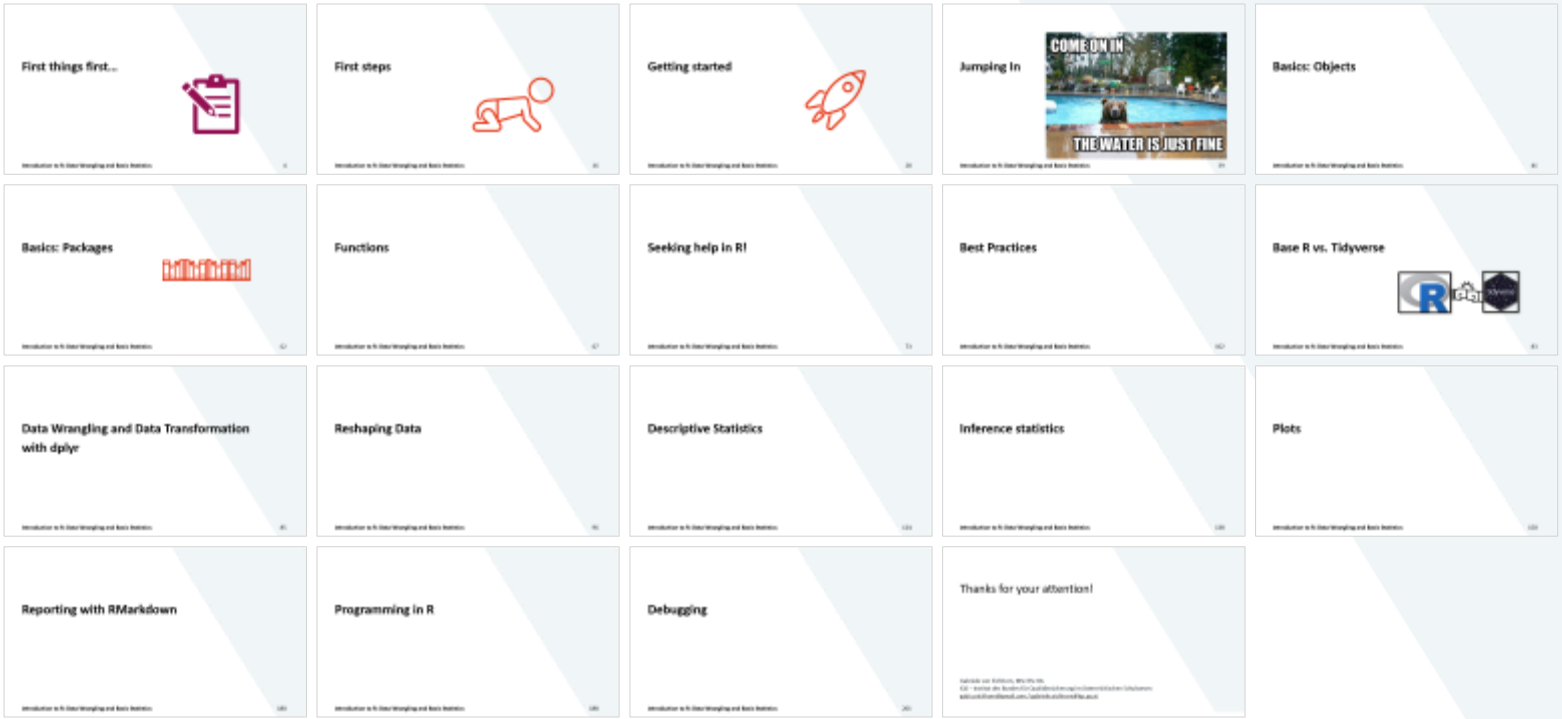
STRANGER THINGS

Getting started with R

Data Wrangling and Basic Principles

Gabriele von Eichhorn, MSc BSc BA
Elisabeth Rothe, Dipl.-Psych.
IQS – Institut des Bundes für Qualitätssicherung im österreichischen Schulwesen
RLadies, 24.04.2024


First things first...
First steps
Getting started
Jumping In
COME ON IN
THE WATER IS JUST FINE
Basics: Objects
Basics: Packages
Functions
Seeking help in R!
Best Practices
Base R vs. Tidyverse
Data Wrangling and Data Transformation with dplyr
Reshaping Data
Descriptive Statistics
Inference statistics
Plots
Reporting with RMarkdown
Programming in R
Debugging
Thanks for your attention!

First things first…

Who are we?

- Elisabeth Rothe
- Diploma (MSc) in Psychology in Munich
- Psychometrician at BIFIE/IQS since march 2018
- During my studies a seminar with R (still without RStudio)
- Experience with other programming languages
- Learning on the job (from other people, scripts, books, internet…)

- Gabriele von Eichhorn (Gabi)
- MSc in Psychology and BA in Education Sciences in Salzburg
- psychometrician at BIFIE/IQS since march 2018
- Introductory course to R in my master's studies → analysed my theses with R → some more introductory courses → learning by doing at IQS

Materials

- https://github.com/squirrellina/Getting-started-with-R

small survey

https://www.menti.com/j4xzhwprkt

menti.com → Code 6308 9997

Survey (I)

- How would you judge your previous experience with:
  - **Statistics**
    - 1: descriptive statistics –
    - 3: linear regressions –
    - 5: multidimensional multilevel models
  - **Programming**
    - 1: formulas in Excel –
    - 3: SPSS syntax –
    - 5: R/Python/SAS/Stata
  - **R / RStudio**
    - 1: never opened on own account –
    - 3: opened before, have executed ready-made code –
    - 5: have written code by myself

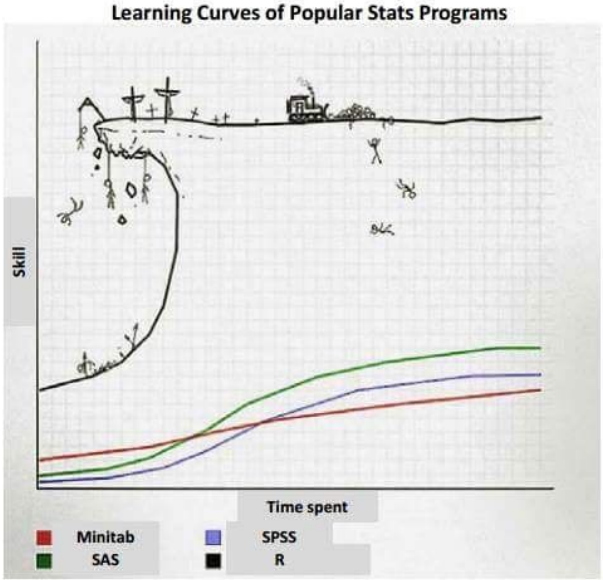
Survey (II)

- I want to learn / I am especially interested in…
  1. Data Wrangling / Pre-Processing
  2. Descriptive Statistics
  3. Inference Statistics
  4. Plots and Visualisation
  5. Reporting (with RMarkdown/Quarto)
  6. Web Applications with Shiny
  7. Programming & Writing Custom Functions
  8. Debugging
  - …

Workshop Goals (specific and abstract)

- You…
  - know RStudio
  - understand objects and functions in R and how they look
  - know different types of objects and data structures
  - can read and edit data from different common formats
  - can wrangle data for the most common use cases
  - can look at basic descriptive statistics
  - have a basic representation of two R dialects (base and tidyverse)

- We want…
  - to give a general insight into the R language and RStudio as a GUI
  - to motivate – show possibilities – put aside reservations
  - to provide tools, tips and tricks for learning
  - to build up resources for self-help to avoid situations like in the cartoons ☺

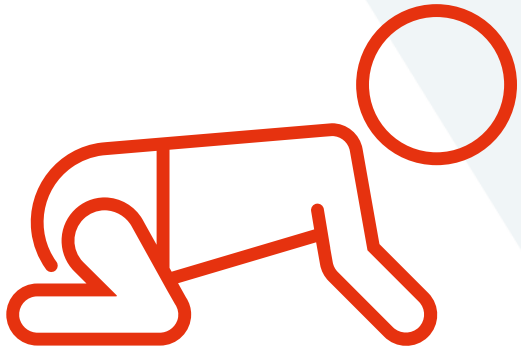

Learning Curves of Popular Stats Programs
Skill
Time spent
Minitab
SAS
SPSS
R


Good Times
Bad Times
I love you R!
...ok
JUST DO THE DAMN T-TEST
...no

Info

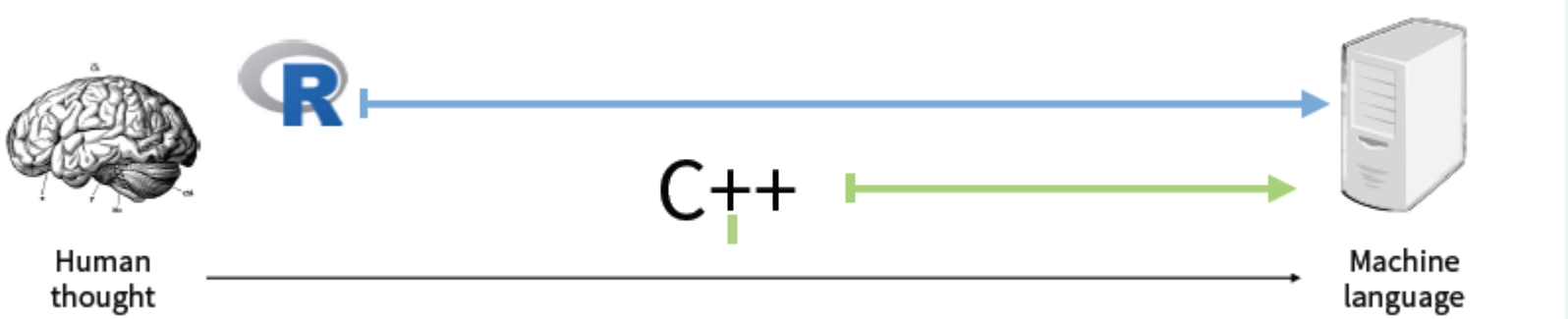
- Workshop is mostly derived from:
  - „R in 10 Schritten – Einführung in die statistische Programmierumgebung“ von Rainer W. Alexandrowicz
  - „R in Action“ von Rob Kabacoff
  - „YaRrr! The Pirate‘s Guide to R“ von Nathaniel D. Phillips

First steps



What is R?

- a programming language and programming environment for data analysis and graphics
- an open source software



Graphik from McNamara (2019)

Why R? Advantages


Open Source


100% free


active community, great support system


flexible & versatile, all in one solution


expandable

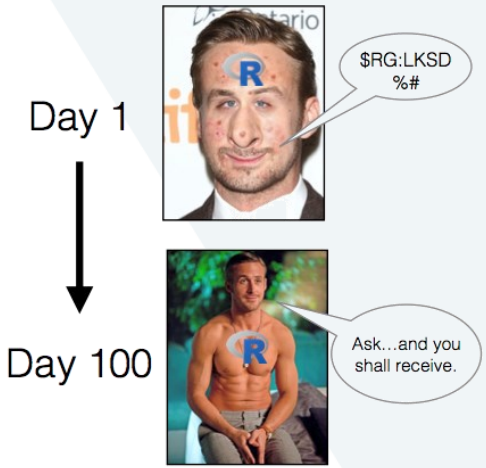

reproducible & transparent

R kann alles (… *man muss nur wissen wie*) ☺

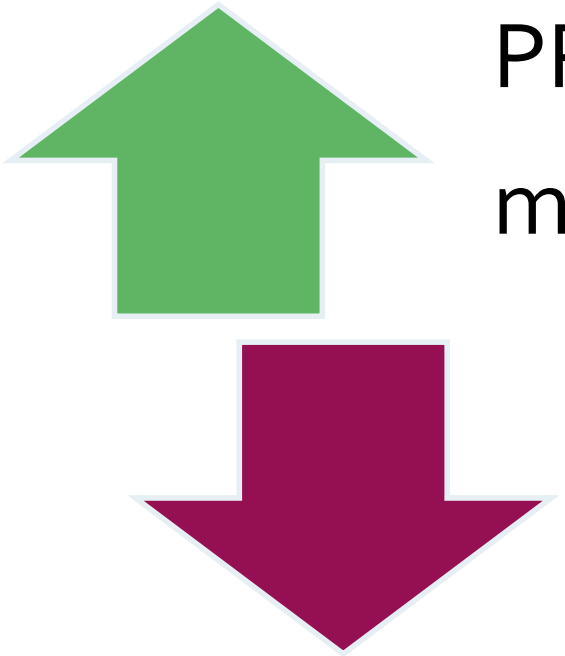
R can do everything (… *you just have to know how*) ☺

Why not R?

- There‘s no guarantee for correctness: *R is free software and comes with ABSOLUTELY NO WARRANTY*. → BUT: standard packages are developed by experts and are sufficiently tested. Source code can be viewed, which is not always the case with commercial software.
- Learning R requires time and a basic motivation for programming. → BUT: You learn basic logic and calculation steps of statistical methods.
- Data input is possible in R, but not recommended. → BUT: Data can be read in effortlessly from e.g. Excel.
- R is an interpreter language (code is not compiled), which can lead to loss of speed with large amounts of data. → BUT: Interface to C/C++ allows efficient programming (and speed compared to SPSS is blazing...)
- R does not provide a GUI (Graphical User Interface). → BUT: If you want one, there are some packages that provide one (or you start using Jamovi/JASP which are GUIs that are similar to SPSS but based on R and open source)



In summary



PRO:

more freedom

CON:

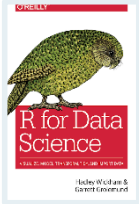
more freedom

R literature (free)

Nathaniel D. Phillips „YaRrr! The Pirate's Guide to R"

https://bookdown.org/ndphillips/YaRrr/

very playful and easy to read, good exercises



Hadley Wickham und Garrett Grolemunds „R 4 Data Science"

https://r4ds.had.co.nz/

all in one with Tidyverse

Robert I. Kabacoff „R in Action"

https://www.manning.com/books/r-in-action

1. edition should hopefully still be free

datacamp

Datacamp

https://www.datacamp.com/

Hands-on, guided exercises in the browser (introductory courses are free of charge, otherwise it's a subscription service)

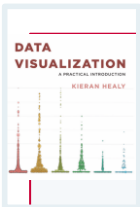
Auf Deutsch:

Wikibooks GNU R

https://de.wikibooks.org/wiki/GNU_R

Statistik mit R und RStudio

https://www.produnis.de/R/index.html



Kieran Healys „Data Visualization - A practical introduction"

http://socviz.co/

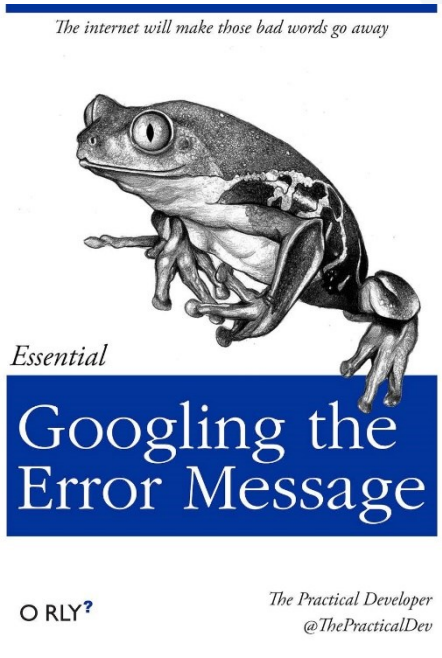
detailed introduction to R and ggplot2

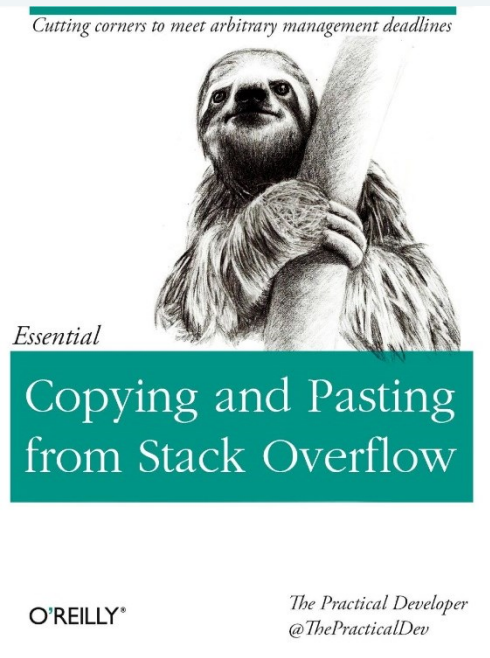
Reference books

- R Reference Card (https://cran.r-project.org/doc/contrib/Short-refcard.pdf)
- RStudio Cheatsheets (https://www.rstudio.com/resources/cheatsheets/)
- Quick-R (https://www.statmethods.net/)
- R Cookbook (http://www.cookbook-r.com/)
- Tidyverse Style Guide (https://style.tidyverse.org/) / Google's R Style Guide (https://google.github.io/styleguide/Rguide.xml)
- R Markdown (https://bookdown.org/yihui/rmarkdown/)
- Programming: “Advanced R” (http://adv-r.had.co.nz/)
- Packages: “R Packages” (http://r-pkgs.had.co.nz/)
- Text Mining: “Text Mining with R” (https://www.tidytextmining.com/)

more specific

- for specific problems:
  - documentation (https://www.rdocumentation.org/)
  - search engine (https://rseek.org/)
  - blogs (https://www.r-bloggers.com/)
  - StackOverflow (https://stackoverflow.com/)

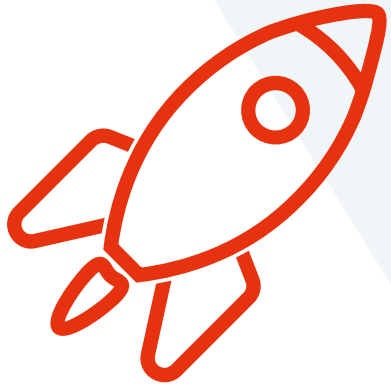




Resources to get inspired

- https://www.r-graph-gallery.com/ (plots + code)
- https://rmarkdown.rstudio.com/gallery.html (R Markdown documents in various output formats and designs)
- http://shiny.rstudio.com/gallery/ (various Shiny web applications + code)
- https://twitter.com/hashtag/rstats (what's happening in the community? new packages, visualizations and memes ;-))
- Youtube: David Robinson oder Julia Silge (Tidy Tuesday Screencasts → preparation, exploration and analysis of new datasets)

Getting started



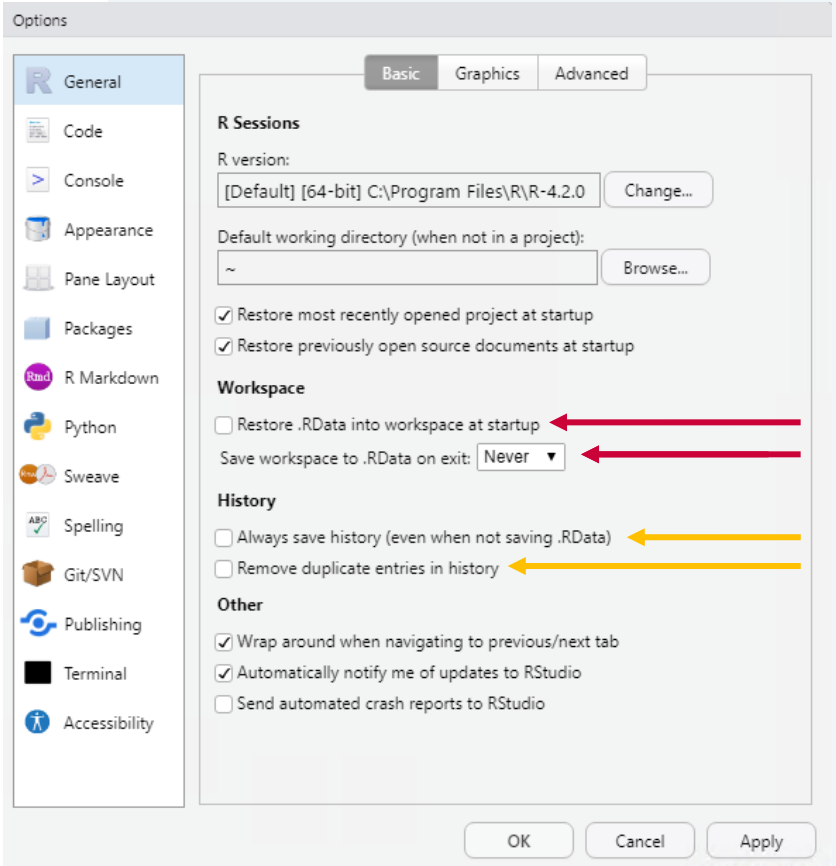
Getting started

- Installation:
  - install R: https://cloud.r-project.org/
  - install RStudio: https://posit.co/download/rstudio-desktop/

Set up RStudio – Basics

- Tools > Global Options > General
  - de-select „Restore .RData into workspace at startup“ & „Save workspace to .RData on exit“ to „Never“ (https://rstats.wtf/save-source.html)
  - „Always save history“: is needed very rarely (most important code is in the script)

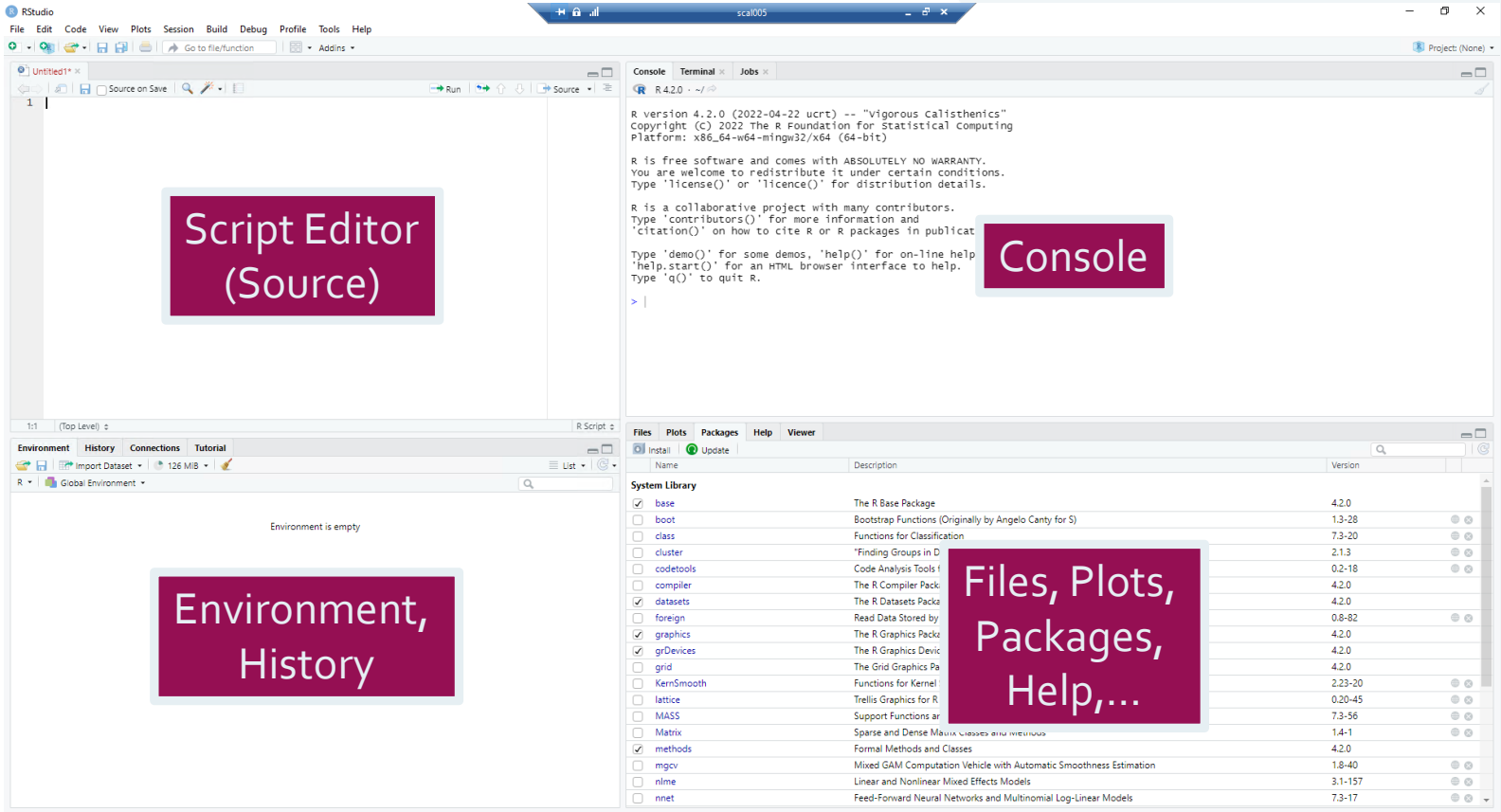
- Tools > Global Options > Code
  - Rainbow Parentheses
  - Native Pipe Operator?
  - Text Encoding?

