

# Einführung in R:

## II. Working with Data



Marie A. Jakob

SUMMER SCHOOL KOGNITIVE MODELLIERUNG 2022

Image Credit: Allison Horst,

<https://github.com/allisonhorst/stats-illustrations>



# Ablauf

---

- Das Drumherum: Daten einlesen, RStudio, Ordner-Strukturen und Projects
- Umgang mit Daten
- ***PAUSE***
- ***Übungsaufgaben***
- Plots
- ***Übungsaufgaben***
- ***PAUSE***
- Getting Help – Debugging, Hilfe und R Ressourcen

# Das Drumherum

# Computer Stuff



Image Credit: [https://en.wikipedia.org/wiki/Matrix\\_digital\\_rain](https://en.wikipedia.org/wiki/Matrix_digital_rain)

# Warum brauche ich zwei Programme?

---

- R: Programmiersprache (primär) für statistische Anwendungen
- RStudio = „Integrated Development Environment“ (IDE)
  - Zur benutzer:innenfreundlichen Anwendung von R 😊
  - Kann R-Skripte öffnen und ausführe
  - + diverse hilfreiche Dinge (z.B. RStudio Projects)

**[Demo in R]**

# Daten einlesen

---

- Wo kommen Daten her?
    - Fragebögen (eingegeben in SPSS / Excel), Online-Umfrage, Computer-Experiment etc.
  - Was sind gängige Datenformate?
    - .xlsx / .xls: Excel-Dateien
    - .sav: SPSS-Datenfiles
    - .csv / .tsc / .txt: Textdateien (.csv = „comma separated values“)
- etc.

# Daten einlesen – Befehl

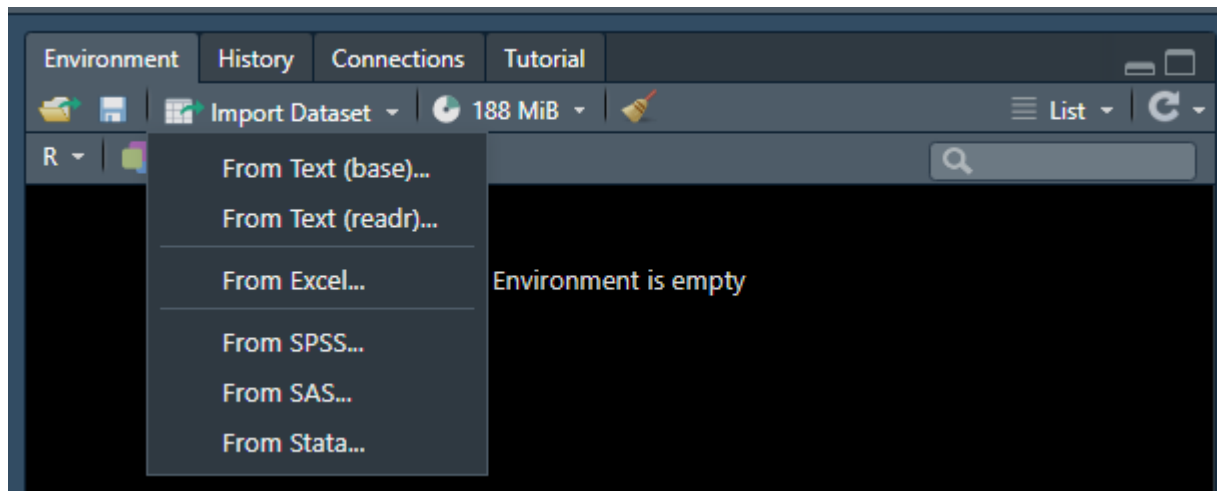
Befehl für .csv-Dateien: `read.csv()`

Benutzung:

```
data_raw <- read.csv("~/path/to/file", # path to the csv-file
                    header = T, # first row contains variable names
                    sep = ",", # values are comma-separated
                    dec = ".", # decimal character is "."
                    ...)
```

# Daten einlesen – Rstudio Shortcut

- Button in RStudio „Import Dataset“:



- Vermeidung von Pfad-Fehlern ☺
- Wichtig: generierten Code in Skript einfügen → Reproduzierbarkeit

**[Demo in R]**

# Packages

---

- Einlesen von Excel- und SPSS-Dateien erfordert Installation sog. Packages:
  - → R Packages stellen spezialisierte Funktionen für (mehr oder weniger) spezifische Aufgaben bereit
  - ... wie bspw. das Einlesen spezieller Dateiformate

- Installation (einmal)

```
install.packages("package")
```

- Laden (bei jeder Anwendung)

```
library(package)
```



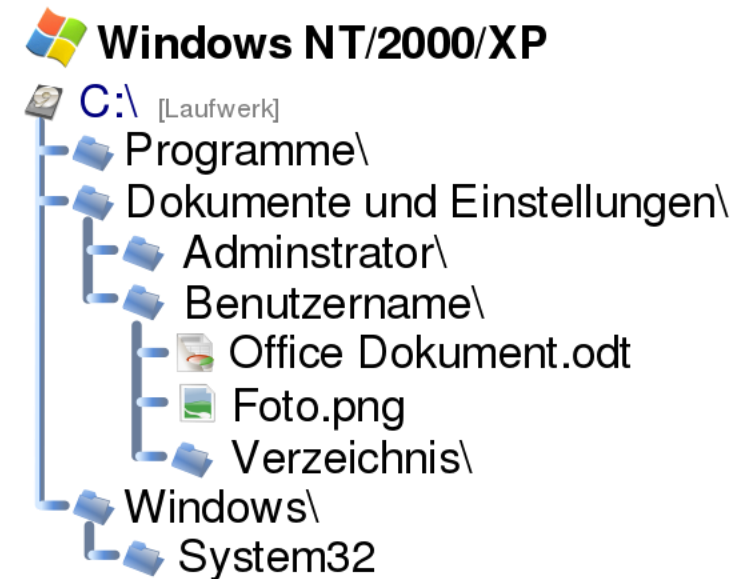
# Exkurs: Pfade & Struktur von PC-Dateisystemen

