|  |
| --- |
| IU |
| Statistical Computing |
| DLBDBSC01\_E |

# Learning Objectives

With the ever-increasing amount of data collected by institutions and companies all over the world, the task of distilling information and insights from that data becomes increasingly important. At the same time, that extensive amount of data presents challenges for the computation of statistics, which in many cases requires the use of a specialized statistics program. The course **Statistical Computing** gives you a hands-on introduction to statistical analyses using such a program. The program and programming language used in this course is the widely-used R statistics package.

As a first step, you will learn how to set up R on your computer and learn about the basic commands and properties of that language. After an introduction to data management in R (importing and exporting data from external files and formats), you can start analyzing that data. Beginning with simple descriptive statistics, you will from there move on to inferential analysis and then more complex investigations of data structures and relationships between data series.

# Unit 1 – Introduction to Statistical Computing

**Study Goals**

On completion of this unit, you will be able to

* define statistical computing.
* distinguish between statistical programs and statistics programming languages.
* set up a development environment for the statistical program R.

## Introduction

One main task of statistics is identifying information and structures in data, formulating models for them, and then interpreting the information, structures, and models to generate knowledge. In his 1982 book *Megatrends*, John Naisbitt wrote “We are drowning in information but starved for knowledge” (Naisbitt, 1982, p. 24). And what was true in 1982, is certainly still true today. The amount of collected data is constantly increasing and much of that data is easily available worldwide with the help of the internet (Reinsel et al., 2017). Fortunately, technological progress has allowed for an increased amount of data collection, storage, and provision, as well as the increased computing power required for the use of modern statistical methods (Hilbert, 2011).

## 1.1 Definition and Delimitation

**Computer Science**

This subject includes the study of computers and computing, including their hardware, software, and use for information processing.

The term ”statistical computing” is often used synonymously with ”computational statistics”. However, they should be distinguished according to Lauro's recommendation (see figure below). He defines the aim of statistical computing to be the design of algorithms for implementing statistical methods using computers (e.g., the development of statistical software), whereas statistical computing is the application of **computer science** to statistics. Both designations, therefore, have similarities but are nonetheless different (Lauro, 1996).

Distinction between Computational Statistics and Statistical Computing

Diagram

Description automatically generated

So, simply put, computational statistics refers to those areas of statistics that require computationally intensive calculations (such as Monte Carlo simulations). Statistical computing refers more to the application of computer-aided, numerical methods in statistics (Lauro, 1996).

Thisted (1988) describes this intersection between statistics and computer science as follows:

Statistics deals with...

* the collection of information,
* how information can be best extracted from data,
* how data can be collected so that the information content can be maximized, and
* how conclusions that contribute to expanding knowledge can be drawn from data.

Computational science is concerned with...

* how the calculations can be carried out optimally (both numerically and symbolically) to derive the right insights from data and to prepare and present them accordingly.

### Self-Check Questions

1. True or false: Statistical computing and computational statistics are synonyms for the same field.

* True
* *False*

**Programming language**

A programming language is a system of notations that enables describing a sequence of steps to be performed by a computer. (Horowitz 2012).

## 1.2 Statistics Program vs. Statistics Programming Language

A statistics program is an application that enables statistical analyses. A statistics programming language is a **programming language** that allows the user to define command sequences that can be compiled or interpreted to perform statistical analyses.

The work of McNamara (2018) summarizes some essential properties that a modern statistics program should have, such as availability, support for numerous analysis methods, and simple presentation of the results. These properties are fulfilled by the program R.

What is R and Why R?

R is both a statistics programming language and a statistics program. It is a self-contained environment for calculations and can also display graphical output. In R, a wide variety of numerical and statistical tasks can be performed. Simple numerical calculations, as you would normally do on a hand calculator, can be carried out, but also complex algorithms can be programmed and executed. The results can be displayed immediately in a graphic window. This integrative approach of R is one of its main advantages. Being an open and transparent programming language with the ability to directly display graphical results has turned it into one of the most popular statistical programs (R Core Team, n.d.-a).

Programming in R is done either by directly entering commands in a console (the R console) or by writing a script (e.g., in an editor like Notepad++ or an R-specific programming environment). Scripts combine sequences of commands and function calls that are written in the programming language R and are then loaded (e.g., by copying the script to the R console) and executed.

**GNU**

The name for this extensive collection of free software is a recursive acronym for 'GNU's Not Unix!'. (Stallman, n.d.)

R originated from the language S, which was developed by Becker, Chambers, and Wilks (1988) and is now implemented in the commercial variant S-Plus. R, on the other hand, is the freely available version of S and is therefore often referred to as **”GNU**-S”. This open access has meant that it has spread rapidly throughout the statistics community and has thus been further improved, expanded, and developed, from the expansion and improvement of available integrated development environments (**IDE**) to the creation of a vast number of “packages” that have been developed for the execution of special tasks (R Core Team, n.d.-a). These are two of the major advantages of R: its free availability and the extensive number of packages covering a vast range of application areas.

**IDE**

For R and many other programming languages, integrated programming environments (IDEs) provide comprehensive features to facilitate software development.

Finally, the programs created in R, which are saved in the form of scripts, are platform-independent. This means that scripts that are, for example, written on a Linux operating system can be run on Windows or Mac without changes (R Core Team, 2022b). This highest possible flexibility is another reason for the widespread use of R.

One property of R that many beginners initially perceive as a disadvantage is that it is command-driven. This means that any kind of data analysis must be performed by the user by entering commands in text form. At first glance, this seems to speak against R and in support of easier-to-use spreadsheet programs such as Excel, where you can select the desired analysis tools in a graphical user interface. However, over time, numerous graphic interfaces have been developed for R (e.g., RStudio, Jupyter) that continue to allow commands to be entered directly in text form. This ensures increased transparency and maintainability. It also makes it easier to hand programs over to third parties, since everything is written in the command sequence. Other developers with R knowledge can then understand that command sequence and easily adapt it to their own needs.

By simply using and linking the numerous functions already integrated into R, more complex analyses can be carried out with little programming knowledge. For the enormous advantages of R to be exploited, the student must be willing to acquire a certain vocabulary of commands and to learn the command syntax. Like any other programming language, R is intolerant of typing errors, which can lead to frustration when trying to use R for the first time. To help avoid this, we will first give a brief introduction to the basic functions of R as a programming tool, before going into the statistics program in a graphical user environment.

R is an **interpreter language**, not a compiler language. In classic compiler programming languages (e.g., Fortran, C, or C++), a sequence of commands (algorithms) must be entered into an editor and then translated (i.e., compiled) into a machine language. For interpreter languages, the code is not converted into a machine language (compiled) but is executed using a so-called interpreter (Adler, 2010).

**Interpreter Language**

An interpreter or scripting language allows for writing scripts that can be executed without having to be compiled first into a machine language program.

Interpreter languages perform significantly less efficiently on algorithms that require many loop iterations (such as "for" and "while” loops). In most cases, however, such time-consuming loops can be avoided through clever programming. In addition, functions and programs from other languages such as C++ can be easily included in R. This makes it possible to write time-consuming arithmetic tasks in an external, compiled program and utilize them in the R script if necessary. This can significantly speed up the runtime of the program but is usually no longer necessary with today's computer performance. Plus, the functions integrated into R and the available additional packages are precompiled so there is no or very little loss of speed (Morandat et al., 2012).

SPSS

IBM SPSS Statistics is a commercial program that is particularly popular in the social sciences and psychology (Gunarto 2019). A key reason for its popularity is its ease of use as it is menu driven and you can get results with just a few mouse clicks. There are cheap license agreements for students with terms of one year for the basic program, but you must pay extra for each expansion pack (IBM, 2020).

SAS

One of the most expensive statistical programs is SAS. It includes many additional and chargeable packages and its own programming environment (SAS, n.d.). This makes it significantly more difficult to learn.

### Self-Check Questions

1. What are some advantages of R?

* It is purely menu-based and therefore does not require the user to learn a new program or programming language.
* *There are many packages for many different statistical analyses available that allow to easily perform complex analyses with this open source program.*

1. R is a scripting language, also called ...

* *… an interpreter language.*
* … a compiler language.

## 1.3 Setting up the Working Environment

The best way to get to know R is "learning by doing". For this, it is necessary to install it on a computer. You can download and install R from the official R site (R Core Team, 2019). Here you will find the current version of R and numerous documents (tutorials) as well as additional packages that can be installed later. Although the R language and scripts are platform-independent, there is a precompiled version for each operating system (Windows, Linux, Mac OS X). The following figure shows the home page of R-CRAN (the Comprehensive R Archive Network) (as of April 2022).

Main page of R-CRAN for the download of R and additional information

Graphical user interface, text, application, email

Description automatically generated

You can select a local address via the “Mirrors” option in the menu on the left. Under ‘Download and Installation’, you will be directed to the desired operating system version, where you should first select the base program under “base”, which, for this document, is sufficient for the time being. You will then be guided through the installation program step-by-step. It is recommended that you install R as an administrator, but you can also install it as a normal user in any folder on the computer. The folder must be created beforehand. If you encounter any problems during installation, FAQs and online tutorials are available to help. In the Ubuntu Linux version, R can easily be set up via the Software Center (with the search input "r-base" and subsequent selection of the installation option).

First Steps

The following steps refer to R on Windows (R on Linux can be easily started in a terminal with the "R" command after a successful installation). After starting R.exe (by default, it is located in C:\Program Files\R\R-4.1.3\bin) the following R console, also known as the command window, appears:

Input console after starting the R program under Windows

Text

Description automatically generated

Commands are now entered by the user after the prompt character (input request), which is a greater-than sign (">"), as can be seen in the previous figure. The command itself is executed by R when the <Enter> key is pressed. If the command for R can be read without errors, the new prompt ">" appears after the result is displayed on a new line. If, however, the command is incomplete, a "+" sign appears on a new line to prompt you to complete the command you have started. If you enter an incorrect command, an appropriate error message is displayed. All commands entered in R are stored in the main memory and can be re-called, edited, and executed again. They can be reached with the arrow keys. By pressing the ESC key (or Ctrl + C under Linux), running processes can be stopped and the prompt symbol appears again (Baker & Kim, 2017).

As can be read in the header when the R program is started (see the figure "Input console after starting the R program under Windows"), the R program can be closed by entering q() as a short form for quit(). This is one of the first basic functions to remember. More on functions will follow later.

### Self-Check Questions

1. True or false: As R is platform-independent, the same precompiled binaries for the installation of R can be used for all operation systems.

* True.
* *False.*

1. After installing R from the official website...

* … you also have to install an IDE to be able to execute commands.
* *… the R terminal provides a first environment to execute commands.*

Summary

In modern times, statistical problems are usually solved with dedicated computer programs. Statistical computing is part of the intersection between computational science and statistics. Statistics programs like spreadsheet programs that guide the user to the desired analysis with the help of a graphical interface and menus are user-friendly, but often are expensive and have limits regarding expandability, transparency, and maintainability. Programming languages specially designed for statistics require more learning effort but allow the user the greatest possible freedom of use and the resulting scripts can be maintainable and transparent. The R program is a statistics programming language for which graphical interfaces have been developed and which, therefore, combines the advantages of both approaches. R is freely available over the internet and can be downloaded and installed for any operating system (see R Foundation 2022). There you will also find numerous guides and tutorials as well as a vast selection of additional packages that have been developed by R users for specific tasks and are freely available to the statistics community. After successful installation, R’s input window, the R console, appears when the program starts.

# Unit 2 – Basics of programming with R

**Study Goals**

On completion of this unit, you will be able to

* use R as a simple hand calculator.
* differentiate between assignments and expressions.
* explain the basic elements and data structures required for programming in R.

## Introduction

Traditionally, learning a new programming language is started by learning the commands to produce the output “Hello World”. The steps to do so depend on the language and the intended purpose. For R as a statistics programming language, it makes sense to start with learning how to execute simple mathematical calculations. This will illustrate the first core concepts of the language using concrete examples before diving into more general topics like data types and functions.

In the following, inputs and outputs are marked in a monospace font. Initially, we will show inputs with the prompt character included, which is given by R and is not to be entered by the user. An entry is always concluded with the <Enter> key so this will not be explicitly mentioned in the following (Baker & Kim, 2017).

## 2.1 R as Pocket Calculator

The first, quite simple use case shows the use of R as a hand calculator. This will also illustrate the first core principles of programming in R. One of these core principles is the distinction between the input of expressions and assignments.

When an expression is entered in the console and the <Enter> key is pressed, the expression’s results are displayed directly in the console or an extra window (Baker & Kim, 2017). The results of calculations are usually shown in the terminal itself; the generated images of graphic commands are displayed in a new window. However, the results of these expressions are not saved and are no longer available after the command has been executed.

With assignments, the results of the calculations are assigned to a variable but are not directly displayed.

R takes into account the known calculation rules, such as the order of operations (parentheses first, then exponents, then multiplication/division, then addition/subtraction), so these rules must be considered when entering (R Core Team, 2022b). The following table lists a selection of the most important mathematical and arithmetic functions in R.

|  |  |
| --- | --- |
| Selection of mathematical and arithmetic functions in R | |
| abs | Absolute value |
| ceiling | Closest, bigger, integer value |
| floor | Closest, smaller, integer value |
| trunc | Closest, integer value towards zero |
| sqrt | Square root |
| exp | Exponential function |
| log | Natural logarithm (to the base e) |
| log10 | Logarithm to the base 10 |
| gamma, lgamm | Gamma function and its natural logarithm |
| cos, sin, tan | Trigonometric functions |
| cosh, sinh, tanh | Hyperbolic trigonometric functions |
| acos, asin, atan | Inverse trigonometric functions |
| acosh, asinh, atanh | Inverse hyperbolic trigonometric functions |
| max | Maximum value of a vector |
| min | Minimal value of a vector |
| length | Length of a vector |
| \* | Multiplication |
| + | Addition |
| - | Subtraction |
| / | Division |
| ^ | Exponentiation |
| %% | Modulo operator |
| %/% | Euclidean division |
| %\*% | Matrix multiplication |
| pi | Number pi |

Most functions in R are called using the function name followed by parentheses (). Any additional information that the function needs to be executed is written inside those parentheses. Those values are called parameters or arguments. Depending on the definition of the function, multiple parameters can be required or optional – this will be covered in more depth in section 2.5 (R Core Team, n.d.-b).

The square root function requires one parameter, the value to take the square root of. So, the square root of 36 can be computed as follows:

> sqrt(36)

[1] 6

As in many (but not all) other programming languages, the basic mathematical operations work slightly differently to comply with the syntax known from mathematics (R Core Team, n.d.a). So, the sum of 4 and 6 can be calculated as follows:

> 4 + 6

[1] 10

The first exercise uses some of the functions from the previous table to compute the mean of a given sequence of numbers.

### Exercise 1

Compute the mean of the number sequence: 1, 2, …, 9, 10.

Without using the function mean, which is already implemented in R, this could be done manually as follows. The arithmetic mean can be calculated as the sum of the values divided by their number (Tammadge, 1970). After the prompt sign > in the R console one can enter the following expression:

> (1+2+3+4+5+6+7+8+9+10)/10

Using R as a pocket calculator only utilizes a fraction of R's potential and merely serves as an introductory example.

### Self-Check Questions

1. How are simple arithmetic operations like addition and division represented and done in R?

* *They are functions in R, but can be entered as you would on a pocket calculator. The normal order of operations is adhered to.*
* They are not functions*.*
* They are functions and need to be called like all other functions in R, so the only option to add the two numbers 4 and 5 would be the command +(4,5).

1. R can be closed using the quit function. Which of the following is true?

* R does not differentiate between the two commands q and q().
* *R does not differentiate between the two commands* q() *and* quit()*.*

## 2.2 Assignments and Variables

Most calculations require more than just one step, which will require us to save and access intermediate results to use in later computation steps. This is achieved by assigning those intermediate results to variables. That way you can, for example, assign an expression to the sequence of numbers from Exercise 1, which is then available to be used in further calculations. The character string '<-' can be used as the assignment character to distinguish it from the equals sign '=' (R Core Team, n.d.-b).

Variable names must begin with a letter and are, like all designations in R, case-sensitive so that a distinction is made between upper- and lower-case letters (Baker & Kim, 2017). A variable represents a storage space that can be flexibly filled with values of different data types (Knuth, 2005). As an example, the sequence of the numbers from 1 to 10 used in the first exercise is assigned to a variable with the name number\_sequence using the <- character as follows:

> number\_sequence <- c(1,2,3,4,5,6,7,8,9,10)

Since this is a vector, i.e., a series of numbers, the individual numerical values separated by commas must be combined into a series using the command c (R Core Team, n.d.-b).

Note: In R, an effortless way to create a sequence of numbers is with the function seq(from,to,by). This is also the first example of a function with more than one input parameter within the parentheses. Here, from is the first value of the series, to is the last value of the series, and by is the step size. So, in this example, the same sequence as defined above can be generated with seq(1,10,1) (R Core Team, n.d.-b).

> number\_sequence <- seq(1,10,1)

To check the content of a variable (for example to see if the assignment worked as intended), enter the name of the variable as an expression:

> number\_sequence

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

This variable can now be passed to the functions implemented in R or used in further calculation steps. Passing the variable named number\_sequence to the mean function implemented in R results in the calculation of the arithmetic mean:

> mean(number\_sequence)

[1] 5.5

It should be noted again that the assignment of variable names is case-sensitive. Also, you should avoid overwriting expressions already defined in the system by assigning new values or functions to them. The following names should not be used as variable names: c, s, q, t, F, I, C, D, T, mean, diff, pi, tree, var, range, or rank (Baker & Kim, 2017).

To check if a variable name is already defined, you can type it in the console as an expression to see if any output is produced.

