FUNCTIONS FOR ECONOMISTS

A table of data in R is called »data frame«. Such a table can be arbitrarily manipulated and analyzed with R functions and R commands. An example of such a table, which we call mydata, is given below:

	ID	Age	Height	Gender	Place
1	1	30	180	1	LJ
2	2	25	170	0	LJ
3	3	50	175	1	MB
4	4	40	170	0	LJ
5	5	16	190	1	KP
6	6	22	178	1	NM
7	7	29	177	0	GO

Table is manipulated by `mydata[ , ]`. The notation before the comma refers to rows, the notation after the comma refers to columns. Examples:

```
mydata[c(-4, -5), ]
mydata[ , c(1, 2, 3)] or mydata[ , c(1:3)]
mydata[4, 3] <- 172
```

In the first example, we delete the 4th and 5th rows from the data table, in the second example, we select the 1st, 2nd and 3rd variables (columns), and in the third example, we replace the element in the 4th row and the 3rd column (i.e. 170) with the value 172. The single variable is accessed with the $ character. Example: `mydata$Age`.

The data table can be ordered according to the selected variable with the function `order`:

```
mydata[order(mydata$Age), ]
mydata[order(-mydata$Height), ]
```

In the first case, the data table is arranged in ascending order according to the age of the persons, in the second case in descending order according to the height of the persons.

The character <- (assign left) assigns a function or the command on the right to the selected object on the left.

The `%>%` (pipe) sign means »then«. To use it, we need to install the library `dplyr`.

R functions and commands are written in R Markdown, and all text must be enclosed in quotes. The R program distinguishes between upper and lower case, punctuation, brackets, etc., but it is not sensitive to spaces.

Wide format of the data table

	ID	Trust_A	Trust_B
1	1	8	3
2	2	5	5
3	3	4	6
4	4	7	2
5	5	2	9

Long format of the data table

	ID	Trust	Factor
1	1	8	A
2	2	5	A
3	3	4	A
4	4	7	A
5	5	2	A
6	1	3	B
7	2	5	B
8	3	6	B
9	4	2	B
10	5	9	B

Data specified in a frequency table should be converted to raw data using the `rep` function (see function explanation below):

Frequency table

	Members	Frequency
1	1	2
2	2	2
3	3	3
4	4	1

Raw data

	Members	Frequency
1	1	2
1.1	1	2
2	2	2
2.1	2	2
3	3	3
3.1	3	3
3.2	3	3
4	4	1

BASIC R FUNCTIONS and COMMANDS

install.packages()
library()

Installing and activating the new library.

```
install.packages("psych")
library(psych)
```

Basic functions for installing and activating new libraries that are not yet installed in the R program. The new library needs to be installed only once, after that it should only be activated. Basic libraries (e.g. `base`, `stats`) are installed and activated automatically.

help()

Help in using functions and commands.

```
help(read.table)
```

It explains how each function or command works and how it should be used. Help can also be accessed by clicking F1 when the name of the function is selected with the mouse in R Markdown.

read.table()

Reading a .csv data file.

```
mydata <- read.table("./data.csv",
                     header = TRUE,
                     sep = ";",
                     dec = ",")
```

Reading a .csv file that contains variable names in the first line. We specify how the variables are separated (`sep`) and the type of decimal separator (`dec`).

read_xlsx()

Reading an Excel file with data.

```
library(readxl)
mydata <- read_xlsx("./data.xlsx")
mydata <- as.data.frame(mydata)
```

Reading files saved in Excel format. To use the function, we need to install and activate the `readxl` library. When we import data, we also use the `as.data.frame` function to convert the object (table with data) into data.frame format.

data.frame()

Manual data entry creating a data table.

```
mydata <- data.frame("Age" = c(30, 25, 50),
                     "Height" = c(180, 170, 175))
```

We have created a data table with three units and two variables.

c()

Creating a vector of elements.

```
c(4, 5, 10)
c("M","M","F")
```

If you specify more than one element, write `c` before the parenthesis. Letters/texts must always be enclosed in quotation marks.

sort()

Sorting values by size.

```
sort(mydata$Age, decreasing = FALSE)
```

We sort the values of the variable in ascending or descending order.

factor()

Creating factor.

```
factor(mydata$Gender,
       levels = c(1, 0),
       labels = c("M", "F"))
```