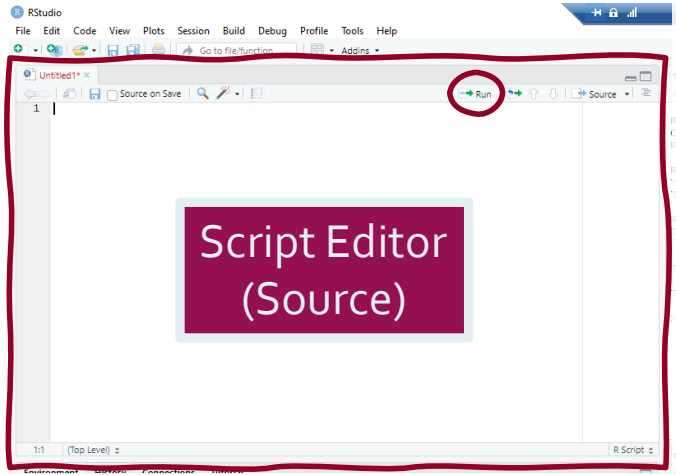


Set up RStudio – Make yourselves at Home

- Tools > Global Options > Appearance
  - Editor Theme – dark or bright?

- Tools > Global Options > Pane Layout
  - Console on the right side or below? (controversial opinion: most screens are wider than they are high → it‘s easier to look from left to right than from up to down)
  - I tend to prefer the console on the right side, when I have a larger screen and down when I‘m working on a laptop


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Script Editor (Source)
Console
R version 4.2.0 (2022-04-22 ucrt) -- "Vigorous Calisthenics"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publicat
Type 'demo()' for some demos, 'help()' for on-line help
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
Environment is empty
Environment, History
Files, Plots, Packages, Help,...
System Library
base The R Base Package 4.2.0
boot Bootstrap Functions (Originally by Angelo Canty for S) 1.3-28
class Functions for Classification 7.3-20
cluster "Finding Groups in D 2.1.3
codetools Code Analysis Tools f 0.2-18
compiler The R Compiler Pack 4.2.0
datasets The R Datasets Packa 4.2.0
foreign Read Data Stored by 0.8-82
graphics The R Graphics Packa 4.2.0
grDevices The R Graphics Devic 4.2.0
grid The Grid Graphics Pa 4.2.0
KernSmooth Functions for Kernel 2.23-20
lattice Trellis Graphics for R 0.20-45
MASS Support Functions ar 7.3-56
Matrix Sparse and Dense Matrix Classes and Methods 1.4-1
methods Formal Methods and Classes 4.2.0
mgcv Mixed GAM Computation Vehicle with Automatic Smoothness Estimation 1.8-40
nlme Linear and Nonlinear Mixed Effects Models 3.1-157
nnet Feed-Forward Neural Networks and Multinomial Log-Linear Models 7.3-17



Here code is written and saved to a script!

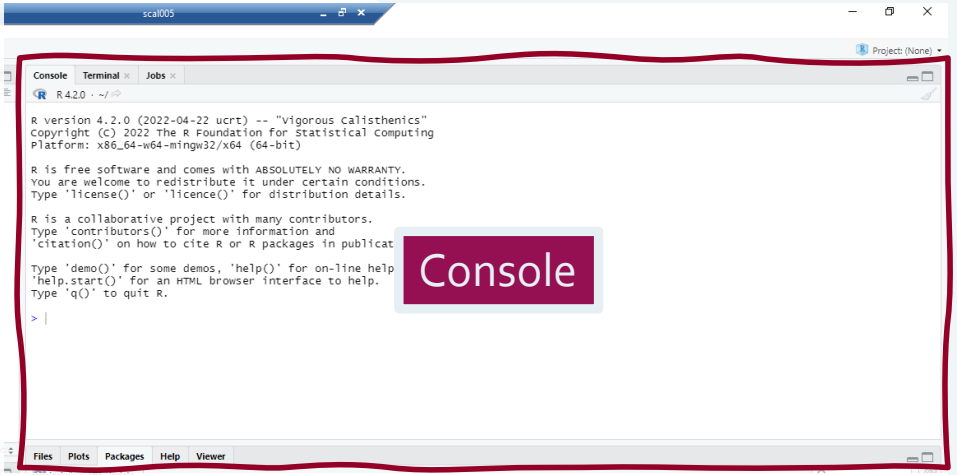
The code is not executed until it is sent to the console

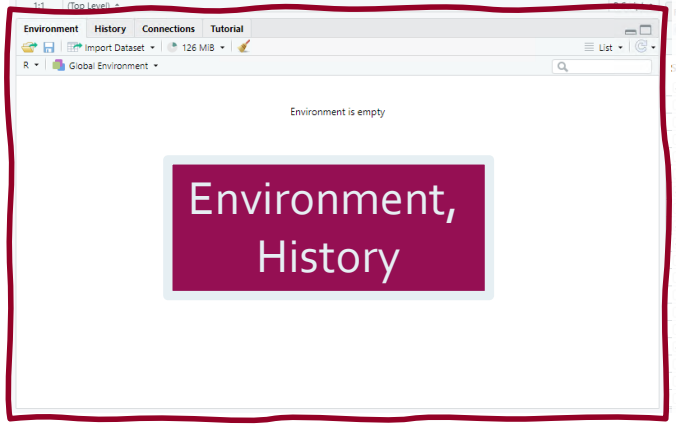
(**Ctrl + Enter** or **„Run“**)

The console executes code.

Here you can see executed **code** and its **results**.

The console is also useful for **quick calculations** that do not need to be saved. The code is then executed with **Enter**.

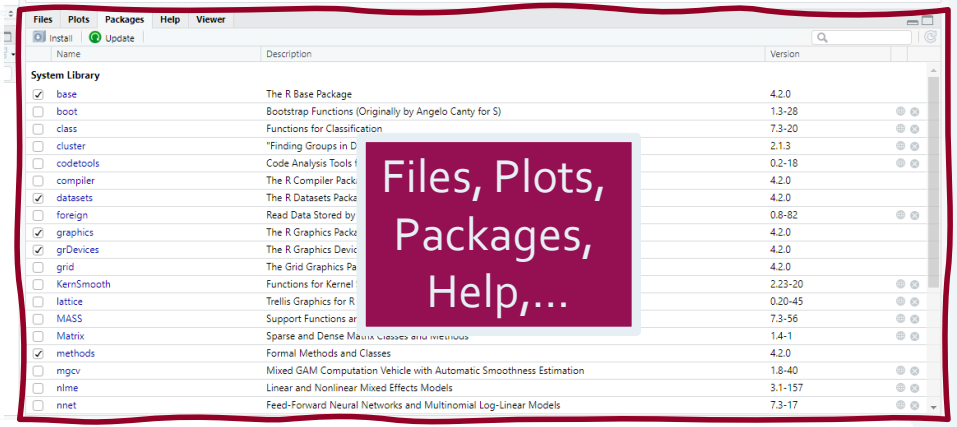




Here you can see which objects are stored in the **workspace** (**Environment**) or which code has already been executed (**History**).

Here are some help functions: you can…

- **navigate directories** (Files),
- look at **plots** (Plots),
- get an overview of installed **R packages** (Packages) and the
- look up **R help** (Help)



Mechanism of Programming in R

Everything that exists is an object.
Everything that happens is a function call.

Writing, executing and stopping code

- Code is written to the script editor and executed with **Ctrl+Enter** or **"Run"**.
- In the console the code is executed by only pressing **Enter**.
- Commands go until all brackets are closed – they may span multiple lines.
- The execution of code can be stopped by pressing Esc or the Stop sign (top right of the console).
- (1) define objects, (2) execute functions on those objects, (3) rinse and repeat!

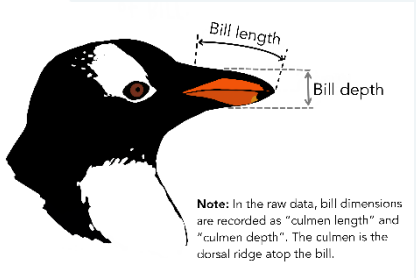
Language basics – what‘s important?

- **Upper and lower case**: object, Object, oBjEcT and OBJECT are all different objects and will be treated differently by R
- **Decimal separator**: period instead of comma (difference between `c(3,4)` and `c(3.4)`)
- **object names:** may not begin with a number and may not contain spaces: my.object, my_object, myobject, MyObject
- comments start with a **#**: everything to the right of the # will not be interpreted by R
- **<-** means assignment: the term on the right of the arrow is assigned to the object on the left of the arrow (`x <- 5`)

Jumping In

Palmer‘s Penguins

- open script: „00_Jumping_in.R“
- script contains some explorations on a dataset with penguin data



- https://allisonhorst.github.io/palmerpenguins/
- Gorman KB, Williams TD, Fraser WR (2014). Ecological sexual dimorphism and environmental variability within a community of Antarctic penguins (genus *Pygoscelis*). PLoS ONE 9(3):e90081. https://doi.org/10.1371/journal.pone.0090081

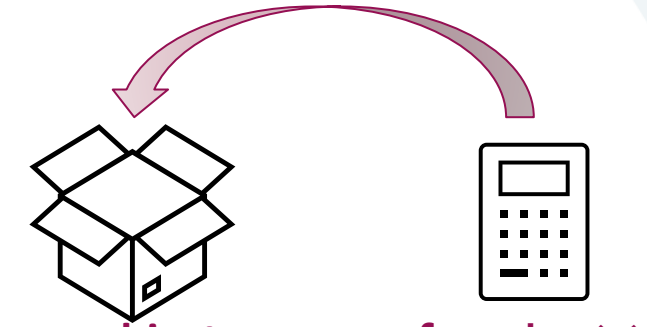
Basics: Objects

What are objects?

- **Everything in R is an object** (strictly speaking). There are simple objects (e.g. vectors, data frames) and complex objects (e.g. results from statistical methods)
- Objects are created and put in the environment by assigning something with the assignment symbol / arrow (`<-`). You can view available objects with the function `ls()`.
- Objects can be addressed without quotation marks.

```
> a <- 35   # a wird durch Zuweisung zu einem Objekt
>
> # 3 Möglichkeiten ein Objekt auszugeben
>
> a
[1] 35
> print(a)
[1] 35
> (a <- 35)
[1] 35
```

Assignment operator <-

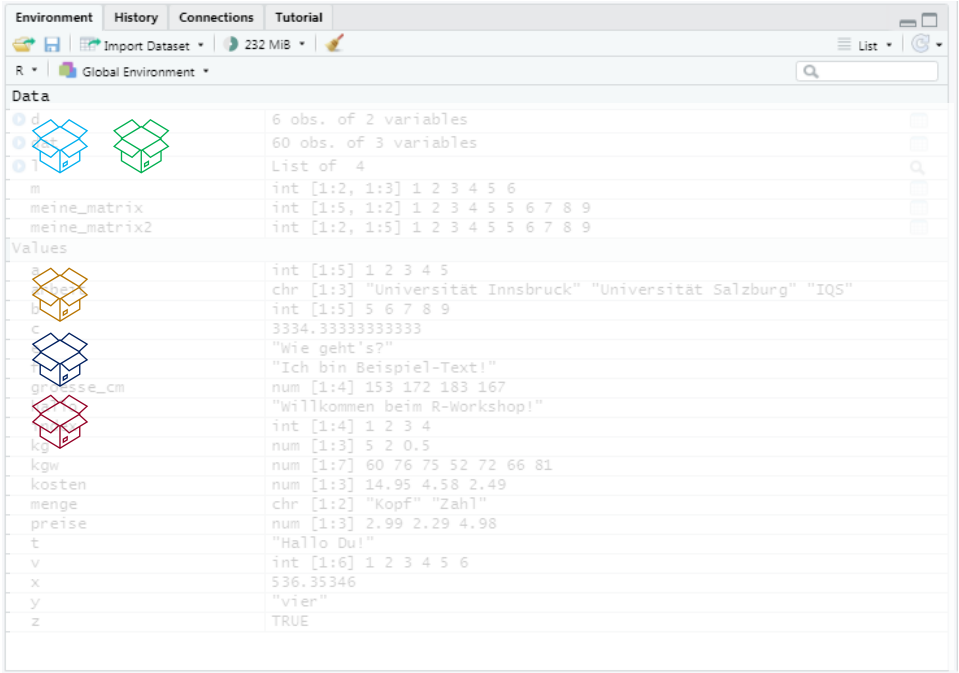


❷ and put it in this box.

❶ calculate the term on the right

```
in R: my_object <- function(x)
```

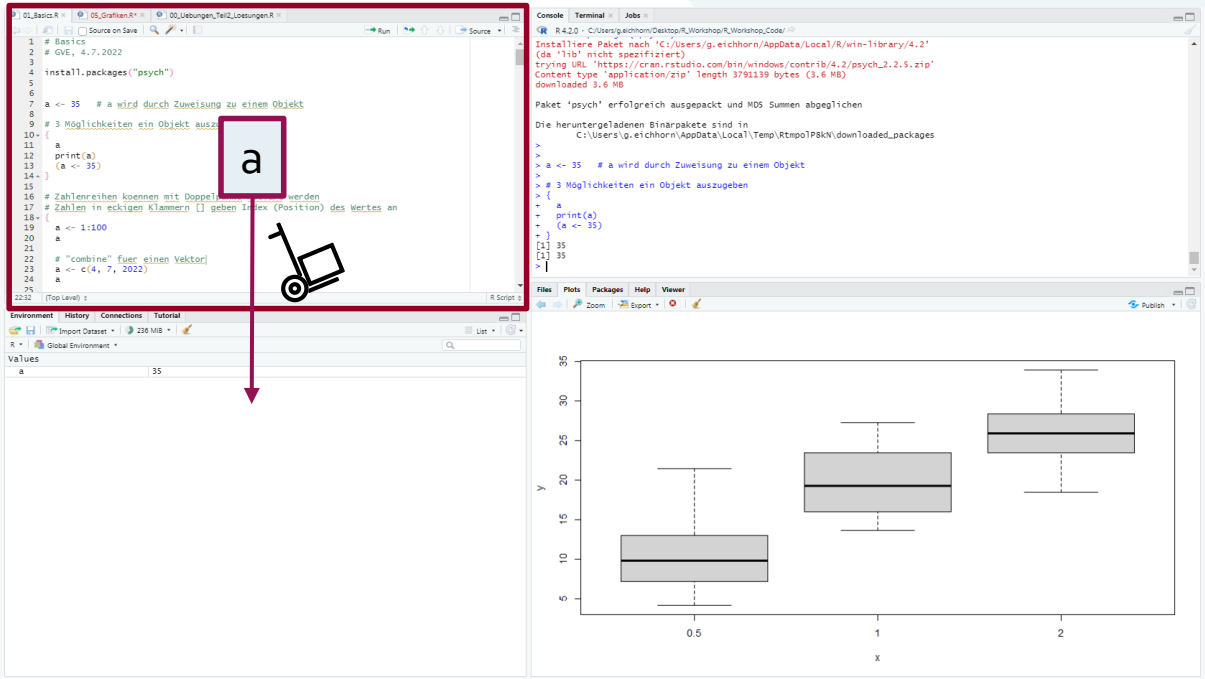
Objects are put in the environment



- Objects in the environment can be displayed in the console, …
  - `print( )`
- processed further and …
  - + 5
- saved into different or the same objects.
  - <- + 5

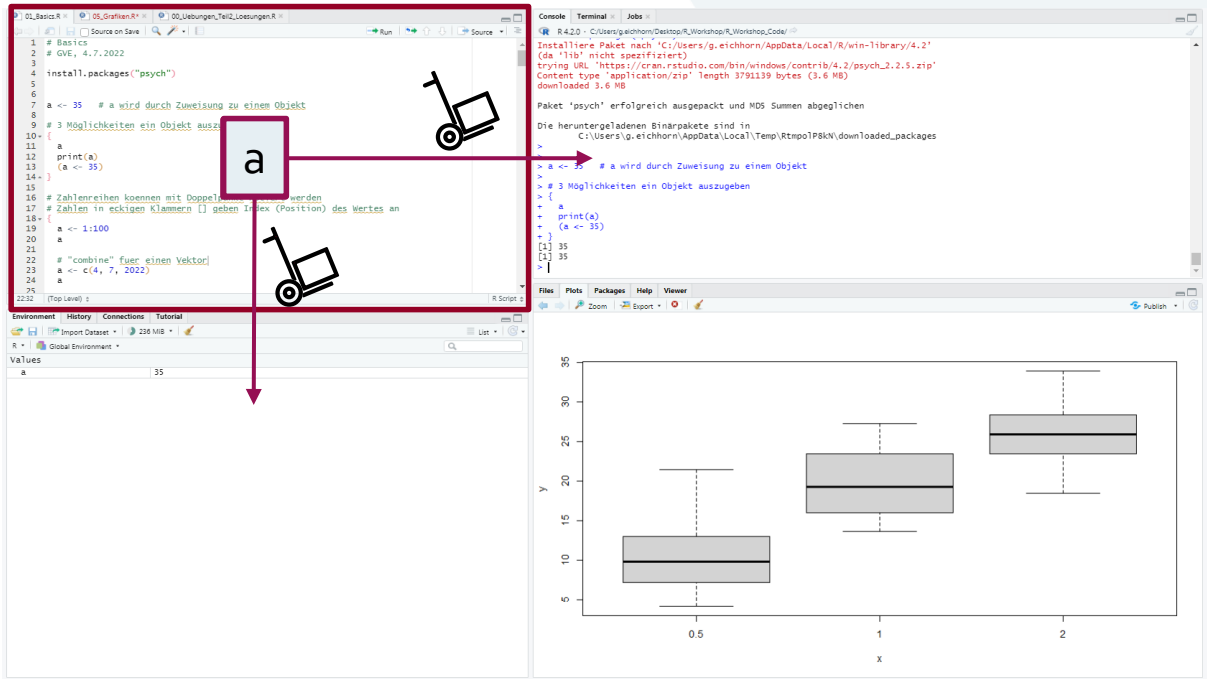
Saving objects to the environment

- `a <- 35`



Saving AND printing objects at the same time

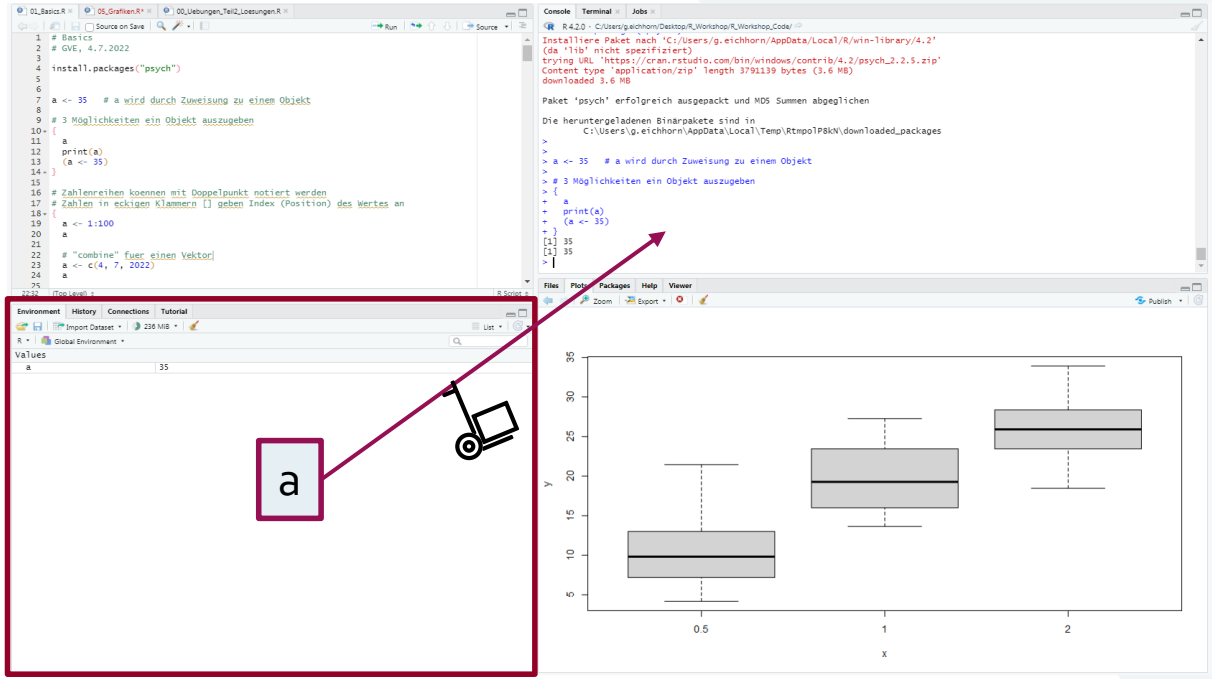
- `a <- 35`
- `(a <- 35)`



Printing objects

- `a <- 35`
- `(a <- 35)`
- `a`
- `print(a)`

- → if things are displayed only in the console and are not visible in the environment, you have to assign them



Exercise – Object Names

1. Create a new R script. Write your name, the date and "Exercise script“ in comments at the beginning of the script. Here you can work on your exercises!
2. Which of the following object names are valid / invalid? Try to guess first and then try out what happens if you assign something to these object names. (see slide 47 for help)

- `thisone <- 1`
- `THISONE <- 2`
- `1This <- 3`
- `this.one <- 4`
- `This.1 <- 5`
- `This/One <- 6`
- `ThIS…ON…E <- 7`
- `This!On!e <- 8`
- `lkjasdfkjsdf <- 9`

