Data Analysis using R

R Basics

Sven Werenbeck-Ueding

25.09.2023

What is R?

> R is a free, not-commercial, open-source software language for statistical computing and graphics. It comes with a lot of advantages, especially for data analysis projects.

Origin

- `R` is based on the programming language `S` ('dialect of S'), which in turn is based on the core programming language `C`
- First version published by **R**oss Ihaka and **R**obert Gentlemen in 1993 (name is based on their first initials)
- Intention of `R`: creating an environment where one doesn't consciously think of programming

Ihaka and Gentleman (1996)

What is R?

> R is a free, not-commercial, open-source software language for statistical computing and graphics. It comes with a lot of advantages, especially for data analysis projects.

General Information

- R is case sensitive
- R and his basic statistical functions can easily be extended by packages
- Comprehensive R Archive Network (CRAN) is the main-platform for packages (extensions)
- Over 20,000 packages in 2021 (roughly 4,000 packages in 2013)

Open-Source and R-Core-Team (2023)

What is R?

> R is a free, not-commercial, open-source software language for statistical computing and graphics. It comes with a lot of advantages, especially for data analysis projects.

Advantages/Strengths

- Free software
- Open-source: contributions from a large active and vibrant community (huge functionality)
- Great data visualization options
- Interfaces with many other languages (e.g. LaTeX, C, Markdown, Python, Java, HTML, CSS)
- Runs on almost every platform (through `C++` even on the PlayStation)

Wickham and Grolemund (2016)

What is R?

> R is a free, not-commercial, open-source software language for statistical computing and graphics. It comes with a lot of advantages, especially for data analysis projects.

Need additional help?

- Use the built-in help system in `R` (`help("...")` and `?"..."`)
- Use online-forums like Stack Overflow or GitHub (developers are active here as well)
- Use the documentation files provided on the official `R`-Website (manuals, FAQs, books)

> To understand computations in R, two slogans are helpful: Everything that exists is an object. Everything that happens is a function call.
>
> Chambers (2014), Co-Founder of S

> Good coding style is like using correct punctuation. You can manage without it, but it sure makes things easier to read. [...] you really should use a consistent style. Good style is important because while your code only has one author, it'll usually have multiple readers. This is especially true when you're writing code with others.
>
> Wickham (2019)

Prerequisites

Download the appropiate versions of R and RStudio for your operating system

- R can be downloaded via the Comprehensive R Archive Network (CRAN)
- Best integrated development environment (IDE), i. e. where you write and compile your code, is RStudio and can be downloaded here

Basic Calculations

Math operators

- `+`: addition
- `-`: subtraction
- `*`: multiplication (elementwise!)
- `/`: division
- `^`: power operator (e.g. 2^3)

Commands

- `abs()`: absolute value
- `sqrt()`: square root
- `exp()`: exponential function
- `log()`: natural logarithm
- `factorial()`: factorial (e.g. 8!)
- `round( , digits = x)`: round to x digits

Note: These operations work on different types (e.g. scalars, vectors, matrices, ...)

Basic Calculations

```
3 + 5 * 2
```

```
## [1] 13
```

```
(3 + 5) * 2
```

```
## [1] 16
```

```
sqrt(8.33)
```

```
## [1] 2.886174
```

```
round(sqrt(8.33), 2)
```

```
## [1] 2.89
```

Caution: Decimal numbers use a dot instead of a comma (convenient in the US and in the UK)

R-Scripts

- Within R there are several file types to create
- R-Scripts are simple text files (`<file_name>.R`) with a collection of commands (and comments)
- Useful to store and document your statistical analyses
- A new R-Script can be generated by clicking on `File > New File > R Script`
- Run code from R scripts by highlighting the code snippet and typing `Ctrl + Enter`

Comments

- For better documentation, you can place comments in your R-Script using the symbol `#`
- Everything placed behind the `#` and belonging to the same line is treated as a comment

```
# Multiply two by five
2 * 5
```

```
## [1] 10
```

```
10 / 4 # Divide ten by four
```

```
## [1] 2.5
```

R-Packages

- R has a lot of built-in commands delivered by the installation of R
- Often, these built-in commands are not sufficient (or convenient) to conduct more complex statistical analyses
- Further commands can be used by installing specialized packages