Categorical variables, whose values are usually encoded with numeric codes, are converted into a factor. Only then will they be treated correctly as categorical variables by the R program. In the argument `levels` we write down the codes with which the variable is entered in the data table, and in the argument `labels` we write down what these codes mean.

```
mydata$GenderF <- factor(mydata$Gender,
                         levels = c(1, 0),
                         labels = c("M", "F"))
```

We have created a new variable GenderF and stored it in an existing data table.

head()
print()
tail()

Display a table of data.

```
head(mydata)
print(mydata)
tail(mydata)
```

We display the first 6 rows of the data table (`head`), the entire table (`print`) or the last 6 rows of the data table (`tail`).

round()
floor()
ceiling()

Rounding values.

```
round(mydata, 2)
```

The values in the data table are rounded to two decimal places.

```
floor(214.8)
```

Round the value down to the first integer. The result of the function in this particular case is 214.

```
ceiling(214.8)
```

Round the value up to the first integer. The result of the function in this particular case is 215.

ifelse()

Conditional command.

```
ifelse(mydata$Age > 22,
       yes = 1,
       no = 0)
```

A function is used to execute a conditional command. In the concrete example, the result of the function is a vector of values, where people older than 22 years have the value 1, otherwise 0.

```
ifelse(mydata$Age >= 22,
       yes = 1,
       no = 0)
```

A function is used to execute a conditional command. In the concrete example, the result of the function is a vector of values, where people older or equal to 22 years have the value 1, otherwise 0.

seq()

A sequence of values.

```
seq(from = 0, to = 100, by = 20)
```

The function is used to create a vector of values with a constant difference between them. In the concrete case, the result is `c(0, 20, 40, 60, 80, 100)`.

log()
exp()

Calculation of logarithm and exponential function (anti-logarithmization).

```
mydata$lnHeight <- log(mydata$Height)
mydata$lnHeight <- log(podatki$Height, base = 10)
```

Calculation of the logarithm. In the first case we calculate the natural logarithm, in the second case the decimal logaritm.

scale()

Standardization

```
mydata$Height_z <- scale(mydata$Height)
```

Standardization of the individual variables. In the concrete case, we assign the new variable to the existing data table.

```
mydata_std <- scale(mydata)
```

Data table standardization.

sqrt()

Square root.

```
sqrt(mydata$Height)
```

Calculation of root values.

aggregate()

Aggregation.

```
aggregate(mydata$Height, by = list(mydata$Gender),
          FUN = sum)
```

Aggregation of a variable with respect to a categorical variable. In this particular case, we add up all the heights of the individuals for each gender separately. Enter any function (`sum`, `mean` etc.) in the argument `FUN` to calculate the parameter of interest.

recode()

Value recoding.

```
library(car)
recode(mydata$Height,
       "150:160 = 155; 160:170 = 165; 170:180 = 175; 180:190 = 185")
```

Recoding a range of values to a new value. In this particular case, all values between 150 and 160 cm are recoded to 155 cm, and so on. (Threshold values are recoded to the value which they first fall according to the written code; in this particular case, the value 160 is recoded to the value 155, not 165. If we were to write the code in the order "160:170 = 165; 150: 160 = 155...", the value 160 would be recoded to the value 165). To use the function, we need to install and activate the library `car`.

rep()

Reproduction of value.

```
mydata[rep(1:nrow(mydata), times = mydata$Frequency), ]
```

With the function we reproduce each row in the data table as many times as the value of frequency (argument `times`). The function is used, for example, when we have data in sorted form (freqency table – aggregated data) and we need to convert it to unsorted form – raw data.

replace_with_na() — Replacing the value with NA.

```
library(dplyr)
library(naniar)
mydata <- mydata %>%
      replace_with_na(replace = list(parliament = c(77, 88, 99),
                                     police = c(77, 88, 99),
                                     politics = c(77, 88, 99)))
```

Replace numeric codes representing missing values with the value NA. To use the function, we need to install and activate the library `dplyr` and `naniar`.

drop_na() — Removing units with missing values.

```
library(tidyr)
mydata <- drop_na(mydata)
```

Removing units that have a missing value for any of the variables. To use the function, we need to install and activate the library `tidyr`.

order() — Sorting units by values.

```
print(mydata[order(-mydata$Points, mydata$Time), ])
```

We display a table of data sorted in descending order by the number of points scored, and units with the same number of points scored are sorted in ascending order by writing time

DESCRIPTIVE STATISTICS

mean() median() sd() var() range() min() max() sum() — Estimation of statistical parameters: mean, median, standard deviation, variance, range, minimum, maximum, sum.

```
mean(mydata$Age)
median(mydata)
```