<https://de.wikipedia.org/wiki/Verzeichnisstruktur>

- Pfad auf meinem Computer:

"K:\Eigene Dateien\Lehre\2022-modeling-summer-school\02\_R\_Data"

- Verzeichnisstruktur hat Baum-Struktur:
- Absolute Pfade: vom Laufwerk (der „Wurzel“ des Baums) aus
  - "K:\Eigene Dateien\Lehre\2022-modeling-summer-school\02\_R\_Data"
  - → Auf jedem PC unterschiedlich
- Relative Pfade: vom working directory aus
  - "~\2022-modeling-summer-school\02\_R\_Data"



# Workflow & RStudio Projects

Für jede Datenanalyse:

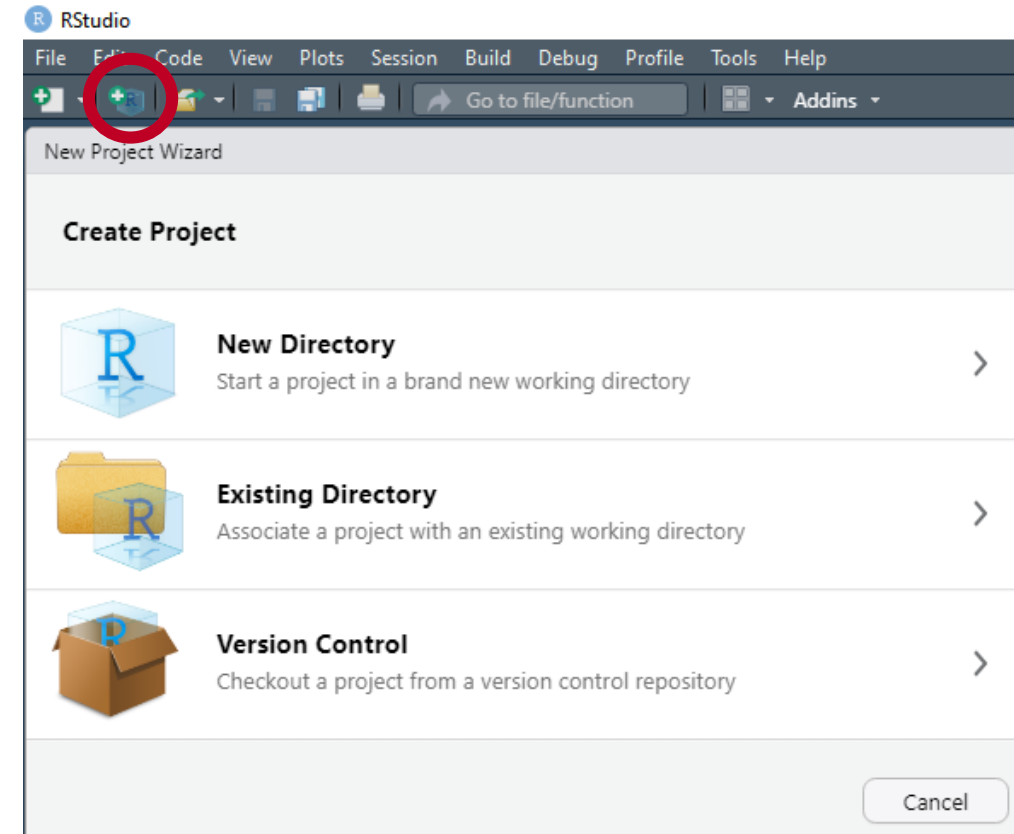
- Ein Projektordner
- Unterordner für Daten, Skripte, Plots, Ergebnisse etc.
- README, das Struktur und Dateinamen erklärt
- Alle Pfade relativ zum Projektordner

## → RStudio Projects:

- Setzen working directory auf Projektordner
- (+ tun andere nützliche Dinge, die man nicht verstehen muss)

Besonders sinnvoll für Kollaborationen

**[Demo in R]**



# Style



<https://www.youtube.com/watch?v=XzAA5Qq9jzc>

# Skript-Struktur

[Demo in R]

