Analyse und Dokumentation

BSc Psychologie SoSe 2024

Belinda Fleischmann

Datum	Einheit	Thema	Lehrperson
10.04.24	Seminar	(1) Ethik und Ethische Formalitäten	BF
17.04.24	Seminar	(2) Wissenschaftliche Berichte	BF
24.04.24	Seminar	(3) Offenheit und Transparenz	BF
01.05.24	Tag der Arbeit		
08.05.24	**Seminar**	**(4) R, Quarto und Zotero**	**BF**
15.05.24	Praxisseminar	Offene Übung	BF
22.05.24	Präsentationen	Einfache Lineare Regression	JS
29.05.24	Präsentationen	Korrelation	JS
05.06.24	Präsentationen	Einstichproben-T-Test	JS
12.06.24	Präsentationen	Zweistichproben-T-Test	JS
19.06.24	Präsentationen	Einfaktorielle Varianzanalyse	BF
26.06.24	Präsentationen	Zweifaktorielle Varianznalyse	BF
03.07.24	Präsentationen	Multipe Regression	BF
10.07.24	Präsentationen	Kovarianzanalyse	BF
26.07.24	Klausurtermin		
Feb 2025	Klausurwiederholungstermin		

(4) R, Quarto und Zotero

R Tools

Quarto

Zotero

R Tools

Quarto

Zotero

Hilfreiche Quellen

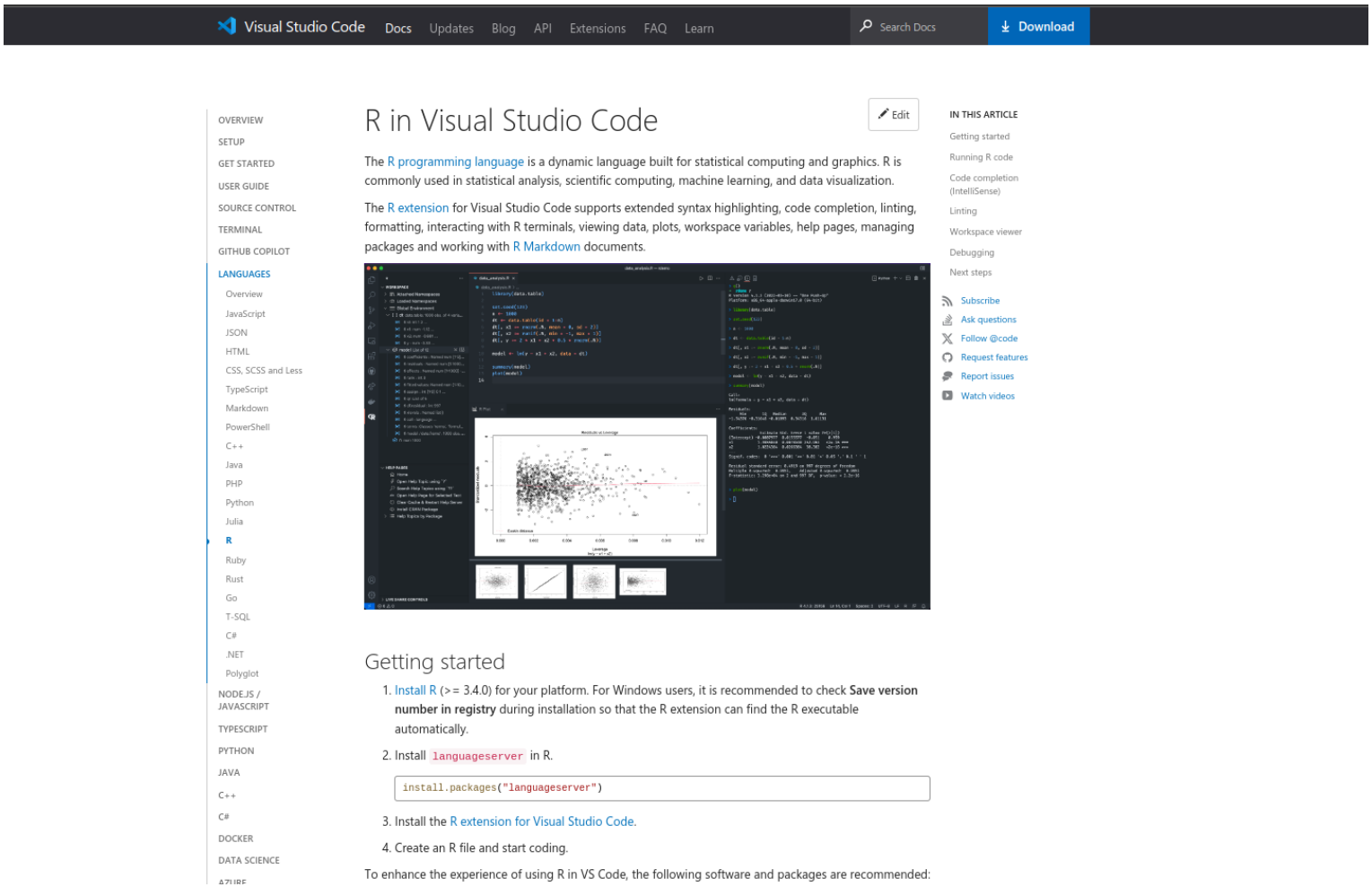
Visual Studio Code (VS Code) Website

VS Code-R Wiki

R for Data Science (2e)