Exercise 2

Calculate the length of the side of a right triangle with the Pythagorean theorem

.

The lengths of the other two sides are and .

In this case, it is canonical to use the variable names a and b for the input values and c for the result.

> a <- 4; b <- 6

As seen here, it is also possible to write multiple commands or assignments on the same line, but they must be separated by a semicolon (R Core Team, 2022b). In most cases, this is not desirable as it makes the code harder to read.

> c <- sqrt(a^2+b^2)

> c

[1] 7.211103

### Self-Check Questions

1. What is a major advantage to use variables in a script?

* Variables improve the computation efficiency in R.
* *Variables allow to re-use intermediate results and therefore allow to write more complex scripts.*

1. Which of the following variable names are feasible?

* 5day\_mean
* *day\_mean5.*
* $day\_mean5

## 2.3 Vectors and Matrices

In R, vectors are among the most important data structures, and understanding how they are constructed is crucial to using R successfully and efficiently. Vectors are composed of values of the same data type (e.g., numeric numbers) and their indices, that is, the components are numbered.

In addition to numeric (number values), there are also logical or character data types. For values of data type character, the individual values must be enclosed in quotation marks so that R can recognize them as a character type. With the function c, you can then easily create a vector of type character, analogous to the numeric vector shown above (R Core Team, n.d.-b).

> x <- c("Friday", "Saturday", "Sunday")

> x

[1] "Friday" "Saturday" "Sunday"

One can access the individual values of a given vector by entering the index of the value to be read in square brackets (R Core Team, 2022b). It should be noted that in R indexing starts with 1 and not with 0 as it does in many other programming languages, like Python, C++, or Java. So, the command

> number\_sequence[3]

[1] 3

results in the value 3 as in the vector number\_sequence the value 3 is saved in the third cell. Analogously "Saturday" is saved in the second cell of the vector x:

> x[2]

[1] "Saturday"

Calculations in R are carried out in a vector-oriented manner (R Core Team, 2022b). This is one of the great strengths of R, but this concept takes some getting used to for beginners. For example, if one wants to increase all values of a vector by 2, this is done by simply adding 2 to the vector.

> new\_number\_sequence <- number\_sequence + 2

> new\_number\_sequence

[1] 3 4 5 6 7 8 9 10 11 12

Likewise, the values of a vector can be increased by a factor of 10 simply by multiplying it by 10:

> number\_sequence\*10

[1] 10 20 30 40 50 60 70 80 90 100

Note that the original vector is not altered by either of these computations.

**Object**

Every variable, whether it is a single value (scalar), a set of values (vector or matrix), or describes a function, is an object in R and is stored in memory.

> number\_sequence

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

So, a vector in R is a one-dimensional **object** of values with the same data type. By connecting two or more numeric vectors of the same length, a two-dimensional object is created, which is called a matrix. The prerequisite for creating a matrix from individual vectors is that the individual vectors are of the same length and of type numeric (R Core Team, 2022b).

A matrix can be formed in R by combining several numeric vectors of the same length by columns (cbind, where c stands for column) or by rows (rbind, here r stands for row) (R Core Team, n.d.-b). The two example vectors number\_sequence and new\_number\_sequence can be combined column by column into a matrix using cbind and assigned to a variable with the name mat:

> mat <- cbind(number\_sequence, new\_number\_sequence)

> mat

number\_sequence new\_number\_sequence

[1,] 1 3

[2,] 2 4

[3,] 3 5

[4,] 4 6

[5,] 5 7

[6,] 6 8

Since a matrix consists of rows and columns, indexing is also done using two indices in square brackets [row,column] (R Core Team, 2022b). In the matrix mat created above the value 6 is stored in the fourth row of the second column (with the column name new\_number\_sequence) and can be accessed as follows.

> mat[4,2]

new\_number\_sequence

6

There are numerous functions available in R for working with matrices, which can speed up calculations enormously. Note that higher-dimensional matrices are called arrays and can be created with the array command (R Core Team, 2022b).

### Self-Check Questions

1. Values of which types can a vector hold?

* Only numerical values.
* *Numerical, logical or character values, but all values have to be of the same type.*
* Numerical, logical or character values*.*

1. Values of which types can a matrix hold?

* *Only numerical values.*
* Numerical, logical or character values, but all values have to be of the same type.
* Numerical, logical or character values*.*

## 2.4 Logic

**Boolean Variables**

Variables of type Boolean serve to check the truth of a logical statement and can take on two truth values: true or false.

In addition to numbers and character strings (characters), R can also handle variables that contain a logical value, so-called **Boolean variables**. A logical value is also referred to as a truth value and can be true (TRUE, T for short) or false (FALSE, F for short). These variables are particularly important in programming, for example, if a command should only be executed after checking whether a condition is true (TRUE) (R Core Team, 2022b).

Logical operators are used to compare individual values or vectors or to check greater-smaller relationships (R Core Team, 2022b).

|  |  |
| --- | --- |
| Selection of Logical Operators in R | |
| == | is equal |
| != | is not equal |
| > | is greater |
| <= | is less or equal |
| & | and |
| | | or |

For example:

Determine if one value is less than another or whether they are equal:

> 17 <= 3

[1] FALSE

or not equal:

> 88 != 87

[1] TRUE

### Self-Check Questions

1. Which values can a Boolean variable have?

* *True and false.*
* True, false and undecided.

1. Which of the following operations would result in a Boolean value?

* *4 <= 5*
* x <- 5
* 4 + 5
* *7 >= 5 & 3 != 4*

## 2.5 Functions

Functions are called by typing the name of the function with parentheses at the end. As already mentioned, after entering q() without any further information within the parentheses, the program will be terminated after a query asking whether you want to save the objects stored in R's main memory (R Core Team, n.d.-b).

It is important to note that R only recognizes that a function is to be executed by adding the parentheses at the end of a function call. If the function is called without parentheses, the command sequence stored in R for executing this function is displayed in the console. For example, the q command (without parentheses) produces the following display in the console:

The command “q” in the console

Text, application, timeline

Description automatically generated  
By entering help with the function name in brackets, the corresponding help page of R internal functions is opened in a browser. This can be done in an even more concise manner by typing a ? followed by the function name. (R Core Team, n.d.-b)

For example, typing ?round (or help(round)) gives information on how to use the round function. The first part of this help page is shown in the following figure.

The help page of the round function in R

Graphical user interface, text, application, email

Description automatically generated  
The help page for the round function gives you information on all rounding functions in base R. It provides a brief description of each function, shows how to use the function, and discusses its arguments or parameters. The function is also described in more detail, mathematically as well as technically.

Under “Usage” one sees that the round function’s **type signature** is round(x, digits = 0). The digits = 0 part indicates that the default number of decimal places the function rounds to is 0. This implies that the digits parameter is optional and if not set specifically, the default value of 0 is used. So, the round function itself has one mandatory argument (the x to be rounded) and one optional argument (the number of digits or decimal places).

**Type signature**

The definition of input parameters and outputs form the type signature of a function (Sedgewick & Wayne, 2016).

> round(7.3526, digits = 2)

[1] 7.35

> round(7.3526, digits = 3)

[1] 7.353

> round(7.3526)

[1] 7

It is also easy to write your own functions for frequently used command sequences and assign them to a function name. Functions can be passed parameters when they are called and return one (or more) value(s) as a result. Existing functions can also be used in a new function. All commands to be executed by the function are written inside curly braces { } (R Core Team, 2022b).

**Standard deviation**

The standard deviation is a measure of the spread of the individual values x in a series of length n around their mean value μ. (Bland & Altman, 1996)

The next code snippet defines a function for the **standard deviation** of a series of numbers:

> stdw <- function(x){

+ resulting\_stdw <- sqrt(var(x))

+ return(resulting\_stdw)

+ }

stdw is the name under which the function is saved and can be called later, and x is the parameter or argument for the values for which the standard deviation is to be calculated. Within the function, the R functions sqrt for the square root and var for the variance are used. The result of that is first assigned to a variable named resulting\_stdw and then returned as an output using the return function.

To calculate the standard deviation for the numerical values of the vector stored under the variable name number\_sequence, the function stdw is now called with the parameter x = number\_sequence:

> stdw(number\_sequence)

[1] 3.02765

For the calculation of the length of with the Pythagorean theorem (see Exercise 2) one can use the following function with the two function parameters a and b.

> pythagoras <- function(a, b){

+ c <- sqrt(a^2+b^2)

+ return(c)

+ }

To now calculate side c for the triangle with a = 3 and b = 5, the following function call is sufficient to get the result with c = 5.831:

> pythagoras(3, 5)

[1] 5.830952

### Self-Check Questions

1. Which of the following is true?

* Other functions can not be called inside a function.
* *Functions allow to write often used command sequences only once.*
* Function names are not case-sensitive.

## 2.6 Data Types and Data Structures

In contrast to some other programming languages, R does not require the user to predefine the data type of a variable. R recognizes and decides for itself whether a variable is of the type numeric, character, or logical. In addition, R has the factor type for data that are available as nominal or ordinal scale values.

To be able to summarize data of different types in one table, there is an additional data type in R called data.frame. Similar to a matrix, several vectors of the same length can be combined in columns or rows into a data frame. In contrast to the matrix, which may only consist of numeric values, the data frame allows you to combine different data types. For example, a column of type character or factor can be appended to a series of numbers of type numeric (R Core Team, 2022a).

This is particularly important when looking for relationships between several variables depending on a factor (for example, by gender, where the factor vector consists of the values “female”, “male” or “diverse”). As with matrices, a data frame is indexed by specifying the row and column in square brackets [r,c] (R Core Team, 2022a).

As an example, a short table consisting of a column with the factor gender and a column with the associated (numerical) heights is shown here:

> height <- c(1.8,1.7,1.9,1.6,1.7,1.8)

> gender <- c("m","f","m","f","f","d")

> height\_gender <- data.frame(x = gender, y = height)

> height\_gender

x y

1 m 1.8

2 f 1.7

3 m 1.9

4 f 1.6

5 f 1.7

6 d 1.8

In the data type list, various data types can be combined into a list with several sub-lists and different lengths. Vectors, matrices, and data frames can then be stored in it. The individual objects in a list can be indexed using double square brackets [[ ]] (R Core Team, 2022a).

### Self-Check Questions

1. What would be a feasible data type to store the results of a marathon (runners and their times) in?

* A vector.
* A matrix.
* *A data frame.*

1. How is the type of a variable decided in R?

* *R automatically determines the type of a variable depending on the value assigned to it*.
* Each variable has to be defined with a type.

Summary

To use R to its full potential, some basic concepts must first be learned. We demonstrated a first use of the R console by using R as a hand calculator for the simple input of arithmetic expressions and functions.

The assignment of values and commands to freely selectable expressions (variables) allows you to work more efficiently since this means that you no longer must enter values repeatedly. Likewise, recurring command sequences can be defined as functions and variable names can be assigned.

All data used in R are stored in memory as objects with a data type assigned by R and are available for subsequent computations. Single data values in R can be of type numeric, character, logical, or factorial. The structure of the data ranges from simple scalars (values) through vectors, matrices, and data frames to lists with increasing complexity.

# Unit 3 – Accessing Data

**Study Goals**

On completion of this unit, you will be able to

* enter data in R.
* import and export data in R.
* explain how stored data are structured and managed in R.

# 1. Accessing Data

## Introduction

## The basis of every statistical analysis is data, so every software-based data analysis must start by making the data accessible from within that software. Also, the quality and reliability of the results depend heavily on the data basis. To put it simply: the more data and the better the quality of the data, the more reliable the result. Therefore, it is usually not practical to enter the data to be examined for analysis by hand. Larger amounts of data should always be saved and backed up in separate files to be called up when required. In R, data can be entered directly or loaded from external files or databases. It provides various options to process these external data files, which will be explained in this lesson.

## 3.1 Enter Data

In R, numerical or character values can be easily entered and combined into a data vector as an assignment. However, this method is very inefficient for larger data sets. A slightly easier way to enter such data is with R's built-in editor, which can be opened with the edit() command. The entered data are saved by assigning them to a variable name (R Core Team, n.d.-b).

The command data.frame() creates an empty data frame, which can then be filled with values using the edit() command (R Core Team, n.d.-b). The following example creates an empty data frame object named ex.dat and opens the editor, in which values can be entered and stored interactively:

> ex.dat <- edit(data.frame())

Opening the Built-in Editor in R

A screenshot of a computer

Description automatically generated

In the editor, the individual column names (by default var1, var2, ...) can be changed by clicking on them. The individual columns can be defined as numeric or character and filled with values accordingly. Empty cells are interpreted as missing values, which are called NA (Not Available) in R.

Entering Values in the Built-in Editor in R

A screenshot of a computer

Description automatically generated with medium confidence

After closing the editor, the entered data are available in R and can be accessed under the chosen variable name.

> ex.dat

x y

1 0 4

2 NA NA

3 3 NA

4 3 3

5 2 2

6 1 3

7 4 3

Data already stored in R can be easily opened and modified using the fix() command. This command opens the editor for the stored data and automatically saves all changes made in that editor (R Core Team, n.d.-b).

The data from the above example are displayed in the editor by calling

> fix(ex.dat)

and new data can be added, or existing data can be changed or deleted. After closing the editor, the changes are saved in the object ex.dat.

### Self-Check Questions

1. What is the difference between the functions edit() and fix()?

*The function* fix() *automatically saves the changes to the data in R, edit() does not.*

1. How will missing values be presented in R?

* *As NA (not available)*.
* As -Inf for number values and an empty string “” for character values.
* As empty cells.

## 3.2 Import and Export of External Files

This method of entering the data manually using the built-in editor is only recommended for very small data sets. Generally speaking, you will want to import larger files from external data sources into R and you will want to make calculation results available for other users and other tools. There are different methods for this depending on the data format and data source. The easiest way is to read data stored in a text format (also known as **ASCII** format). For example, Excel files exported as .csv or .txt are of such a format (Microsoft Office Support, n.d.).

**ASCII**

The abbreviation stands for American Standard Code of Information Interchange, in which each character is assigned a bit code (0, 1) consisting of 7 bits. (Mackenzie, 1980)

Before importing data perform the following check. If the data set consists of several variables (that is, multiple columns, for example, height and weight), the number of values for each variable must be the same. For example, if you want to examine the relationship between height and weight for 100 people, you should have 100 measurements for height and 100 values for weight, resulting in a rectangular matrix with 100 rows and two columns. If there are incorrect values, these should be identified in the table with a uniform identifier. If these conditions are met, these files can be read in table format with the read.table() command and saved as data frame objects (R Core Team, n.d.-b).

For simplified handling of the diverse data formats, which can vary greatly depending on the origin of the file, the command read.table allows numerous arguments to be passed, which are displayed with the help function (?read.table) (R Core Team, n.d.-b). The most important parameters are summarized here:

|  |  |
| --- | --- |
| Important Arguments of the Command read.table | |
| Argument | Description |
| file | File name in quotation marks (can include a path if the file is not saved directly in the directory from which R was started). Other options are to import data from the clipboard (copied from another program) with file="clipboard" or directly from the Internet by specifying the address of the website, file=url("http:// ..."). |
| header | Does the table to be imported have names for each column? If so, header=TRUE must be set. |
| sep | Identification of how the table columns are separated. Spaces are set as default. However, other separators can also be specified, e.g., commas (sep=",") or tabs (sep="\t"). |
| dec | Decimal separator. The default is the point; can for example be changed to comma by dec=",". |
| na.string | This is the identifier of the missing values. The default setting of "NA" can be changed accordingly, e.g. na.string="-9999". |

Variants of read.table are the read.csv command specifically for the frequently used .csv format with a point as the decimal separator or read.csv2 with a comma as the decimal separator (R Core Team, n.d.-b).

Special libraries are available in R for importing data directly from other applications (such as Excel, SAS, etc.) without having to save the files in .txt or .csv format beforehand.

### Libraries for Import and Export

Libraries are additional packages that were written for special tasks and are only loaded when required in order not to overload the main memory. Libraries can be created by any user and made available to all R users after **quality control** (Wickham, 2015). These numerous libraries are one of the reasons for the popularity of R since they cover a wide spectrum and new libraries are constantly added on the R-CRAN website and are available for download.

**Quality Control**

Before a package is released to the public on CRAN, it must pass certain tests, such as whether the library can be installed correctly on all computer systems.

The easiest way to install such libraries is with the install.packages() command. When you call this function for the first time, you are asked which CRAN server you want to access (see figure), and then a list of all available packages is displayed (R Core Team, n.d.-b).

Choosing a Mirror to Install Additional Packages

Graphical user interface, text

Description automatically generated

Alternatively, the desired package can be passed directly to the installation command as an argument (with quotation marks). For example, the XLConnect library was developed for importing Excel files. This package can now be installed by calling

> install.packages("XLConnect")

in R. All necessary auxiliary packages will also be installed automatically. It should be noted that installed packages are loaded into the main memory after the call library(<package name>)is executed – beforehand, they are not available to the user (R Core Team, n.d.-b).

After loading the XLConnect library (library(XLConnect)), functions for reading individual tables (worksheets or worksheets) or an entire workbook can be used. The associated function calls are:

* readWorksheetFromFile() with the option of specifying individual worksheets and rows as well as rows in an Excel file or
* loadWorkbook() to import an entire Excel file. (CRAN, n.d.)

Tutorials for the packages are also usually provided on the CRAN website or can be accessed after loading the library with the vignette(<package name>) command (Wickham, 2015).

**SQL**

Structured Query Language (SQL) is a database query language.

The reading of SAS and SPSS data files is made possible by external libraries and the connection to databases (e.g., **SQL** databases like MySQL or PostgreSQL) is made possible by the package RODBC. The XML library enables the reading of **XML** formats. For more information on these import options, refer to the CRAN help pages (CRAN, n.d.).