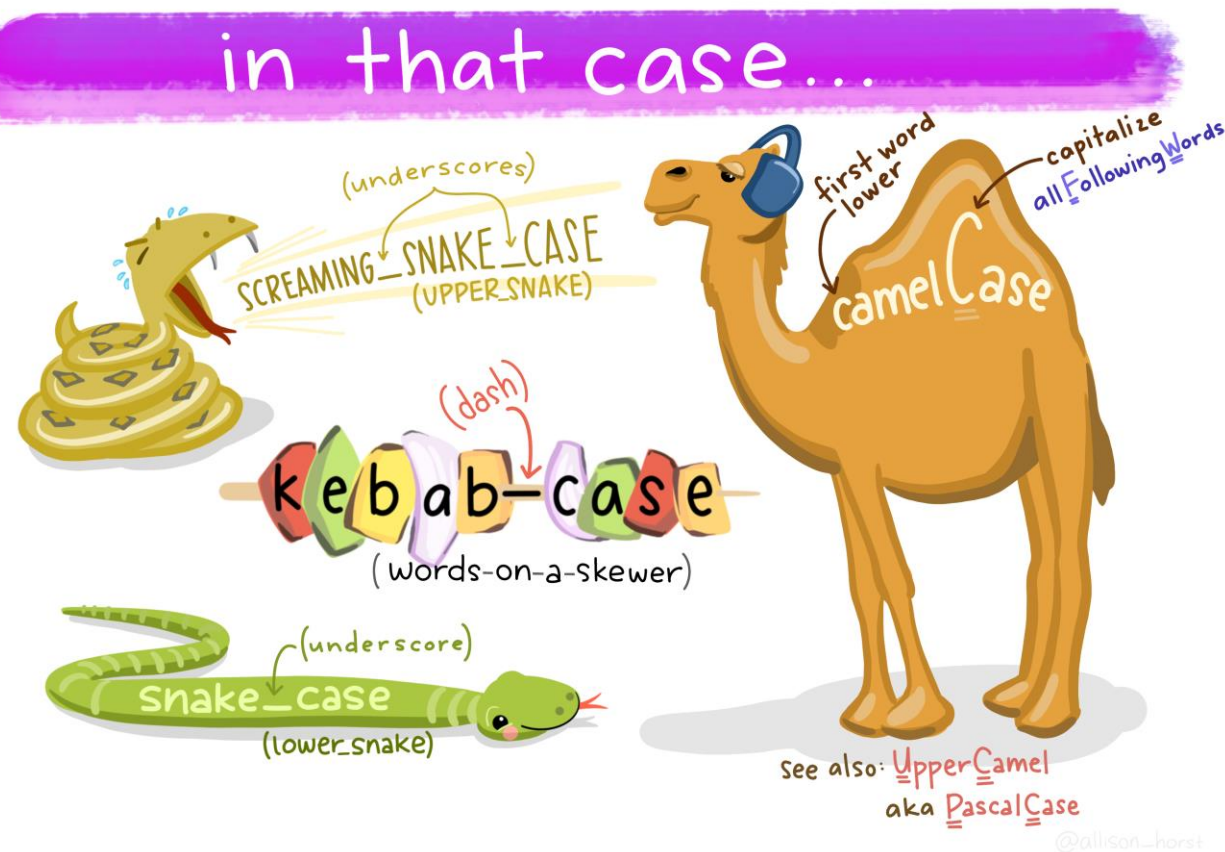
Wie schreibt man übersichtliche und verständliche Skripte?

- Header erstellen mit Datum, Script Info und Autor:in
- Alle benötigten Packages und Datensätze *am Anfang des Skripts* laden
- Skript in Abschnitte einteilen

```
#### A new section ####
```

- Variablen „sinnvoll“ benennen (... I know...)
- Kommentieren: nicht was genau passiert (`# loops through a list and computes the minimum`) sondern wozu der Schritt dient (`# extracts and adds minima from all rows`)

# Variablennamen



- Zusammengesetzte Variablennamen sollten einem Schema („case“) folgen

- In R: in der Regel snake case

> raw\_data

> mean\_response\_times

> results\_anova

etc.

