm(1,0,10)
+ }
> plot(y, xlim=c(0,210))           # plots the time index t against y[t]
> acf(y, lag.max=10, plot=FALSE)   # autocorrelation function
Autocorrelations of series 'y', by lag
    0    1    2    3    4    5    6    7    8    9   10
1.000 0.917 0.856 0.779 0.706 0.644 0.574 0.517 0.437 0.355 0.290
> fit <- ar(y, order.max = 1, method = c("mle")) # fit autoregressive model by ML
> fit$ar
[1] 0.9180788 # ML estimate of phi parameter in model
> pred10 <- predict(fit, n.ahead=10)           # predict next 10 observations on y
> pred10$pred # predicted y for next 10 observations
Start = 201    End = 210
[1] 66.146 68.350 70.374 72.231 73.937 75.503 76.940 78.260 79.472 80.58 4
> pred10$se    # standard errors of predicted values
[1] 10.275 13.949 16.419 18.243 19.649 20.761 21.653 22.378 22.971 23.459
points(201:210, pred10$pred, col="red") # add on plot next 10 predicted y values
```

Figure A7.3 shows the time series. Observations that are close together in time tend to be quite close in value. The plot looks greatly different than if the observations were generated independently, which results for the special case of the model with $\phi = 0$. After generating the data, we use the `acf` function in R to find the sample autocorrelation function, relating to times t and $t+s$ with lag s between 1 and 10. The first-order sample autocorrelation ($s = 1$) is 0.921, and the values are weaker as the lag s increases. We then fit the autoregressive model with the `ar` function in R. The ML estimate of $\phi = 0.90$ is $\hat{\phi} = 0.918$. One can use the fit of the model to generate predictions of future observations as well as standard errors to reflect the precision of those predictions. Predicting ahead the next 10 observations, the R output shows that the predictions tend toward $\mu = 100$ but with standard error that increases as we look farther into the future. Figure A7.3 also shows, in red, the predictions for the next 10 observations.

A more general autoregressive model, having order p instead of 1, is

$$Y_t - \mu = \sum_{j=1}^p \phi_j (y_{t-j} - \mu) + \epsilon_t, \quad t = 1, 2, \dots$$

It uses the p observations before time t in the linear predictor. This process is a higher-order Markov chain, in which Y_t and $Y_{t-(p+1)}$ are conditionally independent, given the observations

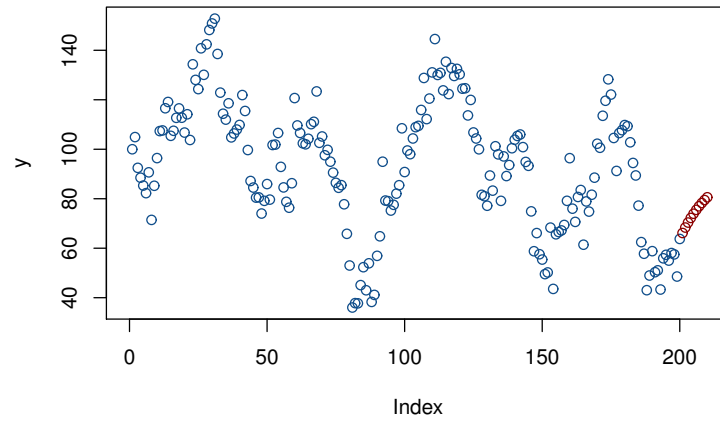


FIGURE A7.3: Observations from autoregressive time series process, with predicted values of next 10 observations shown in red.

in between those two times. An alternative time series model, called a *moving average* model of order p , has form

$$Y_t - \mu = \sum_{j=1}^p \lambda_j \epsilon_{t-j} + \epsilon_t, \quad t = 1, 2, \dots$$

It has $\rho_s = 0$ for $s > p$, so it is appropriate if the sample autocorrelations drop off sharply after lag p . A more general *ARMA* model, more difficult for interpretation but potentially useful for prediction, can include both autoregressive and moving average terms. The models generalize also to include explanatory variables other than past observations of the response variable and to allow observations at irregular time intervals. See Brockwell and Davis (2016) and Cryer and Chan (2008) for introductions to time series modeling.



CHAPTER 8: R FOR CLASSIFICATION AND CLUSTERING

A8.1 Visualization of Linear Discriminant Analysis Results

For linear discriminant analysis, the `ggplot2` package offers options to visualize the data and results. The following code illustrates, for the example using the `Crabs` data file to predict whether a female horseshoe crab has satellites (Section 8.1.2). That section notes that it is sufficient to use weight or carapace width together with color as explanatory variables. Here, we use width and color. Notice that the functions in `ggplot2` require the categorical variables that are used for example to color points to be defined as factors. In `Crabs`, the binary response Y is not a factor, for this we first define it as a factor. We calculate the predicted accuracy (overall proportion of correct predictions) and verify that it is 0.728, higher than 0.688, which was based on a classification using leave-one-out cross-validation (see Section 8.1.3):