**XML**

Extensible Markup Language (XML) is a widely used markup language and data file format (W3, 2013).

There are also numerous options for exporting data sets from R. We will only mention the counterpart to the read.table command here, namely write.table. This command is used to export tables as ASCII text files with arguments similar to the respective import command (see ?write.table for more information about the function) (R Core Team, n.d.-b).

Often, one does not only want to export specified variables to files readable by other programs but also wants to save the current session to later restart working at the same point. All objects created during an R session can be exported and saved as R objects. This can be done as text or in binary format, i.e., it is only readable by the computer and therefore has very efficient storage requirements. The save() and dump() commands are available for this purpose. The objects to be exported can be specified individually, separated by commas, or summarized in a list as arguments for both command functions. Both functions deliver platform-independent files. This means that results calculated on a Linux system can be loaded and processed on a Windows computer after they have been saved. The save.image() command is recommended for saving all objects created during a working session. By default, this will save the current R workspace to a file named .Rdata in the location where R was started (R Core Team, n.d.-b).

Depending on the storage method, the functions for loading are load() for data stored in binary form with the save command, or source() for R files that were stored in text format with the dump command. The data from the last session that was saved with save.image() can be loaded with load(".Rdata") (R Core Team, n.d.-b).

In addition to the load and source commands, there is also the data command, which is only available for selected sample data sets. The data() command without specifying a name returns a list of all data sets available by default in R (R Core Team, n.d.-b).

### Self-Check Questions

1. Which command installs the package ggplot2?

* library(“ggplot2”)
* *install.package(“ggplot2”)*
* library(ggplot2)
* install.package(ggplot2)

1. A library in R is…

* a collection of R manuals.
* a list of currently loaded functions.
* *a downloadable package that provides additional functionality.*

1. Please complete the following sentence:

A data frame can be exported as a .csv file using the command *write.table*.

1. The objects of the current R session can be written to disc with the commands

* *save()*
* session.store()
* store()
* *dump()*
* *save.image()*
* write()

## 3.3 Data Management in R

An overview of all files and objects loaded in R during a session can be obtained with the ls() command, which lists all existing objects. If you want to check whether, for example, a function called my.func is loaded in memory and is therefore available for calculations, you can specify this in the ls function with ls(pattern="my.func"). Objects that are no longer needed can be deleted with the remove command rm(<name of the object to be deleted>). Calling rm(list = ls()) will delete all objects in the main memory (R Core Team, n.d.-b).

By default, the working directory is always the directory from which R is started. This means that all objects that are saved without specifying a path are also saved here. Files that you want to load are also searched for in this directory. The getwd() command shows you the path of the current (working) directory. During a session you can change the working directory with the command setwd("<(path+) directory name>") (R Core Team, n.d.-b).

For clarity, different projects and their associated data and objects should be stored in different directories. When specifying paths in R, it should be noted that these have different directory separators depending on the operating system. Since R has its origins in the Unix environment, the forward slash ("/") is set as the delimiter by default and can therefore also be used on Windows (CRAN, n.d.).

To simplify the handling of data imports from different folders (directories), numerous functions are available in R. The list.files command with arguments path="<path to directory>" and pattern="<search pattern>" will display all files in a directory that match a specific search pattern (R Core Team, n.d.-b). For example, to see all .csv files in the C:\Temp directory, enter and execute the following command:

> list.files(path="C:/Temp", pattern="\*.csv")

Examples of other commands for managing external files from R include: dir.exists, file.exists, file.remove, file.copy, file.rename, file.create, and dir.create (R Core Team, n.d.-b).

An important object type in R is the data frame. Data frames are representations of two-dimensional tables and allow different data types to be combined (such as numerical, character, factor, and logical) (R Core Team, 2022a). As already mentioned, data that are read with the read.table() command are stored in R as data frame objects. In these two-dimensional tables, the various variables are usually arranged in columns with a variable name as a header. The corresponding values of the respective variables are in each row of the data frame (R Core Team, n.d.-b).

There are numerous options available in R for further processing (modification, redesign, reorganization, etc.) of data frame objects, which will now be illustrated using an example data record. See the respective help functions for more detailed information on the functions used (R Core Team, n.d.-b).

First, a data frame object (examp.df) is created with the variables student, score, and gender:

> examp.df <- data.frame(student=c(1,2,3,4,5), score=c(247,284,275,267,276), gender=c("m","f","f","m","m"))

> examp.df

student score gender

1 1 247 m

2 2 284 f

3 3 275 f

4 4 267 m

5 5 276 m

The variable names can be queried with the names() command:

> names(examp.df)

[1] "student" "score" "gender"

The individual variables or values can be accessed by indexing [row,column] or by appending the variable name to the data frame object with the "$" character. Thus, you get the same answer with examp.df[,2] and examp.df$score:

> examp.df[,2]

[1] 247 284 275 267 276

> examp.df$score

[1] 247 284 275 267 276

A new variable is added like this:

> examp.df$age <- c(12,13,12,14,11)

This appends an age column to the existing data frame. To remove a variable (e.g., gender), the corresponding column must be set to NULL:

> examp.df$gender <- NULL

> examp.df

student score age

1 1 247 12

2 2 284 13

3 3 275 12

4 4 267 14

5 5 276 11

If you were only interested in, for example, the students who have a grade better than student 3, this can be extracted as follows:

> examp.df[examp.df$score<270,]

student score age

1 1 247 12

4 4 267 14

Sorting the record by age is done with the order() command:

> examp.df[order(examp.df$age),]

student score age

5 5 276 11

1 1 247 12

3 3 275 12

2 2 284 13

4 4 267 14

The data are sorted in ascending order by default. Data sorted in descending order are obtained with the following command.

> examp.df[order(examp.df$age, decreasing=T),]

student score age

4 4 267 14

2 2 284 13

1 1 247 12

3 3 275 12

5 5 276 11

Often, multiple data frames are loaded from different files and need to be combined. To include this in the example, a second data frame with the names of the students is generated.

> examp2.df <- data.frame(student\_no=c(1,2,3,4,5), name=c("Marc","Lisa","Mary","Lukas","Joey"))

> examp2.df

student\_no name

1 1 Marc

2 2 Lisa

3 3 Mary

4 4 Lukas

5 5 Joey

The command to combine two data frames based on matching columns is merge(). This command identifies columns with the same name and creates a new data frame with the columns of both input data frames (R Core Team, n.d.-b).

The command head() shows the first few rows of a vector, matrix, table, or data frame (R Core Team, n.d.-b). This is useful to get a first understanding of the structure of a loaded data set or to do a quick first check if an expression performed on a data set worked as intended.

> head(merge(examp.df, examp2.df))

student score age student\_no name

1 1 247 12 1 Marc

2 2 284 13 1 Marc

3 3 275 12 1 Marc

4 4 267 14 1 Marc

5 5 276 11 1 Marc

6 1 247 12 2 Lisa

In this case, it is obvious that the command did not work as intended. The identifying columns student and student\_no could not be matched because their names are not equal. There are two options to ensure that the data frames are merged using the correct set of columns. Either the columns can be specified in the function call of merge using the arguments by or by.x and by.y, or the columns in the original data frames can be renamed accordingly – column names need to be changed so that columns that are to be used for matching rows have the same name and others do not. Renaming columns can be done with the names() command.

> names(examp2.df)[names(examp2.df) == "student\_no"] <- "student"

> examp2.df

student name

1 1 Marc

2 2 Lisa

3 3 Mary

4 4 Lukas

5 5 Joey

The same merge command as above now yields the intended result.

> head(merge(examp.df, examp2.df))

student score age name

1 1 247 12 Marc

2 2 284 13 Lisa

3 3 275 12 Mary

4 4 267 14 Lukas

5 5 276 11 Joey

### Self-Check Questions

1. Which command removes the variable score.df from the session, but keeps all other variables?

* rem(score.df)
* rm()
* *rm(score.df)*
* remove()

1. Please complete the following sentence:

By default, R commands like read.table look for the specified file in the *current working directory*.

1. How can a column “stars” be deleted from a data frame “movies.df”?

* *movies.df$stars <- NULL*
* movies.df$stars <- NA
* delete(movies.df$stars)
* del(movies.df$stars)

Summary

Numerous analysis methods are available in R to extract insights from data. However, every analysis requires the data to be read into R.

The simplest, but most inefficient method is to enter data in R by hand in an editor integrated into R for this purpose. However, this is very time-consuming and subject to error (such as reading and writing errors) and is therefore only recommended for very small data sets. In most cases, one will analyze data downloaded via the internet or provided by other sources. If these data are stored in a human-readable text format, it is possible to read them into R with the read.table command or related functions. For most binary data formats that can only be read by machines, special libraries have been developed by R users. These allow importing and exporting data from a wide variety of sources (e.g., Excel spreadsheets).

Numerous functions are available for managing data in the main memory (i.e., data already loaded in R) and from external storage, such as displaying and selecting all files that meet certain search criteria. Likewise, objects in R that are stored as a data frame can be restructured and combined using simple commands (e.g., it can be reordered using the order() command).

# Unit 4 – Descriptive Statistics

**Study Goals**

On completion of this unit, you will be able to

* install and use an IDE for R.
* explain how single-series (univariate) data sets are statistically described.
* analyze relationships between two-series (bivariate) data.

## Introduction

Any statistical analysis should begin with an examination of the underlying data. It is helpful to describe the information contained in the data in a comprehensible and concise manner as a first step. In most cases, these data consist of a sequence of measured values of one (univariate), two (bivariate), or several (multivariate) variables. This lesson takes a closer look at univariate and bivariate variables using descriptive statistics. This can be done either with statistical key indicators (parameters or measures) or by displaying the data graphically. For this purpose, sample data sets from economics are examined in more detail.

The commands can be entered and executed from any R console. However, for more in-depth statistical analysis, it is recommended that you employ a graphical user interface for ease of use. Among other things, this makes it possible to add explanatory text to individual commands and to save command sequences. Two popular graphical user interfaces are introduced and used in the remainder of this work.

## 4.1 Univariate Descriptive Statistics

It is difficult to describe data using a table since even small amounts of data can be difficult to grasp immediately and interpret. In practice, you will usually have to work with large amounts of data, for which a representation of the individual measured variables as a sequence of numerical values is unsuitable and does not aid understanding.

Before examining the EU stock market data as an example for descriptive statistics, two integrated development environments (IDEs) are introduced: RStudio and Jupyter Notebook.

### Digression: IDEs

For more complex development, it is beneficial to use an integrated development environment for R. This is additional software that allows for writing and executing code within one program and that brings additional features to make development easier. Important additional features of RStudio and Jupyter Notebook include (Jupyter Team, n.d. and RStudio, 2022a):

* Syntax highlighting: Here, language keywords like commands are highlighted, and syntax errors are flagged in the code.
* Code completion: The IDE shows known variables (self-defined variable names as well as function names from base R or loaded packages) as completion suggestions for words.
* Menu-based options for common functionalities like saving the R session, saving the script, or changing the working environment.

#### RStudio Desktop

RStudio Desktop is an IDE for R with syntax highlighting, code completion, and smart indentation. Code can be executed directly from the IDE. The R help pages are also integrated, which allows the user to operate R within just one window. RStudio Desktop is available as an open-source version under the AGPL v3 license (RStudio, 2022a).

Installers for several operating systems (Windows, MacOS, and several Linux versions) are available from the RStudio website (RStudio, 2022a). A wizard guides the user through the installation process. After the installation, the RStudio Desktop IDE can be opened. In the following figure, some code lines from the previous section were executed in the RStudio Desktop IDE.

RStudio after Executing Sample Code

Graphical user interface, text, application

Description automatically generated

The main part of the screen is divided into four sections. The lower left section shows the R console as already used in the previous units. Here, code can be directly entered and executed. In the upper left section, code can be written into scripts, which are separate text files containing code. In this case, some sample code from the previous section was entered as a script. Here, syntax highlighting, as well as auto-complete functions, are available. A single line or a selection can be executed by pressing the “Run” button on the upper right of that section or with the shortcut <CTRL>+<Enter>. The lines are then copied into the R console in the lower left section and executed. RStudio allows you to save, edit, and load scripts in the same way standard text editors do.

The results of the executed commands are displayed in the R console as before, but they are also displayed in the right two sections of the RStudio window. If a command produces graphical output, that output is displayed in the lower right section in the tab “Plots”. Using the “Export” button, that output can be saved to the disk as an image. All assigned variables can be inspected on the “Environment” tab of the upper right section.

#### Jupyter Notebook

With the help of Jupyter Notebook, documents can be created online that summarize explanatory text and executable program lines as well as calculation results in a notebook (Rule et al., 2018).

Jupyter Notebook is a web-based client-server application and can therefore be opened with any browser. This allows great flexibility when collaborating with multiple users. The notebook is based on kernels, which process the entered program lines in the appropriate programs (such as R or Python). As Jupyter Notebook also supports other programming languages, it is a particularly valuable tool for data science, in which Python is very often used in addition to R. A dashboard allows for managing the creation of the documents and the kernels used. (Jupyter Team, n.d.).

As mentioned, Jupyter Notebook is a web-based client-server application. The easiest way to run a Jupyter Notebook on your local machine is to use Anaconda, which is also the recommendation of the Jupyter Team (n.d.).

Anaconda is an open-source distribution platform available for all major operating systems and architectures. There is a free license available for students, academics, and hobbyists, but commercial use requires a paid license (Anaconda, 2022a).

After downloading and executing an installer for your operating system, a wizard guides you through the installation process. On opening the Anaconda Navigator, the following screen is opened.

Anaconda Navigator Home Screen

Graphical user interface, application

Description automatically generated

In the “Applications on” dropdown, select R. Jupyter Notebook can then be launched in a browser using the appropriate tile in the navigator.

Jupyter Notebook Dashboard

Table

Description automatically generated with medium confidence

After starting Jupyter Notebook, the Jupyter Notebook Dashboard appears in a browser. In this, you can either open an existing notebook (recognizable by the file ending .ipynb) or create a new document with the *NEW* button. Here, you must select the R kernel to use with R (Jupyter Team, 2021). The following input window then appears after executing the same sample code as above.

Jupyter Notebook after Executing Sample Code

Table, calendar

Description automatically generated

Now let's start with the descriptive analysis of the EuStockMarkets sample data set, which is available in the R base package. First, these data must be loaded into memory using the data command. After that, the time series stored under this new variable name can be displayed in a graphic with the plot command. plot is a universal command that results in different output types depending on the properties of the data. In addition, many function parameters allow the generated image to be modified according to personal preferences; see ?plot for a detailed description (R Core Team, n.d.-b).

Normally, the user is not required to define the properties of the data for the calculations and representations, since R recognizes the data types automatically. In the case of the EU stock market data used here, these are defined as time series (ts), which means that R uses functions (so-called generic functions) available specifically for this attribute. This definition is made by assigning a so-called class attribute to the data record using the class command (R Core Team, n.d.-b).

Every object created in R also has a class that automatically matches the data type but can be overridden by the user. To illustrate this, consider a simple example – as always, all functions are referenced from the R documentation (R Core Team, n.d.-b) and from within R using the help() function. First, two variables x1 and x2 are assigned numeric values.

> x1 <- 1:7

> x2 <- 3:9

These two variables are now combined column by column into a new variable x using the cbind() command.

x <- cbind(x1,x2)

Since the two variables x1 and x2 each contain only integer values, the assigned class attribute is "integer".

> class(x1)

[1] "integer"

By combining x1 and x2 column by column, the variable x forms a matrix with two columns [x1,x2] and seven rows. Therefore, the class attribute of x is not "integer" but "matrix".

> class(x)

[1] "matrix" "array"

If you now call the plot function, the variable x is displayed according to the defaults defined for the "matrix" class.

plot(x)

Plot of a Matrix in R

Chart, line chart, scatter chart

Description automatically generated

If a different class attribute is assigned to the variable x, a different plot may result. For example, the class "matrix" of x can be converted into a time series class ("ts") by overriding class(x) <- "ts" or by transforming x <- as.ts(x). The plot function then produces the following figure:

Plot of Two Time Series in R

Chart, line chart

Description automatically generated

When loading a data set, R checks which class attribute is associated with the object.

Because R identifies the loaded EuStockMarkets data as time series, the plot command interprets the x-values as time and plots the x-axis as a time axis.

Expressions for the three steps load, rename, and plot are now written in R and executed. Comments and explanatory text are also added to the code but must be indicated with the # symbol so that the R interpreter knows that those parts do not need to be executed.

# load EU Stock Markets data set

data("EuStockMarkets")

# assign EU Stock Markets data set to variable EuStock

EuStock <- EuStockMarkets

# plot EU Stock Markets data set

plot(EuStock)

If this command sequence with associated comments is now executed, the following figure is created:

Plot of the EU Stock Markets Data Set

Chart, line chart

Description automatically generated

There are four time series in this data set: the DAY, the SMI, the CAC, and the FTSE. However, in the univariate description, only one feature is examined. Therefore, as a next step, the DAX will be analyzed in more detail as an example.

As you can see in the image of the data set, the DAX is the first time series and can be extracted and assigned to a variable dax by the command

dax <- EuStock[,1]

This new variable is now plotted using the plot command.

Plot of the DAX Data Set

Chart, line chart, histogram

Description automatically generated

### Summary Statistics

**Skewness**

The skewness indicates how much the data deviate from a symmetrical distribution. (Everitt and Skrondal, 2010)

A statistical parameter is a characteristic value used for the statistical description of data (Everitt and Skrondal, 2010). Summary statistics are calculated from the data (samples) to convey as much information as possible as concisely as possible. For a single, measured variable, they often cover

* measures of central tendency,
* measures of variability, and
* measures of shape, for example, for the **skewness**.