→ Am Wichtigsten ist aber  
Konsistenz ☺

Image Credit: Allison Horst,

<https://github.com/allisonhorst/stats-illustrations>

# Übungsaufgaben

02\_Aufgaben\_Wrangling.pdf

→ Aufgabe 0

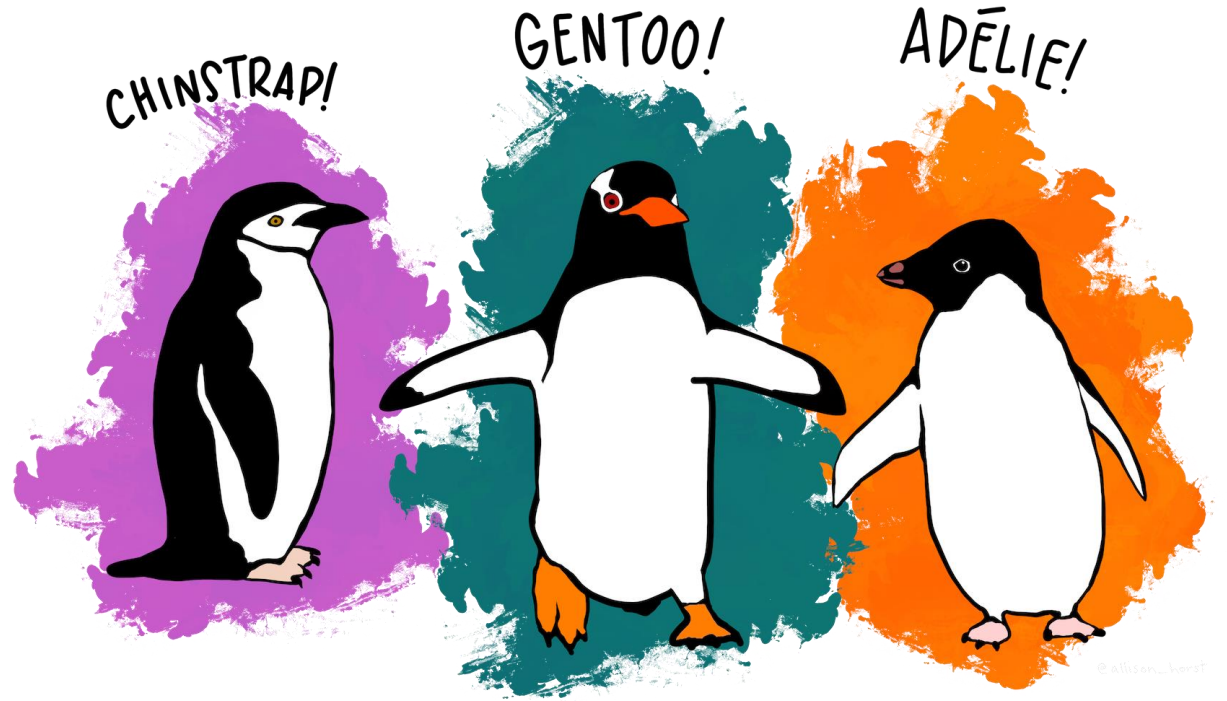
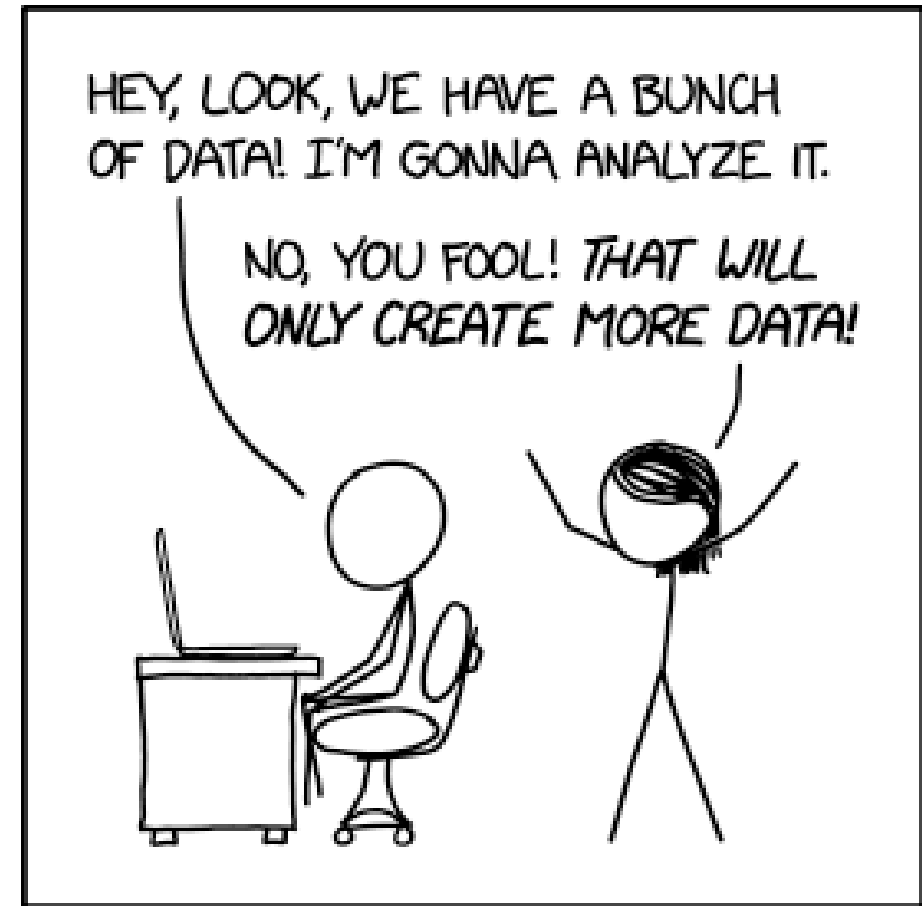


Image Credit: Allison Horst,

<https://github.com/allisonhorst/stats-illustrations>

## II. Data Handling & Manipulation



<https://xkcd.com/2582/>



# Data Handling & Manipulation

---

- Rohe Daten sind selten im Format in dem man sie braucht
- Häufig braucht man:
  - Andere Variablennamen
  - Neue Variablen
  - Ein anderes Format
  - Eine Auswahl an Variablen
  - Einen zusammengeführten Datensatz
  - Einen gefilterten Datensatz
- → R bietet diverse Funktionen dafür an



# Data Handling & Manipulation

---

- Basic Bitches... äh Befehle
  - `View()`, `head()`, `tail()`, `nrow()`, `ncol()`, `names()`, `dim()`, `length()`, `unique()`, `summary()`, `which()`, `is.na()`
- Subsetting (selecting rows and columns)
  - `$`, `[...]`, conditional selection
- Variablen umbenennen und rekodieren
  - `names()`
- Summarizing / Aggregating
  - `table()`, `aggregate()`
- Combining multiple data frames
  - `cbind()`, `rbind()`, `merge()`
- Reshaping data
  - Long vs. wide data; `pivot_longer()`; `pivot_wider()`;

# Basic Befehle

**Aufgabe:** Was tun diese Befehle? Probiert selber in R aus und lest in der Dokumentation nach.

- `View()`  
→ Grafische Ansicht eines Objekts
- `head(); tail()`  
→ Printed die ersten / letzten 5 Zeilen eines Objects in die Konsole
- `nrow(); ncol()`  
→ gibt die Anzahl Zeilen / Spalten eines Objects zurück
- `names()`  
→ gibt einen Vektor mit den Namen eines Objects zurück
- `dim()`  
→ gibt Dimensionen eines Objekts zurück
- `length()`  
→ gibt die Länge eines Vektors oder Faktors zurück
- `unique()`  
→ gibt übergebenes Objekt ohne Duplikate zurück
- `summary()`  
→ printed eine Zusammenfassung eines Objekts
- `which()`  
→ Gibt Indizes aller Einträge zurück, die gegebene Bedingung erfüllen