R-Packages

- R has a lot of built-in commands delivered by the installation of R
- Often, these built-in commands are not sufficient (or convenient) to conduct more complex statistical analyses
- Further commands can be used by installing specialized packages

Installing

- Run `install.packages()` while your PC is connected to the internet
- Each package has to be installed **only once**

```
# Installing the package rio
install.packages("rio")
```

Loading

- If you want to use a command from a specific package you need to activate the package by running the command `library()`
- Packages need only to be activated **once per session**

```
library(rio)
```

Variables

- Variables can take all kinds of different content (vectors, matrices, data sets, regression, ...)
- Variables are created by the assignment operator <-
- When variables are assigned, they are shown in RStudio's environment pane
- Using = to assign variables is considered bad practice

```
# Assign value 4 to variable x
x <- 4

# Print x
x
```

```
## [1] 4
```

```
# Do some calculations with x
x * 2
```

```
## [1] 8
```

```
# Overwrite x
x <- 1
x
```

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

Vectors

- Vectors are the most important data type in R
- What we normally refer to as vector is an "atomic vector" in R: A vector of elements with the same type
  - Scalars in R are atomic vectors as well (length 1 vectors)
- R allows vectors to have elements of different types, a so-called "list"
  - More complex than atomic vectors: Each element can be **any** type
  - Elements can differ in length
  - May contain several atomic vectors of different lengths with different types

Atomic Vectors

- Four primary types:
  1. **Logical**: `TRUE` or `FALSE`
  2. **Integer**: Whole numbers followed by `L`, e. g. `1234L`
  3. **Double**: Decimal number defined by the floating point standard, e. g. `0.1234`
  4. **Character**: Strings surrounded by `"` or `'`, e. g. `"this is a string"` or `'this as well'`
- Integers and doubles are numeric with unique values `Inf`, `-Inf` and `NaN` (not a number)
- Generate vectors with `c(1, 2, ...)`

Atomic Vectors

- Four primary types:
  1. **Logical**: `TRUE` or `FALSE`
  2. **Integer**: Whole numbers followed by `L`, e. g. `1234L`
  3. **Double**: Decimal number defined by the floating point standard, e. g. `0.1234`
  4. **Character**: Strings surrounded by `"` or `'`, e. g. `"this is a string"` or `'this as well'`
- Integers and doubles are numeric with unique values `Inf`, `-Inf` and `NaN` (not a number)
- Generate vectors with `c(1, 2, ...)`

Operations

```
# Generate x and z
x <- c(5, 1, 9, 3)
z <- c(4, 9, 0, 4)
```

```
# Multiplication with scalar
2 * x
```

```
## [1] 10  2 18  6
```

```
# Elementwise multiplication
x * z
```

```
## [1] 20  9  0 12
```

Atomic Vectors

- Four primary types:
  1. **Logical**: `TRUE` or `FALSE`
  2. **Integer**: Whole numbers followed by `L`, e. g. `1234L`
  3. **Double**: Decimal number defined by the floating point standard, e. g. `0.1234`
  4. **Character**: Strings surrounded by `"` or `'`, e. g. `"this is a string"` or `'this as well'`
- Integers and doubles are numeric with unique values `Inf`, `-Inf` and `NaN` (not a number)
- Generate vectors with `c(1, 2, ...)`

Special Commands

```
# vector with 5 ones
rep(1, 5)
```

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

```
# sequence from 1 to 5
seq(1, 5)
```

```
## [1] 1 2 3 4 5
```

```
1:5
```

```
## [1] 1 2 3 4 5
```

Subsetting Vectors

- Sometimes we want to work with specific elements from a vector

    1. Subset via element indices

    2. Subset via logicals

- Specific elements can be extracted by enclosing the index with `[` and `]`

Subsetting Vectors

- Sometimes we want to work with specific elements from a vector
  1. **Subset via element indices**
  2. Subset via logicals
- Specific elements can be extracted by enclosing the index with `[` and `]`

```
# Elements 2
x[2]
```

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

```
# Elements 1 to 3
x[1:3]
```

```
## [1] 5 1 9
```

```
# All elements except the last one
x[-4]
```

```
## [1] 5 1 9
```