A measure of central tendency or location is a central value in the data set; a well-known measure of location is the mean. A measure of variability, dispersion, or spread is a measure that indicates how much the individual values in a sample differ from each other (Gravetter et al., 2020).

Before calculating measures of central tendency, dispersion, or shape, one must determine the scale of measurement of the data, that is, determine their information content. The lowest scale is the nominal scale; data of that scale can only be differentiated according to equality or inequality. An example of this is gender. If the data allow a ranking, it’s on an ordinal scale (e.g., school grades). Values on an interval scale have an even higher scale level. Here the distances between the values represent information (e.g., temperature). It is important to note that lower-level measures of location and spread can be applied to higher-level data (e.g., the mode for ordinal and interval data), but not the other way around (e.g., there is no arithmetic mean for nominal data) (Gravetter et al., 2020).

Measures of location or central tendency aim to determine the single value that is most representative of a distribution or sequence of data values. The following table contains an overview of common measures of central tendency (Gravetter et al., 2020).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Measures of Central Tendency | | | | |
| Measure of Central Tendency | Description | Scale | | |
| Nominal | Ordinal | Interval |
| Mode | Most common value | X | X | X |
| Median | Value below and above which lie the same number of values |  | X | X |
| (Arithmetic) Mean | Average value, computed as the sum of values divided by number of values |  |  | X |

The formulas for calculating the arithmetic mean of the values is (Gravetter et al., 2020)

The median is the value that divides the sample into two equal halves. This means that 50% of all values are equal to or smaller than the median and 50% are equal to or larger than the median. This concept can be generalized to other percentages . Often one wants to know the value for which of the values are equal to or smaller relative to the value for which of the values are equal to or larger. This is referred to as the -quantile. For example, the 0.1 quantile is the value for which 10% of the values are equal to or smaller or 90% are equal to or larger. The median, therefore, corresponds to the 0.5 quantile. The 25%, 50%, and 75% quantiles, which are referred to as quartiles, are often used to describe the location, as they are less influenced by individual, unusual values (so-called **outliers**) than other measures of location like the arithmetic mean. The distance between the top (75%) and bottom (25%) quartiles is also a measure of dispersion and is called the interquartile range (IQR), which is where the middle 50% of the data is located (Dekking et al., 2005).

**Outlier**

These are the values in a series of numbers that are outside the expected range. In most cases, they are defined as the values that are smaller or larger than a limit value (e.g., 1.5 times the interquartile range). (Everitt and Skrondal, 2010)

With the summary command, these important measures of location can be easily calculated (R Core Team, n.d.-b). summary(dax) provides the following output.

> summary(dax)

Min. 1st Qu. Median Mean 3rd Qu. Max.

1402 1744 2141 2531 2722 6186

The so-called five-number summary of a set of observations consists of the five measures minimum, first / lower quartile, median, third /upper quantile, and maximum and forms the basis of the boxplot (Tukey, 1977). In R the boxplot can be generated using the boxplot() command (R Core Team, n.d.-b).

Labeled Boxplot of the DAX Data Set

Chart, box and whisker chart

Description automatically generated

50% of the values are within the box shown, the box's lower and upper limits are the lower quartile (25%) Q1 and the upper quartile (75%) Q3. The median is shown as a line inside the box. The so-called whiskers bordering the box at the top and bottom represent the 1.5-fold expanded IQR. Values outside of this interquartile range are outliers and shown as circles (Dekking et al., 2005).

With the interquartile range, one measure of variability was already introduced. A second measure of dispersion is easily derived from the five-number summary: the range is the distance between the minimum and the maximum value of the data series (Everitt and Skrondal, 2010). The following table lists important measures of variability for interval data (Gravetter et al., 2020).

**Deviation**

The distance between a value and the data’s mean is called deviation (Gravetter et al., 2020).

|  |  |
| --- | --- |
| Measures of Variability | |
| Measure of Variability | Description |
|
| Range | Distance between minimum and maximum. |
| Interquartile range | Distance between upper and lower quartile. |
| Variance | Mean of the squared **deviation**s of the individual values. |
| Standard deviation | Square root of the variance. |

The variance for the values is calculated as

The standard deviation is then the square root of the variance, so (Everitt and Skrondal, 2010).

In R, the standard deviation is calculated by taking the square root (sqrt) of the variance (var) (R Core Team, n.d.-b).

> sqrt(var(dax))

[1] 1084.793

### Frequency Function

A good way of describing data is a graphical representation of frequencies, which shows how often individual values occur. To do so, the data are sorted and, for each value, the entries with that value are counted. Relative frequencies are obtained by dividing the individual frequencies by the total number of values (Gravetter et al., 2020).

A distinction must be made as to whether the data are continuous or discrete. If continuous, the data can assume any value within a certain interval depending on the measurement accuracy (e.g., temperature, time intervals), while discrete data can only take on values out of a finite number of possible values (e.g., numbers on dice) (Everitt and Skrondal, 2010).

For a representation in a frequency diagram (histogram), continuous data must therefore be summarized in groups, also called class intervals (Gravetter et al., 2020). R offers several methods of classing, but in many cases, the default settings for plotting the histogram using the hist() command are sufficient (R Core Team, n.d.-b).

> hist(dax)

Histogram of the DAX Data Set

Chart, histogram

Description automatically generated

The histogram shows the values (class intervals) as automatically classified by R. For ordinal data, these class intervals are displayed in ascending order. Each bar corresponds to the frequency of the values in the corresponding class. So, in the example above, the stock market sample data contained about 600 entries with a DAX between 1500 and 2000.

In this figure, it can be seen that the values of the DAX are not evenly distributed but skewed to the right with a concentration in the range between 1500 and 2500. This can also be seen from the result of the summary statistics with a median (2144) that is significantly smaller than the mean (2531), and a minimum value (1402) that is much closer to the median than the maximum value (6186).

### Self-Check Questions

1. What are advantages of IDEs?

* Available programming support hotlines
* *Syntax highlighting and code auto-completion*
* Additional packages

1. Which measures of central tendency are applicable for data of the data type factor?

* *Mode*
* Median
* Mean

1. What property of a data set is easily seen in a histogram?

* Data accuracy
* Exact number of observations
* *Skewness*

## 4.2 Bivariate Descriptive Statistics

In bivariate statistics relationships between two variables are described. So, if you have two series and with a sample size of values each, the basis for the bivariate analysis is formed by pairs of values (Berry, 2014).

### Categorical Data

For categorical data, i.e., data with nominal or ordinal scalable values, the simplest way to describe a relationship is to use a crosstab, also known as a contingency table. The categories of the first variable form the rows and the categories of the second variable the columns of the table. The table’s entries are the number of occurrences for the respective combination of values for both variables (Everitt and Skrondal, 2010).

The following example creates a contingency table in R for two variables “score” and “team”. In this fictitious example, 100 students were randomly divided into two teams A and B. They each participated in a sports test, and each received a score between 1 and 5 on this test.

This example is also intended to show how random numbers can be generated in R. The variable “score” contains 100 values, which are generated with the command runif() as uniformly distributed random numbers with values between 1 and 5. By applying the round() function, these generated random numbers are rounded to integer values. The “team” variable also contains 100 values, which are drawn at random from the two options team A and team B with replacement (replace=T) using the sample() function. The resulting vectors of values are of the same length and can therefore be combined into a data frame using the data.frame command (R Core Team, n.d.-b).

> score <- round(runif(100,1,5),0)

> team <- sample(c("A","B"),100,replace=T)

> res.df <- data.frame(score, team)

Again, head() is used to inspect the first few rows of the new data frame.

> head(res.df)

score team

1 2 B

2 3 B

3 3 B

4 3 A

5 1 B

6 5 A

The contingency table can be generated with the table() command. The column and row totals can be viewed using the addmargins() command (R Core Team, n.d.-b).

> res.table <- table(res.df)

> addmargins(res.table)

team

score A B Sum

1 5 8 13

2 16 11 27

3 11 11 22

4 12 17 29

5 5 4 9

Sum 49 51 100

As can be seen in this table, the most common score for team A is 2 with 16 people, and for team B it is 4 with 17 people.

The chi-square statistic () is a measure of how much actual values differ from expected values for two series of categorical data (Berry, 2014). Since this is a frequently used measured variable for comparison with the discrete uniform distribution, the calculation is described in more detail here.

The formula for the chi-square coefficient for comparison with the discrete uniform distribution is (Everitt and Skrondal, 2010):

Here, is the number of possible results for the first variable and is the number of possible results for the second variable. So, in the example above (the “score” variable”) and (the “team” variable). is the number of occurrences of the combination of the th value of the first variable and the th value of the second variable. In the example above, the combination of score 2 (2nd value of the first variable) in team A (1st value of the second variable) is expressed as . and are, respectively, the number of occurrences of the th value of the first variable and the th value of the second variable overall. They correspond to the row and column counts in the contingency table. Thus, in the example above there are 27 occurrences of the score 2, therefore . Entering the example data into the formula and then into R (as you would on a pocket calculator) generates the following result:

> (5-49\*13/100)^2/(49\*13/100)+(8-51\*13/100)^2/(51\*13/100)+(16-49\*27/100)^2/(49\*27/100)+(11-51\*27/100)^2/(51\*27/100)+(11-49\*22/100)^2/(49\*22/100)+(11-51\*22/100)^2/(51\*22/100)+(12-49\*29/100)^2/(49\*29/100)+(17-51\*29/100)^2/(51\*29/100)+(5-49\*9/100)^2/(49\*9/100)+(4-51\*9/100)^2/(51\*9/100)

[1] 2.552435

It is easier to get the value with the chisq.test() function, which also provides additional test statistics and warnings.

> chisq.test(res.table)

Pearson's Chi-squared test

data: res.table

X-squared = 2.5524, df = 4, p-value = 0.6353

Warning message in chisq.test(res.table) : Chi-squared approximation may be incorrect

This value still has little significance, so it is converted to the contingency coefficient:

The Pearson contingency coefficient ranges from 0 to 1. Values greater than 0 show a correlation (Hartmann et al., 2018).

For the example data, the contingency coefficient is calculated as follows.

> sqrt(2.5524/(2.5524 + 100))

[1] 0.1577617

For ordinal pairs of values, Spearman's rank correlation coefficient ρ and Kendall's τ are often used to evaluate a relationship. For the calculation of these two coefficients, we consider the ranks (orders) of the data values rather than the frequency (Sprent and Smeeton, 2016).

In R, these measures are calculated with the cor() function. The desired method can be passed as an argument to this function, with the choices being: Pearson, Kendall, or Spearman (e.g.: cor(x,y,method="kendall")) (R Core Team, n.d.-b).

### Interval Data

As a reminder, the means and variances of two interval series and are:

The covariance is defined as

and is the expected value of the deviation of and from their means. From this, the coefficient of determination can be calculated as

which is a measure of the proportion of the variation in one variable that is explained by the other. The square root of is now called the correlation coefficient :

The correlation coefficient can assume values between . The larger the correlation coefficient , the stronger the positive relationship between the series. For there is no relationship and for there is a negative relationship (i.e., the smaller the value of the series is, the larger the value of the row and vice versa) (Everitt and Skrondal, 2010).

This can be illustrated well with a scatter plot. In this diagram, the series with the two pairs of values with are represented as points in an x-y coordinate system (Everitt and Skrondal, 2010). The closer these coordinate values are to a straight line, the stronger the linear connection between the two characteristics. Conversely, if the graphical representation shows an irregular point cloud with a very large scatter, there is little or no connection. To illustrate this, four scatterplots with different correlation coefficients (r = 0.9, 0.7, 0, –0.7) are shown in the figure below.

Scatter Plots with Different Correlation Coefficients

Chart, scatter chart

Description automatically generated

### Digression: Generating Scatter Plots with Different Correlation Coefficients

As a further example of the usage of R and to deepen our understanding of the covariance and the correlation coefficient, we will now explain the generation of the above figure with four scatter plots in detail (see R Core Team, n.d.-b for detailed information on the used functions).

The first step is to generate the data frames behind the plots. To produce a data series with a given correlation coefficient, the package MASS is installed and loaded.

> install.packages("MASS")

> library(MASS)

This package contains the function mvrnorm with the type signature

mvrnorm(n = 1, mu, Sigma, [...])

This function produces n samples from the multivariate normal distributions defined by mu and Sigma. There are additional optional parameters and, as always, these are described on the function’s help page accessible with ?mvrnorm. Here, mu is the vector of the means of the data series to be generated. For n > 1 the function returns a matrix with as many columns as mu has and one sample per row. Sigma is a symmetric matrix that contains the covariances of the data series. It must have mu rows and columns. If you want to generate two data series to plot for the first scatter plot, the function expects mu / and Sigma / as

For data series with variances and , the correlation coefficient equals the covariance :

Therefore, Sigma can be set to

sigma <- rbind(c(1,cor\_target), c(cor\_target,1))

to generate two data series with correlation coefficient cor\_target. To build the given figure, four data frames must be generated, each with two data series. To make the code more concise and readable, this is encapsulated in a function gen\_cor\_dataframe.

> gen\_cor\_dataframe <- function(cor\_target){

+ Sigma <- rbind(c(1,cor\_target), c(cor\_target,1))

+ mu <- c(5, 10)

+ matrix <- mvrnorm(n=2000, mu=mu, Sigma=Sigma)

+ return(as.data.frame(matrix))

+ }

For a given correlation coefficient cor\_target, this function generates a data frame with two data series that have a correlation coefficient of cor\_target and 2000 entries each. The four data frames are then generated with the following code.

> df1 <- gen\_cor\_dataframe(0.9)

> df2 <- gen\_cor\_dataframe(0)

> df3 <- gen\_cor\_dataframe(0.7)

> df4 <- gen\_cor\_dataframe(-0.7)

To check the size of a loaded (or generated) data frame, the command dim() can be used.

> dim(df1)

[1] 2000 2

As expected, the data frame df1 has 2000 rows and two columns. With the head() command the first few rows are displayed.

> head(df1)

V1 V2

1 6.639916 11.091093

2 5.169456 9.778918

3 6.184451 11.101737

4 6.105087 11.272781

5 5.977784 10.873956

6 4.002345 8.884597

The correlation coefficients should be as defined in the function call of gen\_cor\_dataframe(). As a check, the cor() function is applied to df1.

> cor(df1)

V1 V2

V1 1.0000000 0.9015593

V2 0.9015593 1.0000000

As the data series are drawn from a random distribution, the correlation coefficient will differ slightly from cor\_target. Since the object df1 is a data frame with two variables, cor returns a 2 × 2 matrix with a correlation coefficient computed for all possible combinations (V1 – V1, V1 – V2, V2 – V1, V2 – V2). To get just one correlation coefficient for two variables x and y, you need to pass x and y as vectors to the cor function:

> cor(df1[,1],df1[,2])

[1] 0.9015593

Plotting this first data frame df1 produces the following figure.

plot(df1)

Scatter Plot of the Generated Data Frame df1

Chart, scatter chart

Description automatically generated

Now, additional arguments are passed to the plot function. The labels V1 and V2 have no additional information for this generated data and can be removed with xlab="", ylab="". Also, the correlation coefficient is added as the plot title. To achieve the format as in the figure with the four scatter plots above, the correlation coefficient needs to be calculated, rounded to one digit, and prefixed with “r: “.

> paste("r:",round(cor(df1[,1],df1[,2]),1))

[1] "r: 0.9"

Again, this needs to be executed four times, so this is encapsulated into a function text\_cor.

> text\_cor <- function(df,digits=1){

+ return(paste("r:",round(cor(df[,1],df[,2]),digits)))

+ }

This results in the plot() command

plot(df1, xlab="", ylab="", main= text\_cor(df1)).

To arrange multiple plots in one figure, the par() command is used. With par(mfrow=c(2,2)), the next four plots that are generated afterward are arranged in a two-by-two grid. The following code produces the desired figure with four scatter plots.

> par(mfrow=c(2,2))

> plot(df1, xlab="", ylab="", main=text\_cor(df1))

> plot(df2, xlab="", ylab="", main=text\_cor(df2))

> plot(df3, xlab="", ylab="", main=text\_cor(df3))

> plot(df4, xlab="", ylab="", main=text\_cor(df4))

### Case Study: DAX and SMI

Now the connection between the DAX and the SMI is examined in more detail below. To do this, these two variables must first be extracted from the EuStock data set

> daxsmi <- EuStock[,c(1,2)]

You can get a first overview with the summary command:

> summary(daxsmi)

DAX SMI

Min. :1402 Min. :1587

1st Qu.:1744 1st Qu.:2166

Median :2141 Median :2796

Mean :2531 Mean :3376

3rd Qu.:2722 3rd Qu.:3812

Max. :6186 Max. :8412

The comparison of the univariate descriptions of the DAX and the SMI already suggests that there is a connection between these two characteristics. The values for the SMI are larger, but the range (the difference between minimum and maximum) is similar, and both are skewed to the right (the median is smaller than the mean in each case). This is illustrated by calculating the correlation coefficient with the R function cor (R Core Team, n.d.-b).

> cor(daxsmi)

DAX SMI

DAX 1.0000000 0.9911539

SMI 0.9911539 1.0000000

Again, since the object daxsmi is a matrix with two variables, cor returns a 2 × 2 matrix with a correlation coefficient computed for all possible combinations (DAX - DAX, DAX - SMI, SMI - DAX, SMI - SMI). To get just one correlation coefficient for DAX – SMI, the respective vectors are passed to the cor function:

> cor(x=daxsmi[,1],y=daxsmi[,2])

[1] 0.9911539

The plot command is again used for a graphical output of the bivariate analysis. However, as mentioned above, the EU stock market data has the class attribute "ts" and will be interpreted as a time series by the plot command. The result of the plot command. Therefore. does not deliver the desired scatter plot, but the depiction of the two time series.

plot(daxsmi)

Graphic Representation of the Two Time Series DAX and SMI

Chart, line chart

Description automatically generated

To generate a scatter plot, it is necessary to remove the class attribute. The easiest way to do this is with the unclass function (R Core Team, n.d.-b). The following command creates the desired representation with the value pairs consisting of DAX and SMI.

plot(unclass(daxsmi))

Scatter Plot for DAX and SMI

Chart, scatter chart

Description automatically generated

As you can see, the correlation is extremely high. The representation of the DAX versus the SMI forms a clear straight line. This is also reflected in the very high correlation coefficient of 0.99.