# Subsetting

- Zeilen oder Spalten nach Name bzw. Index → `$`, `[]`

```
data_frame$variable; data_frame[rows, columns]
```

- Neue Variablen: ebenfalls mit `$`

```
data_frame$new_var <- data_frame$old_var / 1000
```

- Auswahl von Zeilen, die bestimmte Bedingung erfüllen → `==`, `!=`, `%in%` etc.

```
data_frame[data_frame$variable == "condition", ]
```

**[Demo in R]**

# Rename and Recode Variables

[Demo in R]

- Variablennamen in Rohdaten entsprechend oft keinem (oder nicht dem präferierten) Schema

→ `names()` hilft uns weiter:

```
names(df)[names(df) == "old_name"] <- "new_name"
```

- Dasselbe gilt für die Werte von (character)-Variablen
- → Leerzeichen, Klammern, Länge etc.
- Wenn Variable nur wenige unterschiedliche Werte hat: `ifelse()`
- Für zwei Werte:

```
df$var <- ifelse(df$var == "very ugly old value 1", "new_value_1", "new_value_2")
```

# Summarizing / Aggregating

- Oft von Interesse: Summary Statistics wie

→ Anzahl pro Bedingung → `table()`

→ Mittelwert, Maximum, Minimum, Varianz etc. einer Variable pro Bedingung(en) → `aggregate()`

```
> table(df$condition, df$response)
```

	new	old
new_word	55	45
old_word	47	53

[Demo in R]

```
> aggregate(RT ~ condition, # dv ~ iv1 + iv2...  
            data = dat_long, # name of the dataframe  
            FUN = mean) # aggregation function  
# sd, var, min, max etc. also possible
```

	condition	RT
1	con	821.0
2	incon	723.4

# Combining Objects / Dataframes

---

- Oft sind Daten in unterschiedlichen Objekten gespeichert, die wir zusammenfügen wollen (z.B. demographische Daten in separater Datei)

`cbind()`; `rbind()` → Zusammenfügen von zwei Dataframes spaltenweise bzw. zeilenweise

!!! Spalten bzw. Zeilen der Dataframes müssen in gleicher Reihenfolge sein !!!

`merge()` → Verbindet zwei Dataframes anhand einer oder mehrerer Variablen

```
> merge(data_main, data_demographic, by = "ID")
```

[Demo in R]

# Reshaping Data – Long vs. Wide Format

- Daten können unterschiedlich repräsentiert werden:

ID	con	incon
1	626	843
2	674	685
3	695	872
4	823	664
5	734	652

→ Wide Format

ID	condition	RT
1	con	734
1	incon	606
2	con	856
2	incon	835
3	con	819
3	incon	648
4	con	881
4	incon	687
5	con	815
5	incon	841

→ Long Format

# Reshaping Data – `tidyr`

wide

id	x	y	z
1	a	c	e
2	b	d	f

long

id	key	val
1	x	a
2	x	b
1	y	c
2	y	d
1	z	e
2	z	f

wide

id	x	y	z
1	a	c	e
2	b	d	f



# Reshaping Data – `tidyr`

## Wide → Long: `pivot_longer()`

```
pivot_longer(  
  # name of the dataframe  
  dat_wide,  
  # columns containing the values  
  cols = c("con", "incon"),  
  # new column name for the column  
  # containing the names given in cols  
  names_to = "condition",  
  # new column name for the value column  
  values_to = "RT")
```