Exercise – Object Names (solution)

1. Create a new R script. Write your name, the date and "Exercise script“ in comments at the beginning of the script. Here you can work on your exercises!
2. Which of the following object names are valid / invalid? Try to guess first and then try out what happens if you assign something to these object names. (see slide 47 for help)

- `thisone <- 1`
- `THISONE <- 2`
- `1This <- 3`
- `this.one <- 4`
- `This.1 <- 5`
- `This/One <- 6`
- `ThIS…ON…E <- 7`
- `This!On!e <- 8`
- `lkjasdfkjsdf <- 9`

Exercise – Assignments

- Which calls are valid and what do they do (try to guess first if you're already advanced):
  - `d <- 123 -> e`
  - `pi <= 3.14`
  - `a = b = c = 567 -> x -> y -> z`
  - `x <- (5 + 2)`
  - `2 + 1 <- 3`
  - `3klang <- „c-e-g“`
  - `klang <- „c-e-g“`
  - `( x <- 5 + 3 )`
  - `zz <- 13; zz + 1`
  - `5 <- 3`
  - `5 < -3`

Exercise – first calculations

1. Calculate:
   - `2.6 + 5.8`
   - `(7 - 10.78) / 5`
   - `2^5`
2. Check:
   - `6 == 10`
   - `(3 * 5) > (9 + 1)`
3. Create and calculate:
   - `a = 2`
   - `b = 5`
   - `c = a + b`

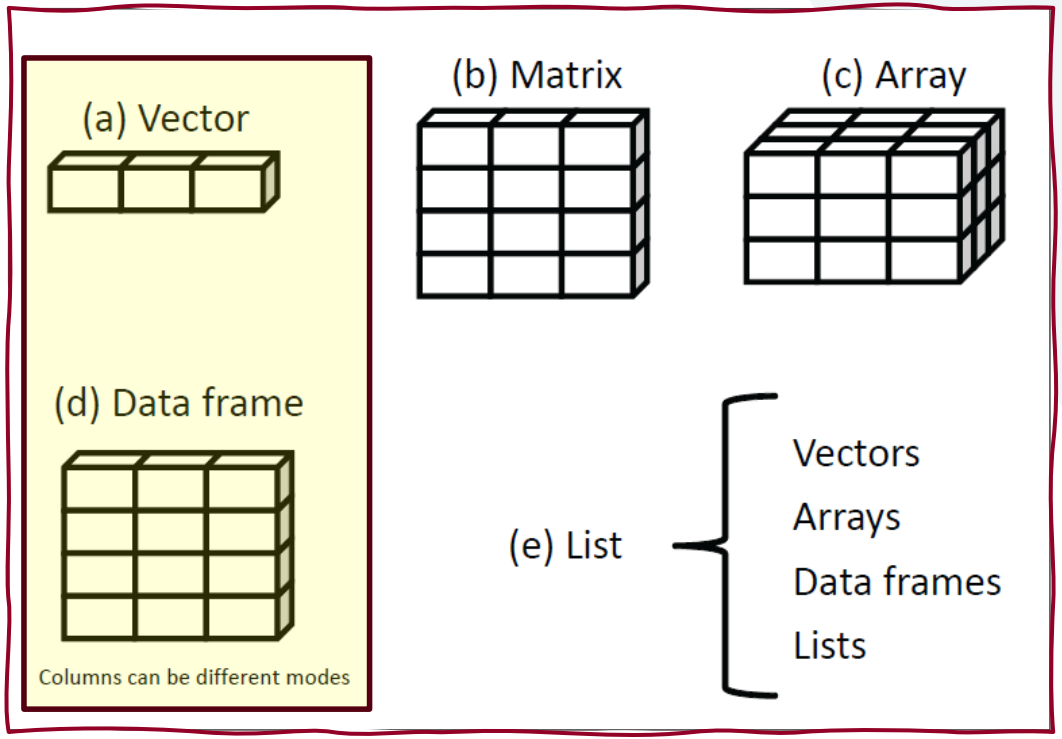
1. Take a look at this code: What will R return after the third line?
   - `a <- 10`
   - `a + 10`
   - `a`

Exercise – Types of objects

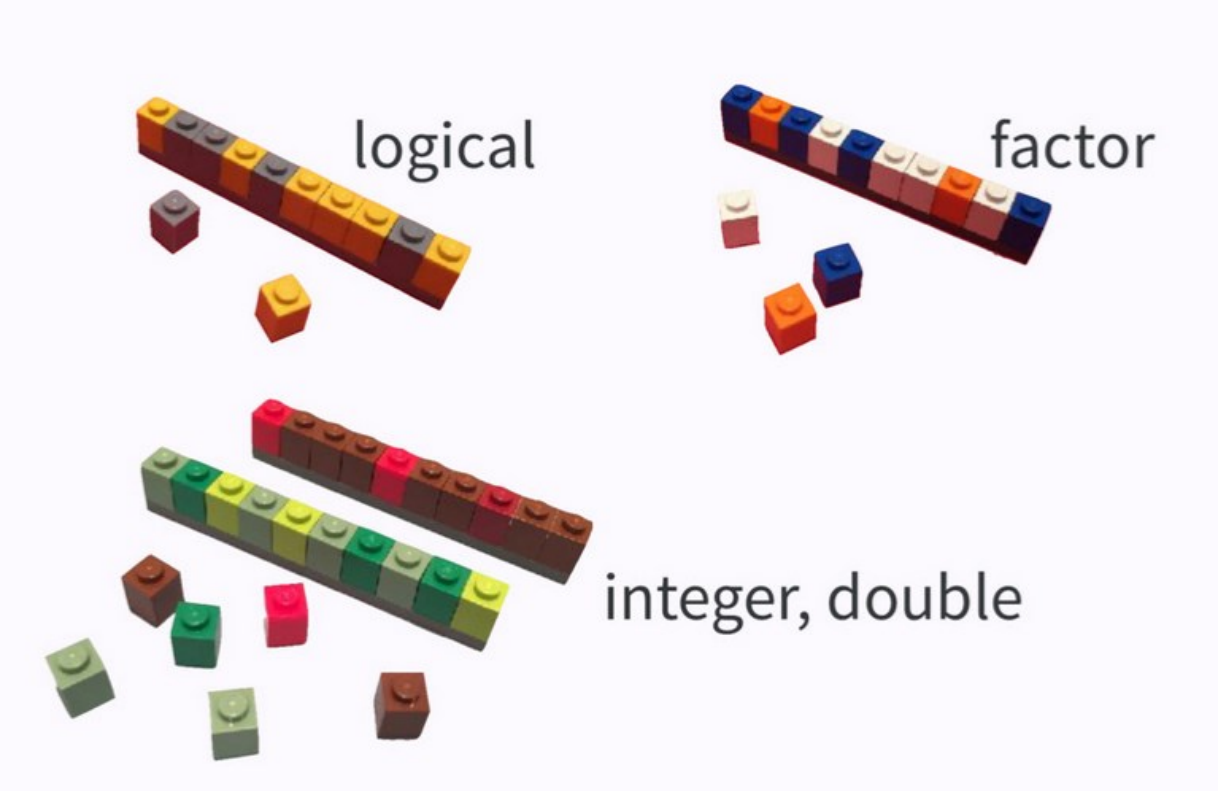
Create a variable for your first name, one for your last name and one for your birthplace.

1. How long is your last name (how many characters)?
2. Using `paste()` combine your data (name, birthplace) to an appealing output ("My name is ..., I was born in ...") The last name should be displayed completely in capital letters.
3. Replace the first two letters of your first name with an X.
4. Create a query: Is your first name the same as your last name?
5. Create another variable for your height in cm. Check if you are shorter than 172 cm.

Data structures



aus Kabacoff, 2011



aus „Data Rectangling" von Jenny Bryan, 2018

Matrices

- A matrix is a two-dimensional object → Rows and columns, similar to an Excel spreadsheet
- contains only data of the same type
- is created with the function `matrix()`
- can be accessed by square brackets `[ ]` with a comma to select rows and columns respectively

```
my_matrix[rows, columns]
```

Indexing matrices

- **my_matrix[3, 2]** corresponds to the 3rd line in the 2nd column

```
my_matrix[row, column]
```

[1, 1]	[1, 2]	[1, 3]	[1, 4]
[2, 1]			
[3, 1]	[3, 2]		
[4, 1]			
[5, 1]			

Indexing matrices

- **my_matrix[3, 2]** corresponds to the 3rd line in the 2nd column
- **my_matrix[3,]** corresponds to the entire 3rd line

```
my_matrix[row, column]
```

[1, 1]	[1, 2]	[1, 3]	[1, 4]
[2, 1]			
[3, 1]	[3, 2]	[3, 3]	[3, 4]
[4, 1]			
[5, 1]			

Indexing matrices

- **my_matrix[3, 2]** corresponds to the 3rd line in the 2nd column
- **my_matrix[3,]** corresponds to the entire 3rd line
- **my_matrix[, 2]** corresponds to the entire 2nd column

```
my_matrix[row, column]
```

[1, 1]	[1, 2]	[1, 3]	[1, 4]
[2, 1]	[2, 2]		
[3, 1]	[3, 2]		
[4, 1]	[4, 2]		
[5, 1]	[5, 2]		

Data Frames

- Data frames are the most important data structure for statistical analysis and most similar to the data structure in e.g. SPSS
- most of the time you will not enter data manually into R, but load it from existing files (Excel, SPSS, ...)
- Base R comes with preinstalled datasets, with which you can run code examples (`library(help = „datasets“)`)

Well-known datasets

- **iris** (150 observations, 5 variables)
  - Petal length and width for 3 species of irises (data by Edgar Anderson, 1935)
- **mtcars** (32 rows, 11 variables)
  - Fuel consumption and other automobile characteristics for 32 automobiles (from the 1974 Motor Trend US Magazine)
- **ToothGrowth** (60 observations, 3 variables)
  - Tooth length of guinea pigs in different experimental conditions (Vitamin C dose * method of administration)
- **ChickWeight** (578 rows, 4 variables)
  - Weight of chicks at multiple measurement time points, by diet condition
- **nycflights13::flights** (336776 rows, 19 variables)
  - Arrival and departure information for all New York airports flights in 2013
- **palmerpenguins::penguins** (344 rows, 8 variables)
  - Measured values of 344 penguins from different islands and species

Exercise – Data Frames

- You are planning a grocery trip and make a shopping list in R. Create a data frame with the columns Product, Department and Quantity.
- You would like to purchase the following products (quantity in brackets): Saft(4), Paprika(1), Zwiebel(2), Banane(1), Apfel(3), Ananas(1) and Bier(2). Assign these products to the respective department in the supermarket: beverages, fruits and vegetables.
- You are cycling to the supermarket and want to know which products may be very heavy. Therefore, have the data frame output so that only the products drinks are displayed anymore.
  - Create a new column “more_than_2” in the dataset which indicates with 0 whether the quantity is less than 2 and with 1 whether the quantity is greater than or equal to 2.

Basics: Packages

Packages („Packerln“)

- With the start of R, ***base R*** is loaded: a mixture of different packages containing most of the basics (base, stats, graphics, utils, datasets, methods, grDevices)
- If you want to use more functionalities, you can download and activate additional packages.
- An R Package is a compilation of (usually specific to one topic):
  - Data
  - Functions
  - Help pages
  - Vignettes / Examples

Packages

Currently, the CRAN package repository features 18276 available packages.

- Packages are like books that you buy and put on the shelf (`install.packages()`). You can access the contents only when you take out the book and put it on the desk (`library()`).

Available CRAN Packages By Date of Publication

Date	Package	Title
2022-06-22	asymmetry	Multidimensional Scaling of Asymmetric Proximities
2022-06-22	bapred	Batch Effect Removal and Addon Normalization (in Phenotype Prediction using Gene Data)
2022-06-22	bizdays	Business Days Calculations and Utilities
2022-06-22	clime	Constrained L1-Minimization for Inverse (Covariance) Matrix Estimation
2022-06-22	clintools	Tools for Clinical Research
2022-06-22	Clustering	Techniques for Evaluating Clustering
2022-06-22	edibble	Designing Comparative Experiments
2022-06-22	emmeans	Estimated Marginal Means, aka Least-Squares Means
2022-06-22	espadon	Easy Study of Patient DICOM Data in Oncology
2022-06-22	fairadapt	Fair Data Adaptation with Quantile Preservation
2022-06-22	GB2	Generalized Beta Distribution of the Second Kind: Properties, Likelihood, Estimation
2022-06-22	GeneralizedWendland	Fully Parameterized Generalized Wendland Covariance Function
2022-06-22	GrassmannOptim	Grassmann Manifold Optimization
2022-06-22	gtsummary	Presentation-Ready Data Summary and Analytic Result Tables
2022-06-22	HaploSim	Functions to Simulate Haplotypes
2022-06-22	html5	Creates Valid HTML5 Strings
2022-06-22	ICsurv	Semiparametric Regression Analysis of Interval-Censored Data
2022-06-22	jacobi	Jacobi Theta Functions and Related Functions
2022-06-22	Jmisc	Julian Miscellaneous Function

Using packages (I)

1

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

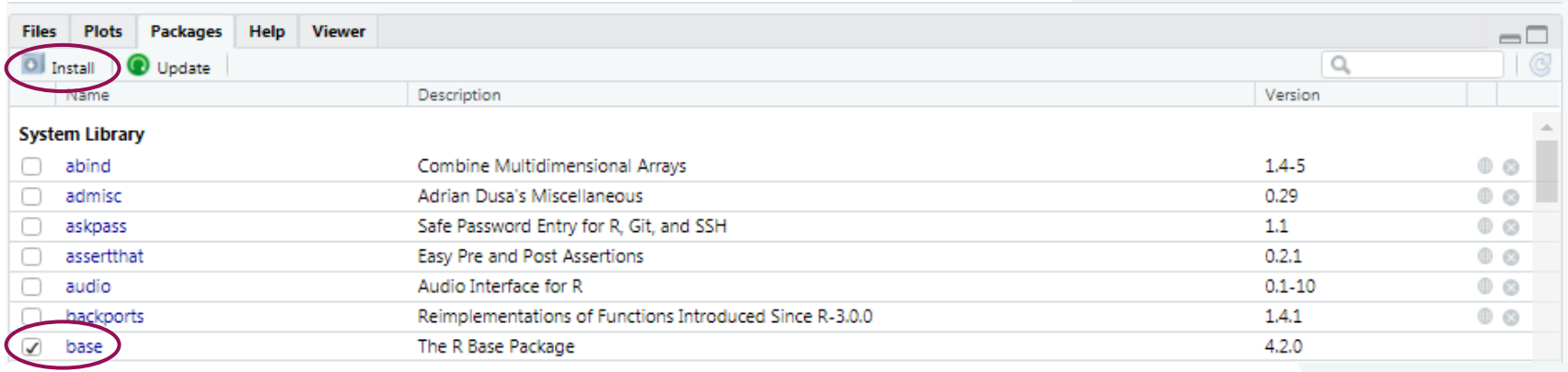
Download the package from CRAN;
1x per computer / user

2

```
> library("psych")
```

loads / activates the package
1x per R session

Using Packages (II)



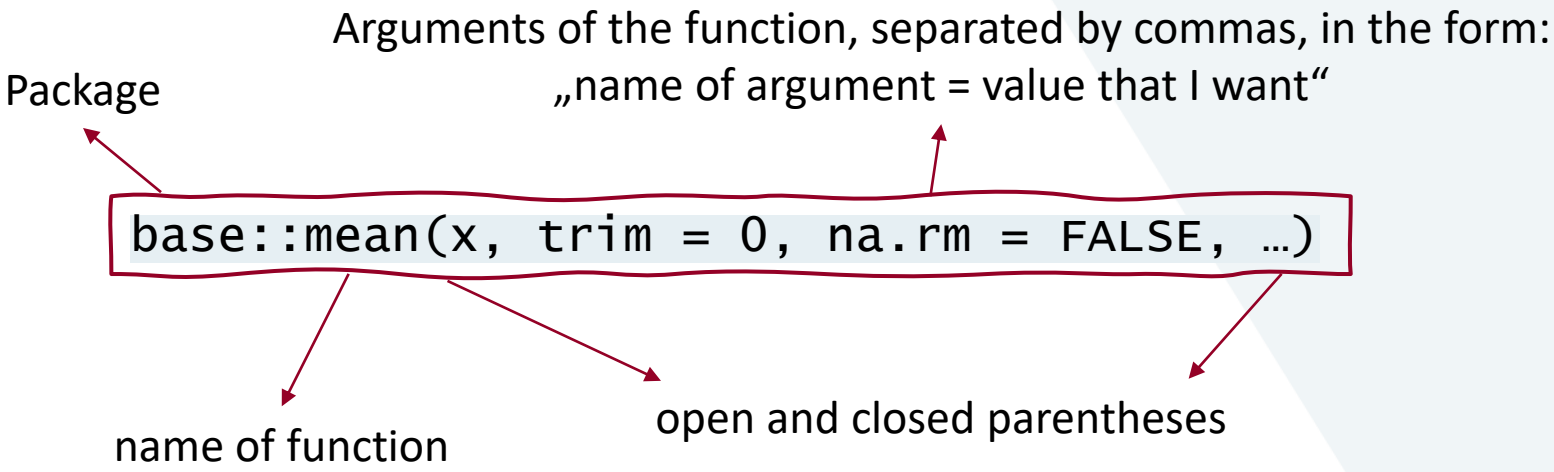
- install via „Install“; activating via the checkbox of the package
- loading packages should rather be done via code (`library()`), because otherwise your code might not be replicable by others or errors in the code may occur
- Tip: `pacman::p_load()` installs and activates at the same time – works really well when cooperating with others (who might not have the same packages as yourself)

Functions

Functions

- **Everything that happens is a function call (in R).**
- Functions are processes :
  - The function **takes** one or more objects,
  - **does** something with them and
  - then **gives something back**.

Standard structure of functions*



* Some functions do not have this form, e.g. operators like + and ==.

Dealing with functions

- The order of the arguments is defined in the function. If the arguments are used in the stored order, their names do not have to be mentioned. Many arguments have default values, these also do not have to be named.
- These variants all lead to the same result (recommended is in the correct order + arguments mentioned):

```
mean(x = my_vector, trim = 0, na.rm = FALSE)
```

```
mean(na.rm = FALSE, trim = 0, x = my_vector)
```

```
mean(my_vector, 0, FALSE)
```

```
mean(my_vector)
```

Digression: Conventions for comprehensible functions

- When writing functions oneself :
  - Name functions with verbs and arguments with nouns
  - data arguments first, then detail arguments
  - data arguments:
    - x, y, z for vectors
    - df for data frames
    - i, j for indices
    - n for length or number of rows
    - p for number of columns
- for more information: https://r4ds.had.co.nz/functions.html, https://www.datacamp.com/courses/writing-functions-in-r

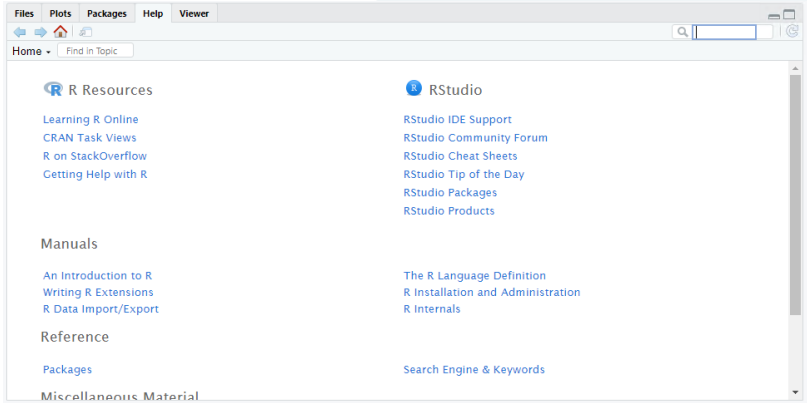
Exercise – Functions

1. Create a variable `values_r_erfahrung` from all responses to the question „Vorerfahrung mit R/RStudio“.
2. Calculate the mean value from the values and store it in the variable `mean_r_erfahrung`.
3. Calculate the mean value excluding the extreme 20%.
4. Add a number to the mean value so that the mean value ends up being 5.

Seeking help in R!

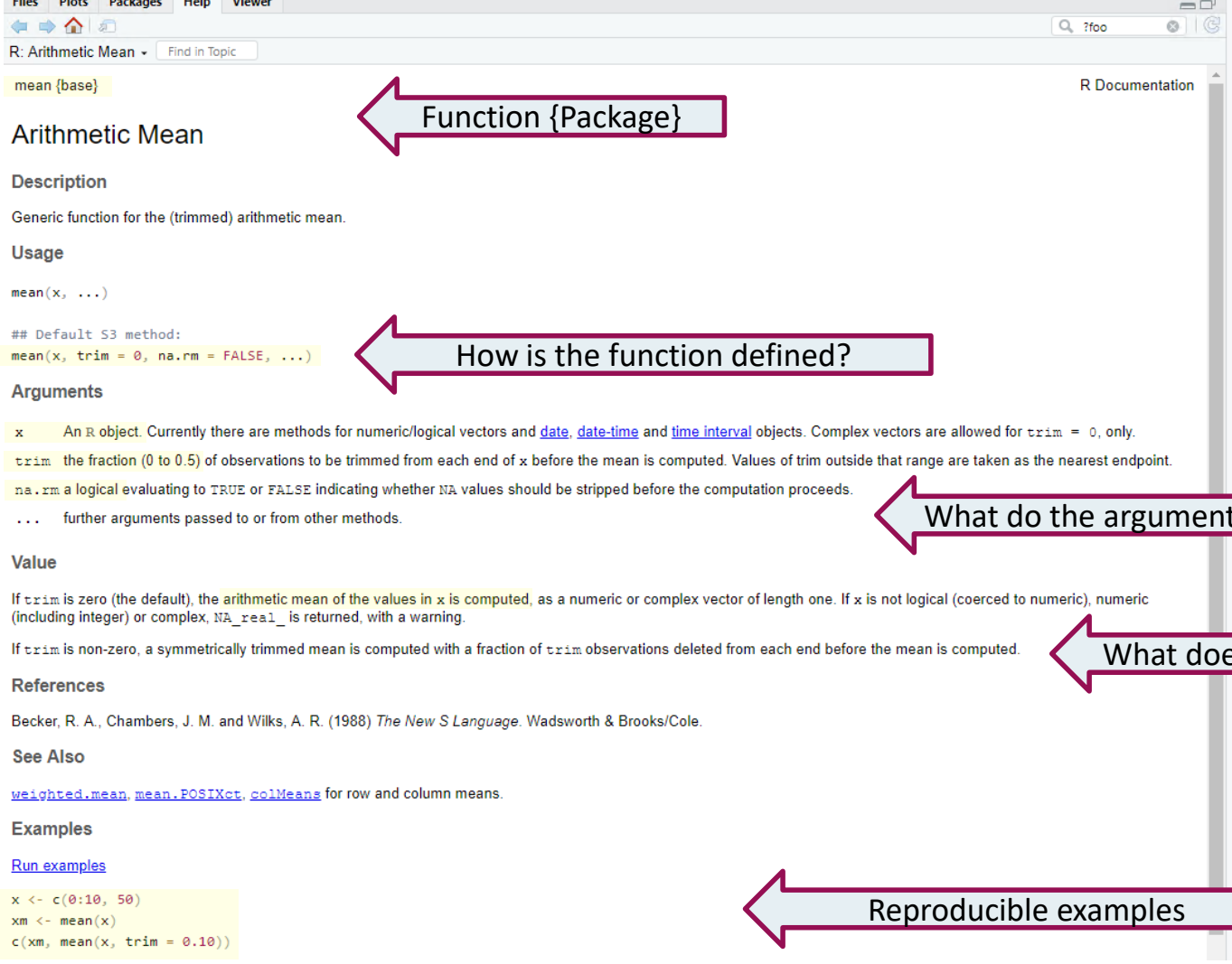
R Help

- `help.start()`
- `?foo`, `help("foo")`, highlight function and press F1
- `??foo`, `help.search("foo")`
- `example("foo")`



Reading help pages

- ***Description***: short description
- ***Usage***: form and arguments of the function, variants
- ***Arguments***: details on the arguments (e.g. which input they expect)
- ***Details***: can include anything and everything
- ***Value***: meaning of the output/return statement of the function
- ***Authors***: authors
- ***References***: books, articles for further reference
- ***See Also***: references to related R functions
- ***Examples***: executable code for the function


Files Plots Packages Help Viewer
?foo
R: Arithmetic Mean
Find in Topic
mean {base}
R Documentation
Arithmetic Mean
Function {Package}
Description
Generic function for the (trimmed) arithmetic mean.
Usage
mean(x, ...)
Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
How is the function defined?
Arguments
x An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for trim = 0, only.
trim the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
na.rm a logical evaluating to TRUE or FALSE indicating whether NA values should be stripped before the computation proceeds.
... further arguments passed to or from other methods.
What do the arguments expect?
Value
If trim is zero (the default), the arithmetic mean of the values in x is computed, as a numeric or complex vector of length one. If x is not logical (coerced to numeric), numeric (including integer) or complex, NA_real_ is returned, with a warning.
If trim is non-zero, a symmetrically trimmed mean is computed with a fraction of trim observations deleted from each end before the mean is computed.
What does the function return?
References
Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) The New S Language. Wadsworth & Brooks/Cole.
See Also
weighted.mean, mean.POSIXct, colMeans for row and column means.
Examples
Run examples
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))
Reproducible examples

Mini Quiz – Type conversion

- What is the result of this function (use help function, then try it out)?
  - `as.logical(0:5)`
- What is the result of this function?
  - `as.numeric(as.logical(0:5))`

Logical Vectors

Description

Create or test for objects of type `"logical"`, and the basic logical constants.