```
> Crabs <- read.table("http://stat4ds.rwth-aachen.de/data/Crabs.dat",
+                   header=TRUE)
> library(MASS)
> fit.lda <- lda(y ~ width + color, data=Crabs)           # linear discriminant analysis
> prob1 <- predict(fit.lda)$posterior[,2]                # estimated P(Y=1)
> predY <- predict(fit.lda)$class                        # predicted class of Y
> library(ggplot2)
> Crabs$y <- factor(Crabs$y)                            # need factor response variable for ggplot2
> qplot(width, color, data=Crabs, col=y)                 # Figure A8.1 (i)
> qplot(width, color, data=Crabs, col=predY)             # Figure A8.1 (ii)
> Crabs$pred.right = predY == Crabs$y
> round(mean(Crabs$pred.right), 3)                      # prediction accuracy
[1] 0.728                                                # without cross validation
> qplot(width, color, data=Crabs, col=pred.right)        # Figure A8.1 (iii)
> ggplot(data=Crabs, aes(x=width, y=color, color=prob1)) + # Figure A8.1 (iv)
  geom_point(alpha=0.6) +
  scale_color_gradient2(low="white", high="darkblue")
```

Figure A8.1 shows the plots. The first one shows the observed y values, at the various width and color values. The second plot shows the predicted responses. At each color, the crabs with greater widths were predicted to have $y = 1$, with the boundary for the predictions moving to higher width values as color darkness increases. The third plot identifies the crabs that were misclassified. The last plot shades the points according to their estimated probabilities.

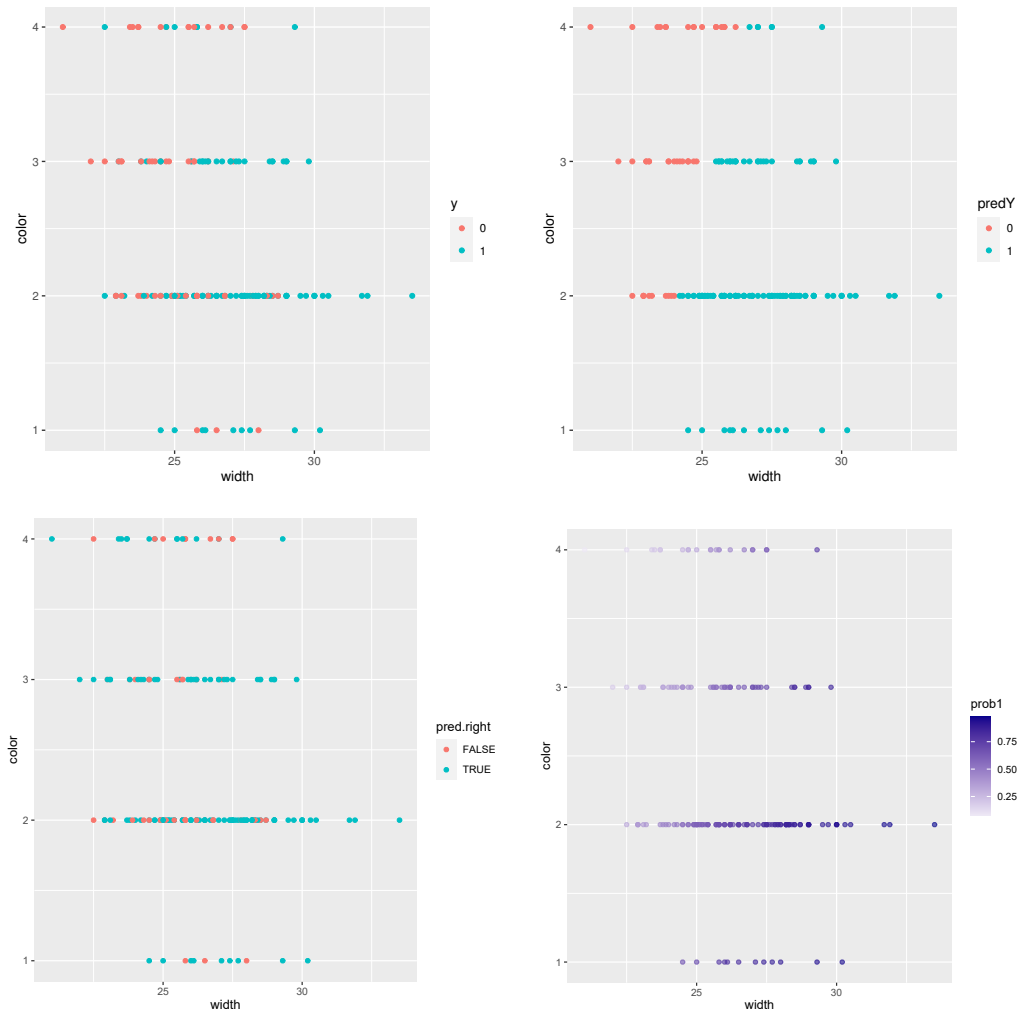


FIGURE A8.1: Scatterplots of width and color values for linear discriminant analysis to predict y = whether a female horseshoe crabs has male satellites, colored after (i) the observed value of y (upper left), (ii) the predicted value of y (upper right), (iii) the missclassification (lower left) and (iv) the estimated probability of having satellites (lower right).