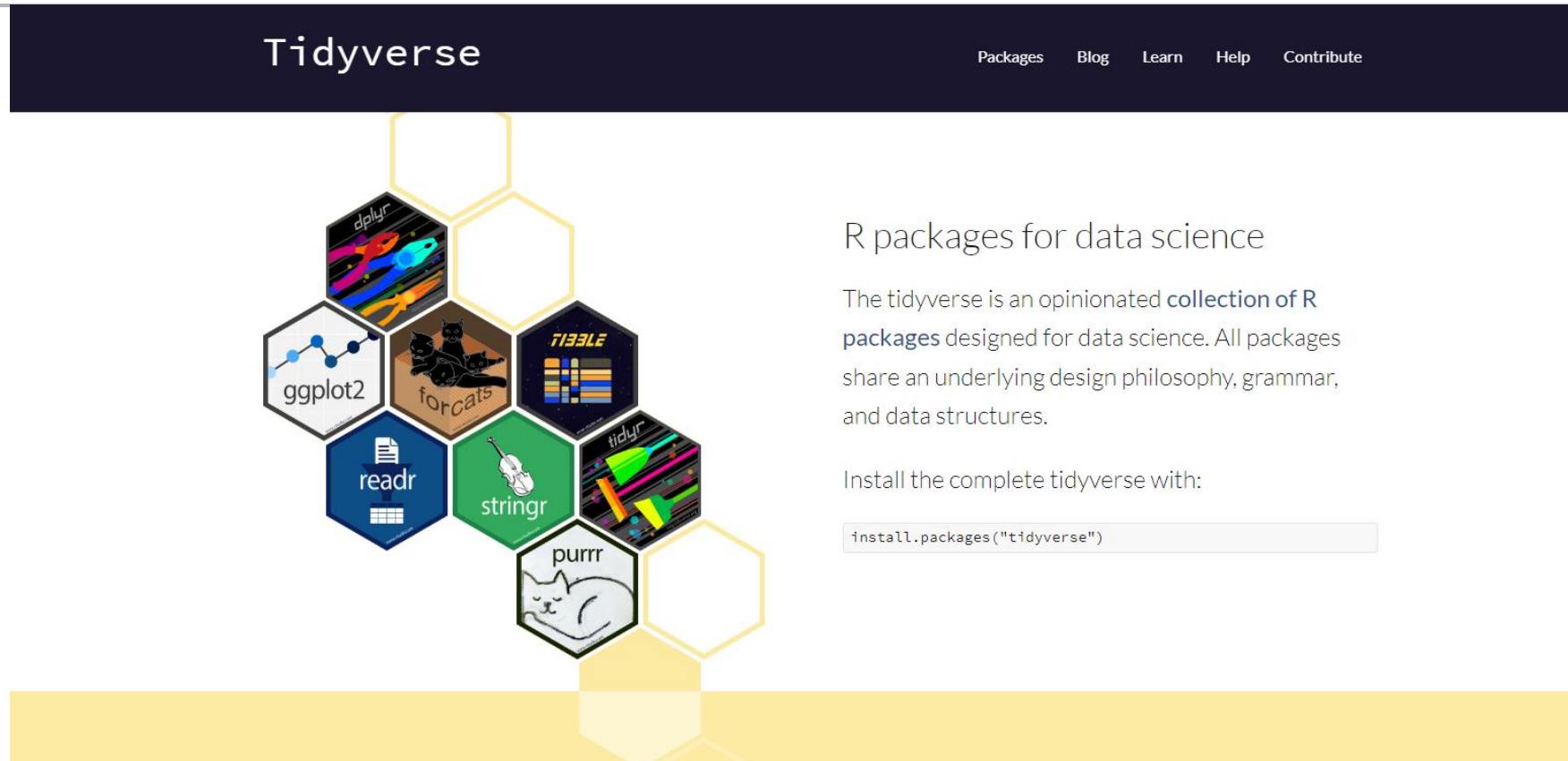
## Long → Wide: `pivot_wider()`

```
pivot_wider(  
  # name of the dataframe  
  dat_long,  
  # column(s) that identify a row  
  id_cols = "ID",  
  # column containing the new names  
  
  names_from = "condition",  
  # column containing the new values  
  values_from = "RT")
```

**[Demo in R]**

# Tidyverse

- Bisher (fast alles): „base R“
- tidyverse ist sehr verbreitet und einflussreich
- Aber: eigene Syntax und (für Einsteiger:innen oft verwirrende) Philosophie
- Deshalb bleiben wir hier bei base R 😊

A screenshot of the Tidyverse website. The header is dark blue with the word "Tidyverse" in white. To the right of the header are links: "Packages", "Blog", "Learn", "Help", and "Contribute". The main content area features a large graphic of a honeycomb structure. Several hexagons contain logos for Tidyverse packages: dplyr (a colorful abstract shape), ggplot2 (a network diagram), readr (a document icon), forcats (a black cat), stringr (a violin), tidyr (a colorful abstract shape), and purrr (a white cat). To the right of the honeycomb graphic, the text "R packages for data science" is displayed. Below this, a paragraph states: "The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures." Further down, it says "Install the complete tidyverse with:" followed by a code block containing the command `install.packages("tidyverse")`.

Tidyverse

Packages Blog Learn Help Contribute

R packages for data science

The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

```
install.packages("tidyverse")
```