Usage

```
TRUE
FALSE
T; F

logical(length = 0)
as.logical(x, ...)
is.logical(x)
```

Arguments

`length`	A non-negative integer specifying the desired length. Double values will be coerced to integer: supplying an argument of length other than one is an error.
`x`	object to be coerced or tested.
`...`	further arguments passed to or from other methods.

Details

`TRUE` and `FALSE` are reserved words denoting logical constants in the `R` language, whereas `T` and `F` are global variables whose initial values set to these. All four are `logical(1)` vectors.

Logical vectors are coerced to integer vectors in contexts where a numerical value is required, with `TRUE` being mapped to `1L`, `FALSE` to `0L` and `NA` to `NA_integer_`.

Value

`logical` creates a logical vector of the specified length. Each element of the vector is equal to `FALSE`.

`as.logical` attempts to coerce its argument to be of logical type. In numeric and complex vectors, zeros are `FALSE` and non-zero values are `TRUE`. For `factor`s, this uses the `levels` (labels). Like `as.vector` it strips attributes including names. Character strings `c("T", "TRUE", "True", "true")` are regarded as true, `c("F", "FALSE", "False", "false")` as false, and all others as `NA`.

`is.logical` returns `TRUE` or `FALSE` depending on whether its argument is of logical type or not.

Tips for Googling

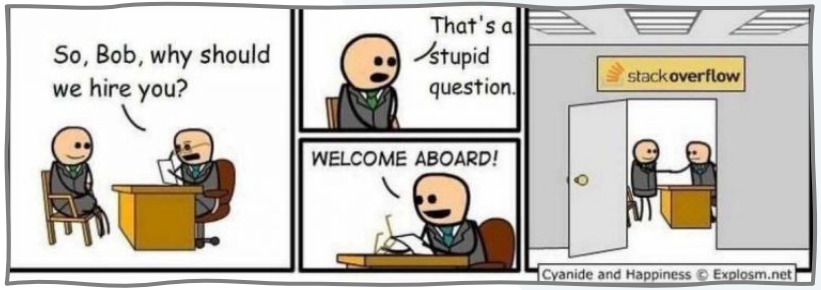
- R alone is an ambiguous term for Google. In most cases, search terms work anyway when using the pattern „r how to …“. „rstats“ or „r programming“ are good search terms as well.
- some tips and tricks:
  - google in English (!!!)
  - Add package and / or function names, e.g. „r how to recode variables with car recode“
  - if possible - specify preferred R dialect in search query: „r how to … in base“, „r how to … in tidyverse / dplyr“, „r how to … in data.table“
  - try different synonyms and guess functions; use R terminology (columns, rows, vectors, logical, numeric, character, data frame, matrix, list, factor, console, loop, regex…)

further tips for Googling

- Enter error messages directly into Google „r object of type closure is not subsettable“
- look up reference books and then google their suggested functions or terms (https://r4ds.had.co.nz/index.html, http://www.cookbook-r.com/)
- Blogs often provide general step-by-step instructions, StackOverflow and forums tend to provide solutions for very specific issues
- rseek.org, if Google does not show R specific search results

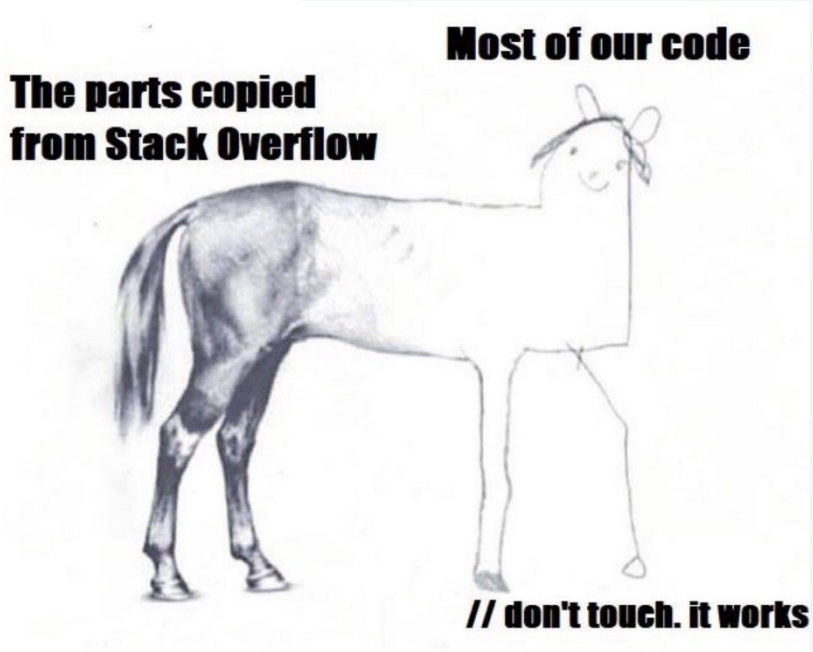
Tips for StackOverflow

- StackOverflow is a platform for software developers that allows users to ask and answer questions.
- Contrary to what the cartoon says, the R community on StackOverflow is very eager and friendly ☺
- Often there are different approaches to the solution, so the chance of success is greater and you can get to know other programming styles.
- There are clear guidelines for "reproducible examples“ (reprex) → if the situation cannot be reproduced by the respondents, the question is not answered.
- Bonus: From time to time you can also find answers from the package developers themselves!

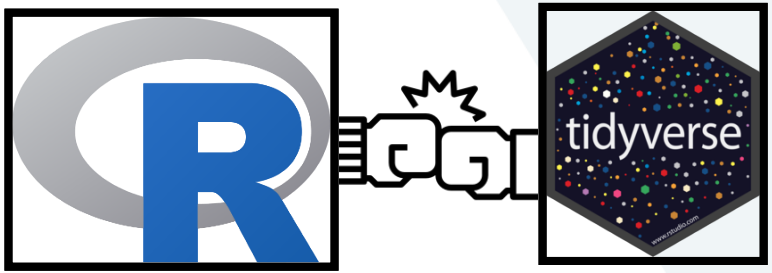


How to read StackOverflow?

- https://stackoverflow.com/questions/14543627/extracting-numbers-from-vectors-of-strings/38712300#38712300
- The answers get **upvotes** from other users and one answer is often accepted as the **best answer**
- I often look at the best answer and its rating first, and then evaluate whether the question helps me. Sometimes the question doesn't cover exactly what you are looking for, but the answer is still useful.



Base R vs. Tidyverse



Code comparison

Base R style	Tidyverse style
`head(dat)`	`dat %>% head()`
`dat$variable`	`dat %>% select(variable)` `dat %>% pull(variable)`

`dat[dat$sex == "female", "height"]`	`dat %>%`
	`filter(sex == "female") %>%`
	`select(height)`

Data Wrangling and Data Transformation with dplyr

„Data Rectangling" von Jenny Bryan, 2018

„Data Rectangling" von Jenny Bryan, 2018

„Data Rectangling" von Jenny Bryan, 2018

Data Wrangling / Data Transformation

dplyr (Tidyverse)

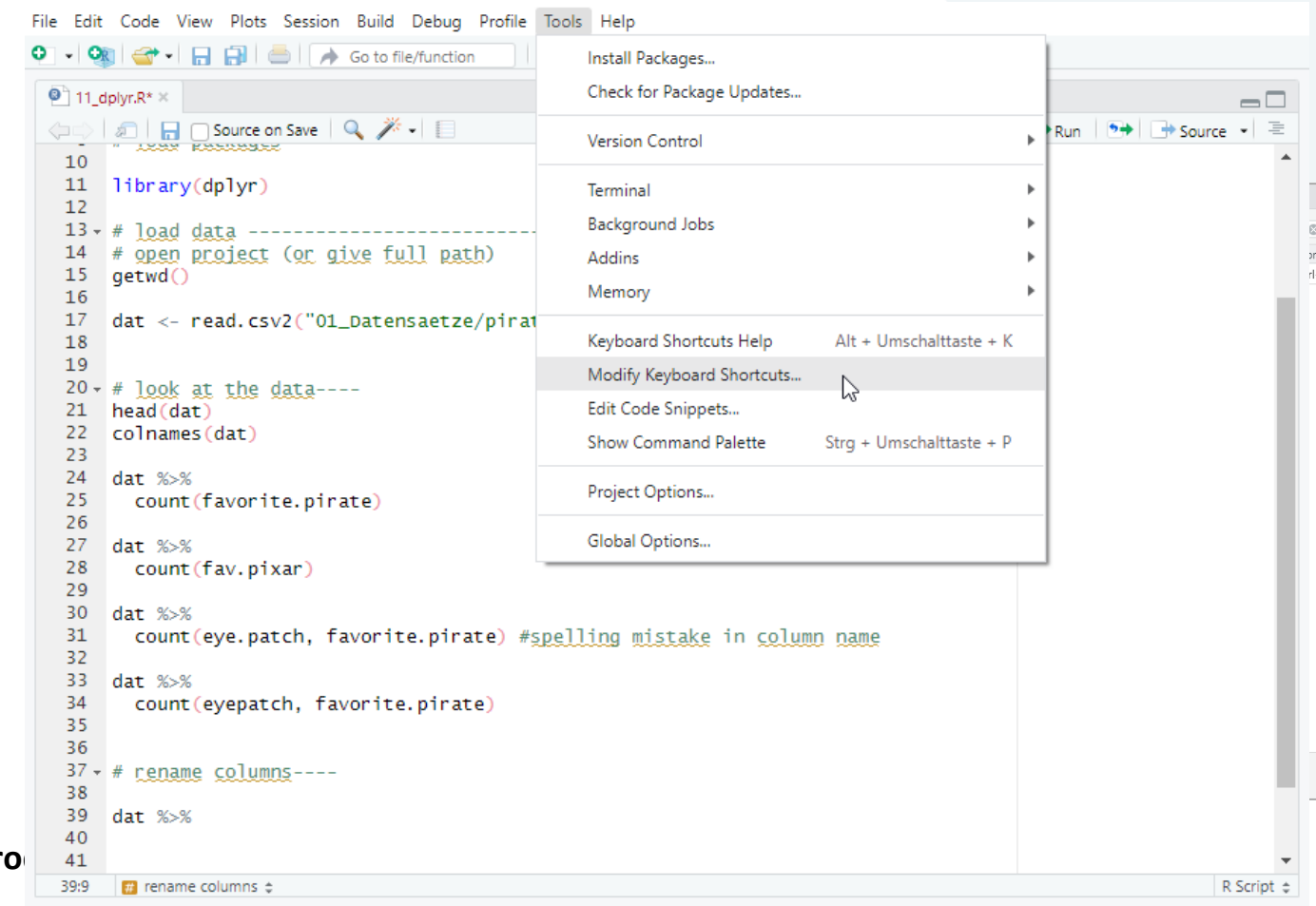
- For data preparation and exploration the package **dplyr** is very popular. It contains the functions:
  - `select()` → selecting columns
  - `filter()` → filtering
  - `mutate()` → creating new variables
  - `arrange()` → sorting

  - `summarise()` → creating summaries
  - `count()` → data summary

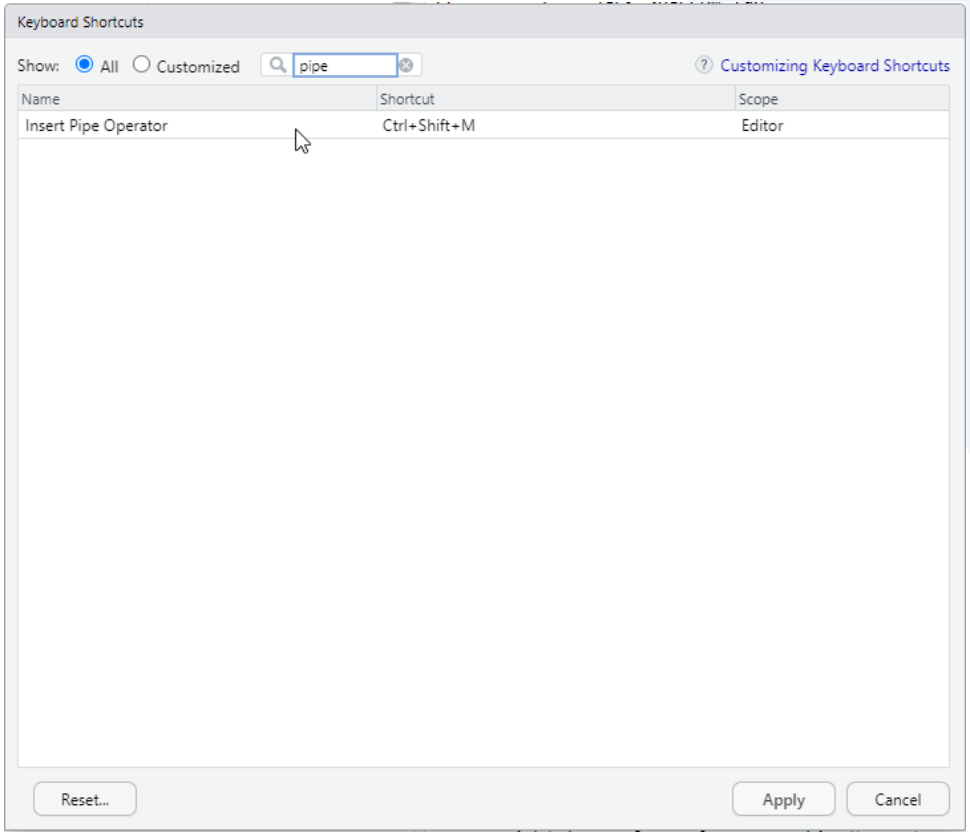
  - `case_when(), case_match()` → recode variables
  - `tidyr::pivot_longer(), tidyr::pivot_wider()` → reshape data

- For an introduction: https://youtu.be/Gvhkp-Yw65U; https://r4ds.had.co.nz/

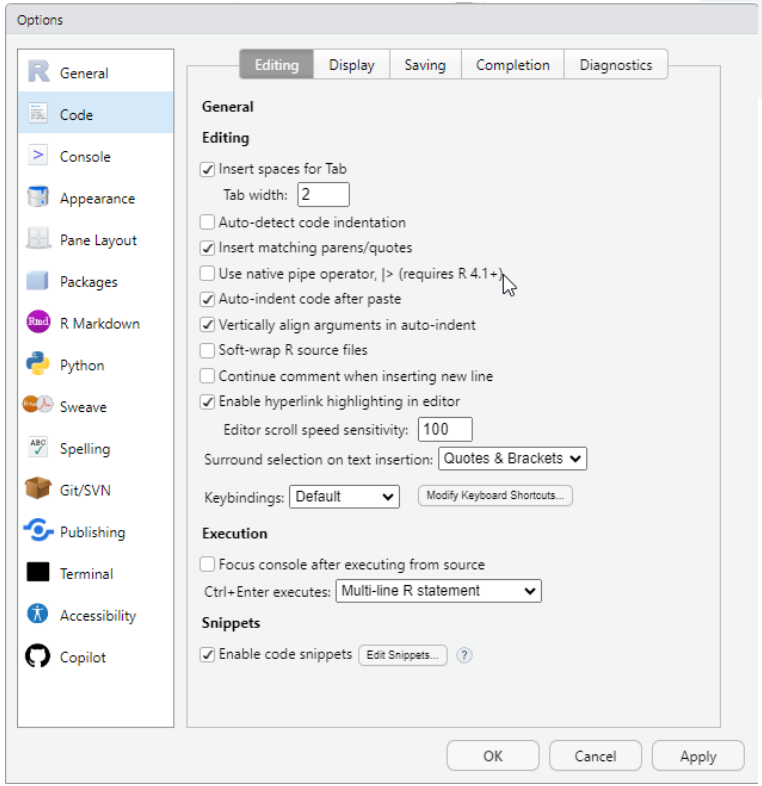
Preparations in RStudio: Shortcut for pipe %>%



Preparations in RStudio

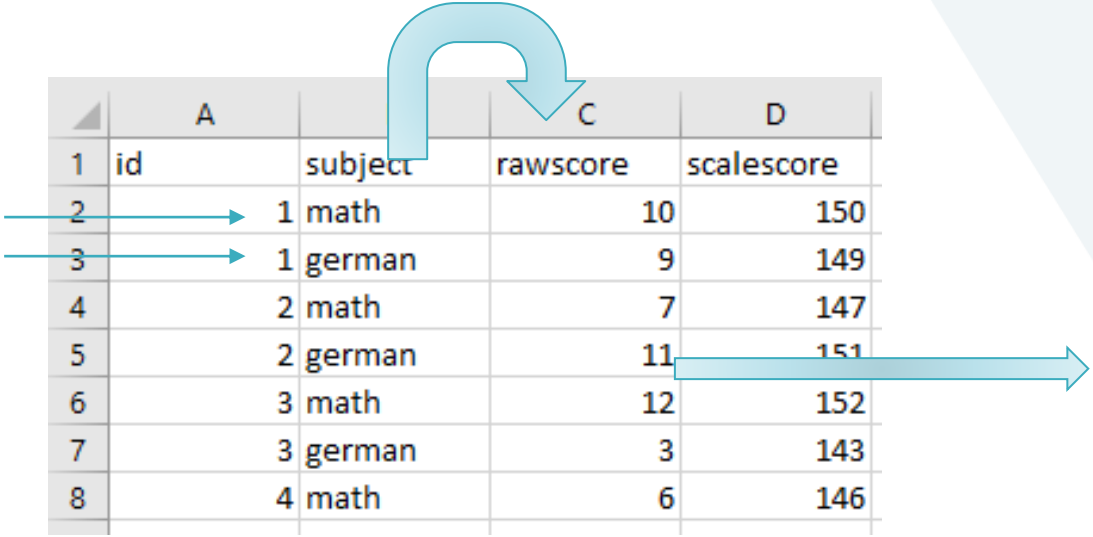


Default pipe



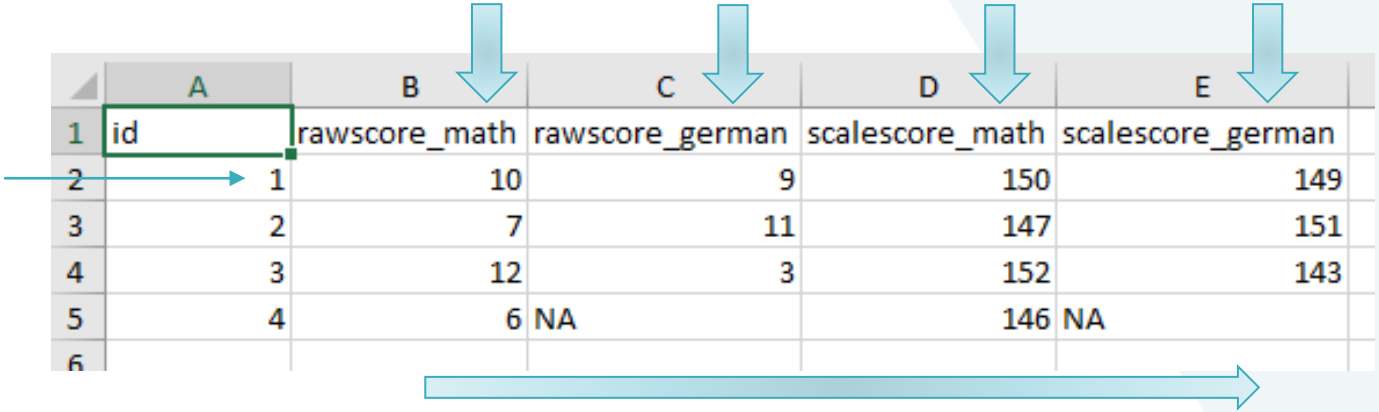
Reshaping Data

From long-format to wide-format



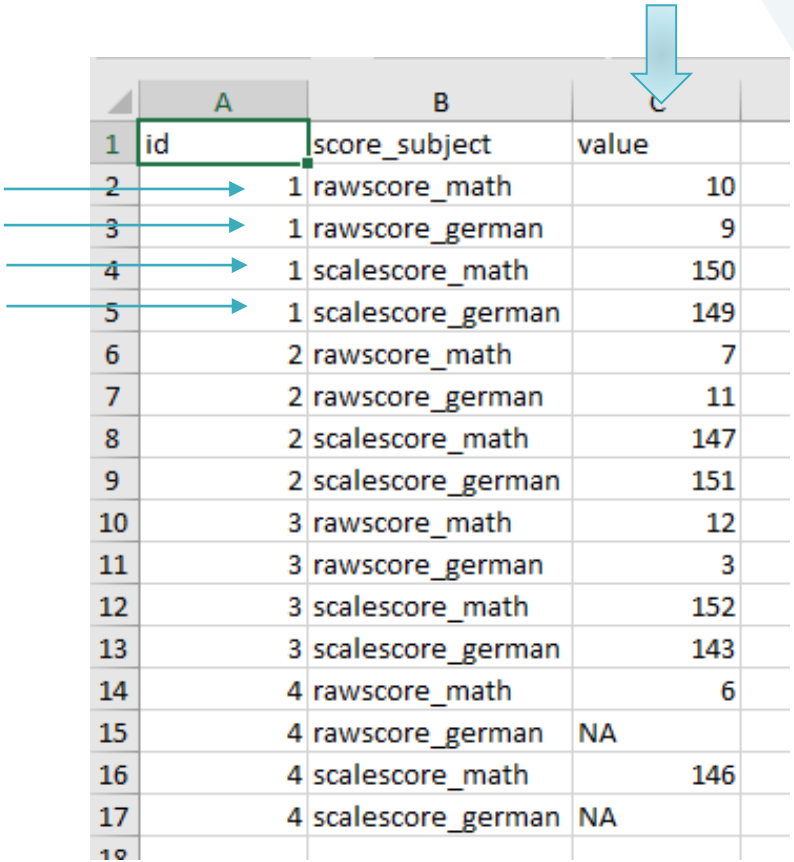
	A		C	D
1	id	subject	rawscore	scalescore
2	1	math	10	150
3	1	german	9	149
4	2	math	7	147
5	2	german	11	151
6	3	math	12	152
7	3	german	3	143
8	4	math	6	146