Subsetting Vectors

- Sometimes we want to work with specific elements from a vector
    1. Subset via element indices
    2. **Subset via logicals**
- Specific elements can be extracted by enclosing the index with `[` and `]`

```
# Subset with TRUE/FALSE
x[c(TRUE, FALSE, FALSE, TRUE)]
```

```
## [1] 5 3
```

Factors

- Vector that contains predefined values
- Used to represent categorical data
- Integer vectors with the following **attributes** (metadata of the vector):
  1. `class`: "factor"
  2. `levels`: Set of allowed values, sorted into increasing order
- Created by the `factor()` function
- Can be ordered when setting the `ordered` argument of `factor()` to `TRUE`

```
x <- factor(c("b", "a", "a", "b", "a"),
            levels = c("a", "b"))

class(x)
```

```
## [1] "factor"
```

```
levels(x)
```

```
## [1] "a" "b"
```

Missing Values

- R reserves `NA` for missing values ("not applicable")
- Most computations with missing values return missing values

```
NA * 10
```

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

- Exception: When identity holds for all inputs, for example when computing x^0

```
NA^0
```

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

- Use `is.na()` to test for missing values

```
x <- c(1, NA, 2, 3)

is.na(x)
```

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

Matrices

- Vectors can have **attributes** (some metadata of the vector)
- Adding the `dim` attribute to an atomic vector makes it a 2-dimensional matrix
- Can be created by assigning `dim()` to an atomic vector or simply calling `matrix()`
- Can also be created by binding atomic vectors with the same data type and length column-wise using `cbind()` or row-wise using `rbind()`

Matrices

- Vectors can have **attributes** (some metadata of the vector)
- Adding the `dim` attribute to an atomic vector makes it a 2-dimensional matrix
- Can be created by assigning `dim()` to an atomic vector or simply calling `matrix()`
- Can also be created by binding atomic vectors with the same data type and length column-wise using `cbind()` or row-wise using `rbind()`

```
# Add dim attribute to atomic vector
x <- 1:6

dim(x) <- c(3, 2) # 3 rows, 2 columns

x
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

Matrices

- Vectors can have **attributes** (some metadata of the vector)
- Adding the `dim` attribute to an atomic vector makes it a 2-dimensional matrix
- Can be created by assigning `dim()` to an atomic vector or simply calling `matrix()`
- Can also be created by binding atomic vectors with the same data type and length column-wise using `cbind()` or row-wise using `rbind()`

```
# Use the matrix command to create a
# matrix from a vector
matrix(x, nrow = 3)
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

```
# Specify ncol instead of nrow
matrix(x, ncol = 2)
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

Matrices

- Vectors can have **attributes** (some metadata of the vector)
- Adding the `dim` attribute to an atomic vector makes it a 2-dimensional matrix
- Can be created by assigning `dim()` to an atomic vector or simply calling `matrix()`
- Can also be created by binding atomic vectors with the same data type and length column-wise using `cbind()`or row-wise using `rbind()`

```
# Fill matrix column-wise instead of
# row-wise
matrix(x, ncol = 2, byrow = T)
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6
```

Matrices

- Vectors can have **attributes** (some metadata of the vector)
- Adding the `dim` attribute to an atomic vector makes it a 2-dimensional matrix
- Can be created by assigning `dim()` to an atomic vector or simply calling `matrix()`
- Can also be created by binding atomic vectors with the same data type and length column-wise using `cbind()` or row-wise using `rbind()`

```
# Bind atomic vectors column-wise
cbind(1:3, 4:6)
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

Matrix Operations

- `t()`: transpose
- `solve()`: inverse
- `%*%`: matrix multiplication

Matrix Operations

- `t()`: transpose
- `solve()`: inverse
- `%*%`: matrix multiplication

```
A <- matrix(c(1, 2, 0, 3), ncol = 2)

A
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    2    3
```

```
t(A)
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    0    3
```

```
solve(A)
```

```
##            [,1]      [,2]
## [1,]  1.0000000 0.0000000
## [2,] -0.6666667 0.3333333
```

Matrix Operations

- `t()`: transpose
- `solve()`: inverse
- `%*%`: matrix multiplication

```
B <- matrix(c(4, 7, 2, 6), ncol = 2)

B
```

```
##      [,1] [,2]
## [1,]    4    2
## [2,]    7    6
```

```
A %*% B
```

```
##      [,1] [,2]
## [1,]    4    2
## [2,]   29   22
```