Evaluation of a single parameter for a selected variable or the entire table.

quantile() — Estimation of quantile.

```
quantile(mydata$Age, p = 0.35)
```

Using the argument `p`, which determines the rank of the quantile, we can estimate any quantile. In this particular case, we estimate the 35th centile.

summary() — Estimation of parameters.

```
summary(mydata)
```

The function provides an estimate of the minimum, first quartile, mean, median, third quartile, and maximum. In the case of variables defined as a factor, the function gives the frequency of each category.

stat.desc() — Estimation of parameters.

```
library(pastecs)
stat.desc(mydata)
round(stat.desc(mydata), 1)
```

The function provides a wide range of parameter estimates. The values are given to 7 decimal places, so it is useful to combine the function with the function `round`. To use the function, we need to install and activate the `pastecs` library.

describe() — Estimation of parameters.

```
library(psych)
describe(mydata)
```

The function gives a wide range of parameter estimates. To use the function, we need to install and activate the `psych` library.

describeBy() — Estimation of parameters, seperated by groups.

```
library(psych)
describeBy(mydata$Age, group = mydata$Gender)
```

The function provides a wide range of parameter estimates separated by categories (groups). To use the function, we need to install and activate the `psych` library.

sapply() — Estimation of selected parameter.

```
sapply(mydata, FUN = mean)
sapply(mydata, FUN = quantile, probs = c(0.01, 0.99))
```

The function gives estimates of the selected parameter for all variables located in the data table.

table() — The number of repetitions of single value.

```
table(mydata$Gender)
```

The function specifies the number of repetitions of a single value for the selected variable.

nrow()
length() — Number of rows. The length of the vector.

```
nrow(mydata)
length(mydata$Age)
```

The number of rows in the data table or length of the vector (variable).

frq() — Frequency table.

```
library(sjmisc)
table(mydata$Height)
```

Frequency table for the selected numerical variable. To use the function, we need to install and activate the `sjmisc` library.

STATISTICAL DISTRIBUTIONS

pnorm() — Calculation of the area under the normal distribution curve.

```
pnorm(q = 5, mean = 3, sd = 2, lower.tail = TRUE)
```

The function gives the area under the probability density curve for the normal distribution to the left of the selected value of the variable (`lower.tail = TRUE`) or to the right of the selected value (`lower.tail = FALSE`).

qnorm() — Calculate the value of the variable for the selected area for the normal distribution.

```
qnorm(p = 0.30, mean = 3, sd = 2, lower.tail = TRUE)
```

The function gives the value of the variable for the selected area under the probability density curve for a normal distribution. The selected area can be located to the left (`lower.tail = TRUE`) or to the right of the calculated limit value (`lower.tail = FALSE`).

rnorm() — Generating random numbers from a given normal distribution.

```
rnorm(n = 20, mean = 3, sd = 2)
```

We generate a selected number (`n`) of random values from a given normal distribution.

dnorm() — Calculation of the density for the normal distribution at the selected value of the variable.

```
dnorm(x = 2.5, mean = 3, sd = 2)
```

We calculate density function at the selected value of the normal distribution.

qt() — Calculation of the value of the variable for the selected area for the *t*-distribution.

```
qt(p = 0.05, df = n-1)
```

The function gives the value of the variable in the selected area under the density curve for the *t* distribution at the selected degrees of freedom (`df`).

SAMPLING

choose() — Number of combinations without repetition.

```
library(combinat)
choose(6, 4)
```

From a population of size 6, we select all possible samples of 4 units without repetition. In this particular case, it is $\binom{6}{4} = 15$. To use the function, we need to install and activate the library `combinat`.

combn() — Print of all possible samples.

```
library(combinat)
combn(mydata, 4)
```

Print the values of the variable for each sample. To use the function, we need to install and activate the library `combinat`.

```
combn(mydata, 4, mean)
```

Output of mean estimates for all possible samples.