A8.2 Cross-Validation in R

For classification methods, leave-one-out cross-validation (*loocv*) provides more realistic values for the probability of a correct classification (Section 8.1.3). For penalized likelihood methods, Section 7.7.2 mentioned that the choice of tuning parameter λ can be based on k -fold cross-validation (k -fold *cv*). The *loocv* is the extreme case of k -fold *cv* for $k = n$. Increasing k , especially for large n , may be computationally difficult. Commonly the values $k = 5$ or $k = 10$ are considered in practice. The choice $k = 10$ is predominant, since it usually performs similarly to *loocv*.¹

Model training refers to randomly partitioning the data frame into a training sample (typically 70%-80% of the observations) and a test sample, fitting the model on the training sample, and checking the fit's accuracy when applied to the testing sample. Cross-validation is used for fitting the model, with 10-fold *cv* by default. For a k -fold *cv*, the choice of k is controlled by the corresponding argument. Thus, one has to set `trControl = trainControl(method = "loocv")` for *loocv* or `trControl = trainControl(method = "cv", number=5)` for a 5-fold *cv*. The *caret* package for classification and regression training applies on models fitted in basic R packages such as *stats*, *MASS*, and *rpart*. One specifies the desired modelling function (`glm`, `lda`,...) in the `method=` argument of the `train` function. The `createDataPartition` function randomly partitions the data frame to training and testing subsamples. We illustrate in the linear discriminant analysis context, for the horseshoe crabs example in Section 8.1.2, using 70% of the observations in the training sample and for 10-fold *cv*:

```
> library(caret); library(ggplot2); library(MASS)
> Crabs$y <- factor(Crabs$y)      # need factor response variable for ggplot2
> index = createDataPartition(y=Crabs$y, p=0.7, list=FALSE)
> train = Crabs[index,]           # training sample, 70% of observations (p=0.7 above)
> test = Crabs[-index,]           # testing sample
> dim(train)
[1] 122  7      # 122 observations of 7 variables in training sample
> dim(test)      # (8 variables if include pred.right in Crabs, as in above code)
[1] 51  7      # 51 observations of 7 variables in testing sample
> lda.fit = train(y ~ width + color, data=train, method="lda",
+               trControl = trainControl(method = "cv", number=10))
> names(lda.fit)      # lists what is saved under "lda.fit"; not shown
> summary(lda.fit)     # output not given
> predY.new = predict(lda.fit, test) # prediction for testing subsample
> table(predY.new, test$y)
predY.new 0 1      # output varies depending on training sample used
          0 5 2
          1 13 31
> round(mean(predY.new == test$y)*100, 2)      # prediction accuracy
[1] 0.706
> qplot(width, color, data=test, col=y)        # first plot in Figure A8.2
> test$pred.right = predY.new == test$y
> qplot(width,color, data=test, col=pred.right) # second plot in Figure A8.2
```

Figure A8.2 provides plots that are analogous to the first and third in Figure A8.1, but now only for the 51 cases in the testing subsample. The first plot shows the actual y values and the second shows the misclassified observations.

Model training is an essential step with neural networks. The *keras* package (see Chollet and Allaire 2018) is useful for this method.

¹e.g., see Molinaro, Simon, Pfeiffer (2005). Prediction error estimation: a comparison of resampling methods. *Bioinformatics*, 21 3301–3307

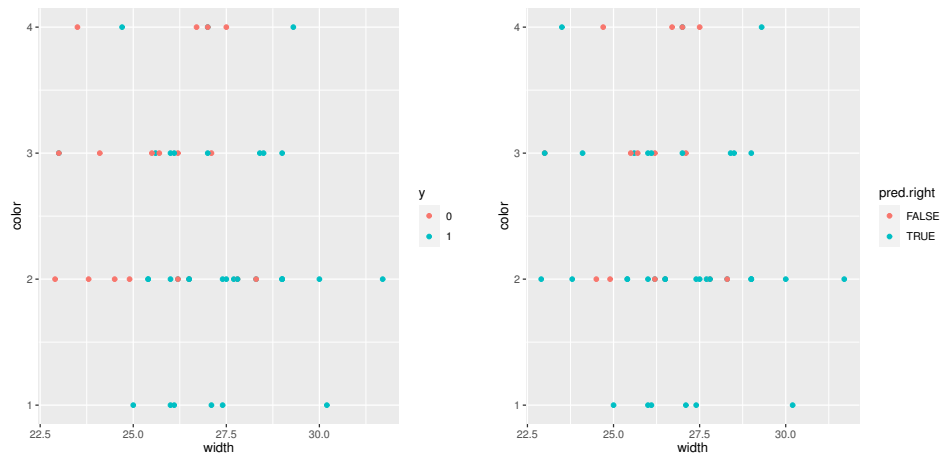


FIGURE A8.2: Scatterplots of width and color values for horseshoe crabs in testing sub-sample, showing (i) observed value of y , (ii) missclassified observations.

A8.3 Classification and Regression Trees

Section 8.1.4 discussed classification trees for binary response variable. The **rpart** and **tree** packages can also handle categorical response variables with multiple categories. Next we provide such an example.

For the GSS2018 survey data introduced in Section A1.2.1, we consider as response the vote of the participants in the 2016 Presidential elections, which has four levels. The classification tree is provided in Figure A8.3 along with the plot of the cross-validated accuracy rate for different values for the complexity parameter λ . Our tree has $\lambda = 0.029$ and we see from the cross-validated accuracy plot that we have an accuracy of about 70%. Observe that finally voters are classified only in two categories (Clinton/Trump), since in none of the classes the other two categories reached a high percentage of voters. (the percentages of all four categories are provided in the second line of every node). The code for this analysis follows:

```
> GSS <- read.table("http://stat4ds.rwth-aachen.de/data/GSS2018.dat", header=T)
> suppressMessages({library(rattle) # for fancy classification tree
  library(caret)
  library(rpart)
  library(tidyverse)})
> GSS2018full <- na.omit(GSS)
> GSS2018full$PRES16 <- factor(GSS2018full$PRES16, levels=c(1,2,3,4),
  labels=c("Clinton", "Trump", "Other", "Not vote"))
> GSS2018f <- GSS2018full %>% select(-PARTYID) %>%
  mutate_at(vars(-c("AGE", "EDUC")), as.factor) %>%
  mutate_at(vars(c("AGE", "EDUC")), as.double)

> sapply(GSS2018f, class) # check the class of the variables

> index = createDataPartition(y=GSS2018f$PRES16, p=0.7, list=FALSE)
> train = GSS2018f[index,]
> test = GSS2018f[-index,]
> dim(train)
> dim(test)
> GSS2018.tree = train(PRES16 ~ ., data=train, method="rpart",
  trControl = trainControl(method = "cv"))
```

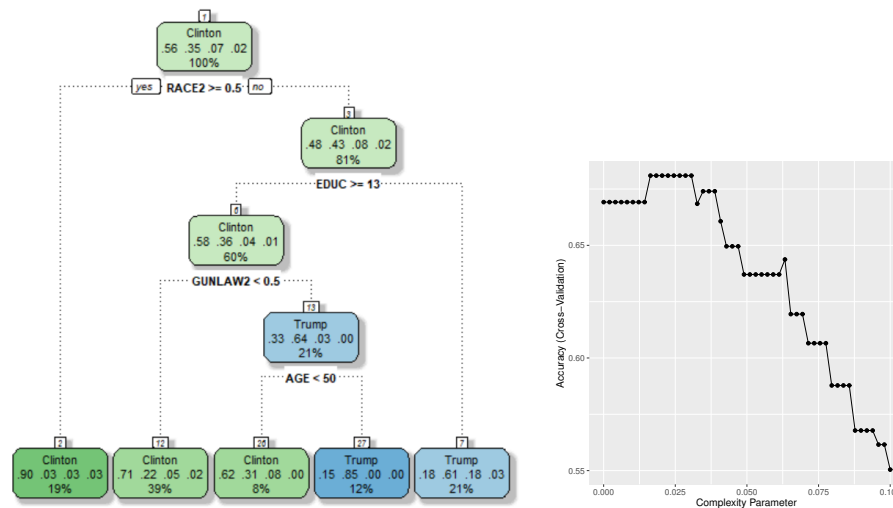


FIGURE A8.3: Decision tree of depth 4 for the Presidential elections 2016 vote of the participants in the GSS2018 survey (left) and cross-validated accuracy rate for different values for the complexity parameter (right).

```
> GSS2018.tree
#----- output -----
CART
160 samples
12 predictor
  4 classes: 'Clinton', 'Trump', 'Other', 'Not vote'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 144, 145, 145, 143, 144, 144, ...
Resampling results across tuning parameters:
   cp      Accuracy   Kappa
0.02857143  0.6811765  0.37562782
0.08571429  0.6451471  0.30188450
0.11428571  0.5876471  0.09631507
Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.02857143.
#-----
> GSS2018.pred = predict(GSS2018.tree, newdata = test)
> table(GSS2018.pred, test$PRES16)
GSS2018.pred Clinton Trump Other Not vote
Clinton      33      8      1      0
Trump         5     15      3      0
Other         0      0      0      0
Not vote      0      0      0      0
> error.rate = round(mean(GSS2018.pred != test$PRES16),3)
> error.rate
[1] 0.262
> fancyRpartPlot(GSS2018.tree$finalModel) # Figure A8.3 (left)
> ggplot(GSS2018.tree)                    # Figure A8.3 (right)
> summary(GSS2018.tree$finalModel) # long output with detailed information (not shown)
```

The `rpart` and `tree` packages can also form trees when the response is quantitative, in which case the display is called a *regression tree*. The predicted value at a

terminal node is the mean response for the subjects in the region of predictor values for that node. The site <https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf> has useful examples of `rpart` for binary, multiple category, and quantitative responses.

A8.4 A Cluster Analysis with Quantitative Variables

Section 8.2 showed a hierarchical cluster analysis for binary variables, using the agglomerative clustering algorithm. An alternative clustering method, called *divisive hierarchical clustering*, applies in the opposite direction, starting with one cluster and splitting them until reaching a prespecified bound for the average dissimilarity. Methods for measuring the dissimilarity between clusters are called *linkage* methods. Section 8.2.3 used the average linkage. Alternative methods of linkage include the complete (farthest neighbor), single(nearest neighbor), centroid and Ward's minimum variance method.

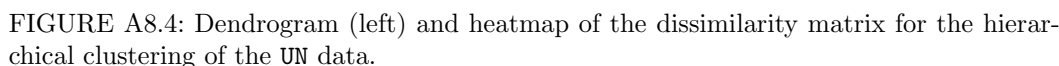
Measures for measuring the dissimilarity between two observations depend on the type of the variables to be clustered. In case of binary variables, this dissimilarity is measured in terms of the *Manhattan* distance (see Section 8.2.3). For quantitative variables, the default dissimilarity measure for most software is *Euclidean distance*, which is a distance measure for multidimensional Euclidean space. For two observations (x_1, \dots, x_p) and (y_1, \dots, y_p) on p variables, this is $\sqrt{(x_1 - y_1)^2 + \dots + (x_p - y_p)^2}$.

We illustrate for the UN data file, using Euclidean distance (all variables are quantitative) and agglomerative clustering. Using the `agnes` function in the `cluster` package, we chose the linkage method (Ward distance) having the highest *agglomerative coefficient*, which is a measure of the strength of the clustering structure, values closer to 1 suggesting stronger structure. The R code for our cluster analysis follows:

```
> UN <- read.table("http://stat4ds.rwth-aachen.de/data/UN.dat", header=T)
> suppressPackageStartupMessages({
  library(tidyverse)
  library(cluster)
  library(gplots)      # heatmap.2()
  library(reshape2) })
> UN_scaled <- UN %>%
  select_if(is.numeric) %>% # selects only numeric variables
# select(-C02) %>%         # command to remove a variable, not activated here
  mutate_all(as.double) %>%
  scale()
> row.names(UN_scaled) <- UN[,1] # Declare column 1 as the row names
> d <- dist(UN_scaled, method="euclidean") # dissimilarity matrix
> hc1 <- agnes(d,method="single"); hc1$ac
[1] 0.6978301
> hc2 <- agnes(d,method="average"); hc2$ac # hierarchical clustering with average linkage
[1] 0.808305 # the agglomerative coefficient for hc2 is printed
> hc3 <- agnes(d,method="complete"); hc3$ac #
[1] 0.8667266
> hc4 <- agnes(d,method="ward"); hc4$ac #
[1] 0.9211221

> hc <- hclust(d,method="ward.D2")
> plot(hc, hang = -1, cex = 0.6) # Figure A8.4 (left)
> dist.f <- function(x) dist(x, method = 'euclidean')
> hclust.w <- function(x) hclust(x, method="ward.D2")
> heatmap.2(d, distfun=dist.f, hclustfun=hclust.w, # Figure A8.4 (left)
  trace="none", dendrogram = "row",
```

Figure A8.4 shows the produced dendrogram (also shown in Figure A.20 of the book) and the heatmap of the dissimilarity matrix.



Cluster analysis is a descriptive method, so there is not really ‘optimal’ number of clusters. The decision has to be taken also having in mind the content of the data and the survey question we focus on. In Figure A8.5 we provide the clustering for two up to five clusters to gain a better insight to our data and how countries are clustered.

```
# k-means Clustering:
> fit2 <- kmeans(UN_scaled, 2) # k = 2 clusters
> attributes(kmeans.fit) # output not shown here

> fit2$centers # Centroids of clusters
      GDP      HDI      GII  Fertility      CO2
1 -1.0260097 -0.9969244  1.0650061  0.5887976 -0.8024088
2  0.6976866  0.6779086 -0.7242041 -0.4003824  0.5456380
      Homicide      Prison      Internet
1  0.5667460  0.07503587 -0.9995534
2 -0.3853873 -0.05102439  0.6796963