Subsetting Matrices

- Specific elements can be extracted using `[` and `]`
- To distinguish the row from the column positions, a comma is used within the brackets **`[row, column]`**

```
C <- matrix(-5:6, ncol = 4)

C
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   -5   -2    1    4
## [2,]   -4   -1    2    5
## [3,]   -3    0    3    6
```

Subsetting Matrices

- Specific elements can be extracted using `[` and `]`
- To distinguish the row from the column positions, a comma is used within the brackets **`[row, column]`**

```
C[1, ]
```

```
## [1] -5 -2  1  4
```

```
C[, 1:3]
```

```
##      [,1] [,2] [,3]
## [1,]   -5   -2    1
## [2,]   -4   -1    2
## [3,]   -3    0    3
```

```
C[1:2, 2]
```

```
## [1] -2 -1
```

Note: Depending on the matrix and choice of subset, the subset may no longer be a matrix!

Useful Statistical/Mathematical Commands

Suppose `x` and `y` are numeric vectors with same length:

Command	Description
`sum(x)`	Sum of `x`
`table(x)`	Frequeny table for `x`
`mean(x)`	Average of `x`
`median(x)`	Median of `x`
`var(x)`	Variance of `x`
`sd(x)`	Standard deviation of `x`
`cov(x, y)`	Covariance matrix of `x` and `y`
`quantile(x)`	Quantiles of `x`
`min(x)`	Minimum value of `x`
`max(x)`	Maximum value of `x`

Note: Some of these commands can also be applied to matrices

Lists

- Each element can be **any** type
- Can be constructed using `list()`
- Just like with atomic vectors, elements in a list can be named
- Subsetting lists is done via double brackets `[[` and `]]` or `$`

```
x <- list(1:5, "a", c(TRUE, FALSE),
          seq(0.2, 0.6, by = 0.2))

typeof(x)
```

```
## [1] "list"
```

```
x
```

```
## [[1]]
## [1] 1 2 3 4 5
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1]  TRUE FALSE
##
## [[4]]
## [1] 0.2 0.4 0.6
```

Lists

- Each element can be **any** type
- Can be constructed using `list()`
- Just like with atomic vectors, elements in a list can be named
- Subsetting lists is done via double brackets `[[` and `]]` or `$`

```
x <- list(
  integer_vector = 1:5,
  character_vector = "a"
)

x
```

```
## $integer_vector
## [1] 1 2 3 4 5
##
## $character_vector
## [1] "a"
```

Lists

- Each element can be **any** type
- Can be constructed using `list()`
- Just like with atomic vectors, elements in a list can be named
- Subsetting lists is done via double brackets `[[` and `]]` or `$`

```
x$integer_vector
```

```
## [1] 1 2 3 4 5
```

```
x[[1]]
```

```
## [1] 1 2 3 4 5
```

Data Frames

- Used when dealing with tabular data
- Essentially a named list of vectors with
  - **same length** of all elements
  - attributes for column **`names`** and **`row.names`**
  - and **"data.frame" as the `class`**
- Created using the `data.frame()` function

```
df <- data.frame(x = 1:10, y = letters[1:10])

df
```

```
##      x y
## 1    1 a
## 2    2 b
## 3    3 c
## 4    4 d
## 5    5 e
## 6    6 f
## 7    7 g
## 8    8 h
## 9    9 i
## 10 10 j
```

Useful Commands

- `View(df)`: Opens `df` in a spreadsheet-style data viewer in RStudio
- `summary(df)`: Produces summary statistics for each variable
- `df$x`: Access variable with name `x`
- `head(df)`: Shows first six rows of `df`
- `tail(df)`: Shows last six rows of `df`
- `nrow(df)`: Number of rows (observations)
- `ncol(df)`: Number of columns (variables)
- `names(df)`: Column names