From long-format to wide-format



	A	B	C	D	E
1	id	rawscore_math	rawscore_german	scalescore_math	scalescore_german
2	1	10	9	150	149
3	2	7	11	147	151
4	3	12	3	152	143
5	4	6	NA	146	NA
6					

„Even longer“ format



	A	B	C
1	id	score_subject	value
2	1	rawscore_math	10
3	1	rawscore_german	9
4	1	scalescore_math	150
5	1	scalescore_german	149
6	2	rawscore_math	7
7	2	rawscore_german	11
8	2	scalescore_math	147
9	2	scalescore_german	151
10	3	rawscore_math	12
11	3	rawscore_german	3
12	3	scalescore_math	152
13	3	scalescore_german	143
14	4	rawscore_math	6
15	4	rawscore_german	NA
16	4	scalescore_math	146
17	4	scalescore_german	NA

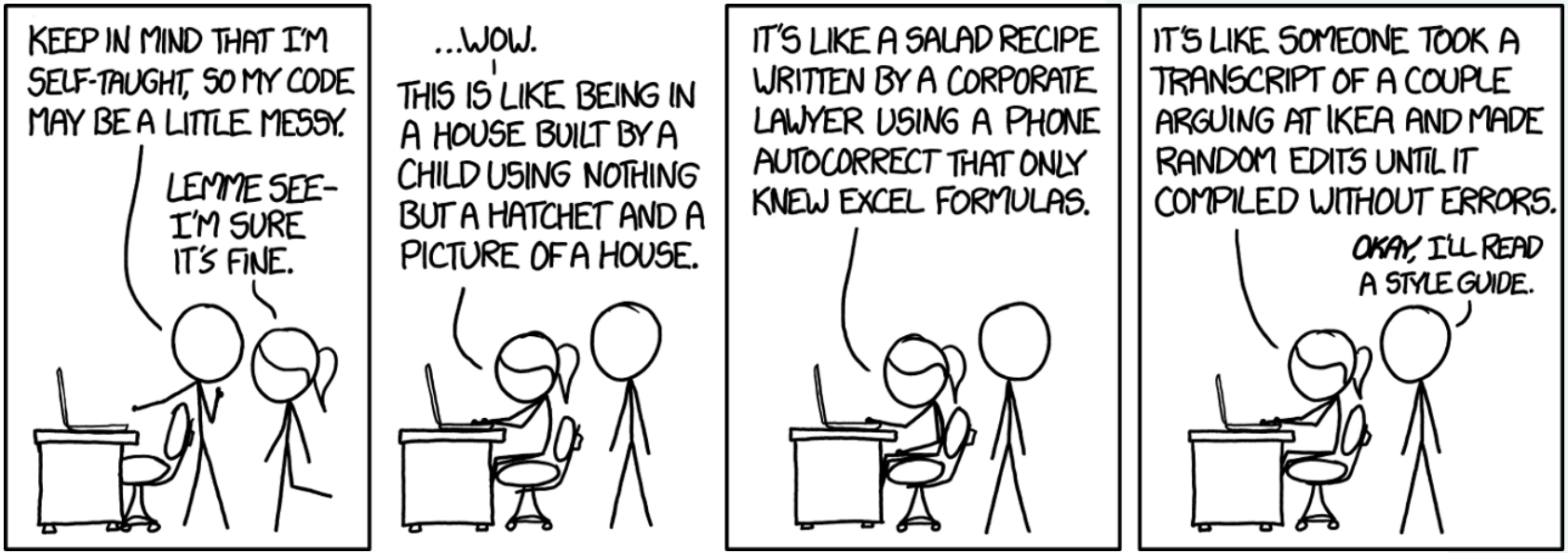
Several datasets (or a `list`)

	A	B	C	D
1	id	subject	rawscore	scalescore
2	1	german	9	149
3	2	german	11	151
4	3	german	3	143

	A	B	C	D
1	id	subject	rawscore	scalescore
2	1	math	10	150
3	2	math	7	147
4	3	math	12	152
5	4	math	6	146

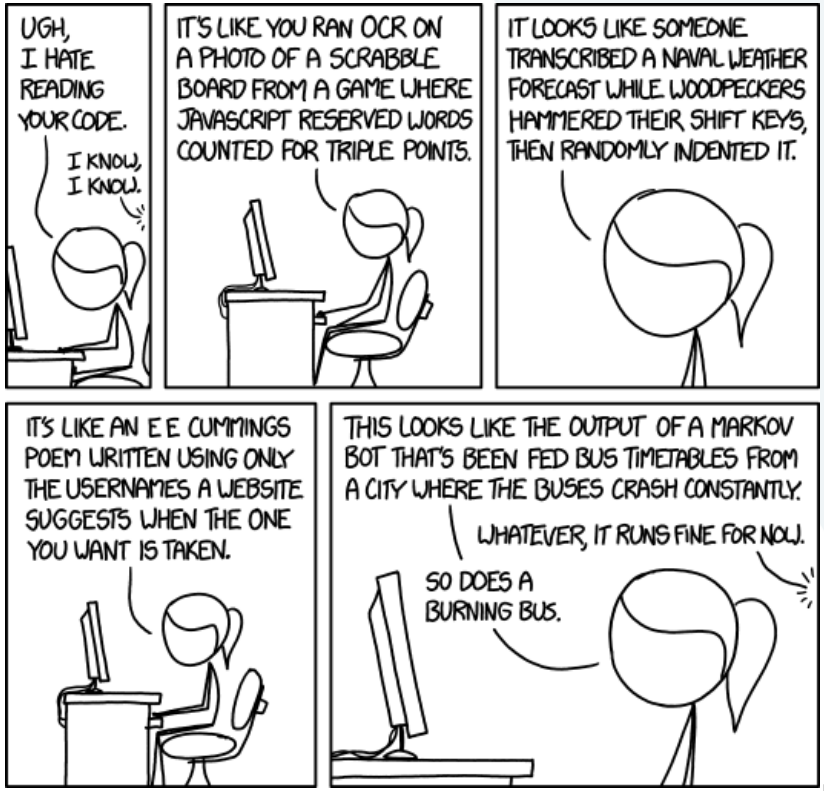
Best Practices

Style Guide (Vol. I)



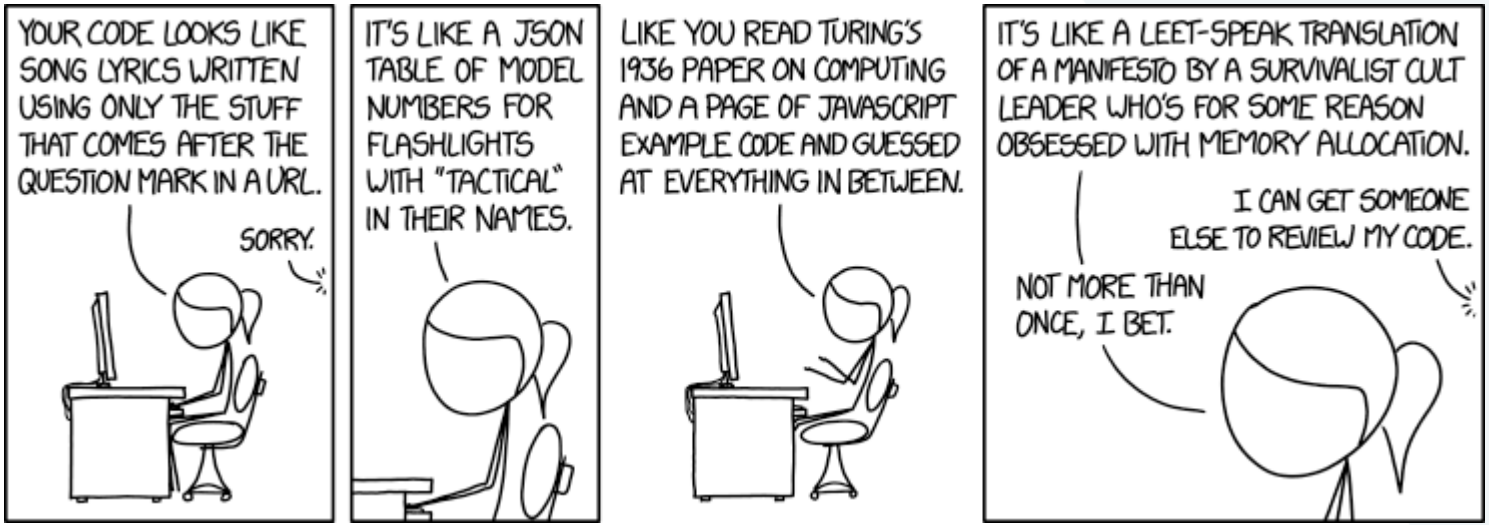
https://xkcd.com/1513/

Style Guide (Vol. II)



https://xkcd.com/1695/

Style Guide (Vol. III)



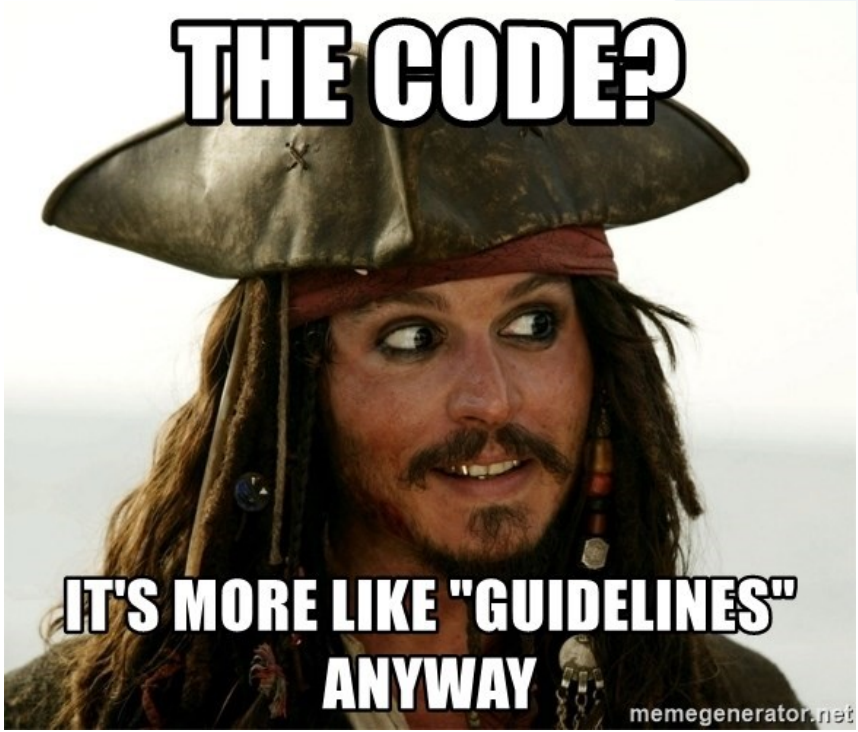
https://xkcd.com/1833/

Why Style Guides?

- Code should be readable for computers AND humans!
- A few simple measures:
  - **Headings** with ####, ----, ==== or Code > Insert Section (Ctrl+Shift+R), optical separators, e.g. #****
  - **put sections in { }** – that way they can be carried out section by section, are indented and visually easy to recognize
  - **Comment**: R lines are free, so be sure to use them to comment your code
  - **Spaces**: put spaces around =, ==, +, -, <- and after commas
  - **line breaks**: when the line gets too long (80 characters as a guideline)
  - choose **meaningful object names** : find balance between „df“ and „March2015Group1OnlyFemales“

Even more Style Guide!!!

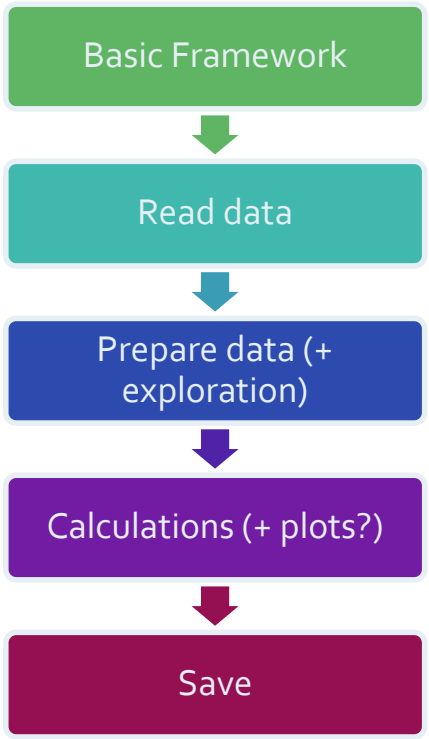
- https://style.tidyverse.org/index.html
  - Object names with underscores (periods are already reserved for base functions, methods and class names).
  - Variables = Nouns; Functions = Verbs
  - no function names as object names (T, F, c, mean, …)
  - Name arguments if the default is not used (z.B. mean(x, **na.rm =** TRUE))
  - Use <- for assignment (not =)
  - „“ instead of ‚‘
  - write TRUE and FALSE instead of T and F
  - for long function calls, put each argument and the parentheses on a separate line
- https://google.github.io/styleguide/Rguide.html (Addition to Tidyverse Guide)


THE CODE?
IT'S MORE LIKE "GUIDELINES"
ANYWAY
memegenerator.net

Shortcuts in RStudio

Shortcut	Function
Strg + Enter	Execute code from the script
Strg + Alt + B	Run code from beginning to this point
Strg + Shift + C	Comment out code (#)
Strg + I	Indent code
Strg + Shift + D	Duplicate line of code
Strg + Shift + R	Insert section
Strg + Shift + M	Insert pipe (base or dplyr, depending on the setting in Options)
F1	Help (for highlighted function) = `?funktion`
Strg + F	Search
Strg + Shift + F10	R restart (useful for a reset if problems occur)

Structure of R scripts




```r
# Beschreibung des Skripts
# Autor
# Datum

# Grundgeruest ####
# Packages laden
library("openxlsx")
# Pfade setzen
pfad <-
  "C:/Projekte/WS_Einfuehrung_in_R"
setwd(pfad)

# Daten einlesen ####
dat0 <- read.xlsx("meine_daten.xlsx", sheet = 1)

# Daten aufbereiten ####
dat <- dat0[1:3, ]   # Daten auswaehlen

# Berechnungen durchfuehren ####
model1 <- lm(y ~ x, data = dat)

# Speichern #####
save(model1, file = "Ergebnisse.Rdata")
```

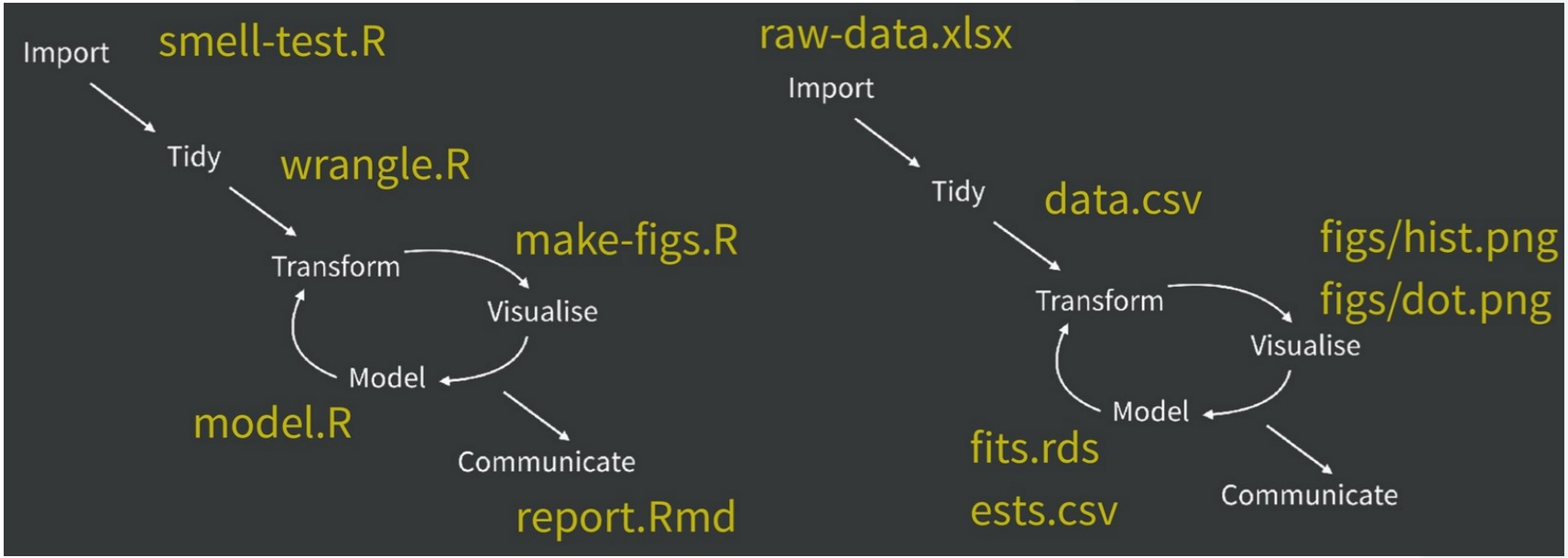
„Data Rectangling" von Jenny Bryan, 2018

Input	Code	Output
raw data	00_smell-test.R	*wisdom*
raw data	01_wrangle.R	data.csv
data.csv	02_model.R	fits.rds
		ests.csv
data.csv	03_make-figs.R	figs/*
fits.rds		
ests.csv		
figs/*	04_report.Rmd	report.html
ests.csv		report.docx
		report.pdf

Structure of bigger analysis projects

- one script for „Make Friends with your Data“ → chaos is allowed here! ☺
- (numbered) scripts for defined substeps
- save substeps (prepared data, models, plots,…)
- put everything you still find important into a report

aus „Zen And The aRt Of Workflow Maintenance“ (Jenny Bryan, 2018)


Import
smell-test.R
Tidy
wrangle.R
Transform
make-figs.R
Visualise
Model
model.R
Communicate
report.Rmd
raw-data.xlsx
Import
Tidy
data.csv
Transform
figs/hist.png
figs/dot.png
Visualise
Model
fits.rds
ests.csv
Communicate

What's in a Name? That which we call a Rose by any other name would smell as sweet.

(Shakespeare, 1595)

- WRONG! Names of objects and files are very important
- Naming files and things well is difficult - but worth the effort. Accurate and well-written names help others and yourself to understand and deal with the code.

Prinzipien für Dateinamen

1. **machine readable**
   - avoid spaces, punctuation, accented characters, case sensitivity
   - deliberate use of delimiters (_, -)
2. **human readable**
   - name contains info on content
3. **plays well with default ordering**
   - put something numeric first (chronologically or logically ordered)
   - left pad other numbers with zeros (“fig01”)

NO

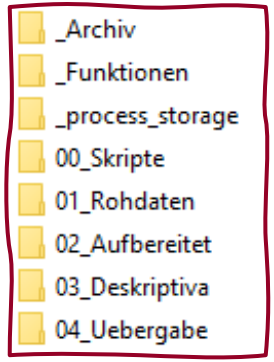
myabstract.docx
Joe’s Filenames Use Spaces and Punctuation.xlsx
figure 1.png
fig 2.png
JW7d^(2sl@deletethisandyourcareerisoverWx2*.txt

YES

2014-06-08_abstract-for-sla.docx
joes-filenames-are-getting-better.xlsx
fig01_scatterplot-talk-length-vs-interest.png
fig02_histogram-talk-attendance.png
1986-01-28_raw-data-from-challenger-o-rings.txt

Directories

- for smaller projects you can manage with one folder - for larger ones you usually start with one folder and regret it afterwards ☺
- Tip: it is better to start too organized than to have to overturn everything afterwards



in the simplest form:

- 01_rawdata
- 02_prepared
- 03_results

optional:

- scripts
- plots
- report
- documentation/additional info
- _process_storage (substeps)
- _archive (for older versions that you do not dare to delete)

Problem with fixed working directories

We need to talk about setwd(„path/that/only/works/on/my/machine“)!