### Case Study: Longley

As a further example, the data from Longley (1967) will now be examined. This data set is also included in the standard package of R and can be loaded with the data command. It includes macroeconomic annual values like the gross national product (GNP) and the number of employed and unemployed citizens (R Core Team, n.d.-b).

With the help of this data set, some previously mentioned analysis tools in R will be explored again. One can get an initial overview of the data after loading with the str command (R Core Team, n.d.-b).

> data(longley)

> str(longley)

'data.frame': 16 obs. of 7 variables:

$ GNP.deflator: num 83 88.5 88.2 89.5 96.2 ...

$ GNP : num 234 259 258 285 329 ...

$ Unemployed : num 236 232 368 335 210 ...

$ Armed.Forces: num 159 146 162 165 310 ...

$ Population : num 108 109 110 111 112 ...

$ Year : int 1947 1948 1949 1950 1951 1952 1953 1954 1955 1956 ...

$ Employed : num 60.3 61.1 60.2 61.2 63.2 ...

As you can see, the data set loaded and stored in the main memory under the name longley has the class attribute data.frame. This class attribute can also be easily retrieved with the class command (R Core Team, n.d.-b).

> class(longley)

[1] "data.frame"

For the EuStock data set examined earlier, the str and class commands return the following result:

> str(EuStock)

Time-Series [1:1860, 1:4] from 1991 to 1999: 1629 1614 1607 1621 1618 ...

- attr(\*, "dimnames")=List of 2

..$ : NULL

..$ : chr [1:4] "DAX" "SMI" "CAC" "FTSE"

> class(EuStock)

[1] "mts" "ts" "matrix"

As you can see, EuStocks is a time series and has three class attributes, namely mts (multiple time series), ts (time series), and matrix.

With the Longley data set, a bivariate analysis between the characteristics GNP and the number of employed versus the number of unemployed citizens is now carried out. First, it is helpful to graph the GNP series. Since the Longley data record is a data frame object, you can select the GNP series either by selecting the relevant column (in this case the second), i.e., with longley[,2], or by directly specifying the name of the desired column with a "$" appended to the object name (longley$GNP) (R Core Team, 2022a). It should be noted that the column names do not need to be written out in full, we only need just enough characters to allow for clear identification. So, to select the employment number (Employed) it is sufficient to write longley$E or longley$U for the unemployment number (Unemployed).

The plot command for displaying the GNP is now:

> plot(longley$GNP,main="Longley",ylab="GNP",xlab="Time")

The default labels suggested by R have been replaced by specifying arguments for the title (main="Longley"), y-axis (ylab="GNP"), and x-axis (xlab="Time") (R Core Team, n.d.-b). The resulting image now looks like this:

Representation of the Gross National Product According to Longley

Chart, scatter chart

Description automatically generated

The correlation coefficients for the strength of the relationship between GNP and employment numbers are now calculated as follows:

> gnpemp <- cor(x=longley$GNP,y=longley$E)

> gnpemp

[1] 0.9835516

The corresponding scatterplot is generated with the following plot command. In this command, the correlation coefficient previously calculated and stored under gnpemp is inserted into the figure using the text function. The arguments to the text function specify the text to be displayed as well as the position and the color of the added text. The x and y coordinates indicate where the text should be positioned. The labels argument is used to pass the text itself. The paste function was also used here since a character expression ("r:") is combined with a variable (the correlation coefficient stored under gnpemp). In addition, the correlation coefficient is rounded to two decimal places using the round function. The paste function and other specifications, such as col for the color and cex for the character size, are not explicitly available only for the text function but can also be used in other functions, for example in the plot function (R Core Team, n.d.-b).

plot(longley$GNP,longley$Empl,main="Scatterplot",

xlab="GNP", ylab="Employment")

text(250,70,labels=paste("r:",round(gnpemp,2)),

col=4,cex=1.2)

Scatter Plot of the Gross National Product and Employment According to Longley

Chart, scatter chart

Description automatically generated

In the same way, the correlation coefficient and the scatter plot for the relationship between unemployment figures and GNP are now generated:

> gnpunemp <- cor(x=longley$GNP,y=longley$U)

> gnpunemp

[1] 0.6042609

> plot(longley$GNP,longley$Unempl,main="Scatterplot",

xlab="GNP", ylab="Unemployment")

> text(250,450,labels=paste("r:",round(gnpunemp,2)),

col=4,cex=1.2)

Scatter Plot of the Gross National Product and Unemployment According to Longley

Chart, line chart, scatter chart

Description automatically generated

As one can see by comparing the results for employment and unemployment figures, there is a strong correlation between the former and GNP (r = 0.98), while there is a much weaker correlation between unemployment figures and GNP (r = 0,6).

The graphics generated with Jupyter Notebook or RStudio can be copied or exported directly as an image. However, to produce higher quality graphics, e.g., for scientific work and publications, there are several options in R to choose from. PostScript graphics can be generated by calling the postscript function before the plot command, which requires the name of the image to be generated as an argument. For the output of subsequent commands to be displayed on the screen again, the graphic export must be terminated with the dev.off() command (R Core Team, n.d.-b).

### Self-Check Questions

1. Which minimum scale level must the data have to be displayed in a contingency table?

* *Nominal*
* Ordinal
* Metric

1. You have read a data frame big.df from a .csv file. It had too many rows to be displayed in Excel or a notepad. A good first step before beginning to analyze the data would be to display the first few rows with the command *head()*.

Summary

Descriptive statistics aims to summarize the content of a data set (univariate) or the relationship between two (bivariate) or several (multivariate) data sets. To do this, parameters or measures are calculated that summarize essential aspects of the samples with just a few numbers. The most important parameters of univariate statistics are those that describe the central tendency (e.g., mean and median) and the spread of the data around the central tendency (e.g., variance). The correlation coefficient can be used as a descriptive parameter to characterize the linear relationship between two samples.

In addition to the numerical description with parameters, it is often helpful to prepare and present the essential information in the form of diagrams. Valuable insights into the structure and distribution of a data set can be obtained by displaying the data in histograms and boxplots. For bivariate analysis, the scatterplot is a valuable tool. This diagram shows the degree of correlation in a simple, visual way by comparing two data series in one figure.

# Unit 5 – Inferential Statistics

**Study Goals**

On completion of this unit, you will be able to

* explain which distributions are commonly used to describe the properties of a population.
* infer certain properties of the population from samples.
* test statistical hypotheses using the t-test.

# 5. Inferential Statistics

## Introduction

In descriptive statistics, the data at hand are analyzed. Often, these data have been either collected in experiments, acquired through surveys, or perhaps, they represent the past performance data of a live system. In nearly all relevant cases it is not possible to take those measurements for all possible combinations of independent variables – the available data usually only consist of a limited number of possible value combinations and is, therefore, referred to as a sample. Usually, one wants to make statements that are generally valid and do not only apply to the collected sample. In this unit, we will use inferential statistics to ascertain the properties of the underlying population. In other words, we will attempt to derive generally valid statements from the information contained in the limited data available.

To summarize, it is usually impossible to capture a complete population (e.g., to survey all voters in an election). Therefore, one is forced to limit the data collection to a small number of randomly selected values (e.g., people), which is referred to as a random sample. From that random sample, one aims to describe the population (e.g., the expected voting behavior of all eligible voters) as well as possible. To do so, it is necessary to derive probability distributions from the samples that best describe the unobserved population. These distributions are described by parameters that can be estimated from the sample. The selections of the distribution and the parameter estimation are based on a hypothesis or assumption, the correctness of which must be checked with statistical tests.

The first part of this unit covers the relationship between frequency distributions of samples and probability distributions. This is followed by an explanation of how the parameters of distributions are estimated from random samples. Finally, we describe the t-test, which plays an essential role in testing hypotheses in inferential statistics.

## 5.1 Distributions

In descriptive statistics, data are described using the distribution of frequencies. In inferential statistics, on the other hand, one deals with the distribution of probabilities, i.e., how the probabilities of possible random results are distributed.

The basis for correct conclusions about the underlying population is a representative sample that is as large as possible. Nevertheless, a random sample is only a section of the total population and is subject to errors. A crucial question in inferential statistics is whether certain properties and phenomena of a sample can be generalized to the population or whether they are a special feature of the available random sample. It is therefore important to indicate probabilities for how reliably the sample describes the population. This requires probability distributions, which can be estimated from random samples.

All assumptions of inferential statistics assume that the samples were drawn at random. The selection of the sample itself can thus be described as a random experiment. The set of all possible outcomes of a single experiment is called the sample space and is denoted with ; its elements are denoted with . For example, when tossing a coin, consists of the elements , heads, and , tails. A random variable, denoted by an uppercase letter, is a function on the sample space. Any realization of the random variable is denoted by lower-case letters (Beichelt, 2018).

For example, if the IQ values of a sample of randomly selected students are determined, the assignment of a measured IQ value to each student would be the random variable . The experiment (sample selection) of the students could be described as follows: the set of all randomly selected students corresponds to . Each individual student from would be an ω of the sample. The random variable now assigns an IQ to each student, which could be any positive number .

One distinguishes between discrete and continuous random variables. Discrete random variables can only assume a finite or countably infinite number of values. This applies, for example, to integers (e.g., a number between 1 and 6 on the dice). Continuous random variables, on the other hand, can assume any value, e.g., body size or temperature (Beichelt, 2018).

If a probability is now assigned to each outcome of the random variable, this assignment is referred to as a probability function for discrete variables and as a probability density for continuous variables. In this way, every number on a die can be assigned the same probability of 1/6, so (where stands for probability). However, with continuous variables such as age, the probability that a person is exactly 10 years, 3 months, 12 days, 3 hours, 4 minutes, 40 seconds, etc., is equal to zero. However, one can specify so-called densities, namely how large the probability is that the random variable is greater than or less than an exact value: or . If the individual probabilities are summed up, one speaks of a distribution function or **c**umulative **d**ensity **f**unction (CDF) (Beichelt, 2018).

In R, the most important statistical distributions are available as functions. These include functions for the generation of random numbers that follow a specific distribution as well as for the description of the probability, density, and quantile functions. These functions begin with r (for random), d (density), p (probability), and q (quantile), followed by the distribution name. In this way, for example, random numbers for the binomial, Poisson, and geometric discrete distributions are generated with rbinom, rpois, and rgeom. Random numbers derived from continuous distributions such as the normal, exponential, t, and gamma distribution are generated with rnorm, reexp, rt, and rgamma. An overview of the distributions implemented in R and their parameters can be obtained with the command help("Distributions") (R Core Team, n.d.-b).

Each of these distributions has a different number of parameters that characterize the distribution and must be estimated from the sample. For example, the normal distribution is described by the expected value (mean) and the spread (variance or standard deviation), so it is a two-parameter distribution. For this distribution, the two parameters can be estimated directly from the sample. For other distributions, the estimation is more complex, and different methods are often available, such as the method of moments and the maximum likelihood estimator (see for example Martin et al., 2013 for more detailed information on different methods of parameter estimation). Estimating parameters is another focus of inferential statistics. In general, a distinction is made between point estimators, i.e., the specification of the individual parameters that are as precise as possible, and interval estimators, which specify confidence intervals within which the estimated parameters are located (Sahu et al., 2015).

Understanding the relationship between the sample and the population is essential for inferential statistics. Therefore, the remainder of this section covers an illustrative example. Suppose you could record the IQ score of all students in a country. One would thus know the properties of the entire population. The purely theoretical distribution derived from this fictitious example is shown in the following figure. As can be seen, the mean IQ value of all students in that country would be 100. The form of the normal distribution follows a Gaussian bell curve and is determined by the expected value and the variance . In the following figure, the dotted blue line marks the theoretical mean value of 100, which is called the expected value.

Theoretical Distribution of IQ Values

Chart, histogram

Description automatically generated

This figure is generated with the following code. Here, the lines() command connects the points in the two vectors(x, hx). With the polygon() command, the area underneath the drawn curve is colored red. The abline() command adds a straight line to the current plot. The argument v=100 indicates that a vertical line is to be drawn at the x-value 100. The previously seen command text()adds text at a specified position. The polygon drawing is explained in detail int the next section (R Core Team, n.d.-b).

# Normal distribution parameters mean and sd

mean <- 100

sd <- 15

x <- seq(-4,4,length=100)\*sd + mean

# Calculation of the density function at the points x

hx <- dnorm(x,mean,sd)

# Limits for display

j <- x>=40 & x<=160

plot(x, hx, type="n", xlab="IQ Values",

ylab="", main="Theoretical Distribution",axes=FALSE)

lines(x, hx)

polygon(c(40,x[j],160), c(0,hx[j],0), col="red")

abline(v=100,lty=2,col=4)

text(102,hx[50]+0.0005, expression(mu),col=4)

axis(1)

For cost reasons, however, it is not possible to test all students. Therefore, one needs to limit oneself to randomly selected samples. The results for these samples then allow conclusions to be drawn about the population, for which the result is unknown.

Therefore, 100 students are selected randomly, and the frequency distribution of their IQ values is calculated. This is simulated here using the rnorm() function, which draws random numbers from a normal distribution. The number of values (in this case 100) and, optionally, the parameters of the normal distribution (a mean value of 100 and a standard deviation of 15) are passed to the function as arguments. If no parameters are passed, the parameters of the standard normal distribution (a mean value of 0 and a standard deviation of 1) are used (see also ?rnorm). The results of the sample students’ IQ tests are then plotted in a scatter plot (R Core Team, n.d.-b).

rx1 <- rnorm(100,100,15)

plot(rx1,xlab="Students",ylab="IQ")

Scatter Plot of the IQ Values of Students in a Random Sample

Chart, scatter chart

Description automatically generated

hist(rx1,main="Histogram",xlab="IQ", ylab="Frequency",col=8)

abline(v=mean(rx1),lty=2,lwd=2)

text(101,24.5,expression(bar(x)))

Histogram of the IQ Values of Students in a Random Sample

Chart, histogram

Description automatically generated

The form of the frequency distribution of the sample is right-skewed, with the most common IQ values being between 90 and 100 with a mean x of 98.8. The question now is whether one can draw any conclusions about the population from this sample and how reliable these conclusions may be.

### Self-Check Questions

1. What is the difference between the functions rnorm() and runif()?

*Both draw random numbers according to a distribution. But rnorm() draws from the normal distribution and runif() from the uniform distribution.*

1. What arguments can be passed to rnorm()?

* Minimum and maximum possible value.
* *Mean and standard deviation.*
* Median and range.

1. What arguments can be passed to runif()?

* *Minimum and maximum possible value.*
* Mean and standard deviation.
* Median and range.

## 5.2 Samples

Samples are randomly selected from the population. Therefore, the parameters, or measures derived from it (such as the mean), are random variables – each sample will deliver a different mean. For this reason, the mean value itself has its own distribution with its own measures of central tendency, variability, and so on (Montgomery and Runger, 2010).

In the simplest case, several samples of a population are available, e.g., from an experiment that can be repeated several times. The repeated selection of random samples can then reduce or eliminate randomly occurring effects in a single random sample that does not correspond to the population. This can be achieved by determining the mean for each random sample and generating a distribution of mean values from the different parameter values obtained in this way. According to the **Central Limit Theorem**, the distribution of these means approaches a normal distribution as the number of samples increases, regardless of the shape of the frequency distribution of the individual samples. The mean of the mean values of the individual samples, therefore, approaches the expected value of the underlying population (Mukhopadhyay, 2020).

**Central Limit Theorem**

This important theorem states that the sampling distribution of the means will be asymptotically normally distributed, regardless of the underlying distribution of the data (Montgomery and Runger, 2010).

So, once again, in inferential statistics, one is interested in making statements about the properties of the population based on a sample. This property is characterized by parameters, which need to be estimated from the sample. Three important criteria that the parameter estimation from the sample should meet are:

* 1. unbiased: The mean of the distribution of the estimated parameter is equal to the population's parameter.
  2. efficient: The estimated value of the parameter should have the smallest possible variance.
  3. consistent: As the sample size increases, the probability of obtaining the true parameter increases (Schumacker, 2014).

In the example with the IQ values for students above, if the random selection of students were repeated several times, the expected value of the means would approach 100. The question now arises as to how reliable a statement based on a sample distribution of the mean values calculated in this way is. In other words, how big is the error in the estimated mean, or in which range does the mean lie with a selected probability of, for example, 90%?

To determine that error, the standard deviation for the population is needed. This standard deviation is unknown and must therefore also be estimated (estimates are denoted by a hat over the symbol). Similar to the standard deviation of a sample, this is defined as:

The difference to the formula for calculating the standard deviation of the sample is the reduced correction factor of (Steinberg and Price, 2020).

The standard error of the mean   now indicates how far the mean value determined from the sample deviates from the expected value of the population (Landers, 2018).

As we are inferring from a sample to a population, this standard error expresses how large the error of our estimate is. The range in which the calculated mean lies with a certain probability can now be specified. This range is called the confidence interval (Rees, 1987).

The limits of the confidence interval for the true (unknown) mean of a population with unknown variance a is computed as

**Degrees of Freedom**

The number of independent data points in the sample that can be used to estimate a parameter is called degrees of freedom.