Useful Commands

- `View(df)`: Opens `df` in a spreadsheet-style data viewer in RStudio
- `summary(df)`: Produces summary statistics for each variable
- `df$x`: Access variable with name `x`
- `head(df)`: Shows first six rows of `df`
- `tail(df)`: Shows last six rows of `df`
- `nrow(df)`: Number of rows (observations)
- `ncol(df)`: Number of columns (variables)
- `names(df)`: Column names

```
summary(df)
```

```
##        x               y
##  Min.   : 1.00   Length:10
##  1st Qu.: 3.25   Class :character
##  Median : 5.50   Mode  :character
##  Mean   : 5.50
##  3rd Qu.: 7.75
##  Max.   :10.00
```

Useful Commands

- `View(df)`: Opens `df` in a spreadsheet-style data viewer in RStudio
- `summary(df)`: Produces summary statistics for each variable
- `df$x`: Access variable with name `x`
- `head(df)`: Shows first six rows of `df`
- `tail(df)`: Shows last six rows of `df`
- `nrow(df)`: Number of rows (observations)
- `ncol(df)`: Number of columns (variables)
- `names(df)`: Column names

```
df$x
```

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

Useful Commands

- `View(df)`: Opens `df` in a spreadsheet-style data viewer in RStudio
- `summary(df)`: Produces summary statistics for each variable
- `df$x`: Access variable with name `x`
- `head(df)`: Shows first six rows of `df`
- `tail(df)`: Shows last six rows of `df`
- `nrow(df)`: Number of rows (observations)
- `ncol(df)`: Number of columns (variables)
- `names(df)`: Column names

```
nrow(df)
```

```
## [1] 10
```

Useful Commands

- `View(df)`: Opens `df` in a spreadsheet-style data viewer in RStudio
- `summary(df)`: Produces summary statistics for each variable
- `df$x`: Access variable with name `x`
- `head(df)`: Shows first six rows of `df`
- `tail(df)`: Shows last six rows of `df`
- `nrow(df)`: Number of rows (observations)
- `ncol(df)`: Number of columns (variables)
- `names(df)`: Column names

```
names(df)
```

```
## [1] "x" "y"
```

Working with Data Frames

Create a new column:

```
df$z <- seq(0.1, 1, by = 0.1)

df
```

```
##     x y   z
## 1   1 a 0.1
## 2   2 b 0.2
## 3   3 c 0.3
## 4   4 d 0.4
## 5   5 e 0.5
## 6   6 f 0.6
## 7   7 g 0.7
## 8   8 h 0.8
## 9   9 i 0.9
## 10 10 j 1.0
```

Working with Data Frames

Select multiple columns by name:

```
df[, c("x", "z")]
```

```
##     x   z
## 1   1 0.1
## 2   2 0.2
## 3   3 0.3
## 4   4 0.4
## 5   5 0.5
## 6   6 0.6
## 7   7 0.7
## 8   8 0.8
## 9   9 0.9
## 10 10 1.0
```

Working with Data Frames

Select a subset of observations based on a condition:

- `df$x <= 5` evaluates to a vector of logicals with same length as the atomic vector `df$x`
- Logical vector selects rows to keep when subsetting
- Can of course also subset using a vector of indices

```
df[df$x <= 5,]
```

```
##   x y   z
## 1 1 a 0.1
## 2 2 b 0.2
## 3 3 c 0.3
## 4 4 d 0.4
## 5 5 e 0.5
```

Importing Data Sets

- Data sets are usually stored in specific file formats such as `.csv`, `.xlsx` or Stata's `.dta`
- Base R offers functions for the most common data storage formats
- For some formats we have to use packages such as `haven` for `.dta` files (or write functions ourselves)
- Import the `.csv` file stored under `data/processed/happiness.csv`, data from Wooldridge (2013)

```
df <- read.csv("data/processed/happiness.csv")
```

Note that...

... you can change the working directory by using the command `setwd("YOUR_PATH")` (recommended!)

... you can also change the working directory by clicking on `Session > Set Working Directory > Choose Directory`

... as a rule, `R` accepts "`/`" but does not accept "`\`" (which is often used on german PCs)

... if your working directory is specified you do not need to set the whole path for future operations (only the relative path)

Linear Regression Models in R

Let's build a linear probability model using the happiness data!