- There are disadvantages to using the default working directory or setting your own working directories
  - in cooperation with other persons
  - when changing the PC or changing folder structures
  - when switching between analysis projects
- Following some guidelines, `setwd()` is fine:
  - use only in obvious places (at the beginning of the script)
  - set to a fixed location (highest level of a project)

Project-oriented workflow

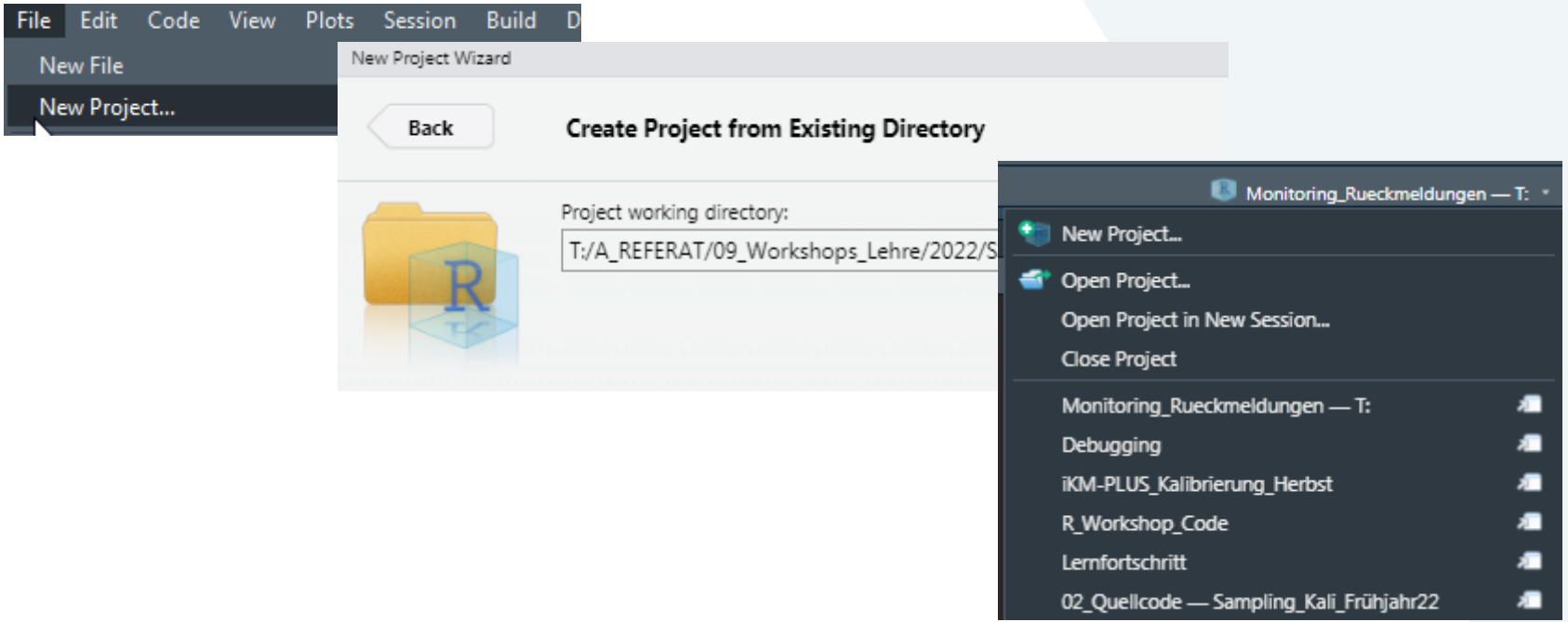
- **File system discipline**
  - all relevant files are stored in designated folders (data, code, graphics, notes,...)
  - Organization in subfolders is often useful
- **Working directory intentionality**
  - the working directory in project A is always in project As folder (ideally this is not achieved by hardcoded paths in the code)
- **File path discipline**
  - all paths are relative to the designated parent folder

https://rstats.wtf/projects.html

Projects and the here package

- RStudio projects set the working directory to the location of the project
- the here package recognizes the project and can output the path
- Projects can also have their own settings (Tools > Project Options) or a profile (.rprofile) whose content is executed when the project is opened (e.g. to load packages)
- Advantages:
  - no manual copying of folder paths
  - no search-replace with backslashes
  - the project runs (if used correctly with relative paths) also with other persons or on other PCs
  - you can easily work on multiple projects at the same time through multiple R sessions

Creating projects



Small change – big effect

- with setwd()

```
library(ggplot2)
setwd("/Users/jenny/cuddly_broccoli/verbose_funicular/foofy/data")
df <- read.delim("raw_foofy_data.csv")
p <- ggplot(df, aes(x, y)) + geom_point()
ggsave("../figs/foofy_scatterplot.png")
```

- with project in a parent folder and here::here()

```
library(ggplot2)
library(here)

df <- read.delim(here("data", "raw_foofy_data.csv"))
p <- ggplot(df, aes(x, y)) + geom_point()
ggsave(here("figs", "foofy_scatterplot.png"))
```

Descriptive Statistics

Principle

- read data
- Prepare data, prepare for analysis
- (maybe exploration)
- **ANALYSIS!**
- save result

Overview

Function	Description
`mean()`	Mean value (arithmetic mean)
`median()`	Median
`quantile()`	Quantile
`var()`	Variance
`sd()`	Standard deviation

Overview

Function	Description
`summary()`	Summary of (any) object
`psych::describe()`	important characteristic values (e.g. min, max, mean value)
`range()`	minimum, maximum
`cor()`	Correlation (by default: Pearson. Kendall or Spearman are possible) Significance test for correlations with cor.test(x,y)
`cov()`	Covariance

To get an overview of data …

- `psych::describe()`

```
> describe(a$Spalte1)
   vars n mean   sd median trimmed mad min max range skew kurtosis   se
X1    1 6 0.33 0.52      0    0.33   0   0   1     1 0.54    -1.96 0.21
```

- `summary()`

```
> summary(a)
    Spalte1          Spalte2
 Min.   :0.0000   Min.   :0.0
 1st Qu.:0.0000   1st Qu.:0.0
 Median :0.0000   Median :0.5
 Mean   :0.3333   Mean   :0.5
 3rd Qu.:0.7500   3rd Qu.:1.0
 Max.   :1.0000   Max.   :1.0
```

Mini-Quiz:

You want to calculate the mean and SD of the variable "dose" from the dataset med. Which code is correct?

a) `mean(dat$dosis)`
b) `mean(med$dosis)`
c) `Mean(med$dosis)`
d) `sd(med$dosis)`
e) `mean(med§dosis)`
f) `mean(med$Dosis)`
g) `mean(dosis)`
h) `library(psych); describe(med$dosis)`

Mini-Quiz:

You want to calculate the mean and SD of the variable "dose" from the dataset med. Which code is correct?

a) `mean(dat$dosis)`
b) **`mean(med$dosis)`**
c) `Mean(med$dosis)`
d) **`sd(med$dosis)`**
e) `mean(med§dosis)`
f) `mean(med$Dosis)`
g) `mean(dosis)`
h) **`library(psych); describe(med$dosis)`**

Row-by-row or column-by-column calculations

- `rowMeans()`
- `colMeans()`

```
> a <- data.frame ("Spalte1" = c(0, 1, 0, 0, 1, 0), "Spalte2" = c(0, 0, 0, 1, 1, 1))
> print(a)
  Spalte1 Spalte2
1       0       0
2       1       0
3       0       0
4       0       1
5       1       1
6       0       1
> rowMeans(a)
[1] 0.0 0.5 0.0 0.5 1.0 0.5
> round(colMeans(a), 2)
Spalte1 Spalte2
   0.33    0.50
```

Frequency tables

- `tab <- table()`
- `prop.table(tab)`
- `addmargins(tab)`

```
> table(Arthritis$Improved)

  None    Some Marked
    42      14      28
> prop.table(mytable)

           Placebo    Treated
None    0.34523810 0.15476190
Some    0.08333333 0.08333333
Marked  0.08333333 0.25000000
> prop.table(mytable, margin = 2)

           Placebo   Treated
None    0.6744186 0.3170732
Some    0.1627907 0.1707317
Marked  0.1627907 0.5121951
> addmargins(mytable)

        Placebo Treated Sum
None         29      13  42
Some          7       7  14
Marked        7      21  28
Sum          43      41  84
```

Analyses by group

```
tapply(variable, grouping variable, function)
```

Treatment	Age
Treated	27
Treated	31
Placebo	55
Placebo	40
Placebo	65

Mean

Mean

```
> tapply(Arthritis$Age, Arthritis$Treatment, mean)
 Placebo  Treated
52.18605 54.58537
>
> tapply(Arthritis$Age, Arthritis$Treatment, sd)
 Placebo  Treated
12.12289 13.45358
>
> tapply(Arthritis$Age, Arthritis$Treatment, function(x) mean(x))
 Placebo  Treated
52.18605 54.58537
>
```

Mini-Quiz: Which code is correct?

a) `tapply(Arthritis$Age, Arthritis$Treatment, mean`

b) `tapply(Arthritis$Age, Arthritis$Treatment, mean)`

c) `tapply(Age, Treatment, mean)`

d) `tapply(Arthritis$Age, mean)`

e) `tapply(Arthritis$Age, Arthritis$Treatment, function(z) mean(z))`

Mini-Quiz: Which code is correct?

a) `tapply(Arthritis$Age, Arthritis$Treatment, mean`

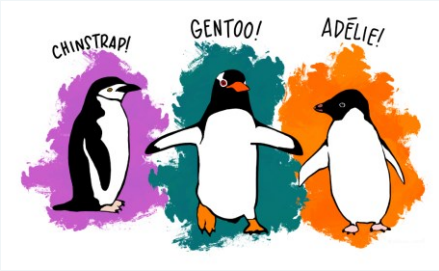
b) **`tapply(Arthritis$Age, Arthritis$Treatment, mean)`**

c) `tapply(Age, Treatment, mean)`

d) `tapply(Arthritis$Age, mean)`

e) **`tapply(Arthritis$Age, Arthritis$Treatment, function(z) mean(z))`**

Exercise



1) Load the dataset “penguins.csv“ (read.csv2())
2) Generate a cross table with `table()` using the variables *species* and *sex* and assign the table to the tab1 or tab2 variable.
   a) without consideration of missing values (NA) → tab1
   b) with consideration of missing values → tab2
3) Generate a table with edge frequencies from tab2 (with missing values).
4) Generate a table of relative frequencies from tab2, rounded to 2 decimal places. Assign this table to a new variable tab3.
5) Convert tab3 to percentages.

Inference statistics

Tests for independence

- **Chi-Square Test:** `chisq.test()`
- **Fisher's Exact Test:** `fisher.test()`
- Preparation: create a crosstab. (`xtabs()`)

Console | Terminal | Jobs

R 4.2.0 · ~/

```
> library (vcd)
Loading required package: grid
> data ("Arthritis")
> head(Arthritis)
  ID Treatment  Sex Age Improved
1 57   Treated Male  27     Some
2 46   Treated Male  29     None
3 77   Treated Male  30     None
4 17   Treated Male  32   Marked
5 36   Treated Male  46   Marked
6 23   Treated Male  58   Marked
> mytable <- xtabs( ~ Treatment + Improved , data = Arthritis)
> mytable
         Improved
Treatment None Some Marked
  Placebo   29    7      7
  Treated   13    7     21
> chisq.test(mytable)

        Pearson's Chi-squared test

data:  mytable
X-squared = 13.055, df = 2, p-value = 0.001463

> fisher.test(mytable)

        Fisher's Exact Test for Count Data

data:  mytable
p-value = 0.001393
alternative hypothesis: two.sided
```

Covariance und Correlation

- `cov()`
- `cor()`
- `cor.test()`

```
> cov(zuf$swls, zuf$fs)
[1] 24.31863
> cor(zuf$swls, zuf$fs)
[1] 0.6519828
> cor.test(zuf$swls, zuf$fs)

        Pearson's product-moment correlation

data:  zuf$swls and zuf$fs
t = 20.565, df = 572, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.6022803 0.6966499
sample estimates:
      cor
0.6519828
```

T tests

- `stats::t.test()`
  - One-sample T test (to test against a fixed value)
    `t.test(x, mu = 0)`
  - t-test for dependent samples
    `t.test(x, y, paired = TRUE)`
  - t-test for independent samples
    `t.test(x ~ group, var.equal = FALSE)`
-

```
> pirate.ttest <- t.test(formula = tattoos ~ sex,
+                        data = dat)
> pirate.ttest # Print result

        Welch Two Sample t-test

data:  tattoos by sex
t = -0.016792, df = 952, p-value = 0.9866
alternative hypothesis: true difference in means between group female
 and group male is not equal to 0
95 percent confidence interval:
 -0.4313270  0.4240083
sample estimates:
mean in group female   mean in group male
            9.431034             9.434694
```

Welch-Korrektur

Mini Exercise

We want to know if a group of students could improve math skills by taking a course. To do this, we have a performance score from each student from a math test before and after the course.
(For this purpose we want to calculate a t-test.)
What is the command in R?

id_student	test1	test2
1	20	34
2	13	12
3	1	7
...	...	...

t.test(, ,)

dat$test1

==

paired

TRUE

true

dat$id_student

dat$test2

=

FALSE

false

Mini Exercise 2

We have 2 groups of mice. One group we fed normally, the other with a special diet. We would like to know if the two groups differ in weight

(For this purpose we want to calculate a t-test.)
What is the command in R?

maus	futter	gewicht
1	normal	20
2	normal	13
3	spezial	19
...	...	...

t.test(, ,)

~

futter

==

paired

TRUE

true

maus

gewicht

=

FALSE

false

Linear Regression

- `mod <- lm(av ~ predictor, data = datensatz)`
- `summary (mod)`

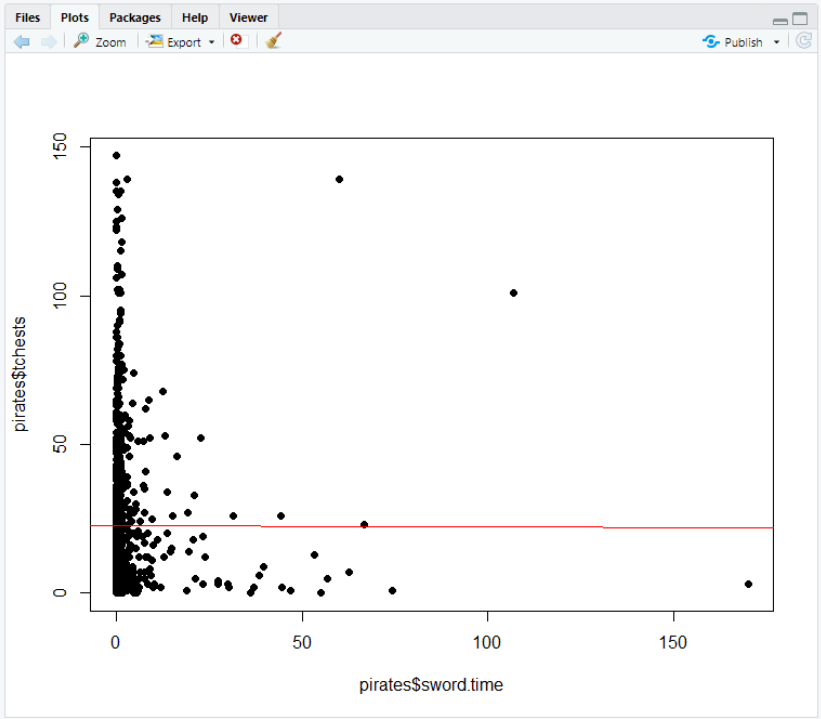
```
> mod1 <- lm(tchests ~ sword.time, data = pirates)
> summary(mod1)

Call:
lm(formula = tchests ~ sword.time, data = pirates)

Residuals:
    Min      1Q  Median      3Q     Max
-22.701 -16.700  -7.701   7.301 124.299

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 22.701452   0.802128  28.302   <2e-16 ***
sword.time  -0.004897   0.082988  -0.059    0.953
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 24.47 on 998 degrees of freedom
Multiple R-squared:  3.489e-06, Adjusted R-squared:  -0.0009985
F-statistic: 0.003482 on 1 and 998 DF,  p-value: 0.953

> coefficients(mod1)
 (Intercept)   sword.time
22.701452051 -0.004897273
```

Linear Regression

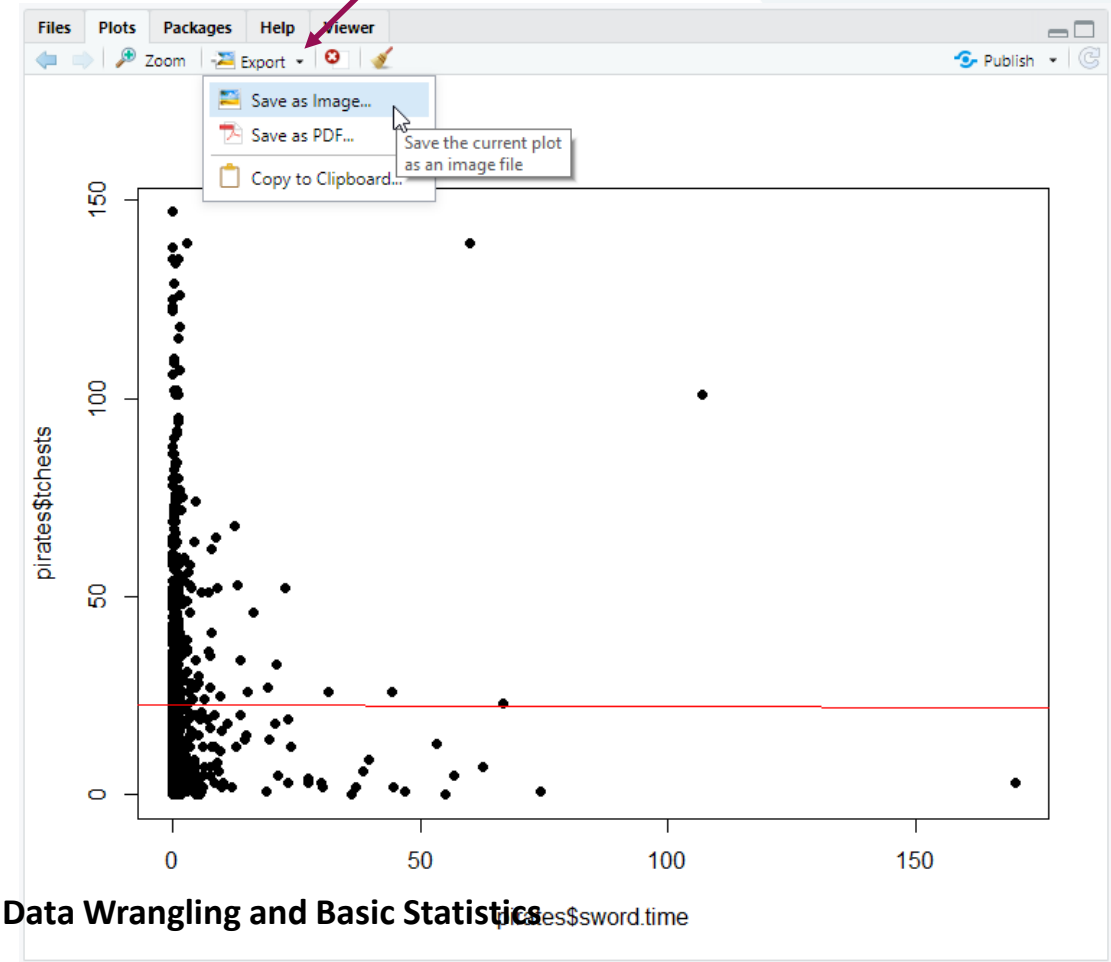
- `plot(av ~ uv)`
- `abline(mod, col="red")`

```
> plot(tchests ~ sword.time, data=pirates, pch=19)
> abline(mod1, col="red")
```

```
> plot(pirates$tchests ~ pirates$sword.time, pch=19)
> abline(mod1, col="red")
```



Save plot


Files
Plots
Packages
Help
Viewer
Zoom
Export
Publish
Save as Image...
Save as PDF...
Copy to Clipboard...
Save the current plot as an image file
pirates$tchests
0
50
100
150
0
50
100
150
pirates$sword.time

Exercise linear regression

1) Load the dataset **pirates.csv** (read.csv2)
2) Calculate a linear regression with criterion weight and the predictor height.
   Create a plot with the regression line.
3) Calculate a linear regression with criterion tattoos and predictor height.
   To do this, create a plot with regression line.
4) Use a t-test to check whether male and female pirates (sex) differ in beard length (beard.length).
   (To do this, first restrict the data set to sex == "male" or "female").
5) Load the dataset **bfik.sav**
   Check if the variables sex and uni are statistically independent.
   (bfik$sex once contains the value 99. First, recode this to NA.)

Plots

Basically …

- Plots have multiple purposes, for which different workflows are suitable.
  - **exploratory** plots: get to know data, get impression, fast, not meant for others
  - **expository** plots: plots for publications or reports
- for fast, explorative graphics the standard plots of R (with base R) are suitable
- for detail-oriented plots the functions from the package ggplot2 are suitable
- We first look at base plots!

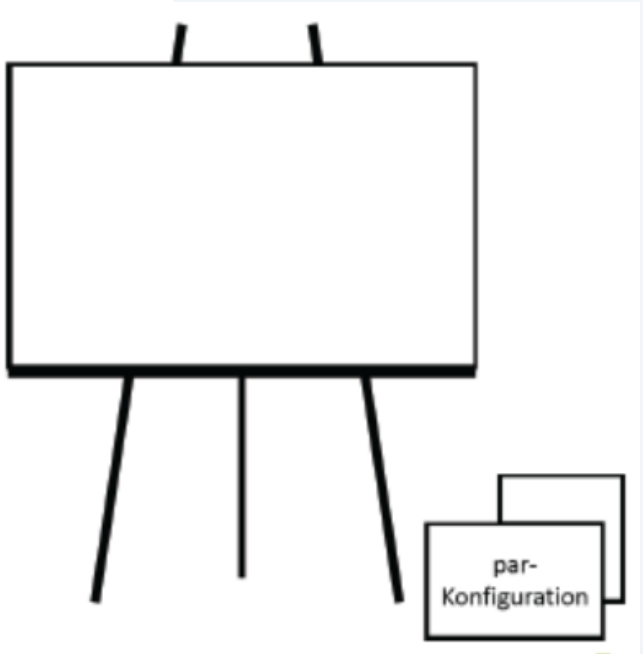
Logic of plots in R

- Plots in base are applied in three layers:

1. **Graphic parameters** (layout, size,…)
2. **high-level plot functions** (scatterplot, bar diagram, histogram,…)
3. **low-level plot functions** (linies, dots, legend, titles,…)

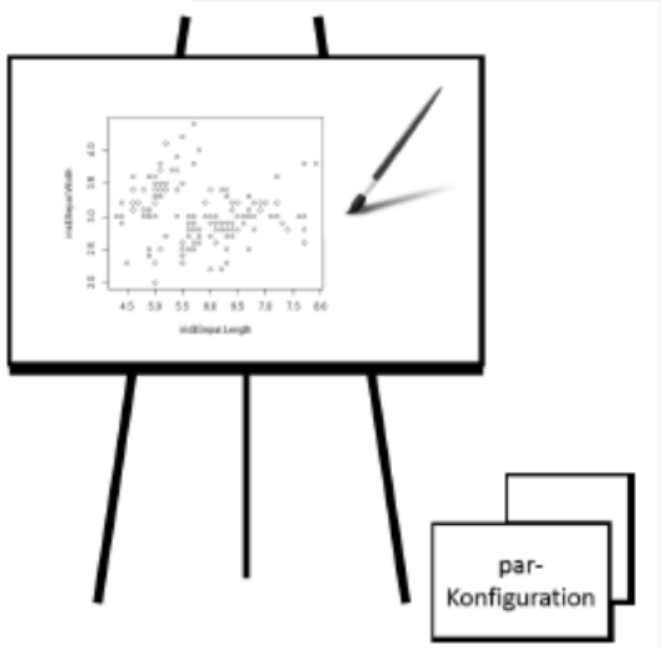
Graphics parameters

- The graphic parameters determine the "canvas" on which the plot is mapped.
- layout, size,…



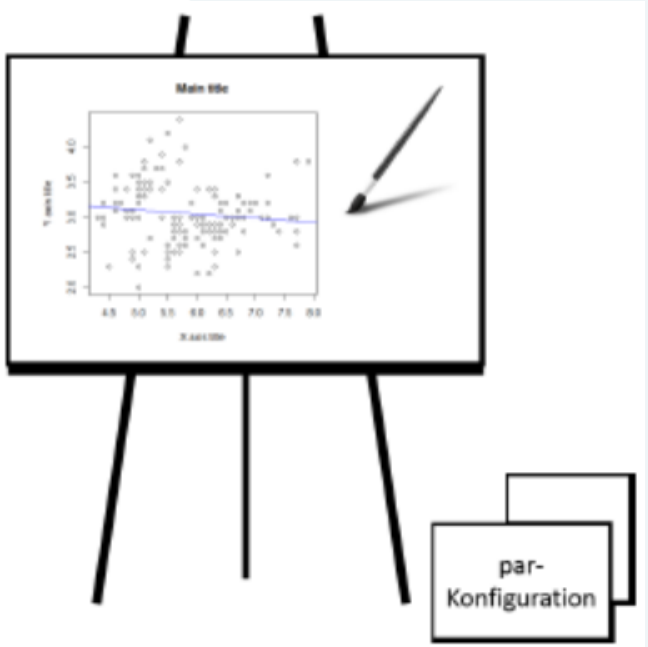
High-Level plot functions

- These functions determine the plot type. The application of such a function causes a new plot to be drawn.
- scatterplot, bar diagram, histogram,…



Low-Level plot functions

- Low-level plot functions are "painted" or added to existing graphics.
- lines, dots, legend, titles,…



First plots in R

Function	Description
`plot()`	context-dependent, generic function
`barplot()`	Bar diagram
`boxplot()`	Boxplot
`hist()`	Histogram
`density()`	Density distribution

The function `plot()` is generic and can be used on different types of objects. Depending on the input `plot()` draws different diagrams.