<https://www.tidyverse.org/>

# PAUSE



# Übungsaufgaben

02\_Aufgaben\_Wrangling.pdf

→ So weit ihr kommt 😊

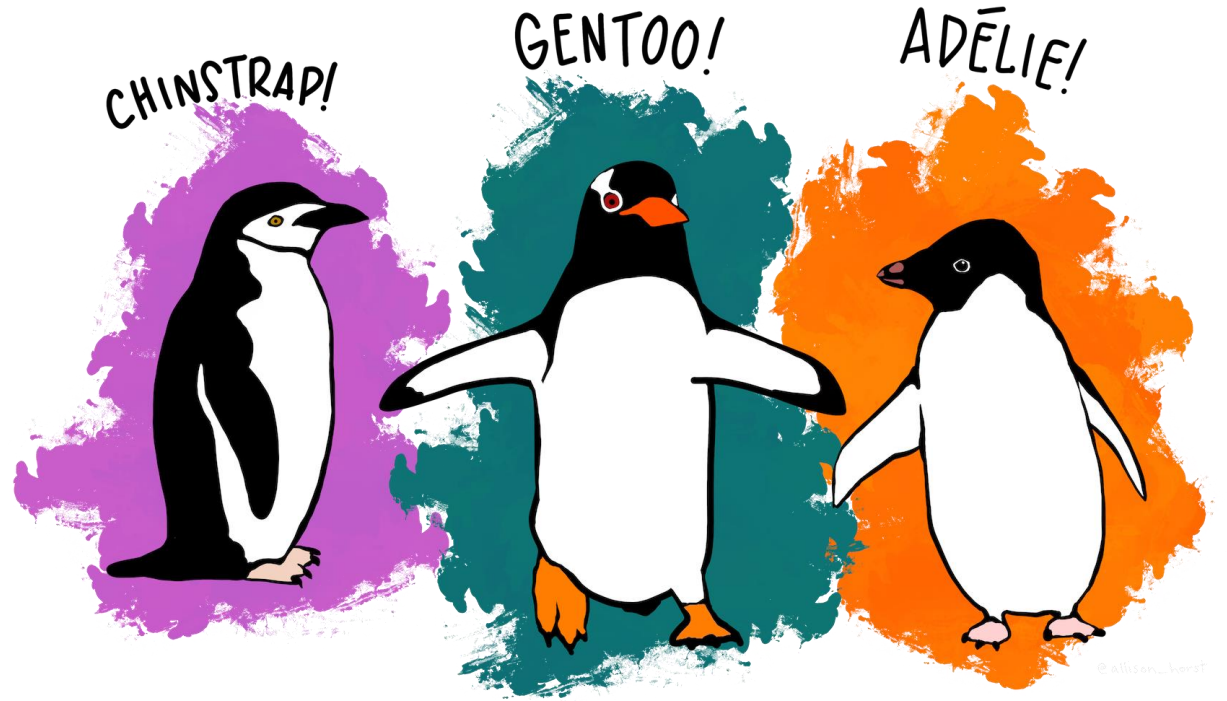
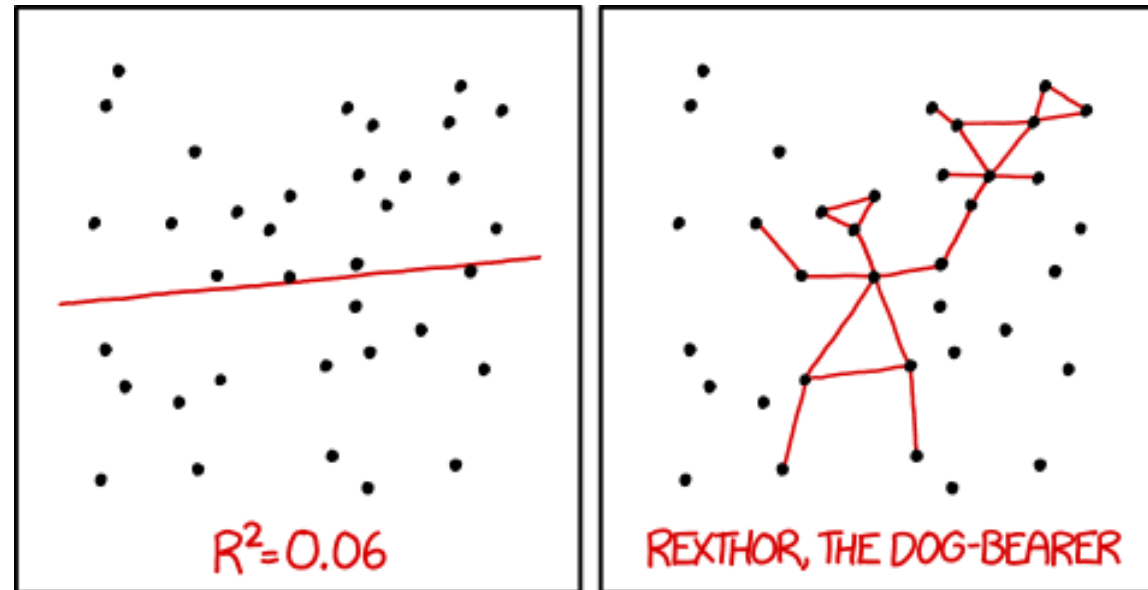


Image Credit: Allison Horst,

<https://github.com/allisonhorst/stats-illustrations>

# Plots



I DON'T TRUST LINEAR REGRESSIONS WHEN IT'S HARDER  
TO GUESS THE DIRECTION OF THE CORRELATION FROM THE  
SCATTER PLOT THAN TO FIND NEW CONSTELLATIONS ON IT.

<https://xkcd.com/1725/>

# Was für Arten von Plots kennt ihr?

---

- Boxplots
- Histogramme
- Scatterplots
- Balkendiagramme

# R: base R vs. ggplot2

- R ist bekannt dafür, dass man damit wunderschöne (und vor allem informative) Grafiken erstellen kann

## Möglichkeit 1: base R

- Basiert auf standard R Syntax & Logik → recht leicht, basics zu lernen
  - Mit den defaults nicht soooo schön
- Sinnvoll für „quick and dirty“ Visualisierungen
- Machen wir hier ☺

## Möglichkeit 2: ggplot2 package

- Eigene (andere) Syntax & Logik → für Anfänger:innen etwas unintuitiv
  - Schneller „schöne“ Ergebnisse
- Sinnvoll für Abschlussarbeiten, Vorträge und Publikationen
- Tutorial am Ende

# Histogramm: hist()

```
hist(  
  X, # vector to plot  
  breaks, # number of cells  
  xlim, ylim, # axis limits  
  xlab, ylab, # axis labels  
  main # plot title  
)
```



# Scatterplot: plot()

plot(

*x, y, # the variable to plot to each axis*

*type, # the type of plot (in this case, "p")*

*col, # color (can also be a factor!)*

*pch, # type/shape of points*

*cex, # size of points*

*... # xlim, ylim, xlab, ylab, main... different graphics parameters*

)

# Boxplot: boxplot()

boxplot(

*y ~ x1\*x2, # formula indicating dv ~ factor(s)*

*data, # data frame containing y, x1, x2*

*notch, # draws a notch to compare the medians*

*cex.lab, # size of axis labels*

*cex.axis, # size of tick labels*

*cex.main, # size of title*

*xlab, ylab, main, ylim, # you know these*

*xlim, # works, but doesn't really make sense*

)