Here is the sample size, is the mean of the sample, and its standard deviation. corresponds to the values of the t-distribution with **degrees of freedom** at the **significance level** (Smithson, 2003).

In R, the t-values can be determined with qt() with the parameters degrees of freedom df and significance level  (R Core Team, n.d.-b).

**Significance Level**

The significance level is the risk of rejecting the null hypothesis even though it is correct.

We consider a short example in which we are given a sample of 10 people whose heights were measured.

> x <- c(170,160,165,181,175,167,200,158,173,177)

> m <- mean(x)

> s <- sd(x)

> m; s

[1] 172.6

[1] 12.08488

The mean of the sample is 172.6 with a standard deviation of around 12.1.

With a selected significance level of , the t-value in R is obtained with the qt() function, and the upper and lower bounds can be calculated from this, within which the expected value lies with a probability of 95%.

> alpha <- 0.05

> n <- length(x)

> h <- qt(1-alpha/2,n-1)\*sd(x)/sqrt(n)

> lb <- m-h

> ub <- m+h

> lb; ub

[1] 163.955

[1] 181.245

The values 163.95 and 181.25 are the lower and upper limits. This means that there is a 95% probability that the expected value of the population lies between these two values.

### Self-Check Questions

1. Which distribution is used to determine the width of the confidence interval for the mean of a population?

* The distribution of the underlying population.
* The normal distribution.
* *The t-distribution*.

## 5.3 t-Tests

Not only can the confidence interval for the expected value of normally distributed populations be determined with the help of the t-distribution, but the hypothesis as to whether a given expected value (e.g., based on experience) corresponds to the expected value estimated from the sample can also be checked. This is called a one-sample t-test. It requires the values of the sample to come from a normally distributed population (with unknown mean and variance) (Boslaugh, 2012).

As an example, the average service life of a device is given by a manufacturer. With several randomly selected devices, this sample can be used to check whether this statement is reliable using the t-test. The t.test() function is available in R for this purpose (R Core Team, n.d.-b).

The t-test checks the so-called null hypothesis that the true and unknown expected value corresponds to the specified value , so . The alternative hypothesis states that there are significant differences, . This test is called two-tailed because if the values differ, one can be either less than or greater than the other. The one-tailed test, on the other hand, only tests systematic deviations in one direction (e.g., whether the true and unknown expected value is systematically smaller than the provided value) (Herzog et al., 2019).

Continuing with the example of the manufacturer, 100 random values for the service life of devices in years are drawn. This is done using the sample() command to draw 100 values between 3 and 9 with replacement (meaning that values that have already been drawn can be drawn again). To be able to later reproduce the example with the same values, i.e., to draw the same random numbers in a later execution of the script, the **random seed** must be set to a number (in the example it is set to 10000) using the command set.seed()before using the random number generator. (R Core Team, n.d.-b)

**Random Seed**

The value with which a pseudorandom number generator is initialized is called random seed.

set.seed(10000)

x <- sample(3:9,100,replace=T)

Now we must check whether the average service life that the manufacturer has specified for this device can be verified using the random sample. The average service life provided by the manufacturer is years. Therefore, the null hypothesis is that the true mean of the population is 5.8 years. The mean estimated from the sample is years. The result of the t-test is now:

> t.test(x,mu=5.8,alternative="two.sided")

One Sample t-test

data: x

t = 0.60665, df = 99, p-value = 0.5455

alternative hypothesis: true mean is not equal to 5.8

95 percent confidence interval:

5.52751 6.31249

sample estimates:

mean of x

5.92

In addition to the t-value, the result also provides the p-value, which indicates the probability of incorrectly rejecting the null hypothesis. Depending on the desired level of significance (probability of error), we must reject or keep the null hypothesis. If the p-value is less than the significance level (e.g., 0.05), differences between the provided value and the value derived from the sample have a statistical significance of level , therefore the null hypothesis that the estimated mean equals the provided value is rejected. If the null hypothesis is falsely rejected (meaning one concludes that the values differ when they actually do not), this is called a type I error. In a t-test, the significance level equates to the risk of having a type I error (Field et al., 2012).

The calculated p-value of 0.5455 is larger than the selected limit of and is therefore in the range in which the null hypothesis cannot be rejected. If, on the other hand, the manufacturer had given an average lifetime of 5.4 years, the t-test() method would have provided the following result.

> t.test(x,mu=5.4,alternative="two.sided")

One Sample t-test

data: x

t = 2.6288, df = 99, p-value = 0.009934

alternative hypothesis: true mean is not equal to 5.4

95 percent confidence interval:

5.52751 6.31249

sample estimates:

mean of x

5.92

In this case, the null hypothesis would have to be rejected for the significance level of , since the p-value of 0.006 is smaller than .

The following figure shows the confidence interval of the example above for the significance level of with the corresponding t-distribution displayed on the example’s scale. The green lines mark the lower and upper limits of the confidence interval. The blue lines represent the calculated mean values specified by the manufacturer of years (left) and years (middle). It can be seen that the smaller mean of years is left of the marked lower limit of the confidence interval and the null hypothesis must therefore be rejected.

The t-Distribution and Mean Values on Example Scale

Chart, line chart, histogram

Description automatically generated

With the t-test, either individual samples can be tested to determine whether they come from a population (the so-called one-sample t-test), or two samples (or populations) can be tested to determine whether they show significant differences. The latter is called a two-sample t-test. This test checks whether the expected values and of two samples (or populations) are equal. This hypothesis is tested using the means and of the two given samples.

### Digression: Generating a Plot Based on a Distribution

Generating meaningful figures based on data and statistical analyses is a key area of using R. This short digression explains how the figure “The t-Distribution and Mean Values on Example Scale” was generated. This will allow you to deepen your understanding of many of the R commands used. As always, detailed information on the commands used can be found in the R manual (R Core Team, n.d.-b).

As preparation, some variables are assigned to be used later in the script. The number of data points is assigned to n, the significance level is set as alpha <- 0.05, and the mean of x is calculated and assigned to the variable mean. Then the lower and upper bounds of the confidence interval are calculated using the t-distribution function qt() and assigned to variables lb and ub.

n <- 100; alpha <- 0.05

mean <- mean(x)

h <- qt(0.5\*alpha,df=n-1)\*sd(x)/sqrt(n)

lb <- mean+h

ub <- mean-h

As a first step of creating the desired curve, the data to be plotted are generated. A vector y\_t is created with 5000 equidistant values between -4 and 4. The vector hy is then generated comprising the y-values to the values in y\_t under the distribution function of the t-distribution with n-1 degrees of freedom. The values in y\_t are then transformed to fit the scale of the original example data; the resulting vector is assigned to a variable y.

y\_t <- seq(-4,4,length=5000)

hy <- dt(y\_t,df=n-1)

y <- y\_t\*sd(x)/sqrt(n)+mean

The plot() command is then used to plot y and hy. The argument type=”l” generates a line graph. Additionally, both axes are removed with the axes=FALSE argument. On the x-axis, all relevant x-values should be displayed. These values are saved as a vector variable axis\_points with their labels being written to the variable axis\_labels. The x-axis is then added using the command axis(1) with additional arguments for the axis points and labels.

plot(y, hy, type="l", ylab="", xlab="", axes=FALSE)

axis\_points <- c(5.2,5.4,lb,mean,6,5.8,ub,6.6)

axis\_labels <- sprintf("%.03f",axis\_points)

axis(1,at=axis\_points,las=1,labels=axis\_labels)

The t-Distribution on Example Scale

Chart

Description automatically generated with medium confidence

In the next step, the dashed lines and labels are added. The abline() command is used to add vertical dashed lines. The arguments lty and col are additional graphical parameters that also work in other commands. Here, lty defines the line type (2 stands for dashed, the full list can be found in the documentation of ?par). The argument col sets the color. It can be provided in different encodings, for example as the name of a color or as an integer, where each integer stands for a specific color.

The labels are then added with the text() command. The first two arguments to this command specify the position of the text, while the third argument specifies the text to be displayed. The expression() command prints mathematical formulas.

abline(v=lb,lty=2,col="dark green")

text(lb+0.02,0.25, expression(frac(alpha, 2)),col="dark green")

abline(v=ub,lty=2,col="dark green")

text(ub-0.02,0.25, expression(frac(alpha, 2)),col="dark green")

abline(v=mean,lty=2,col=4)

text(mean+0.02, 0.2, expression(mu),col=4)

abline(v=5.4,lty=2,col=4)

text(5.38, 0.2, expression(mu[1]),col=4)

abline(v=5.8,lty=2,col=4)

text(5.78, 0.2, expression(mu[0]),col=4)

The t-Distribution on Example Scale with Dashed Lines and Labels

Chart, line chart

Description automatically generated

For the graphic to be generated, the areas under the left and right tails of the bell curve need to be colored in red. This is done by constructing two polygons that correspond exactly to the part to be colored, one for each tail. Each of these polygons has its border along the plotted curve, the dashed line down to the x-axis, and then along the x-axis.

For the left polygon, a logical vector j is defined. This vector has the same size as x. It contains TRUE for all entries where the corresponding entry in x is less than or equal to lb and FALSE otherwise. One can index a vector with a logical vector of the same size. X[j] is the vector of all entries for which the entry in j was TRUE, and analogously for hx[j]. To ensure the area below the curve is colored, the point (lb,0) is added to the polygon. The polygon is then added to the current plot using the command polygon(). As arguments, the vectors for the x and y coordinates as well as the color are provided. The polygon for the right tail is generated analogously.

j <- y<=lb

polygon(c(2,y[j],lb), c(0,hy[j],0), col="red")

k <- y>=ub

polygon(c(ub,y[k],10), c(0,hy[k],0), col="red")

### Self-Check Questions

1. What is the null hypothesis of the two-tailed t-test?

*The null hypothesis of the t-test states that the mean of the sample equals the mean of the population.*

1. What does the p-value indicate?

* The probability of correctly rejecting the null hypothesis.
* *The probability of incorrectly rejecting the null hypothesis*.

1. Which information does the t-test() command return (mark all that apply)?

* *t-value and p-value.*
* The prediction interval for the mean of the population based on the sample.
* The sample estimate for the standard deviation.
* *The sample estimate for the mean.*
* *The confidence interval for the mean of the population based on the sample.*

Summary

In descriptive statistics, characteristic values are calculated from the data to describe the information contained in the data with just a few values (e.g., using the mean value and the variance). However, the series of numbers to be examined usually only represent a small, randomly selected part of the population, so-called random samples. The main interest in statistics is in making statements that are valid for the underlying population, not just for the sample at hand. To do this, it is necessary to draw conclusions about the underlying population from the limited information in a sample using inferential statistics. Therefore, methods from probability theory are used since the unknown populations must be described using probability distributions.

In this unit, a brief overview of different probability distributions has been given and their practical application in R has been shown. Another focus of inferential statistics is testing how well the samples and their estimated parameters describe the population. For this purpose, the t-test was introduced. This test allows us to check whether the mean value estimated from the sample agrees with the expected value of the population and within what limits it is most likely to be found.

# Unit 6 – Analysis of Variance

**Study Goals**

On completion of this unit, you will be able to

* differentiate between a t-test and an analysis of variance.
* explain what an analysis of variance is.
* perform an analysis of variance with one factor (one-way analysis of variance) and two factors (two-way analysis of variance).

## Introduction

Analysis of variance (ANOVA) is used to describe differences in a measured target variable depending on other variables. Essentially, one examines to what extent the mean values and variances of one target variable differ depending on the combination of values of the other variables. While the t-test is used to examine the differences in the means between two groups (e.g., between the sample and the population), analysis of variance examines the relationships when more than two groups are to be analyzed (Robert and Russo, 2014).

Analysis of variance examines the effect of one (or more) independent variables on one (or more) dependent variables. Independent variables are variables whose values are intended to explain the realization of one or more other, dependent variables. In the context of analysis of variance, the independent variables are also referred to as factors, which are usually in the form of discrete parameter values (factor levels). For example, the variable purchasing power could be divided into the three factor levels weak, medium, and strong. Depending on whether the effect of one factor (with more than two distinguishing features), two factors, or more than two factors is examined, one speaks of one-, two-, or multi-factorial analysis of variance (Robert and Russo, 2014).

## 6.1 Principles and Delimitation to the t-Test

The aim of analysis of variance is to find and describe possible relationships between different groups (factor levels) of one or more independent variables (factors) and a measurable target variable. An example is the goal of determining whether and to what extent different treatment methods affect a target variable (e.g., state of health).

For example, you could test whether the factor sleep deprivation affects learning behavior. The independent variable, in this case, would be the sleep deprivation variable, which is divided into different levels such as (A) no sleep deprivation, (B) one day, and (C) two days of sleep deprivation. The learning behavior could be checked by presenting each person with a number of pictures, which they must memorize in a given amount of time.

Analysis of variance provides a way of testing for such an effect. It tests whether the mean values of the individual groups of the independent variables agree or whether there are significant differences between the individual groups. If significant differences between the factor levels can be identified, one can conclude that the level of a factor influences the target variable. To determine this, the total variance of the dependent variables is broken down into the variance of the groups’ mean values relative to the variance of each group’s individual values around the means of that respective group. (Field et al., 2012)

In the example with the three levels of sleep deprivation (0, 1, and 2 days), one considers on the one hand how large the variance of the individual groups is and on the other hand the variability of the three groups as a whole. If there are large differences between the means of the individual groups, that is, the variance of the three groups together is very large, but at the same time the values within a group have a small spread around their means, then it can be concluded that the three groups very likely have different effects on the target variable and do not come from a common population.

As with the t-test, the null hypothesis would be that all expected values of a factor with levels are the same:

The alternative hypothesis, on the other hand, states that at least two expected values are different. The difference to the t-test is the number of independent variables. Analysis of variance thus represents a generalization of the t-test. (Field et al., 2012)

In the t-test, only two levels are compared. With the one-sample t-test, the relationship between a sample and the underlying population is analyzed, with the two-sample t-test, the relationship between the mean values of two samples or the populations derived from them is analyzed. One could now consider extending the t-test to several groups by applying it individually to each pair of groups. However, this leads to the so-called problem of multiple comparisons (Robert and Russo, 2014).

Assume for example that a sample with the average weight is collected in three countries and then tested separately for equality, the individual null hypotheses for the three expected values would be:

As with any test analysis, a probability of error is selected (e.g., 5%). This expresses the probability of rejecting the null hypothesis even though it is correct, which is also referred to as a type I error. For the entire null hypothesis: , the probability that no type I error occurs is therefore , i.e., significantly lower than . This means that with multiple comparisons of means, the probability of a type I error increases, and the null hypothesis is rejected even though it is correct.

This cumulative effect can be reduced by correction factors. In R, the pairwise.t.test() function enables performing this repeated t-test with appropriate corrections (R Core Team, n.d.-b). However, instead of performing multiple t-tests with correction factors, an analysis of variance should be carried out.

### Self-Check Questions

1. How high is the overall probability that a type I error occurs when conducting t-tests to test the null hypothesis of equal means of five groups, each with a probability of error of = 5%?
2. Which command allows to conduct multiple t-tests with appropriate correction factors?

* *pairwise.t.test()*
* t.test(mode = pairwise)
* corrected.t.test()

## 6.2 One-Way Analysis of Variance

In the simplest case, analysis of variance consists of only one independent variable (one factor) with more than two different levels, and one dependent variable (that has, at least interval scale). The data set can then be divided into groups. The analysis of variance is based on the variability of the target variable values (using the variance), which can be broken down into two parts:

* 1. Between-group variance – the variance due to group membership, expressed as the deviations of the group means of each group from the overall mean.
  2. Within-group variance – the remaining variance that is not due to group membership, expressed as the deviations of individual data point values from the respective group mean (Weinberg et al., 2020).

Mathematically, this decomposition of variance in a data set with data points can be formulated as

Here, is the overall mean, is the mean of group . is the number of data points in group , so . The total spread (Sum of Squared Total) is the sum of the spread within the groups () and the spread between the groups (). The mean square deviations are given by the variance divided by the associated degrees of freedom ( or ):

The derived test value of the analysis of variance is the value. This is the ratio of the mean square deviations between the groups to the mean square deviations within the groups, so

As with the t-test, the decision whether to accept or reject the null hypothesis can again be made by comparing the calculated value with a critical value, where is the value of the -distribution with degrees of freedom and and significance level . As is the case for other parametric tests based on the normal distribution, certain requirements need to be fulfilled to get a reliable result with the -test. The variances of the groups need to be similar, and the data points need to be independent (Field et al., 2012).

In R, an analysis of variance can be calculated using one of two methods: using the command anova() applied to the result of the lm() function or using the aov() command. In the first method, lm stands for “linear model”, which is also used to calculate linear regression models, while both anova and aov stand for analysis of variance. The results are identical for both methods (R Core Team, n.d.-b).

This section is closed with an example of how to carry out the one-factor analysis of variance in R. As always, detailed descriptions of all commands used can be found in the R documentation (R Core Team, n.d.-b).

A manufacturer offers a product in three different product variants A, B, and C. The aim of the analysis is now to determine whether all three products have the same effect on the total revenue or whether there are differences between the product variants and they, therefore, have different effects on the total revenue. For the independent target variable revenue, 30 random numbers between 3000 and 5000 are drawn based on the uniform distribution using the runif command (runif(30,1000,5000)) and rounded to integer values using the round function. A vector good is then built that contains the product variants (A, B, or C) that correspond to each revenue value in the revenue vector. Each product variant was sold ten times.