We are interested in analyzing how owning a gun has an effect on whether individuals see themselves as "very happy":

$$vhappy = \alpha + \beta_1 \times educ + \beta_2 \times income + \beta_3 \times owngun + \varepsilon,$$

where

- $vhappy = 1$ if the individual is very happy
- $educ$: Years of schooling completed
- $income$: Income brackets
- $owngun = 1$ if the individual owns a gun

Inspect the Data

```
summary(df[,c("vhappy", "educ", "income", "owngun")])
```

```
##      vhappy            educ          income            owngun
##  Min.   :0.0000   Min.   : 0.00   Length:17137       Length:17137
##  1st Qu.:0.0000   1st Qu.:12.00   Class :character   Class :character
##  Median :0.0000   Median :13.00   Mode  :character   Mode  :character
##  Mean   :0.3069   Mean   :13.32
##  3rd Qu.:1.0000   3rd Qu.:16.00
##  Max.   :1.0000   Max.   :20.00
##                   NA's   :44
```

- Contains `NA` ➡ need to be removed for the estimation
- `income` and `owngun` are of type character ➡ transform columns to correct format

Clean the Data

Remove missing values using `na.omit()`:

```
# Create a new data frame in the environment with a subset of columns used for the analysis
df_clean <- df[,c("vhappy", "educ", "income", "owngun")]

# Remove all rows containing NAs
df_clean <- na.omit(df_clean)

summary(df_clean)
```

```
##      vhappy           educ          income            owngun
##  Min.   :0.0000   Min.   : 0.00   Length:9969        Length:9969
##  1st Qu.:0.0000   1st Qu.:12.00   Class :character   Class :character
##  Median :0.0000   Median :13.00   Mode  :character   Mode  :character
##  Mean   :0.3074   Mean   :13.39
##  3rd Qu.:1.0000   3rd Qu.:16.00
##  Max.   :1.0000   Max.   :20.00
```

Clean the Data

Recode owngun to a dummy variable using `ifelse()`:

```
# Recode owngun to a dummy variable
df_clean$owngun <- ifelse(df_clean$owngun == "yes", 1, 0)

summary(df_clean)
```

```
##      vhappy            educ          income             owngun
##  Min.   :0.0000   Min.   : 0.00   Length:9969        Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:12.00   Class :character   1st Qu.:0.0000
##  Median :0.0000   Median :13.00   Mode  :character   Median :0.0000
##  Mean   :0.3074   Mean   :13.39                      Mean   :0.3692
##  3rd Qu.:1.0000   3rd Qu.:16.00                      3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :20.00                      Max.   :1.0000
```

Clean the Data

The `income` is given in the form of income brackets and stored as a character vector:

```
# Take a look at unique values of the income
unique(df_clean$income)
```

```
##  [1] "$15000 - 19999" "$10000 - 14999" "$25000 or more" "$20000 - 24999"
##  [5] "$6000 to 6999"  "$5000 to 5999"  "$3000 to 3999"  "$1000 to 2999"
##  [9] "$8000 to 9999"  "$7000 to 7999"  "$4000 to 4999"  "lt $1000"
```

➡ Convert to factor

Clean the Data

Convert `income` to an ordered factor:

```
df_clean$income <- factor(df_clean$income,
                          levels = c(
                            "lt $1000", "$1000 to 2999", "$3000 to 3999", "$4000 to 4999",
                            "$5000 to 5999", "$6000 to 6999", "$7000 to 7999", "$8000 to 9999",
                            "$10000 - 14999", "$15000 - 19999", "$20000 - 24999", "$25000 or more"
                          ),
                          ordered = T)

summary(df_clean)
```

```
##      vhappy            educ                    income        owngun
##  Min.   :0.0000   Min.   : 0.00   $25000 or more:6422   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:12.00   $20000 - 24999: 873   1st Qu.:0.0000
##  Median :0.0000   Median :13.00   $10000 - 14999: 813   Median :0.0000
##  Mean   :0.3074   Mean   :13.39   $15000 - 19999: 751   Mean   :0.3692
##  3rd Qu.:1.0000   3rd Qu.:16.00   $8000 to 9999 : 266   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :20.00   $7000 to 7999 : 153   Max.   :1.0000
##                                   (Other)       : 691
```

Linear Model: `lm()`

- The function `lm()` is used to fit (multivariate) linear models
- For the model defined before, we simply need:
  1. `formula`: `vhappy ~ educ + income + owngun`
  2. `data`: The `df_clean` data frame we prepared

```
model <- lm(vhappy ~ educ + income + owngun,
            df_clean)
```