> fit2$cluster # Cluster Id per case (not shown here)
> fit2$size # Cluster size
```

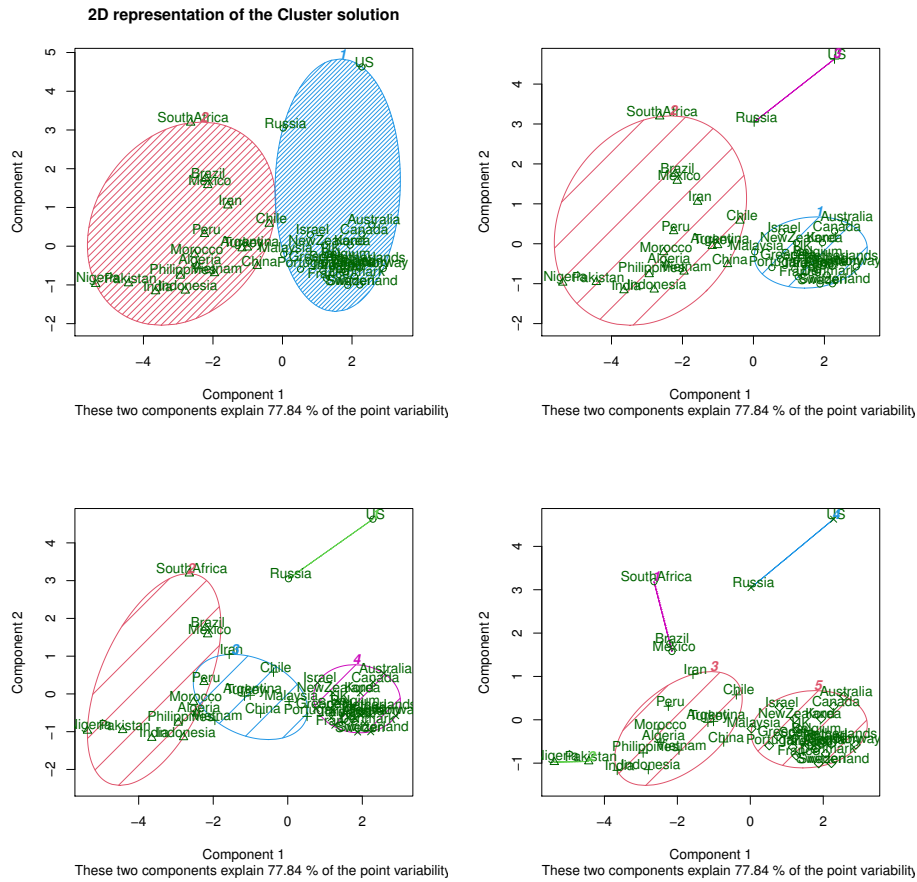


FIGURE A8.5: Clustering plots of the UN data, corresponding to k-means clustering (from 2 up to 5 clusters).

[1] 17 25

```
> clusplot(UN_scaled, kmeans.fit$cluster, main=' ', # Figure A8.5
  color=TRUE, shade=TRUE,
  labels=2, lines=0)
```

In general, in *k*-means clustering the optimal number of clusters can be estimated based on methods minimizing the total within clusters sum of square (wss), the average silhouette width ² or based on gap statistics ³. Figure A8.6 provides the optimal number of clusters plot based on the three methods discussed above for the *UN* data file, produced as follows:

```
> library(factoextra)
> library(gridExtra)
> layout(matrix(c(1,2,3),1,3))
> p11 <- fviz_nbclust(UN_scaled, FUNcluster = kmeans,
```

²Rousseeuw (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis, *Journal of Computational and Applied Mathematics*, 20, 53–65.

³Tibshirani, Walther, Hastie (2001). Estimating the number of clusters in a data set via the gap statistic, *J. R. Statist. Soc. B*, 63, 411–423.