Procedure for the creation of plots

- plots are built up step by step in the script
- `par()` sets plot parameters
- To save plots, you use various graphic devices with the functions: `pdf()`, `png()`, `jpeg()`, `bmp()`, `tiff()` before the plot functions and `dev.off()` after

```
par(lwd=2)    # Linienstaerke festlegen
pdf(file.path(pfad, "Output", "Grafik1.pdf"))    # als PDF speichern
plot(dose, drugA, type="b")    # Plot
title("Meine Grafik")    # Titel hinzufuegen
dev.off()    # PDF fertigstellen
```

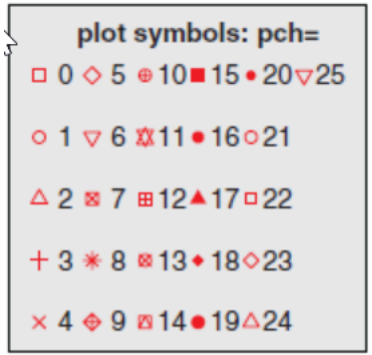
Plot parameters

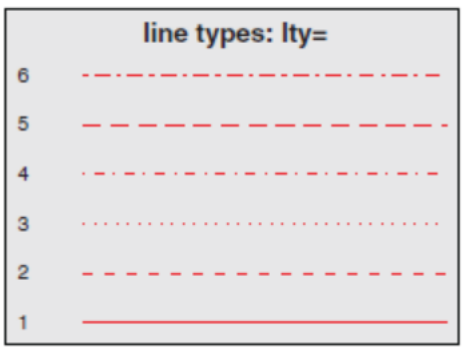
- the following parameters can be used with `par()` or in plot functions:

Parameter	Description
axes	Axes should (not) be drawn
bg	Background color
cex	Size of a dot / letter
col	Colors
las	Alignment, axis labeling
lty, lwd	Line type, line width
main, sub	Title, subtitle
mfcol, mfrow	multiple plots in one image
pch	Symbol for a dot
type	Type (l = line, p = dot, b = both, n = nothing)
xlab, ylab	Axis labeling
xlim, ylim	Limits for axes

Plot parameters: symbols and lines

- `plot(dose, drugA, type="b", lty=3, lwd=3, pch=15, cex=2)`





Plot parameters: colours

- Colours can be displayed with index, name, hexadecimal representation, as RGB (red, green, blue) or HSV (hue, saturation, value): `col=1`, `col="white"`, `col="#FFFFFF"`, `col=rgb(1,1,1)`, `col=hsv(0,0,1)`.
- with the functions `colors()` you can view all possible colours.
- Additional functions generate vectors with color schemes: e.g.: `rainbow()`, `heat.colors()`…
- `scales::show_col()` can display colors in the viewer window
- List with all colors at: http://research.stowers.org/mcm/efg/R/Color/Chart/ColorChart.pdf
- wesanderson Package: https://github.com/karthik/wesanderson

low-level plot functions

- After a plot is generated, low-level functions can be used to add and modify text, titles, legends and axes.

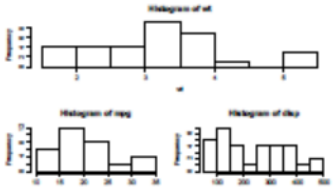
Parameter	Description
`abline()`	Line
`arrows()`	Arrows
`axis()`	Axes
`grid()`	grid
`legend()`	legend
`lines()`	lines (step by step)
`points()`	points
`polygon()`	(filled) polygons
`pretty()`	better division of the axes
`text()`	text
`title()`	Labeling

Combination of plots

- The functions `par()`, `layout()` enable the mapping of multiple plots simultaneously.
  - `par(mfrow=c(nrows, ncols))`: Fill row by row
  - `par(mfcol=c(nrows, ncols))`: Fill column by column, z.B.
  - `par(mfcol=c(1, 2))` results in two graphics side by side
  - `layout(mat)`: Matrix contains coordinates of the individual plots

```
layout(matrix(c(1,1,2,3), 2, 2,
       byrow = TRUE))
hist(wt)
hist(mpg)
hist(disp)
```

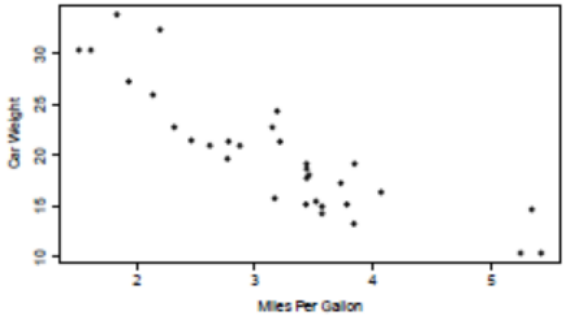
```
     [,1] [,2]
[1,]    1    1
[2,]    2    3
```



Scatterplots

- Scatterplots are created by `plot(x, y)` and show the correlation of two variables.

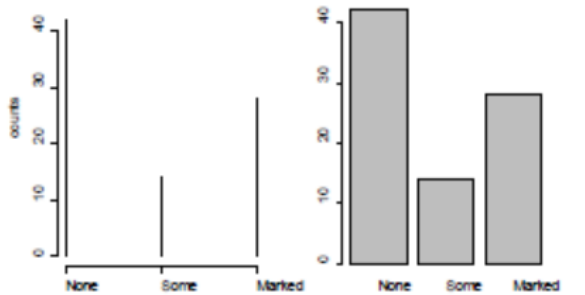
```
plot(mtcars$wt, mtcars$mpg, pch=19,
     xlab="Miles Per Gallon",
     ylab="Car Weight")
```



Bar diagrams

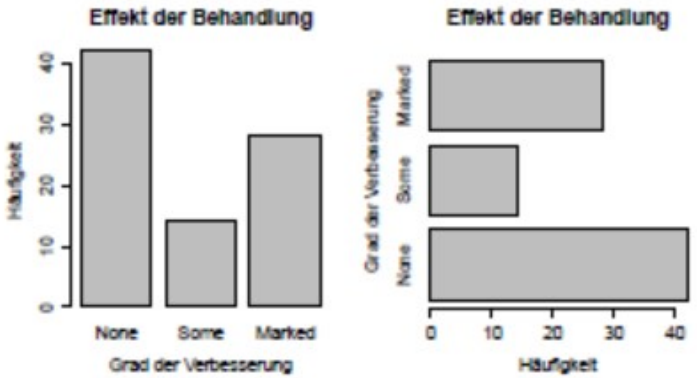
- Bar diagrams are created with `barplot(height)` and are important for representation of frequency distributions(`table()`) with categorical data.

```
counts <- table(Arthritis$Improved)
par(mfcol=c(1,2))
plot(counts)
barplot(counts)
```



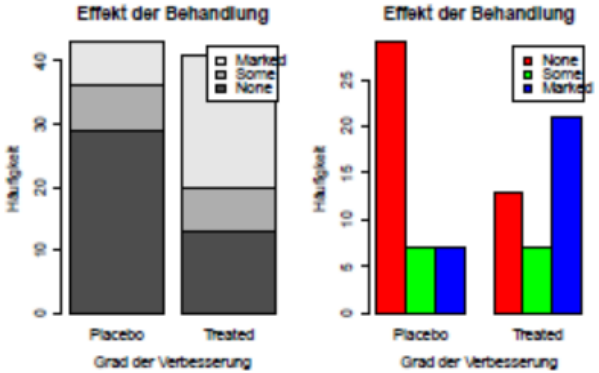
Bar diagrams – new arguments

```
barplot(counts, main = "Effekt der Behandlung",
        ylab = "Grad der Verbesserung",
        xlab = "Häufigkeit",
        horiz = TRUE)
```



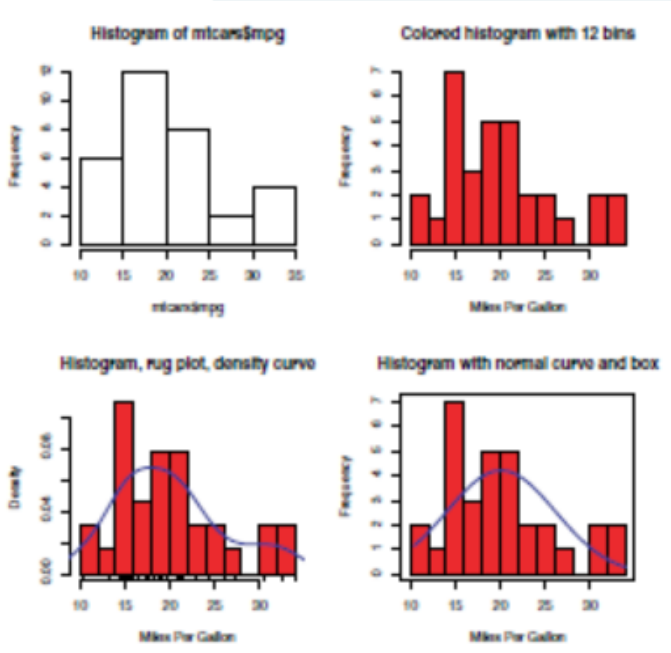
Bar diagrams – multidimensional matrices / data frames

```
counts <- table(Arthritis$Improved, Arthritis$Treatment)
barplot(counts, main = "Effekt der Behandlung",
        xlab = "Grad der Verbesserung", ylab = "Häufigkeit",
        legend = rownames(counts),
        col = c("red", "green", "blue"),
        beside = TRUE)
```



Histograms

- Histograms are created with `hist()`, `density()` and are used to represent distributions for continuous variables.

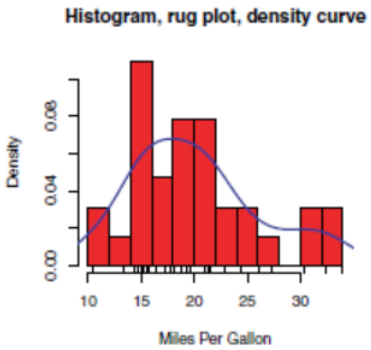


Histograms

- Histogram with a Rug Plot and Density Curve

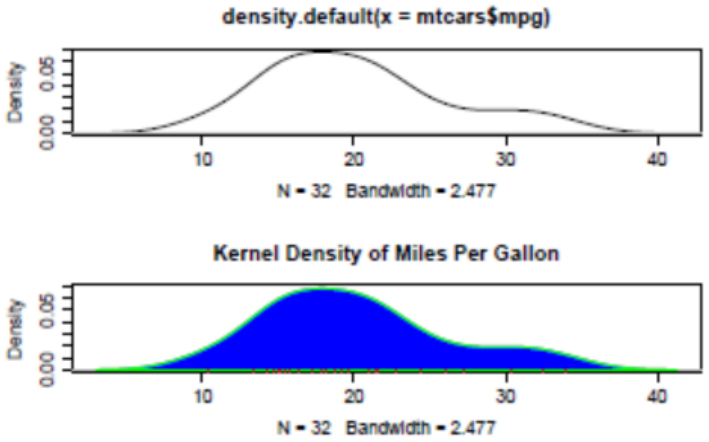
Beispiel: Histogramm mit rug plot und density curve

```
hist(mtcars$mpg,
     freq=FALSE, # Dichte statt Häufigkeit
     breaks=12,  # Anzahl Kategorien
     col="red",  # Farbe
     xlab="Miles Per Gallon",
     main="Histogram, rug plot, density curve")
rug(jitter(mtcars$mpg)) # jitter() bei vielen ties
lines(density(mtcars$mpg), col="blue", lwd=2)
```



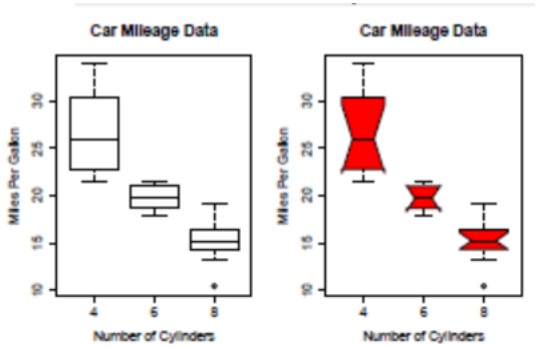
Density Plots

```
d <- density(mtcars$mpg)
plot(d, main="Kernel Density of Miles Per Gallon")
polygon(d, col="blue", border="green") # Für Füllung
rug(mtcars$mpg, col="brown")
```



Boxplots

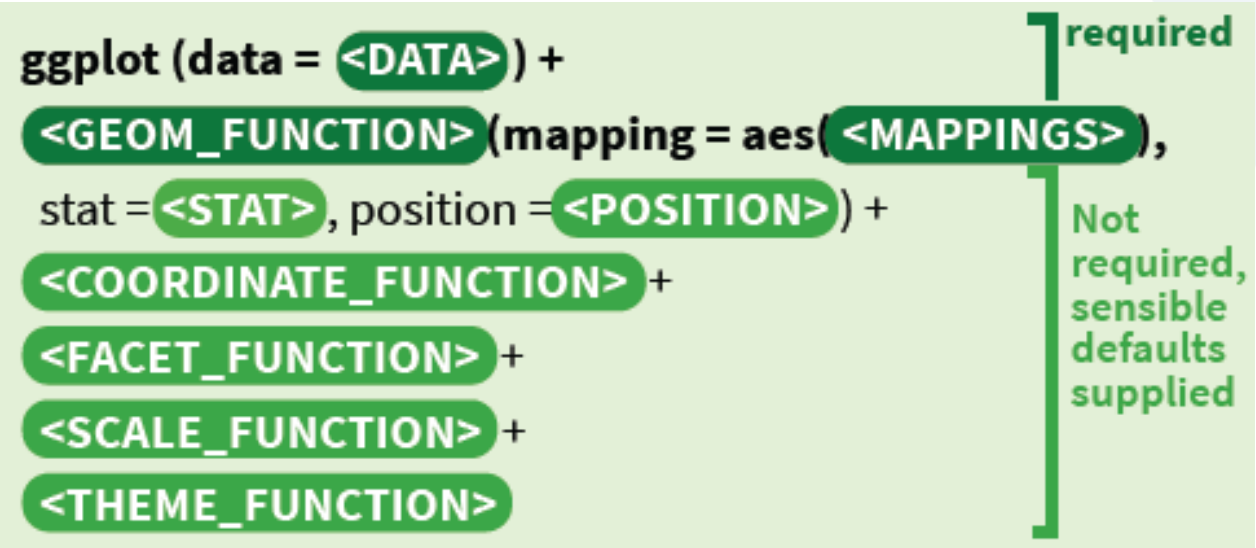
- Box-and-whiskers Plots graphically show the five-number statistic: min, lower quartile, median, upper quartile, max (`boxplot.stats()`).
- Boxplots require the formula notation y ~ A*B.



If this is not enough: ggplot2

- What is ggplot2?
  - ggplot2 is a popular package for creating plots. It is more flexible than base-graphs, but also needs a little more training.
  - ggplot2 is based on the grammar of graphics, the idea that every plot consists of the same components: a data set, geometric elements and a coordinate system (more on that here: Wickham, 2010).
  - Compared to base plots, ggplot2 plots require a bit more code, but are customizable in all details.
  - While base plots are displayed immediately, ggplot2 plots are objects that are customizable.
  - ggplot2 works best with tidy data, i.e. each row is an observation and each variable is a column (see Wickham, 2014)

Syntax of ggplot2

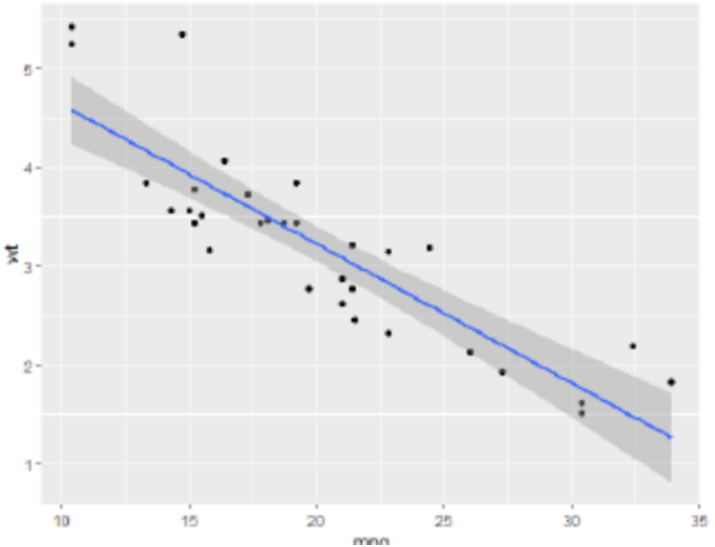



```
ggplot() +
    geom_*(aes()) +
    stat_*() +
    coord_*() +
    facet_*() +
    scale_*_*() +
    theme_*()
```

- Cheatsheet here: https://www.rstudio.com/resources/cheatsheets/

Syntax of ggplot2 (Example)

```
ggplot(mtcars, aes(x = mpg, y = wt)) +
  geom_smooth(method = "lm") +
  geom_point()
```



The most common geom functions

Function	Description	Pic
geom_bar(); geom_col()	Bar chart - e.g. frequency of categorical expressions	
geom_point()	Scatterplot - Relationship between two continuous variables	
geom_smooth()	Fit line for a scatterplot, e.g. linear regression or smoothing	
geom_line()	Line chart - e.g. change over time	

geom_abline();
geom_hline();
geom_vline()
	single lines (diagonal, horizontal, vertical), e.g. as reference	

geom_histogram()	Histogram - distribution of a variable	
geom_violin()	Violinplot - distribution of a variable grouped by categories	
geom_boxplot()	Boxplot - distribution of a variable grouped by categories	

Most common other functions

Function	Description
theme_set(theme_bw())	Fix the theme for the upcoming plots (in this case theme_bw())
coord_equal()	x- and y-axis get the same coordinate system
coord_flip()	Switch x- and y-axis (e.g. for long labels)
facet_grid(~ variable) / facet_wrap()	Split plot according to one or more categorical variables
scale_x_continuous(breaks = …, limits = …)	Boundaries and hyphens for coordinate axes (also works with y)
scale_fill_manual()	determine own colors for fill aesthetics

other useful functions

Function	Description
annotate()	single note in the plot (e.g. correlation, p-value,...)
labs(title, subtitle, caption, x, y, fill,...)	Title for the plot, the axes, the signature…
theme(legend.position = „none“)	Remove legend
theme(axis.text.x = element_text(angle = 90))	Rotate text, in this case labeling the x-axis by 90°.

Help for Plots

- base:
  - R Cookbook: http://www.cookbook-r.com/Graphs/
  - R Base Graphics - An Idiot's Guide: https://bit.ly/2YrlDg8
  - The Base Plotting System in R: https://bit.ly/2UVxFMH
- ggplot2:
  - ggplot2-Homepage: https://ggplot2.tidyverse.org/index.html
  - ggplot2-Cheatsheet: http://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf
  - Package: ggpubr for publication ready plots (https://rpkgs.datanovia.com/ggpubr/)
  - Kapitel "Data visualisation" in "R 4 Data Science": https://r4ds.had.co.nz/data-visualisation.html
  - Kieran Healys "Data Visualization - A practical introduction": http://socviz.co/
- both:
  - R Graph Gallery: https://www.r-graph-gallery.com/

Exercise – Plots

- Create a colored bar chart with legend from tab1 (from the last exercise).
- Load the dataset zuf.sav and create a histogram of the variable swls and a density plot (package car) of the variable fs. Both graphs should appear side by side in one window.
- Load the data set data(WorldPhones) with data(WorldPhones); dat <- WorldPhones. In it, find the number of landlines between 1951 and 1961 in 7 world regions. Create a common barplot for North America and Europe that shows the progression over the years, using different colors for them.
- (Choice between base and ggplot2 - bonus points if both are used!)

Exercise – Plots

- we read the dataset penguins with
  - `penguins <- palmerpenguins::penguins` (`install.packages(„palmerpenguins“)`, if not yet available)
  - Explore the data set and some questions that come to mind about it using plots.
  - Afterwards, feel free to switch to your own data sets and explore them with plots.

Reporting with RMarkdown

What is RMarkdown?