# Balkendiagramm: barplot()

```
barplot(  
    height, # height of bars  
    names.arg, # category names (not necessary)  
    width, # width of bars (no effect without xlim)  
    ... # graphics parameters  
)
```

→ Fehlerbalken gehen auch, sind aber etwas umständlich

# PAUSE



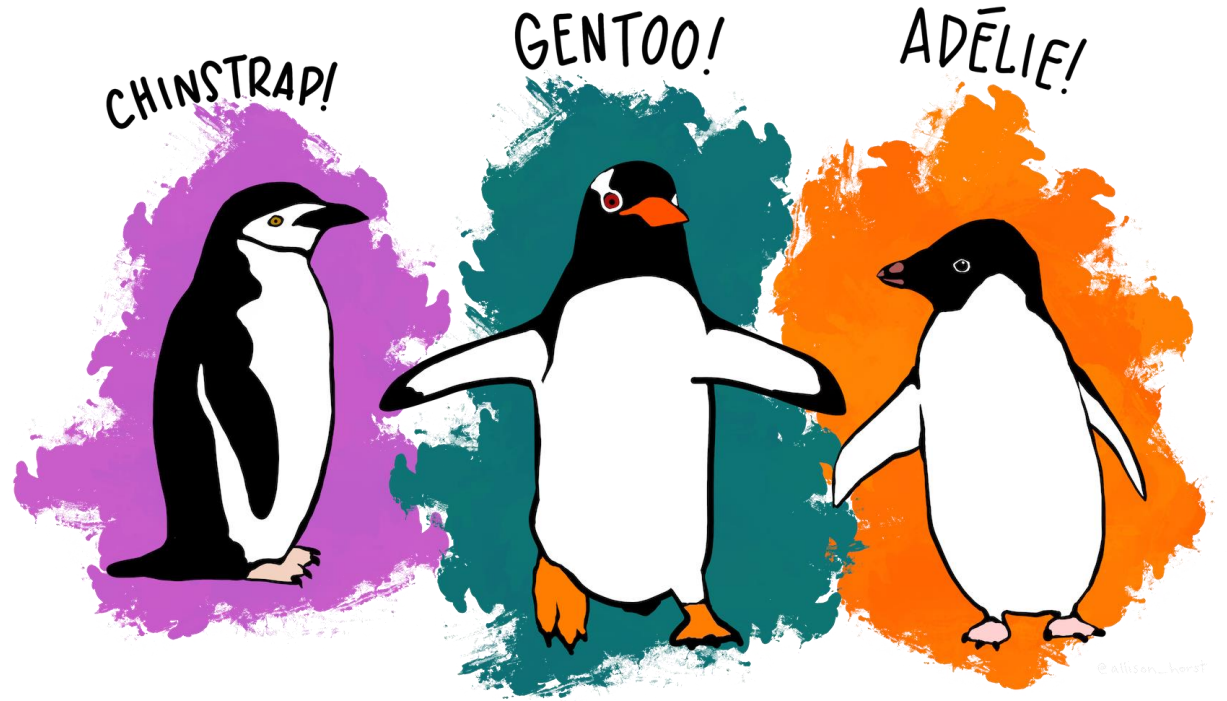


Image Credit: Allison Horst,

<https://github.com/allisonhorst/stats-illustrations>

# Übungsaufgaben

02\_Aufgaben\_Plots.pdf

→ Fangt an und schaut wie weit ihr kommt 😊



# Getting Help

<https://tenor.com>



# Getting Help

<https://tenor.com>

# HELP



- Für Funktionsweise bestimmter Funktionen
- Für spezifische Probleme / Aufgaben
- „error bars in barplots base R“
- Für Fehlermeldungen

- Hilfe-Funktion in R: `?function` → öffnet Dokumentation

> `?mean`

- Ausführliche Debugging Tipps:  
<https://www.greenteapress.com/thinkpython/thinkCSpy.pdf> (Appendix A) --> Für Python, sind für R aber genau so anwendbar



# Wenn gar nichts mehr geht

## A.3.4 I'm really, really stuck and I need help.

First, try getting away from the computer for a few minutes. Computers emit waves that affect the brain, causing these effects:

- Frustration and/or rage.
- Superstitious beliefs (“the computer hates me”) and magical thinking (“the program only works when I wear my hat backward”).
- Random-walk programming (the attempt to program by writing every possible program and choosing the one that does the right thing).

If you find yourself suffering from any of these symptoms, get up and go for a walk. When you are calm, think about the program. What is it doing? What are some possible causes of that behavior? When was the last time you had a working program, and what did you do next?

Sometimes it just takes time to find a bug. We often find bugs when we are away from the computer and let our minds wander. Some of the best places to find bugs are trains, showers, and in bed, just before you fall asleep.

Aus: „How to Think like a Computer Scientist“

# Heute Nachmittag

---

- Beendet die Aufgaben von heute morgen
- Wer danach noch Lust auf R hat: <https://swirlstats.com/>
  - Interaktives R Tutorial in R Studio
  - Sinnvolle Lessons für die Summer School:  
1: Basic Building Blocks, 3: Sequences of Numbers; 4: Vectors; 6: Subsetting Vectors; 8: Logic; 9: Functions; 13: Simulation; 15: Base Graphics

Bei Fragen / Problemen etc.: Raum 4032 (bis 17:00) oder per Mail an [marie.jakob@psychologie.uni-freiburg.de](mailto:marie.jakob@psychologie.uni-freiburg.de)

Danke für Eure Aufmerksamkeit!