VIZUALIZATION OF DATA

hist() — Histogram.

```
hist(mydata$Age,
     main = "Distribution of age",
     ylab = "Frequency",
     xlab = "Age",
     breaks = seq(from = 20, to = 50, by = 5))
```

We display the data with a histogram. The `main` argument gives the title of the graph, `ylab` and `xlab` give the axis names, and the width of the columns is given by the `breaks` argument.

boxplot() — Boxplot.

```
boxplot(mydata$Age)
```

We plot the data using a boxplot. It consists of a rectangle defining the first, second, and third quartiles and a vertical line defines the minimum and maximum of the variables under study. Potential outliers are marked with circles.

ggplot() — Tool for graphical representation of data.

```
library(ggplot2)
ggplot(mydata, aes(x = Age)) +
  geom_histogram(binwidth = 5, colour = "gray") +
  ylab("Frequency")
```

```
library(ggplot2)
ggplot(mydata, aes(y = Age)) +
  geom_boxplot()
```

```
library(ggplot2)
ggplot(mydata, aes(y = Height, x = Age)) +
  geom_point()
```

We select the table with the data, and with the argument `aes` we specify which variable we draw on each axis. Then we build the chart using + signs. The argument `geom_` specifies the chart type. To use the function, we need to install and activate the `ggplot2` library. A more detailed explanation can be found in the document ggplot2.pdf.

scatterplot() — Scatterplot.

```
library(car)
scatterplot(mydata$Height ~ mydata$Age,
            smooth = FALSE,
            boxplots = FALSE,
            regLine = FALSE,
            ylab = "Height in cm", xlab = "Age in years")
```

Scatter plot for a pair of numerical variables. The first variable is on the y-axis, the second on the x-axis. With the arguments `smooth = FALSE` we turn off smoothing, with the argument `boxplots = FALSE` we turn off the display of the boxplot on the axes, with the argument `regLine = FALSE` we turn off the linear regression function, and with the arguments `ylab` and `xlab` we name the axes. To use the function, we need to install and activate the library `car`.

```
library(car)
scatterplot(mydata$Height ~ mydata$Age | mydata$Gender,
            smooth = FALSE,
            boxplots = FALSE,
            ylab = "Height in cm", xlab = "Age in years")
```

Scatter plot, separated by categorical variable.

scatterplotMatrix() — Matrix of scatterplots.

```
library(car)
scatterplotMatrix(mydata, smooth = FALSE)
```

Matrix of scatter plots for each pair of numeric variables. Non-numeric variables must be excluded from the analysis. To use the function, we need to install and activate the library `car`.

HYPOTHESIS TESTING

t.test() — Hypothesis about the arithmetic mean.
Hypothesis about the difference between two arithmetic means.

```
t.test(mydata$Age,
       mu = 25,
       alternative = "two.sided")
```

Testing of the hypothesis about the value of the population arithmetic mean. The value of the null hypothesis is entered into the `mu` argument, and the `alternative` argument is used to determine whether it is a two sided test (`"two.sided"`) or one sided test: (`"less"`) or (`"greater"`).

```
t.test(mydata$Weight1, mydata$Weight2,
       paired = TRUE,
       alternative = "less")
```

Testing of the hypothesis about the difference betwen two arithmetic means for the dependent samples (`paired = TRUE`).

```
t.test(mydata$Height ~ mydata$Gender,
       paired = FALSE,
       var.equal = FALSE,
       alternative = "greater")
```

Testing of the hypothesis about the difference between two arithmetic means for independent samples (`paired = FALSE`). With the argument `var.equal = FALSE` we choose the Welch correction for the possible difference of the group variances.

anova_test() — Analysis of variance for the dependent samples, rANOVA.

```
library(rstatix)
anova_test(dv = Trust,
           wid = ID,
           within = System,
           data = mydata_long)
```

We define the dependent variable (`dv`), we define the unit identifier (`wid`), we define the factor indicating what the dependent variable measures (`within`), and the data must be in a long format. To use the function, we need to install and activate the `rstatix` library.

aov() — Analysis of variance for the independent samples, ANOVA.

```
aov(Salary ~ Country,
    data = mydata)
```

We define the dependent variable and the factor that separates the units into groups.

prop.test() — Hypothesis about the proportion.
Hypothesis about the equality of two proportions.

```
prop.test(x = 150, n = 250,
          p = 0.50,
          correct = FALSE,
          alternative = "two.sided")
```

We determine the number of events (`x`) and the number of all trials (`n`). Using the argument `p`, we determine the null hypothesis, exclude the correction, and determine the alternative hypothesis.

```
prop.test(c(195, 45), c(234, 89),
          correct = TRUE,
          alternative = "two.sided")
```

Testing of the hypothesis about the equality of two proportions. In the specific case, we check whether the proportion 195/234 is statistically different from the proportion 45/89.