- R Package
- Reports can be created (e.g. as PDF or HTML).
- They contain R code and can calculate and update their own content.
- Advantage: easily adaptable, e.g. to new data.
- file format: .Rmd or .Qmd

- You need:
  - Packages: rmarkdown, knitr
  - maybe further packages (e.g. kableExtra for tables)
  - .Rmd file
  - maybe. LateX, pandoc (is usually installed automatically)

Elements of an .Rmd file

- YAML header

```yaml
---
title: "RMarkdown Teaser"
author: 'Workshop: Einführung in R'
date: "28.06.2022"
output:
  pdf_document: default
  html_document:
    self_contained: yes
    theme: readable
    highlight: default
    toc: no
---
```

- Comments
  - <!-- in text-->
  - # in code

- Code Block

Options for code block

```
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
library(knitr)
library(kableExtra)
```
```

- Text with headings

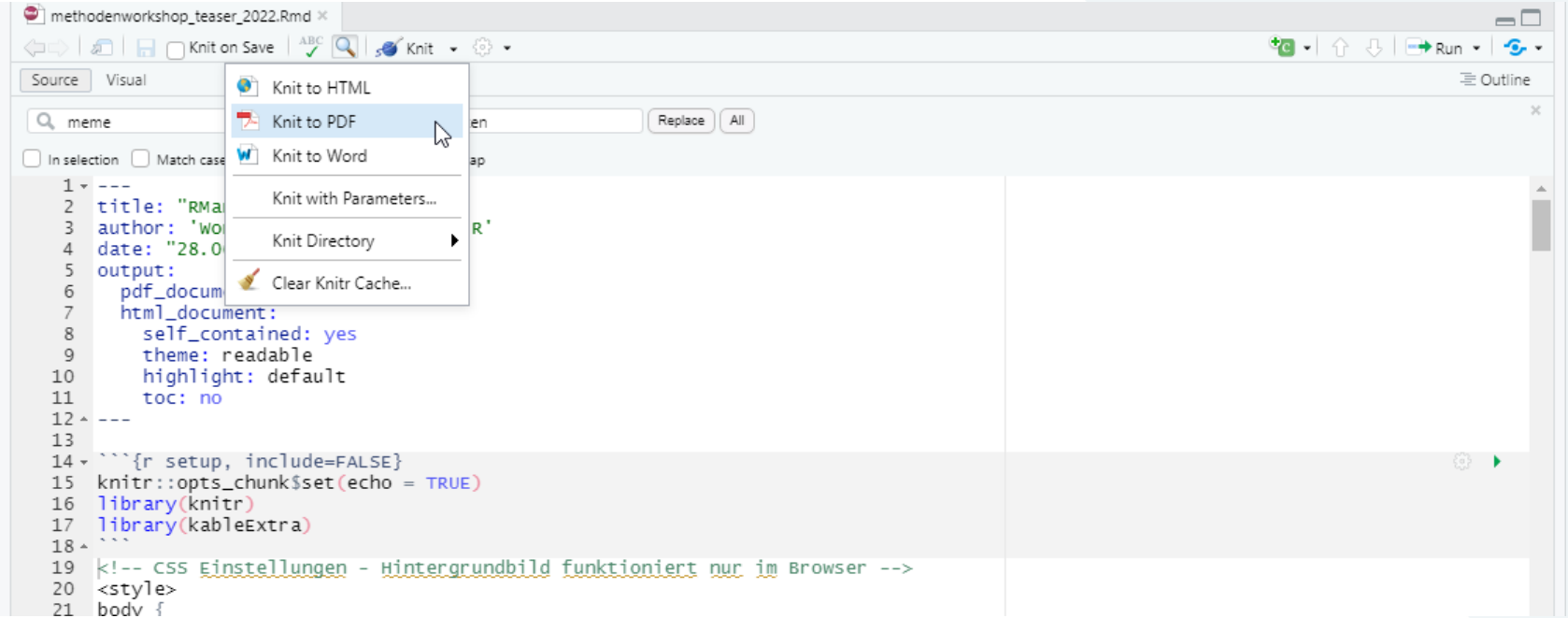
```
### R Markdown Dokument

Das hier ist ein *R Markdown* Dokument. Es kann
automatisierte Berichte erzeugen, die mit neuen


### Formatierung

Mit ein paar einfachen Tricks kann man ein schön
z.B. mit `# Überschrift` bzw. `## Überschrift` k
```

„Knitting“ an RMarkdown makes a PDF (or…)



RMarkdown Teaser

Workshop: Einführung in R

28.06.2022

Markdown

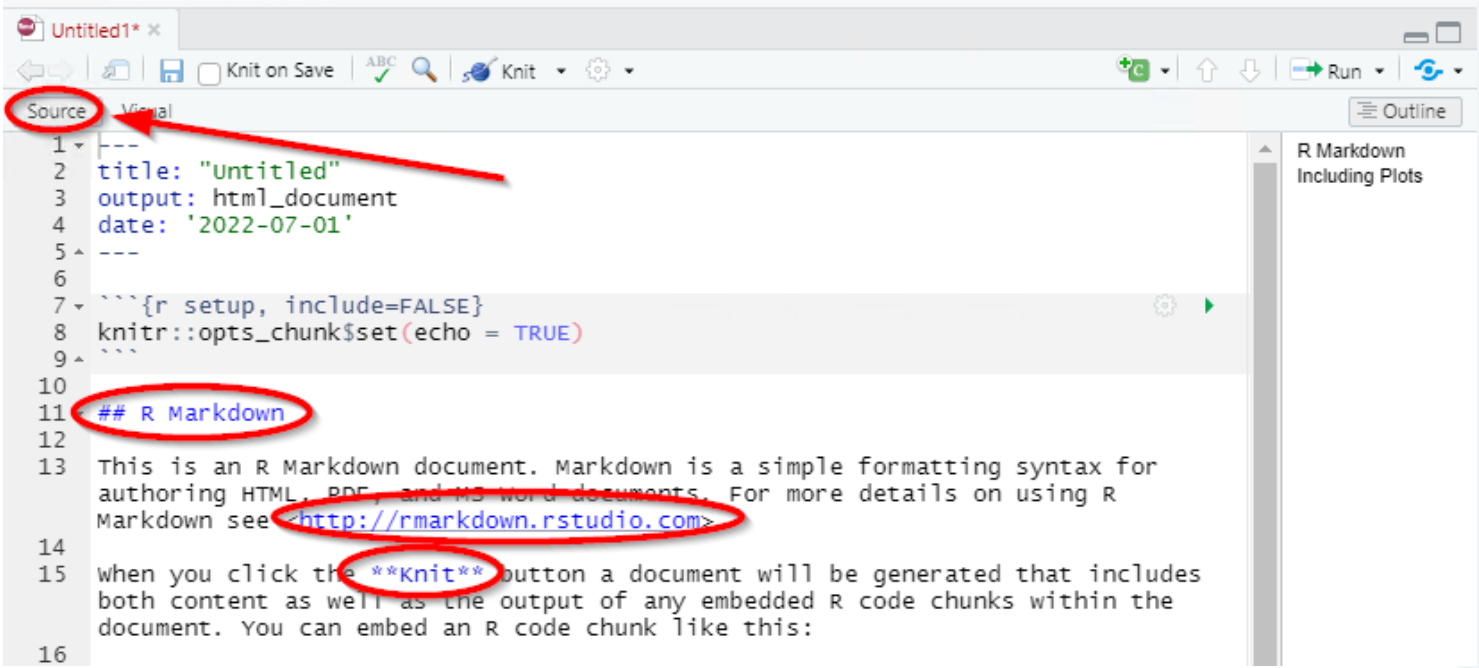
R Markdown Dokument

Das hier ist ein *R Markdown* Dokument. Es kann R Code ausführen und die Ergebnisse darstellen. Damit lassen sich z.B. automatisierte Berichte erzeugen, die mit neuen Daten vorgefertige Analysen ausführen.

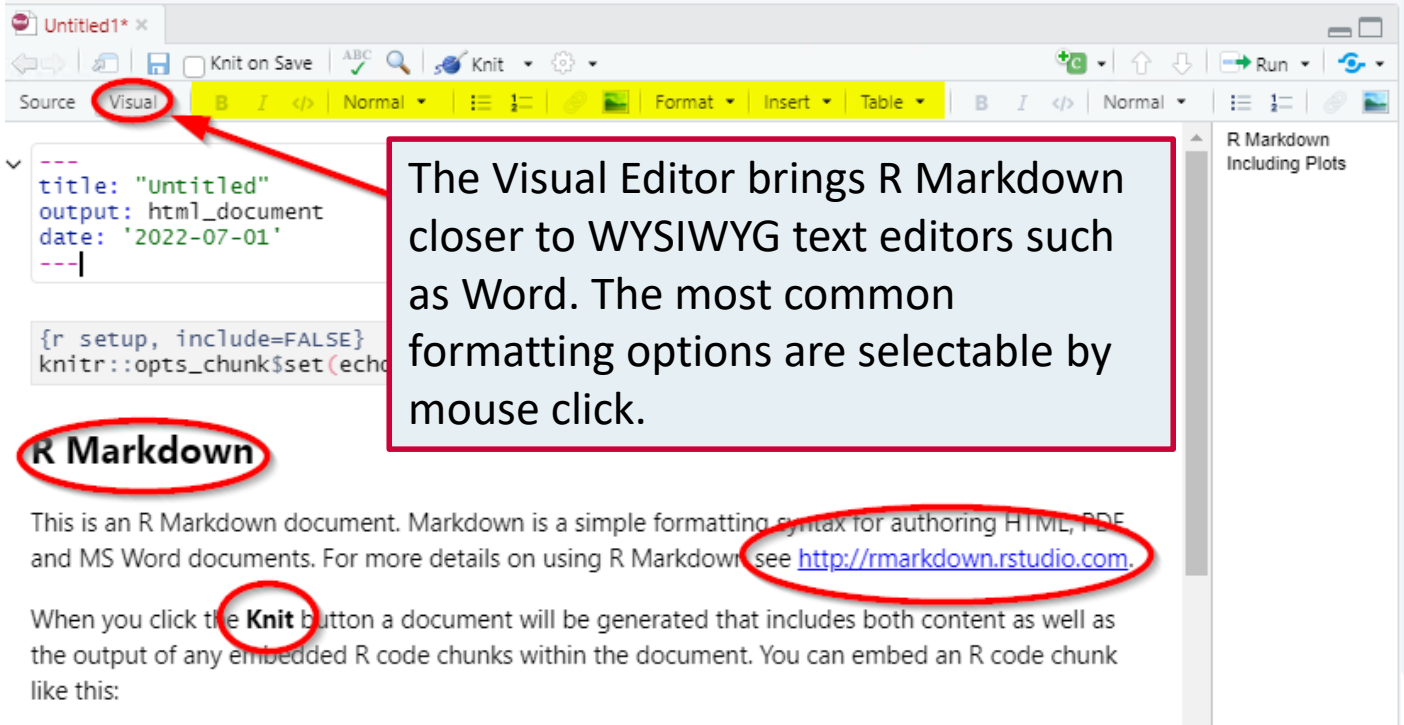
Formatierung

Mit ein paar einfachen Tricks kann man ein schönes und ev. interaktives **HTML- oder pdf-Dokument** erstellen.
Z.B. mit `# Überschrift` bzw. `## Überschrift` kann man Überschriften in unterschiedlicher Größe erzeugen

Source Editor



Visual Editor



Programming in R

for loops

- One would like to repeat commands
- *for loop*
  for(*looping variable* in *area*)
  {*LOOP.CODE*}
  - *while* loop
    while(*looping variable* in *area*)
    {*LOOP.CODE*}

```
> for(ii in 1:5){
+   print(ii)
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
>
```

- You want to execute commands only under certain conditions
  - *if loop*

```
> ii <- 1
> while(ii < 5){
+   print(ii)
+   ii <- ii + 1
+ }
[1] 1
[1] 2
[1] 3
[1] 4
```

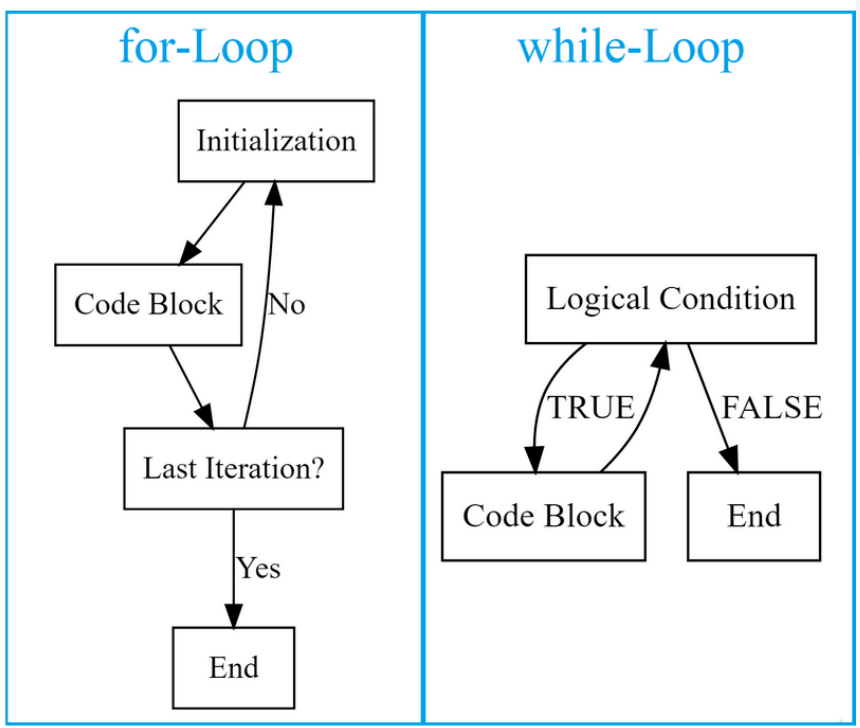
for loops

- *for loop* to create multiple plots.

  for(*looping variable* in *vector*)
  {*LOOP.CODE*}

```
> par(mfcol=c(2,2))
> vars <- c("geschl", "alter", "swls", "fs")
> for(ii in vars){
+   #   ii <- vars[1]
+   dat <- zuf[, ii]
+   hist(dat, main=ii)
+   print(ii)
+ }
[1] "geschl"
[1] "alter"
[1] "swls"
[1] "fs"
```

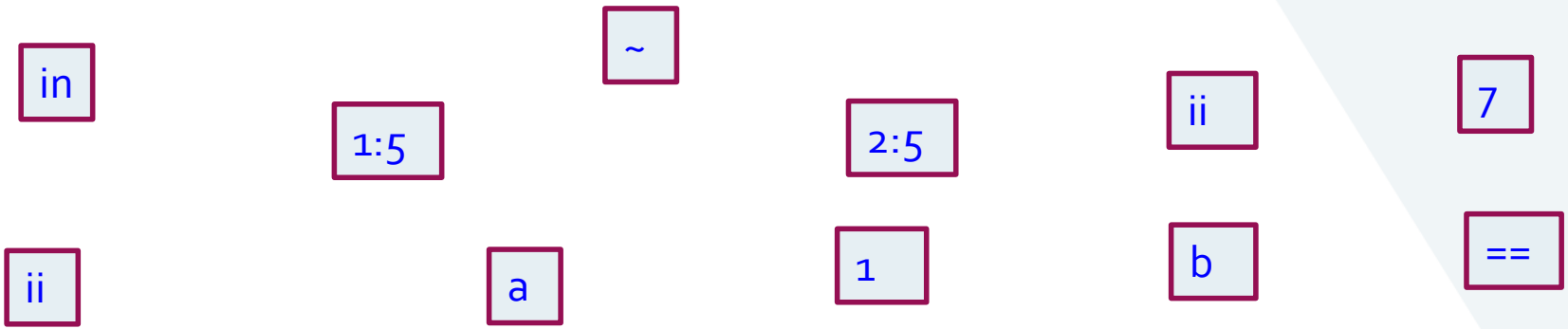
Loops


for-Loop
Initialization
Code Block
No
Last Iteration?
Yes
End
while-Loop
Logical Condition
TRUE
FALSE
Code Block
End

Mini Exercise 1

We want to output the numbers 2 to 5 to the console. For this we want to use a loop. What is the command in R?

for( ) { print( ) }

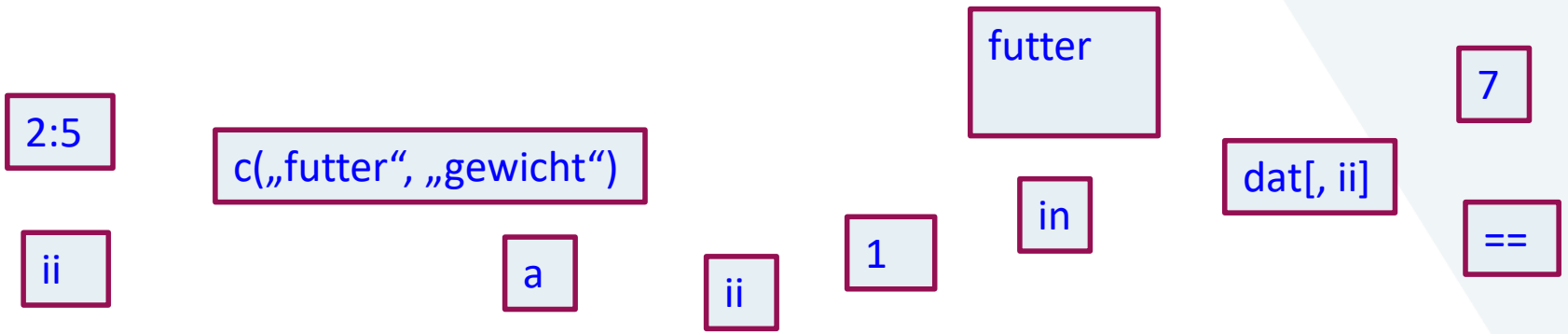


Mini Exercise 2

We want to output a histogram of each of the variables food and weight? For this we want to use a loop. What is the command in R?

maus	futter	gewicht
1	normal	20
2	normal	13
3	spezial	19
...	...	...

for( ) {hist( ) }



if loops

- You want to execute commands only under certain conditions
- *if loop*
  else if
  else

  if(*condition*)
  {*LOOP.CODE*} else {

  ALT.CODE}

  if(*condition1*)
  {*LOOP.CODE*} else if (*condition2*) {

  ALT.CODE} else {OTHER.CODE}

```
> # if Schleife
> x <- sample(c(TRUE, FALSE), 1)
> if(x == TRUE) {print("x was true!")}
> if(x == FALSE) {print("x was false!")}
[1] "x was false!"
```

Examples

- if statement

```
if(x > 0){
print("Positive number")
}
```

- if…else statement

```
if(x > 0){
  print("Non-negative number")
  } else {
  print("Negative number")
  }
```

- if…else ladder

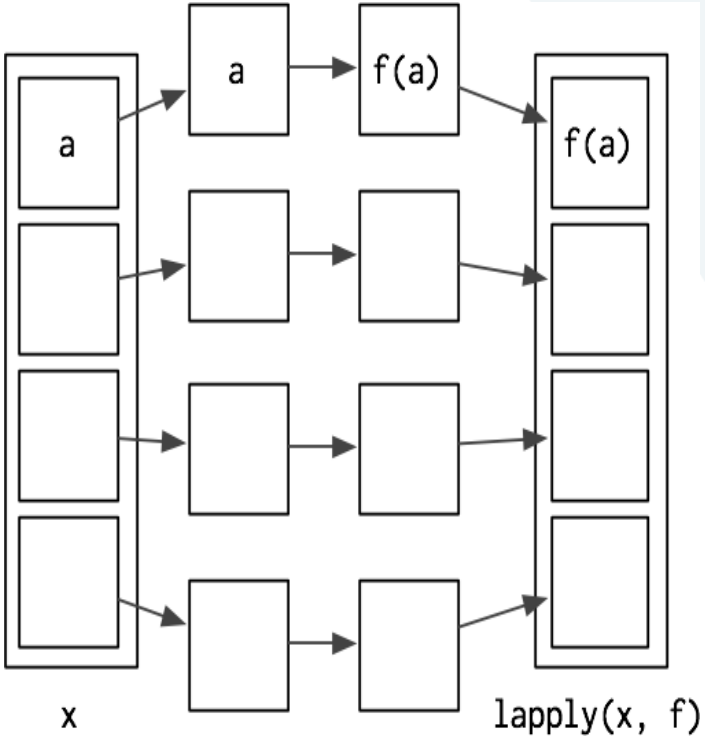
```
if(x < 0){
      print("Negative number")
      } else if(x > 0) {
      print("Positive number")
      } else {
      print("Zero")}
```

apply functions

Apply functions can be used instead of for-loops (and are often but not always faster):

- `apply(data, direction, function)`:
  for data frames, matrices and arrays.
- `lapply(list, function)`: for lists and vectors (output = list)
- `sapply(list, function)`: for lists and vectors (output = simplified structure)
- `tapply(data, groups, functions)`: for groups (output = table)

lapply function



- Principle: apply(data, direction, function)
- direction: 1=row wise, 2=column wise

Example:
Here the minimum value (min) is determined for each column (MARGIN = 2).
Additional arguments (na.rm=TRUE) are simply appended.

```
> vars <- c("alter", "swls", "fs")
> apply(zuf[,vars], MARGIN = 2, min, na.rm=T)
alter  swls    fs
   16     9    16
```

Exercise

1) Calculate the square of the numbers 1 to 10.
   Use a for loop and display the respective number and its square.
2) Create a vector with the numbers 1 to 10.
   Randomly draw a number from it (sample(vector, 1)).
   Print the drawn number.
   Check with if()..., whether the drawn number is less than 5 and output "The number is less than 5.".

Debugging

```
Error in setwd(pfad) : cannot change working directory
> #*************************************************************************************
> # Daten einlesen ####
> {
+   # meine Daten
+   dat0 <- read.csv2(file.path(pf_roh, "meine_rohdaten.csv"))
+
+   # meine Fragebogen-Daten dazu
+   fragebogen <- read.xlsx(file.path(pf_roh, "mein_fragebogen.xlsx"), sheet = 1)
+ }
Error in file(file, "rt") : cannot open the connection
In addition: Warning message:
In file(file, "rt") :

Error in file(file, "rt") : cannot open the connection


> #*************************************************************************************
> # Daten aufbereiten ####
> {
+   # Sortieren
+   # Filtern
+   # Rekodieren
+   # Variablen erstellen
+   # Zusammenfuegen
+   # ...
+   dat <- dat0[!is.na(dat0$meine_filtervariable), ]
+ }
Error: object 'dat0' not found
```

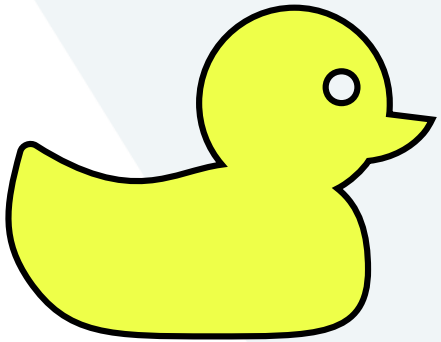
Basis Rules when errors occur

- stay calm
- see if an error message has occurred before
- try the same again
- restart R and then try again (Session > Restart R)
- Break the problem down into small parts: where does R get stuck?
  - am I giving the function what I expect? View data, view substeps
  - for nested functions, execute the interior individually
  - Recreate function with simpler data
  - look up examples of the function (F1, ?function oder help(„function“)) and look for differences in the application

https://speakerdeck.com/jennybc/object-of-type-closure-is-not-subsettable

Rubber duck method

- https://en.wikipedia.org/wiki/Rubber_duck_debugging
- Talk to a friend or colleague (or even a rubber duck) and explain your code to them. They don‘t need to understand it or solve your problem. By explaining the code line by line you might discover the problem by yourself.



Common error messages and their meaning

Error message	Description	common solution
Error: object ‚…' not found	something was executed that is not present in the environment	check whether the requested object is in the environment. If not, assign again. Often this error is not the cause, but the consequence of an error above.
Error: could not find function „…“	Function cannot be found	Package is not loaded (→ library()) or function has a typo. Load package or refer to it directly in function package::function; check for typos.
Error: unexpected numeric constant in „mean(c(1. 4.“	R finds a number where it does not expect it	Are all commas, brackets, etc. where I want them? The numbers should have been separated by commas, not a period. "Unexpected..." also often comes up with typos, spaces,....
Error: incorrect number of dimensions	Indexing hits different dimensions than expected	Is what I want to index in the structure I expect? e.g. my_df$variable[,] Through the $ my data frame becomes a vector and therefore does not need a comma in the square brackets anymore
Error in file(file, „rt“): cannot open the connection	a file cannot be found	File path is wrong or there are no read rights to the file, is getwd() where I expect it?
Error: object of type closure is not subsettable	type closure = function	Here R interprets something as a function that is not a function. Occurs when e.g. square brackets are confused with round brackets: mean[] (function "mean" can not be indexed)
R is not ready (>)	R does nothing despite running lines	R is either ready for new code (> in the console) or waiting for more code (+ in the console) Here a closing bracket is often missing) or } Wait status can be canceled by pressing Esc

Thanks for your attention!

Gabriele von Eichhorn, MSc BSc BA
IQS – Institut des Bundes für Qualitätssicherung im österreichischen Schulwesen
gabi.v.eichhorn@gmail.com / gabriele.eichhorn@iqs.gv.at