> set.seed(10000)

> revenue <- round(runif(30,1000,5000),0)

> good <- rep(c("A","B","C"),each=10)

> revenue

[1] 2767 2895 2331 3403 2630 2329 4012 4129 3115 1223 1489 1512 3612 3891 1579 1903 3751 4963 4894 1576 3951 2164 2055 3306 2517 1472 2224 3176 2772 4675

> good

[1] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "B" "B" "B" "B" "B" "B" "B" "B" "B" "B" "C" "C" "C" "C" "C" "C" "C" "C" "C" "C"

To initially describe the data, they are presented as boxplots for the three groups of the factor.

> boxplot(revenue~good, col=2:4)

It should be noted that the function parameters can be specified in the form of a formula revenue~good. Here the tilde (~) separates the dependent variable on the left (revenue) from the independent variables (factors; in this case, the product variant) on the right (good).

Boxplot for the Revenue of the Different Product Variants

Chart, box and whisker chart

Description automatically generated

After that, the -test can be performed using the anova function applied to the linear model result. The linear model itself is irrelevant to the analysis of variance here, but the anova function needs the output of the lm function as an input. Alternatively, the aov function can be applied directly to the data.

> anova(lm(revenue~good))

Analysis of Variance Table

Response: revenue

Df Sum Sq Mean Sq F value Pr(>F)

good 2 37385 18692 0.0149 0.9852

Residuals 27 33811898 1252293

Since the p-value (Pr(>F)) of 0.985 is significantly larger than the selected -level of 5% error rate, the null hypothesis that the three different groups have the same expected values cannot be rejected. This means that the three product variants deliver similar revenues. This was to be expected, as the revenues for the three product variants were drawn from the same distribution.

What would have happened if the revenue of commodity C had been 50% lower? This can be achieved by multiplying the rows of revenue with good=="C" by 0.5. The boxplot of the new revenue clearly shows the effect of the product variant C on the total revenue.

> revenue\_new <- revenue

> revenue\_new[good=="C"] <- revenue[good=="C"]\*0.5

> boxplot(revenue\_new~good,col=2:4)

Boxplot for the Revenue of the Different Product Variants with Altered Revenue for Product Variant C

Chart, box and whisker chart

Description automatically generated

> anova(lm(revenue\_new~good))

Analysis of Variance Table

Response: revenue\_new

Df Sum Sq Mean Sq F value Pr(>F)

good 2 14699226 7349613 7.2058 0.003107 \*\*

Residuals 27 27539005 1019963

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

The result of the analysis of variance is now a p-value of 0.0031 which is less than the threshold . It can, therefore, be concluded that there are significant differences in revenues depending on the product variant, and the probability of a type I error by incorrectly rejecting the null hypothesis is very low.

One gets the same result with the aov() function:

> a1 <- aov(revenue\_new~good)

> summary(a1)

Df Sum Sq Mean Sq F value Pr(>F)

good 2 14699226 7349613 7.206 0.00311 \*\*

Residuals 27 27539005 1019963

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

What the simple -test does not reveal is which of the levels deviate the most from each other or have the greatest effect. This can be determined with the so-called Tukey's test, in which the significance of the differences is tested in pairs (Moody et al., 2011). In R, this test can easily be performed with the TukeyHSD() command.

> posthoc <- TukeyHSD(x=a1, conf.level=0.95)

> posthoc

Tukey multiple comparisons of means

95% family-wise confidence level

Fit: aov(formula = revenue\_new ~ good)

$good

diff lwr upr p adj

B-A 33.6 -1086.242 1153.4425 0.9969538

C-A -1467.8 -2587.642 -347.9575 0.0083974

C-B -1501.4 -2621.242 -381.5575 0.0069907

The results of the Tukey's test show significant differences between product variants C and A and between C and B.

### Self-Check Questions

1. In a one-way analysis of variance, into what parts is the total sum of squares split? *Into the sum of squares between the means of the factor level groups and the sum of squares within the factor groups.*
2. Which command can be used to perform a one-way analysis of variance in R between the dependent variable price the independent variable color?

* anova(price~color)
* aov(color~price)
* anova(color~price)
* *aov(price~color)*

## 6.3 Two-Way Analysis of Variance

A two-factor analysis of variance examines the effects of two independent factors on a dependent variable. It not only investigates how each factor individually affects the dependent variable, but also the magnitude of the influence of the combination of the two factors, and, thereby, the effects of possible interactions between the two factors are examined. Of course, this increases the computational effort and the number of tests that need to be carried out compared to the one-way analysis of variance, in which the influence of only one factor on the dependent variable is considered. Essentially, however, the examination methods remain the same as in the one-factorial analysis (Field et al., 2012).

In the one-factor analysis of variance, the total variance is broken down into the variance caused by that one factor, i.e., the variance between the groups, and the variance within the groups, which cannot be explained by the factor level. If the effects of two factors on a target variable are to be examined, the variance between the groups consists of the two variances separately for the two factors plus the variance due to the interaction of the two factors. As the number of factors to be examined increases, so does the number of possible combinations and, therefore, the number of variance components. The following figure and table show and summarize the variance decompositions for the two-way analysis of variance (Field et al., 2012).

Decomposition of the Variance in a Two-way Analysis of Variance

Diagram

Description automatically generated

The total explained variance , which explains the variability between the groups, is made up of the variance explained by factor 1, the variance explained by factor 2, and the variance explained by the interaction of factors 1 and 2. The following table contains the formulas for the decomposition elements of the total variance. It uses the following denotations:

* is the total number of observations and is the overall mean.
* Factor 1 has levels, factor 2 has levels.
* is the number of observations in the group with factor level for the first factor and factor level for the second factor. is the mean of that group.
* is the number of observations in the group with factor level for the first factor. is the mean of that group.
* Analogously, is the number of observations with factor level for the second factor, and is the mean of that group.

|  |  |
| --- | --- |
| Formulas for the Decomposition of the Variance in a Two-way Analysis of Variance | |
| Total variance |  |
| Total explained variance |  |
| Variance explained by factor 1 |  |
| Variance explained by factor 2 |  |
| Variance explained by the interaction of factors 1 and 2 |  |
| Unexplained variance |  |

The two-factor analysis of variance requires three different tests, for which the test statistics are again determined from the ratios of the mean sums of squares:

|  |  |  |
| --- | --- | --- |
| Test Statistics for the Two-way Analysis of Variance | | |
| Type of variance | Mean squared sum | -test statistic |
| Total ( |  |  |
| Total between the groups () |  |  |
| Factor 1 () |  |  |
| Factor 2 () |  |  |
| Interaction of factors 1 and 2 () |  |  |
| Within the groups () |  |  |

The following null hypotheses can now be tested using the three F-test statistics:

* Factor 1 does not influence the variance of :
* Factor 2 does not influence the variance of :
* The interaction of factors A and B does not influence the variance of : (Field et al., 2012).

As an illustrating example for the two-way analysis of variance in R, data from a fictitious study on how the factors of smoking and activity level affect the dependent variable are analyzed. As always, details on the used commands and packages can be found in the R documentation (R Core Team, n.d.-b).

The measurable variable is the time it takes the individual test subjects to complete a half marathon. For this purpose, a data frame with the dependent variable time and the two factors smoker and activity is created. The factor smoker consists of the two levels non-smokers (NR) and smokers (R). The activity level factor is divided into three groups:

* 0 – no regular physical activity,
* 1 – moderate regular physical activity, and
* 2 – high level of regular physical activity.

After generating the test data, the factors can be displayed as a function of the mean values of with the plot.design command.

> datasm <- data.frame(

+ time = c(120, 130, 150, 110, 99, 140, 100, 90, 120, 240, 200, 180, 150, 170, 220, 119, 130, 115),

+ smoker = factor(c(rep("NR", 9), rep("R", 9))),

+ activity = factor(rep(c(rep(0, 3), rep(1, 3), rep(2, 3)),2)))

> plot.design(datasm,ylab="Average Time",xlab="Factors")

Plot of Univariate Effects of the Average Times by Factors Smoker and Activity

Calendar

Description automatically generated with low confidence

This plot shows the overall mean (as a long horizontal line segment) as well as the mean values for the individual groups of the factors (as one short horizontal line segment per group of that factor). These mean values are necessary for calculating the -tests for the factors without interaction. Boxplots are also useful to illustrate possible interactions. The R package lattice contains numerous additional features to display dependencies for multivariate analyses. The package is not installed by default and must therefore first be installed and loaded.

> install.packages("lattice)

> library(lattice)

Then lattice's own command of boxplots bwplot() can be used. As seen before, the function argument can be passed as a formula. One feature of lattice functions such as bwplot is that one can specify dependencies explicitly with a vertical bar (|). The function call time~activity|smoker expresses that the variable time is dependent on the factor activity displayed in separate box plots according to the groups of the factor smoker.

> bwplot(time~activity|smoker,datasm)

Boxplots of the Time Depending on Activity Grouped by Factor Smoker

Chart, box and whisker chart

Description automatically generated

Of course, the time depending on the smoking factor can also be shown grouped by the activity factor groups (0, 1, and 2).

> bwplot(time~smoker|activity,datasm)

Boxplots of the Time Depending on Factor Smoker Grouped by Activity

Graphical user interface, chart, box and whisker chart

Description automatically generated

You can already see in these boxplots that there are large differences between times in the individual groups of the two factors.

In R, the two-way analysis of variance can be performed just like the one-way analysis, either directly with the command aov on the data frame or with anova applied to the result of a linear model (lm). To be able to account for the interactions between the two factors, the formula must be specified as time~smoker\*activity. The asterisk \*, which is the multiplication operator in R, indicates that the factors smoker and activity, as well as the interaction between smoker and activity, should be considered. If a + is used instead of the \* in the formula (time~smoker+activity), interactions are not considered.

The summary command applied to the results of the analysis returns a table with the most important information: the sums of squares, the mean sums of squares, the -values and p-values for the two factors, and their interaction. For the sake of clarity, the significance of the probability values (Pr(>F)) is again marked with asterisks.

> summary(aov(time~smoker\*activity,datasm))

Df Sum Sq Mean Sq F value Pr(>F)

smoker 1 12013 12013 22.449 0.000482 \*\*\*

activity 2 10172 5086 9.505 0.003358 \*\*

smoker:activity 2 2620 1310 2.448 0.128310

Residuals 12 6421 535

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Since the p-values for the individual factors are very small separately (i.e., less than 0.001 or 0.01), the null hypotheses for the two factors must be rejected separately. This means that both the smoking factor and the activity level factor significantly influence the dependent variable time. The null hypothesis for the interaction of the two factors cannot be rejected, which means that an effect of the combination of smoking and activity level on the time cannot be proven.

The interactions of the factors can be analyzed in more detail with interaction plots. To be able to access the three variables time, activity, and smoker directly without specifying the data frame (e.g., datasm$smoker, or datasm[["smoker"]]), they can be made "visible" in the working environment. This is achieved with the attach() command. The three variables can then be called up and used directly with their names.

> attach(datasm)

> interaction.plot(smoker,activity,time)

Interaction Plot of the Factor Smoker Grouped by Activity

Chart, line chart

Description automatically generated

> interaction.plot(activity\_level,smoker,time)

Interaction Plot of the Factor Activity Grouped by Smoker

Chart, line chart

Description automatically generated

As can be seen in the interaction plots, both factors show an effect on the average time. While the mean time increases when the level of the factor smoker is R, the factor activity causes a decrease in duration. If the combination of smoking and activity together had an influence on the target variable, the lines would intersect, which is not the case here. More on the interpretation of interaction plots and factor analysis can be found in Navaro (2019).

### Self-Check Questions

1. A survey asked test subjects how likely they would be to buy a known product in a new, purple color. The test subjects’ age groups and genders were recorded. The marketing manager assumes that elderly ladies will be more likely to buy, the other groups will not be affected by the new color. If she is correct, which value will be highest?
2. What is the difference between the formulas aov(x~y\*z) and aov(x~y+z)?

* *aov(x~y\*z) also analyzes the effect of the interaction between y and z, aov(x~y+z) does not.*
* aov(x~y\*z)analyzes only the effect of the interaction between y and z, aov(x~y+z) only the effect of x respectively y on their own.
* There is no difference, those are formulas are interpreted equivalently by R.

Summary

The t-test examines relationships between two groups. For example, a t-test may be used to decide whether the mean of a sample is the same as that of a population or whether the means of two samples show significant differences. Analysis of variance is an extension of the t-test, which examines the relationships between more than two groups.

In the one-way analysis of variance, the relationship between one explanatory variable (a so-called factor), which can be divided into more than two different groups, and a dependent variable (the response or target variable) is sought to be clarified. For this purpose, the ratios of the variance between the individual groups and within the groups are evaluated using the -test. Here the size of each group's effect on the target variable is analyzed.

In the two-factor analysis of variance, this analysis is expanded by an additional factor. In this case, the effects of the interactions between the two factors must also be considered.

# Unit 7 – Regression Analysis

**Study Goals**

On completion of this unit, you will be able to

* perform a correlation analysis in R.
* perform linear regressions in R.
* explain generalized regression models.

## Introduction

The main task of regression analysis is to find a mathematical model that reflects a relationship between a dependent variable and one or more explanatory, independent variable(s) as simply and as accurately as possible. A regression model is used to explore the strength of the correlations, to describe them with a simple model, and to derive predictions for any unknown value combinations of the independent variable(s).

The simplest case is that of linear relationships with one independent variable. Here the independent variable has a direct influence on the dependent variable (e.g., height and weight). If this relationship can be described mathematically with sufficient accuracy using a simple linear equation, the expected value of the dependent variable can be determined for each value of the independent variable. Of course, even for well-fitted models, these predictions are based on simplified model assumptions, which are always subject to errors. It is, therefore, essential to specify the reliability and error probabilities for the predictions derived from the regression model.

Also, linear models often fail to describe more complex relationships. As a result, numerous model modifications have been developed, some of which will be discussed at the end of this unit.

## 7.1 Correlation

Correlation analysis is the first analysis of a quantitative relationship and

In this section, we will describe how to infer a population from the relationship between samples. In the simplest case, it is assumed that the two samples follow the normal distribution. Since one does not know the population containing the two samples, one must estimate the unknown correlation of the population from the sample. Pearson's correlation coefficient can be used to estimate the correlation between two normally distributed random variables and of size :

Here are realizations of and realizations of , which come in pairs as . denotes the sample correlation. (Rees, 1987)

The correlation coefficient lies between the values -1 and 1. A correlation coefficient of 0 means that there is no connection between the two random variables and . A correlation coefficient of +1 describes a perfect positive relationship between the two variables; in a scatter plot this would be represented by the points all lying in a straight line with a positive slope. A correlation of -1 indicates a perfect negative (inverse) relationship (anti-correlation), in which case a scatter plot would show the points all lying in a straight line with a negative slope (Rees, 1987).

In the following paragraphs, an example of two normally distributed random variables generated by the rnorm() function is presented. As always, details on the used commands and packages can be found in the R documentation (R Core Team, n.d.-b).

The dependent variable is generated from the normally distributed variable by adding another normally distributed variable to it. Both variables are then displayed in a scatter plot.

> x <- rnorm(1000,0,1)

> y <- x+rnorm(1000,0.1,1)

> plot(x,y)

Scatter Plot of Two Dependent Variables

Chart, scatter chart

Description automatically generated

The correlation coefficient as an estimator for is now obtained with the cor() function. This first correlation analysis on the sample data is still part of descriptive statistics.

> cor(x,y)

[1] 0.6996556

One now wants to know whether the two variables correlate for the underlying population. This is again examined using the t-test. Under the assumption that the two random numbers and are normally distributed, it is tested whether the unknown correlation coefficient has the value zero. The test variable

comes from a t-distribution with degrees of freedom. The null hypothesis is now , which is rejected at the significance level if . (Field et al., 2012).

In R, this test is performed using the cor.test() function:

> cor.test(x,y)

Pearson's product-moment correlation

data: x and y

t = 30.936, df = 998, p-value < 2.2e-16

alternative hypothesis: true correlation is not equal to 0

95 percent confidence interval:

0.6665744 0.7299864

sample estimates:

cor

0.6996556

As mentioned previously, this command tests whether the estimated correlation coefficient of the population is equal or not equal to zero. The very small p-value (<2.2e – 16) shows that the probability of incorrectly rejecting the null hypothesis is less than the 95% significance level and therefore there is a significant correlation. Additionally, this command also calculates the confidence interval within which the estimated correlation coefficient of the population lies with a certain probability (in the example it is between 0.69 and 0.75 with a 95% probability).

However, the t-test is not suitable for random variables that are not normally distributed. In this case, distribution-free correlation coefficients, like rank correlation coefficients, should be used, where the ranks of the random variables are compared and possible relationships between those ranks are evaluated. Examples include Spearman’s rank coefficient (also known as Spearman’s rho) and Kendall’s tau. Since Kendall’s tau has already been described, Spearman's rank correlation coefficient will be discussed at this point. Spearman’s rank coefficient is based on the rank information, that is whether is smaller or larger than a value of the random variable . More precisely, let be the rank of , that is the position of in ordered according to size. Analogously denotes the rank of the associated realization of the variable . Then

with is an estimate for the correlation . (Rees, 1987)

In R, both Spearman's rank correlation coefficient and Kendall’s tau can be calculated, and the respective significance test can be performed by specifying "spearman" or "kendall" as the method argument to the cor.test() function (see ?cor.test) (R Core Team, n.d.-b).

### Digression: Generating Scatter Plots with Additional Information

The scatter plot above only provides a very simple overview. In this digression, a more appealing, more meaningful graphic for the same data is generated to demonstrate some of the more advanced plotting options in R. This new figure shows the bivariate density as a heat map with the frequency distributions of the variables and as histograms on the borders.