Linear Model: `lm()`

```
model
```

```
##
## Call:
## lm(formula = vhappy ~ educ + income + owngun, data = df_clean)
##
## Coefficients:
## (Intercept)         educ     income.L     income.Q     income.C     income^4
##    0.103169     0.009402     0.046546     0.091012     0.054562     0.040568
##    income^5     income^6     income^7     income^8     income^9    income^10
##   -0.037144     0.068211    -0.013639    -0.020895    -0.007898     0.017806
##   income^11       owngun
##   -0.034506     0.031326
```

Model Summary: `summary()`

```
summary(model)
```

```
##
## Call:
## lm(formula = vhappy ~ educ + income + owngun, data = df_clean)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.4289 -0.3506 -0.2375  0.6212  0.9223
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.103169   0.022115   4.665 3.12e-06 ***
## educ         0.009402   0.001646   5.713 1.14e-08 ***
```

Extract Coefficients: `coef()`

```
coef(model)
```

```
##  (Intercept)         educ     income.L     income.Q     income.C     income^4
##  0.103168955  0.009401949  0.046545580  0.091012452  0.054561656  0.040568424
##     income^5     income^6     income^7     income^8     income^9    income^10
## -0.037143658  0.068210608 -0.013638591 -0.020894643 -0.007898189  0.017805968
##    income^11       owngun
## -0.034506034  0.031326131
```

Formulas

- Formulas have a special syntax in R and reference the column names given in the data
- LHS and RHS are separated by `~`
- Variables to be included in the model are separated by `+`
- R automatically assumes an intercept: To remove the intercept, add `-1` at the start or `+0` at the end of the RHS
- May include transformations of the data, e. g. `log()`
- `:` indicates interactions of variables
- `*` denotes factor crossings, i. e. `a*b` is the same as `a + b + a:b`
- `^` is used for polynomials, i. e. `a^2` translates to `a*a`

Formulas: Examples

Formula	Regression Model
`y ~ x1 + x2 + x3`	$y = \alpha + \beta_1 \times x_1 + \beta_2 \times x_2 + \beta_3 \times x_3 + \varepsilon$
`y ~ -1 + x1 + x2 + x3`	$y = \beta_1 \times x_1 + \beta_2 \times x_2 + \beta_3 \times x_3 + \varepsilon$
`y ~ log(x1) + x2 + x3`	$y = \alpha + \beta_1 \times \ln x_1 + \beta_2 \times x_2 + \beta_3 \times x_3 + \varepsilon$
`y ~ x1*x1 + x2 + x3`	$y = \alpha + \beta_1 \times x_1 + \beta_2 \times x_1^2 + \beta_3 \times x_2 + \beta_4 \times x_3 + \varepsilon$
`y ~ x1:x1 + x2 + x3`	$y = \alpha + \beta_1 \times x_1^2 + \beta_2 \times x_2 + \beta_3 \times x_3 + \varepsilon$
`y ~ x1 + x2*x3`	$y = \alpha + \beta_1 \times x_1 + \beta_2 \times x_2 + \beta_3 \times x_3 + \beta_4 \times x_2 \times x_3 + \varepsilon$

References

Chambers, J. M. (2014). "Object-Oriented Programming, Functional Programming and R". In: *Statistical Science* 29.3, pp. 167-180. DOI: 10.1214/13-STS452.

Ihaka, R. and R. Gentleman (1996). "R: A Language for Data Analysis and Graphics". In: *Journal of Computational and Graphical Statistics* 5.3, pp. 299-314. DOI: 10.1080/10618600.1996.10474713.

Open-Source and R-Core-Team (2023). *R Project. Open-source statistical software*. Result of a collaborative Effort with Contributions from all over the World. URL: https://www.r-project.org/.

Wickham, H. (2019). *Advanced R*. 2nd. Chapman & Hall/CRC. URL: http://adv-r.had.co.nz/.

Wickham, H. and G. Grolemund (2016). *R for data science. import, tidy, transform, visualize, and model data*. O'Reilly. URL: https://r4ds.had.co.nz/.

Wooldridge, J. M. (2013). *Introductory Econometrics: A Modern Approach*. 5th. Cengage Learning.