```

method = "wss" ,      # other options: "silhouette", "gap_stat"
diss = d, k.max = 10)
> p12 <-fviz_nbclust(UN_scaled, FUNcluster = kmeans,
method = "silhouette" ,
diss = d, k.max = 10)
> p13 <-fviz_nbclust(UN_scaled, FUNcluster = kmeans,
method = "gap_stat" ,
diss = d, k.max = 10)
> grid.arrange(p11,p12,p13, nrow=1) # to print all plots in a matrix form

```

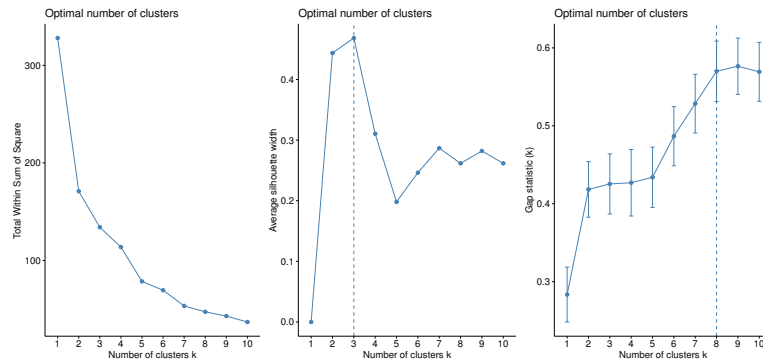


FIGURE A8.6: Optimal number of clusters using the method based on (i) the total within clusters sum of square (left), (ii) the average silhouette width (middle), and (iii) the gap statistics (right).

Beyond the types of clustering algorithms discussed above, which depend only on the data without assuming an underlying stochastic model, clustering methods are also available in a modeling context. For example, *Gaussian mixture models* assume that k multivariate normal distributions generate the observations. Model-based clustering is available with the `mclust` package.



Bibliography

- Agresti, A. (2015). *Foundations of Linear and Generalized Linear Models*. Wiley.
- Agresti, A. (2019). *An Introduction to Categorical Data Analysis*, 3rd ed. Wiley.
- Agresti, A., Franklin, C., and Klingenberg, B. (2021). *Statistics: The Art and Science of Learning from Data*, 5th ed. Pearson.
- Albert, J. (2009). *Bayesian Computation with R*, 2nd ed. Springer.
- Allison, P.D. (2014). *Event History and Survival Analysis*, 2nd ed. Sage.
- Baumer B. S., Kaplan D. T., and Horton N. J. (2017). *Modern Data Science with R*, CRC Press.
- Burnham, K. P., and Anderson, D. R. (2002). *Model Selection and Multi-Model Inference* 2nd ed. Springer-Verlag.
- Carlin, B., and Louis, T. (2008). *Bayesian Methods for Data Analysis*, 3rd ed. CRC Press.
- Casella, G., and Berger, R. (2002) *Statistical Inference*, 2nd edition. Wadsworth and Brooks/Cole.
- Chollet, F., and Allaire, J. J. (2018). *Deep Learning with R*. Manning Publications.
- Cryer, J. D., and Chan, K.-S. (2008). *Time Series Analysis, with Applications in R*. Springer Publishing.
- Davison, A. C. (2003). *Statistical Models*. Cambridge University Press.
- Ellenberg, J. (2014). *How Not to Be Wrong: The Power of Mathematical Thinking*. Penguin Books.
- Everitt, B. S., Landau, S., Leese, M., and Stahl, D. (2011). *Cluster Analysis*, 5th edition. Wiley.
- Fitzmaurice, G., Laird, N., and Ware, J. 2011. *Applied Longitudinal Analysis*, 2nd edition. Wiley.
- Gelman, A., and Hill, J. (2006). *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge Univ. Press.
- Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning*, 2nd ed. Springer.
- Hoff, P. D. (2009). *A First Course in Bayesian Statistical Methods*. Springer.
- Hollander, M., Wolfe, D., and Chicken, E. (2013). *Nonparametric Statistical Methods*, 3rd ed. Wiley.
- Hothorn, T., and Everitt, B. S. (2014). *A Handbook of Statistical Analyses Using R*, 3rd ed. CRC Press, Boca Raton FL.
- Irizarry, R. A. *Introduction to Data Science*. (2019). Chapman and Hall/CRC.
- James, G., D. Witten, T. Hastie, and R. Tibshirani. (2021) *An Introduction to Statistical Learning*, 2nd edition. Springer.
- Kateri, M. (2014). *Contingency Table Analysis: Methods and Implementation Using R*. New York: Birkhäuser.

- Rashid, T. (2016). *Make Your Own Neural Network*. Amazon.
- Salsburg, D. (2002). *The Lady Tasting Tea: How Statistics Revolutionized Science in the Twentieth Century*. W. H. Freeman.
- Shumway, R. H., and Stoffer, D. S. (2017). *Time Series Analysis and Its Applications, with R Examples*, 4th ed. Springer.
- Stigler, S. M. (1986). *The History of Statistics: The Measurement of Uncertainty before 1900*. Harvard University Press.
- Stigler, S. M. (2002). *Statistics on the Table*. Harvard University Press.
- Stigler, S. M. (2016). *The Seven Pillars of Statistical Wisdom*. Harvard University Press.
- Tukey, J. W. (1977). *Exploratory Data Analysis*. Addison–Wesley.
- Tutz, G. 2011. *Structured Regression for Categorical Data*. Cambridge Univ. Press.
- Wickham, H., and Golemund, G. (2017) *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. O'Reilly Media Inc., Sebastopol, CA.