shapiro.test() — Hypothesis about whether the variable is normally distributed.

```
shapiro.test(mydata$Difference)
```

We use the Shapiro-Wilk test to check whether the variable is normally distributed.

pairwise.t.test() — Comparison of all pairs of arithmetic means with t-tests.

```
pairwise.t.test(mydata$Salary, g = mydata$Country,
                paired = FALSE,
                p.adj = "bonf")
```

Testing the hypothesis about the difference of two arithmetic means for dependent samples (`paired = TRUE`) or independent samples (`paired = FALSE`) for each pair of variables. A Bonferroni correction is often used when calculating *p*-values.

cohens_d()
interpret_cohens_d() — Effect size, determined using Cohen's D statistic.

```
library(effectsize)
cohens_d(mydata$Age, mu = 25)
interpret_cohens_d(0.52, rules = "sawilowsky2009")
```

We determine the effect size and then interpret it based on various rules. To use the function, we need to install and activate the `effectsize` library.

wilcox_test() — Wilcoxon test for median.

Wilcoxon test for equality of locations of two distributions for dependent and independent samples.

```
wilcox_test(mydata$Age,
            mu = 23,
            correct = FALSE)
```

The Wilcoxon test for median (nonparametric test) is calculated by determining the median (`mu`) and excluding the correction.

```
wilcox_text(mydata$Weight1, mydata$Weight2,
            paired = TRUE,
            correct = FALSE, exact = FALSE,
            alternative = "two.sided")
```

The Wilcoxon test for equality of two distribution locations for dependent samples (Wilcoxon signed rank test - nonparametric test) is computed by specifying both variables and chosing the argument `paired = TRUE`. We specify the alternative hypothesis and we exclude the correction and the calculation of the exact *p*-value.

```
wilcox_text(mydata$Height ~ mydata$Gender,
            paired = FALSE,
            correct = FALSE, exact = FALSE,
            alternative = "less")
```

The Wilcoxon test about the equality of two distribution locations for independent samples (Wilcoxon rank sum test - nonparametric test) is calculated by specifying the dependent variable and the factor that determines which group the observation refers to. We specify the argument `paired = FALSE` and the alternative hypothesis. The correction and the calculation of the exact *p*-value are excluded.

binom.test() — Binomial test (proportion test).

Sign test.

```
binom.test(x = 150, n = 250,
           p = 0.70,
           alternative = "two.sided")
```

We determine the number of events (`x`) and the number of all trials (`n`). Using the argument `p`, we determine the null hypothesis and specify the alternative hypothesis. If `p = 0.50`, we perform a sign test.

friedman_test() — Friedman ANOVA.

```
library(rstatix)
friedman_test(Trust ~ System | ID,
              data = mydata_long)
```

We define a dependent variable and a factor that indicates what the dependent variable measures, and then an identifier that assigns the measurement to an individual unit. The data must be in long format. This is a nonparametric test. To use the function, we need to install and activate the `rstatix` library.

kruskal.test() — Kruskal-Wallis Rank Sum test.

```
kruskal.test(Salary ~ Country,
             data = mydata)
```

We define the dependent variable and the factor that separates the units into groups. This is a nonparametric test.

summarySE() — Calculation of the parameters of the confidence intervals for the arithmetic mean by groups.

```
library(Rmisc)
summarySE(mydata,
          measurevar = "Height",
          groupvars = "Gender",
          conf.interval = 0.95)
```

We define data containing a numeric variable for which we calculate a confidence interval for the arithmetic mean (`measurevar`), and a categorical variable that divides the values of the variable into groups (`groupvars`). To use the function, we need to install and activate the `Rmisc` library.

VarTest() — Hypothesis about the value of the variance. Hypothesis about the ratio of two variances.

```
library(DescTools)
VarTest(mydata$Height,
        sigma.squared = 5,
        alternative = "two.sided")
```

Testing the hypothesis about the value of the variance. With the argument `sigma.squared` we determine the value in the null hypothesis, and with the `alternative` argument we determine whether it is a two-sided hypothesis (`"two.sided"`), a hypothesis that is directed to the left (`"less"`) or to the right (`"greater"`). To use the function, we need to install and activate the `DescTools` library.

```
library(DescTools)
VarTest(mydata$Height ~ mydata$Gender,
        alternative = "two.sided")
```

Hypothesis about the assumption of equality of two variances (whether their ratio is equal to 1). In the specific case, we check whether the height variances differ between the sexes, and the `alternative` argument is used to determine the alternative hypothesis To use the function, we need to install and activate the `DescTools` library.