Heat Map and Histograms of Two Dependent Variables

Chart, histogram

Description automatically generated

The center part of the figure shows the relationship between the variables and as a heat map. Much of the data is near a straight line with a slope of 1 (denoted by warm colors). If the two variables correlated perfectly, all data points would lie on this straight line. Points that are further away from this straight line are less frequent and are, therefore, displayed in a colder color. The frequency distributions of the and variables are shown on the top and the right-hand side of the map, giving an overview of the distribution of the two samples.

To generate this figure, two libraries must be installed and loaded: RColorBrewer and MASS.

install.packages("RColorBrewer")

library(RColorBrewer)

install.packages("MASS")

library(MASS)

First, the histograms for x and y are generated and the variable names h1 and h2 are assigned to them for later reference. Each variable is divided into 25 segments of equal size. If the plot argument is set to F (shorthand for FALSE), the result is not a graphic, but the definition of a histogram graphic.

h1 <- hist(x,breaks=25,plot=F)

h2 <- hist(y,breaks=25,plot=F)

The next step is to compute the density of data points in the 25x25 grid for the heat map. This is done with the command kde2d (for “2-D kernel density estimate”). To plot these densities as a heat map, a color scheme needs to be defined. This is done with the loaded library RColorBrewer. The command brewer.pal creates a color palette with a specified number of colors (here 11) according to a color palette scheme (here 'Spectral'). This palette is then reversed using the command rev, otherwise, higher densities would be plotted in colder colors. Finally, the command colorRampPalette is applied to the generated palette to interpolate between the 11 colors.

k <- kde2d(x,y,n=25)

rf <- colorRampPalette(brewer.pal(11,'Spectral'))

The variable rf is now a reference to the generated color scheme, which is a function with one argument n, as can be seen when outputting the contents of the variable.

> rf

function (n)

{

x <- ramp(seq.int(0, 1, length.out = n))

if (ncol(x) == 4L)

rgb(x[, 1L], x[, 2L], x[, 3L], x[, 4L], maxColorValue = 255)

else rgb(x[, 1L], x[, 2L], x[, 3L], maxColorValue = 255)

}

<bytecode: 0x00000249ccb387e8>

<environment: 0x00000249ccbc6680>

This function rf returns n colors in hexadecimal encoding. To use this color palette in the figure, a vector of 32 colors is created and passed to the image command along with k. But first, the layout of the plot is set with the layout command.

Before each plot is added to the figure, the command par() is used to set the margin of the next plot. By default, the plot margins are quite big to allow for additional information like axis labels. As there are no axis labels to be displayed between the heat map and the histograms, the margins on the top and the right of the heat map are set to 1. The other two margins are set to 3 to allow space for axis labels here. The image command is used to plot the heat map, and the barplot command to plot the histograms.

layout(matrix(c(2,0,1,3),2,2,byrow=T), c(3,1), c(1,3))

par(mar=c(3,3,1,1))

image(k, col=r)

par(mar=c(0,2,1,0))

barplot(h1$counts,axes=F,space=0,col='red')

par(mar=c(2,0,0.5,1))

barplot(h2$counts,axes=F,space=0,col='red', horiz=T)

### Self-Check Questions

1. The scatter plot of two vectors and show that all points lay on a perfect straight line with slope 0. What is the correlation coefficient of and ?

* Either -1 or 1, as the straight line indicates a perfect relationship.
* *0, as the values of y do not depend on the values of x.*
* Unknown until a correlation analysis is conducted.

1. When would Kendall’s tau be used instead of Pearson’s correlation coefficient?

* For data on interval scale level.
* For data on only nominal scale level.
* *For not normally distributed data.*

## 7.2 Linear Regression

The correlation analysis was used to examine whether there was a linear relationship between two random variables and . Once such a linear relationship has been established, it is now the task of regression analysis to find a function that describes this relationship in a mathematical model. In the case of linear regression, this functional relationship is assumed to be of the form . Such a map can be used to describe the relationship and to make predictions in the form of calculated values of the corresponding dependent random variable for a given value of the independent variable . This can be particularly useful when one wants to fill in missing values of the dependent variable or to determine future values (Rees, 1987).

In this section, it is assumed that there is at least an approximately linear relationship between two characteristics in a population. In practice, this means that the scatter plot of data pairs resembles a straight line. The two realizations of and of are put into a linear equation of the form:

for . is the so-called absolute term (also called regression constant or intercept). is the slope (or increase) and the random error, which is assumed to be normally distributed in . The result is then a regression line with slope and the y-axis intercept that is as close as possible to all data points (Rees, 1987).

A simple method to estimate the parameters and is the method of least squares. The sum of squared deviations or residuals is defined as

The aim is to find values for the parameters and that minimize this squared distance between the function values and the data points (Field et al., 2012).

The value as given in the equation above depends on the values for and . Good estimates for these parameters minimize . These can be found using partial differentiation with respect to and . Doing so results in the following formulas for the estimates and for and . (Rees, 1987)

The total sum of squares can again be split into the deviations explained by the linear regression model and the unexplained part , so . The ratio of explained variance to unexplained variance is the important so-called value with

The -test can be used to assess the quality of the found linear regression. The mean squares are calculated as and . The -value is then

The larger the-value, the better the model fits the data (Field et al., 2012).

In the following example, a regression analysis is performed in R and the results are discussed in detail. The relationship is examined using a sample with measured heights and weights of ten randomly selected people. As always, details on the used commands and packages can be found in the R documentation (R Core Team, n.d.-b).

> weight <- c(60,45,73,87,56,90,63,70,73,53)

> height <- c(173,160,181,183,169,185,179,171,175,169)

> plot(weight,height)

Scatter Plot of Height and Weight

Chart, scatter chart

Description automatically generated

Linear regression is performed with the command lm() (lm stands for linear model). The argument is again an equation in the form of a formula, which is denoted by the tilde character (~). The dependent variable is on the left side, and the independent variable(s) is on the right side of the tilde.

> lm.weight <- lm(height~weight)

The first thing of interest is the regression line calculated by R to see how well the data points are represented by it and how far the individual points deviate from this line.

> plot(weight,height)

> abline(lm.weight)

> segments(weight,fitted(lm.weight),weight,height)

Scatter Plot of Height and Weight with Regression Line

Chart, line chart

Description automatically generated

The vertical distances shown in the graphic as line segments between the data points and the estimated regression line correspond to the residuals. The summary() command provides more information about the quality of the fit.

> summary(lm.weight)

Call:

lm(formula = height ~ weight)

Residuals:

Min 1Q Median 3Q Max

-4.9264 -2.0171 -0.3529 1.6604 6.4019

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 142.64286 5.75728 24.776 7.53e-09 \*\*\*

weight 0.47548 0.08419 5.648 0.000483 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.646 on 8 degrees of freedom

Multiple R-squared: 0.7995, Adjusted R-squared: 0.7744

F-statistic: 31.9 on 1 and 8 DF, p-value: 0.0004827

We will now explain this output in detail. The first piece of information in the output is a simple overview of the distribution of the residuals, i.e., the deviations of the data points from the straight line (represented as vertical line segments in the plot above).

Residuals:

Min 1Q Median 3Q Max

-4.9264 -2.0171 -0.3529 1.6604 6.4019

Because linear regression assumes that these deviations come from a normal distribution with a mean of zero, the median should be close to zero and the minimum and maximum should be roughly equidistant from the mean. Given the small sample size, the results shown are tolerable.

Next, we have an overview of the estimated model parameters and their standard error, as well as the t-values and the p-values derived from them. As before, the asterisks indicate the significance levels. Since both p-values are significant, the null hypothesis can be rejected and there is a significant linear relationship between weight and height.

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 142.64286 5.75728 24.776 7.53e-09 \*\*\*

weight 0.47548 0.08419 5.648 0.000483 \*\*\*

The function that describes the regression line has the form

The extent of the relationship is analyzed in the following lines, with the most important parameter being the coefficient of determination known as multiple R-squared. In a simple linear regression, this corresponds to Pearson's correlation coefficient. With an R-squared of around 0.8, the strong connection between the two variables is again confirmed. The last line shows the result of an -test. This test becomes more important when several independent variables are used in a so-called multiple regression. The -test is then used to check whether the addition of individual variables increases the significance of the identified relationship.

Residual standard error: 3.646 on 8 degrees of freedom

Multiple R-squared: 0.7995, Adjusted R-squared: 0.7744

F-statistic: 31.9 on 1 and 8 DF, p-value: 0.0004827

The residuals can be examined more closely in R using the plot() function.

par(mfrow=c(1,2))

plot(lm.weight,c(1,2))

Fitted Values of the Linear Regression

Chart, line chart, scatter chart

Description automatically generated

These two figures serve to check whether the assumption of the normally distributed residuals is fulfilled. If there are major deviations from the dotted lines (horizontal on the left and 1:1 line on the right), the assumption of a linear regression should be discarded, and alternative methods to find a model should be used.

With this first linear regression analysis, a functional relationship has been established, and its significance confirmed. One can therefore calculate estimates for the values of the dependent variables for other values of the independent variable. For example, to estimate the height of a person whose weight is 80 kg, one could calculate this by substituting the value in the regression equation.

> plot(weight,height)

> abline(lm.weight)

> points(80,180.64,col=2,pch=2)

> segments(80,180.64,80,50,lty=2)

> segments(80,180.64,40,180.64,lty=2)

Prediction of Height Using the Linear Regression Model

Chart, scatter chart

Description automatically generated

Of course, this is easier in R with the predict() function, which, in addition to the estimated value, also provides information about its reliability.

It is often important to know within what range the estimated value for the dependent variable lies with a desired certainty. This is made possible by specifying a confidence interval within which the searched value can be found with a certain probability. A distinction must be made between the confidence interval and the prediction interval. The confidence interval is the range in which the mean value is located with a given probability. This corresponds to the uncertainty of the estimated regression line. The prediction interval calculates the uncertainty about the actual position of unknown data points. Both relate to a chosen probability range. (Abell et al., 1999)

In the predict function, the interval type to be computed can be specified with interval="predict" for the prediction interval and interval="confidence" for the confidence interval.

> weight\_new <- data.frame(weight=80)

> predict(lm.weight,weight\_new,interval="predict")

fit lwr upr

1 180.6812 171.5081 189.8544

> predict(lm.weight,weight\_new,interval="confidence")

fit lwr upr

1 180.6812 177.0151 184.3473

For a weight of 80 kg, the estimated height is 180.68 cm and with a 95% probability, that height is between 171.51 and 189.85 cm (prediction interval). The uncertainty of the estimated mean itself is significantly lower and lies between 177 and 184 cm. The two uncertainty bands can be graphically represented as follows for values between 40 and 90 kg:

prog.frame <- data.frame(weight=40:90)

pp <- predict(lm.weight,int="p", newdata=prog.frame)

pc <- predict(lm.weight,int="c", newdata=prog.frame)

plot(weight,height,ylim=range(height,pp))

prog.weight <- prog.frame$weight

matlines(prog.weight,pc,lty=c(1,2,2),col="blue")

matlines(prog.weight,pp,lty=c(1,3,3),col="red")

Confidence and Prediction Intervals of the Linear Regression Model

Chart, line chart

Description automatically generated

The analysis methods shown so far can be used in the same way for regressions with several independent variables. The linear regression equation for independent variables is . The various explanatory variables must be added with a "+" in the input formula, e.g., lm(height~weight+age+gender).

### Self-Check Questions

1. Assume the data at hand has a Pearson’s correlation coefficient of 1. How high is the sum of squared residuals ?

* -1
* *0*
* 1
* Not known until the regression analysis was performed.

1. How can the residuals part of the summary() of a linear model help as an indicator of -distributed residuals?

* *The median should be close to zero and the lower / upper quantile as well as minimum / maximum should be equidistant from 0.*
* The mean should be close to 0.
* The variance needs to be positive.

1. A linear regression model is to be used to determine the range in which a single value of the dependent variable lies with a probability of 70%. This can be done using the *prediction interval.*

## 7.3 Other Models and Procedures

In the last section, some generalizations of the linear regression model are shown. These are necessary when assumptions such as the normal distribution of the residuals are not met or when the relationship between the independent variables and the dependent variable cannot be described by a straight line. Since the underlying mathematics is often very complex, we will only give a general overview with applied examples in R.

The generalized linear model (GLM) is an extension of the linear model, which includes non-normally distributed response variables as well as transformations for linearization. In R it can be calculated with the glm() function, which is used similarly to the lm() function. Here, the type of model to be computed must be provided, in addition to a formula that describes the dependent and the independent variables. For example, logistic regression models (also known as logit models) can be specified by passing family=binomial("logit") as an argument to the glm function (R Core Team, n.d.-b).

One common GLM model is the binary logistic model. The logit model makes it possible to explain a dependent binary variable (i.e., a variable with only two possible values, such as yes/no or 0/1) and to make a corresponding prediction of the probability with which an event will occur. In contrast to linear regression, in which any value between is feasible, with binary variables only the values 0 and 1 are possible. The effect of the explanatory variables (with and sample size ) on the probability for or is to be determined as

The probability for is not estimated directly from the explanatory variables (as is done in linear regression), but indirectly via the so-called logit. The logit is the logarithmic chance for the occurrence of and is calculated as the logarithm of the probability of occurrence divided by the opposite probability (Tutz, 2011):

As an example, the connection between the condition of the rubber seals and the temperature during the space shuttle accident in 1986 is investigated. The data are contained in the SMPracticals package, which must be installed and loaded beforehand.

> install.packages("SMPracticals")

> library(SMPracticals)

> data(shuttle)

> state <- as.numeric(shuttle$r > 0)

> temp <- shuttle$temperature

Now a generalized linear model is fitted to the data using the glm() command and the results are displayed using the summary() command.

> shut.erg <- glm(state~temp,family=binomial("logit"))

> summary(shut.erg)

Call:

glm(formula = state ~ temp, family = binomial("logit"))

Deviance Residuals:

Min 1Q Median 3Q Max

-1.0611 -0.7613 -0.3783 0.4524 2.2175

Coefficients:

Estimate Std. Error z value Pr(>|z|)

(Intercept) 15.0429 7.3786 2.039 0.0415 \*

temp -0.2322 0.1082 -2.145 0.0320 \*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 28.267 on 22 degrees of freedom

Residual deviance: 20.315 on 21 degrees of freedom

AIC: 24.315

Number of Fisher Scoring iterations: 5

Next, the predictions for the temperature range between 20 and 100 degrees can be computed.

> T.min = 20

> T.max = 100

> Temp.X <- seq(T.min, T.max, by=0.15)

> Temp.Modell <- data.frame(Temperature = Temp.X)

> Temp.Prog <- predict(shut.erg, Temp.Modell, type = "response")

As a result, you can now calculate the probabilities of failure of the rubber seals for different temperatures.

> plot(temp, state, main = "Prediction", xlab = "Temperature", ylab = "Probability P")

> lines(Temp.X, Temp.Prog)

> abline(h = seq(0,1,0.1), lty = 2)

> abline(v = seq(55,80,5), lty = 2)

Probability of Failure Based on Temperature

Chart, line chart

Description automatically generated

The lower the temperature, the greater the likelihood that the rubber seal will leak. Additional information on extended linear regression models can be found in, for example, Faraway (2016).

Another example of modified regression models is non-linear relationships between two variables that can be described using a polynomial function. As an example, the relationship between the price per unit and the quantity of a specific product is to be calculated. The total amount that must be paid is the product of . Normally, this relationship would be linear. However, discounts on purchases above a certain quantity can make this relationship non-linear.

p <- 0.4

q <- seq(from=0, to=20, by=0.1)

y <- 450 + p\*(q-10)^3

plot(q,y,type='l',main='Nonlinear Relationship',lwd=3)

Nonlinear Relationship between Purchase Amount and Price

Chart, line chart

Description automatically generated

The target variable is now overlaid with noise, which is intended to represent the error in the observation. The unknown function for describing is to be estimated from these noisy data.

noise <- rnorm(length(q), mean=10, sd=80)

noise.y <- y + noise

plot(q,noise.y,ylab="y",main="Observation")

Observation Data for a Polynomial Relationship

Chart, scatter chart

Description automatically generated

A third-degree polynomial of the following form is to be fitted as a regression model, which is implemented in R with the poly function.

> model <- lm(noise.y~poly(q,3))

> predicted.int <- predict(model,data.frame(x=q),interval='pred',level=0.95)

> plot(q,noise.y,ylab="y", main="Prediction")

> lines(q,predicted.int[,1], col='blue',lwd=3)

> lines(q,predicted.int[,2], col='green',lwd=1)

> lines(q,predicted.int[,3], col='green',lwd=1)

Generalized Linear Model for a Polynomial Relationship

Chart, diagram

Description automatically generated

> summary(model)$r.squared

[1] 0.8186651

As can be seen from the resulting plot, the model is well suited to describing the target variable with a function, which is also reflected in the high of 0.75. The prediction intervals also contain most of the observed data.

### Self-Check Questions

1. Which type of regression model can be used to model binary data?

* Linear
* Polynomial
* *Logit*

Summary

Regression analysis is about finding functional relationships between variables and describing them with a mathematical model. This assumes that there is a directional relationship between one or more independent variable(s) and a dependent variable. This relationship is first investigated with a correlation analysis. For normally distributed samples, this can be checked using Pearson's correlation coefficient. When the assumption of normally distributed values is violated, rank-based correlation coefficients are used (such as Spearman and Kendall).

If there is a linear relationship, a linear regression model can be used. Here, a best fit line for the data pairs is fitted and its parameters, such as the intercept and the slope parameter, are estimated. The -values can be used to check whether these parameters are significant and how large their uncertainty is.

In the case of a non-linear relationship or non-normally distributed values, numerous methods are implemented in R to derive a functional relationship and formulate a mathematical model. For example, the generalized linear model (GLM) allows for the use of samples that are not normally distributed or do not follow a linear relationship. Non-linear polynomial relationships can also be modeled by including polynomial functions in the lm() function.