ANALYSIS OF CATEGORICAL VARIABLES

chisq.test() — Pearson χ^2-test. χ^2-test for given probabilities.

```
chisq.test(mydata$Gender, mydata$Place,
           correct = FALSE)
```

We use the test to check if there is a relationship between two categorical variables. If both variables have exactly two categories, we use the argument `correct = TRUE`.

```
chisq.test(x = c(150, 200, 350), p = c(0.30, 0.30., 0.40),
           correct = FALSE)
```

We test whether two distributions are the same. In the argument `p` we write the expected probabilities, which are used to determine the expected frequencies.

fisher.test() — Fisher's exact probability test.

```
fisher.test(mydata$Gender, mydata$Place)
```

We use the test to check if there is a relationship between two categorical variables.

CORRELATION AND REGRESSION ANALYSIS

cor()
cor.test()

Estimation of the correlation coefficient. Hypothesis about the value of the correlation coefficient.

```
cor(mydata$Age, mydata$Height,
    method = c("pearson", "spearman"))
```

Estimation of the correlation coefficient between two numerical variables. Either Pearson's or Spearman's correlation coefficients can be calculated.

```
cor(mydata)
```

Estimation of correlation matrix. The data table must contain only numerical variables.

```
cor.test(mydata$Age, mydata$Height,
         method = c("pearson", "spearman"))
```

Hypothesis about the value of the correlation coefficient.

rcorr()

Correlation matrix

```
library(Hmisc)
rcorr(as.matrix(mydata), type = c("pearson", "spearman"))
```

An estimate of the correlation matrix, containing either Pearson's or Spearman's correlation coefficients for each pair of numeric variables. *p*-values are also provided below the matrix to test the hypothesis about the value of the correlation coefficient. To use this function, the library `Hmisc` must be installed and activated.

pcor()

Estimation of the partial correlation coefficient.

```
library(ppcor)
pcor(mydata)
```

An estimate of the partial correlation coefficient for each pair of numeric variables in the data table. To use this function, the library `ppcor` must be installed and activated.

lm()

Linear regression model estimation (OLS method).

```
lm(Height ~ Age + Gender + Age:Gender,
   data = mydata)
```

Estimation of the linear regression function using the least squares method. The dependent variable is indicated to the left of the tilde (~), while the explanatory variables are indicated to the right, separated by the sign +. Interactions are included by indicating the sign between the variables `:`

```
fit <- lm(Height ~ Age + Gender,
          data = mydata)
summary(fit)
```

Showing the results of the estimated regression function.

glm()

Linear regression model estimation (ML method).

```
glm(Height ~ Age + Gender,
    data = mydata)
```

Estimation of the linear regression function using the maximum likelihood method. The dependent variable is indicated to the left of the tilde (~), while the explanatory variables are indicated to the right, separated by the sign +. Interactions are included by indicating the sign between the variables `:`

```
glm(Smoker ~ Age + Gender,
    family = binomial,
    data = mydata)
```

Estimation of a binary logistic regression function.

```
fit <- glm(Smoker ~ Age + Gender,
           family = binomial,
           data = mydata)
summary(fit)
```

Showing the results of the estimated regression function.

ols_test_breusch_pagan()

Breusch-Pagan heteroskedasticity test.

```
library(olsrr)
ols_test_breusch_pagan(fit)
```

Test for the presence of heteroskedasticity. To use the function, we need to install and activate the `olsrr` library.

shapiro.test() — Hypothesis if the variable is normally distributed.

```
shapiro.test(mydata$StdResiduals)
```

We use the Shapiro-Wilk test to check whether the variable is normally distributed.

vif() — Checking for multicollinearity.

```
vif(fit)
```

Checking the degree of correlation between explanatory variables.

lm.beta() — Estimation of standardized partial regression coefficients.

```
library(lm.beta)
lm.beta(fit)
```

Estimation of the standardized partial regression coefficients of the regression model. To use the function, we need to install and activate the library `lm.beta`.

anova() — Comparison of two regression models in terms of fit to data.

```
anova(fit1, fit2)
```

Statistical comparison of two regression models estimated by the least squares method.

```
anova(fit1, fit2, test = "Chi")
```

Statistical comparison of two regression models estimated by the maximum likelihood method.

lm_robust() — Linear regression model estimation with robust standard errors.

```
library(estimatr)
lm_robust(Height ~ Age + Gender,
          se_type = "HC1",
          data = mydata)
```

Estimation of the regression model with robust White's standard errors. The correction is applied when the assumption of homoskedasticity is violated. To use this feature, the library must be installed and activated `estimatr`.

lme() — Estimation of a multilevel linear regression model (ML method).

```
library(nlme)
fit <- lme(Stress ~ Age + GenderF + SizeF,
           random = ~ 1 | ID_hospital / ID_department,
           method = "ML",
           data = mydata)
```

Estimation of multilevel (hierarchical) regression model using the maximum likelihood method. We specify the regression model, and the argument `random` is used to determine the random regression constant. In a spceific case, this is determined at the hospital and department level. To use the function, we need to install and activate the `nlme` library.

PRINCIPAL COMPONENT ANALYSIS AND FACTOR ANALYSIS

cortest.bartlett() — Bartlett's test of sphericity.

```
library(psych)
cortest.bartlett(R, n = nrow(mydata))
```

Bartlett's test of sphericity of the correlation matrix. The input element is the value of the correlation matrix created with the `cor()` function. We also specify the number of units in the sample. To use the function, the `psych` library must be installed and activated

det() — Determinant of the correlation matrix.

```
det(R)
```

Determinant of the correlation matrix, with which we check the degree of correlation between the variables.

KMO() — Kaiser-Meyer-Olkin measure of sampling adequacy.

```
library(psych)
KMO(R)
```

Calculation of KMO statistics and individual MSA statistics to check the adequacy of each variable. To use this function, the library `psych` must be installed and activated.

PCA() | Principal Component Analysis.

```
library(FactoMineR)
library(factoextra)
PCA(mydata,
    scale.unit = TRUE,
    ncp = 3,
    graph = FALSE)
```

Applying the principal component analysis to selected numeric variables. With the argument `scale.unit = TRUE` we perform the normalization of the variables, with the argument `ncp` we determine the number of components. To use the function, we need to install and activate the libraries `FactoMineR` and `factoextra`.

```
pca <- PCA(mydata,
           scale.unit = TRUE,
           graph = FALSE)
```

```
get_eigenvalue(pca)
```

We show the eigenvalues of the principal components.

```
fviz_eig(pca,
         choice = "eigenvalue",
         addlabels = TRUE)
```

Eigenvalue chart.

```
fviz_pca_var(pca, repel = TRUE)
```

Graphic representation of the loadings of the principal components.

```
fviz_pca_biplot(pca)
```

Graphical representation of the position of the units in relation to the values of the first two principal components.

fa.parallel() | Parallel analysis.

```
library(psych)
fa.parallel(mydata,
            sim = FALSE,
            fa = c("pc", "fa"))
```

Parallel analysis to determine the number of principal components or factors to be retained in the final result. We include only numeric variables in the data table, exclude simulation, and use the argument `fa` to indicate whether we are performing principal component (`pc`) or factor (`fa`) analysis. To use the function, we need to install and activate the `psych` library.

corPlot() | Graphical representation of the correlation matrix.

```
library(psych)
corPlot(R)
```

Graphical representation of the correlation matrix. The input element is the value of the correlation matrix created by the `cor()` function. The colors indicate the strength of the correlation for each pair of variables. To use the function, we need to install and activate the `psych` library.

fa() | Factor Analysis.

```
library(psych)
library(GPArotation)
fa(mydata,
   covar = FALSE,
   nfactors = 3,
   fm = "minres",
   rotate = "oblimin",
   inpute = "mean")
```

Performing factor analysis for selected numerical variables. The argument `covar` is used to specify whether the analysis is performed on the covariance matrix (`covar = TRUE`) or on the correlation matrix (`covar = FALSE`), the argument `nfactors` is used to specify the number of factors, the argument `fm` is used to specify the factor method (recommended use of `minres` – minimum residual method), the argument `rotate` is used to specify the type of factorial rotation (oblique `oblimin` or orthogonal `varimax`). Missing values can be replaced by average values of variables (`inpute = "mean"`). To use this function, the libraries `psych` and `GPArotation` must be installed and activated.

```
factor <- fa(mydata,
             covar = FALSE,
             nfactors = 3,
             fm = "minres",
             rotate = "oblimin",
             inpute = "mean")
```

```
print.psych(factor,
            cut = 0.3,
            sort = TRUE)
```

We present the results of the factor analysis. Low factor loadings are not shown (`cut`), and factor loadings are shown in descending order (`sort`).

```
fa.diagram(factor)
```

Graphic representation of the factor model.

alpha() | Cronbach alpha.

```
library(psych)
alpha(mydata, check.keys = TRUE)
```

Calculation of Cronbach's alpha for selected numeric variables. The argument `check.keys = TRUE` ensures that all indicators point in the same direction. To use this function, the `psych` library must be installed and activated.

CLUSTERING

get_dist() | Calculation of distances between units.

```
library(factoextra)
get_dist(mydata, method = "euclidean")
```

Calculation of Euclidean distances between units. The distance type is specified with the argument `method`. The distances "`manhattan`" and "`minkowski`" are also commonly used. To use this function, the `factoextra` library must be installed and activated.

```
distances <- get_dist(mydata, method = "euclidean")
fviz_dist(distances)
```

Graphical representation of the distance matrix.

get_clust_tendency() | Calculation of Hopkins statistics.

```
library(factoextra)
get_clust_tendency(mydata,
                   n = nrow(mydata)-1,
                   graph = FALSE)
```

Calculation of Hopkins statistics, which is used to check whether the data are suitable for carrying out the clustering.

hclust() | Hierarchical clustering.

```
library(dplyr)
library(factoextra)
WARD <- mydata %>%
  get_dist(method = "euclidean") %>%
  hclust(method = "ward.D2")
```

Implementation of hierarchical clustering based on selected numerical variables. In the first step, the data is selected, then (`%>%`) the distances between the units are calculated (`get_dist`), and then hierarchical clustering (`hclust`) is performed based on these distances according to the selected algorithm (`method`). It is recommended to use Ward's algorithm, which is combined with squared Euclidean distance (`ward.D2`). To use the function, the libraries `dplyr` and `factoextra` must be installed and activated.

```
fviz_dend(WARD)
```

Representation of dendrogram (classification tree).

```
cutree(WARD, k = 3)
```

Clustering into a selected number of groups.

hkmeans() | K-means clustering.

```
library(factoextra)
hkmeans(mydata,
        k = 3,
        hc.metric = "euclidean",
        hc.method = "ward.D2")
```

Clustering by the k-means method, starting from the solution obtained by hierarchical clustering. We need to specify the number of groups (`k`) into which we want to group the units, and the arguments `hc.metric` and `hc.method` should match the selected algorithm for hierarchical sorting, based on which we decided on the number of groups. To use the function, we need to install and activate the `factoextra` library.

```
kmeans <- hkmeans(mydata,
                  k = 3,
                  hc.metric = "euclidean",
                  hc.method = "ward.D2")

fviz_cluster(kmeans)
```

Graphical representation of the clustering, with the first and second principal components shown on the axes.

TIME SERIES

lm()

Calculation of linear or exponential trend (OLS method).

```
lm(Overnights ~ t,
   data = mydata)
```

Linear trend function. The variable `t` is the time, defined as `1, 2, 3, …, N`.

```
lm(lnOvernights ~ t,
   data = mydata)
```

Exponential trend function. The variable `t` is the time, defined as `1, 2, 3, …, N`.

ts()

Determination of time series.

```
ts(mydata$Overnights,
   start = c(2020, 1),
   end = c(2022, 12),
   frequency = 12)
```

We define the variable as a time series. The `start` argument is used to determine the first observation (in the concrete case it is 2020, the first month), the `end` argument is used to determine the last observation (in the concrete case it is 2022, the last month), and the `frequency` argument is used to determine how many time units are within a period. In the concrete case, the data is given monthly, so a total of 36 observations.

```
overnights <- ts(mydata$Overnights,
                 start = c(2020, 1),
                 end = c(2022, 12),
                 frequency = 12)

plot(overnights,
     ylab = "Monthly number of overnights")
```

Graphical display of time series.

decompose()

Decomposition of time series.

```
decompose(Overnights,
          type = "multiplicative")
```

Breakdown of the time series into basic elements (trend, cyclical component, period, and irregular component). The variable we analyze must be the result of the `ts()` function. For the type of decomposition, we choose the multiplicative approach.

```
overnights <- ts(podatki$Overnights,
                 start = c(2020, 1),
                 end = c(2022, 12),
                 frequency = 12)

decomposition <- decompose(overnights,
                           type = "multiplicative")

plot(decomposition)
```

Graphical representation of a time series decomposition.