ggplot2: Elegant Graphics for Data Analysis (3e)

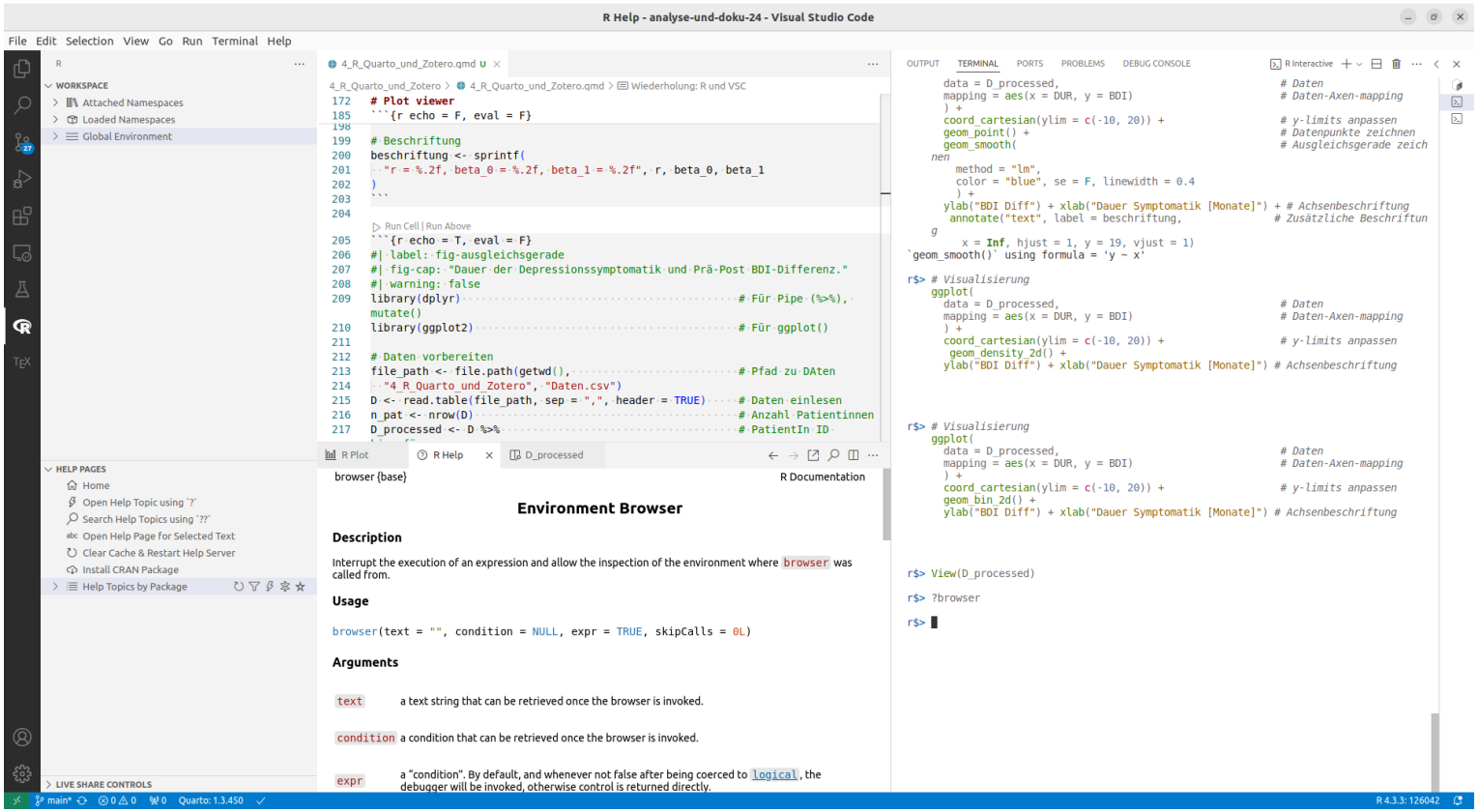
Wiederholung: R und VS Code



VS Code Website

Wiederholung: R workspace und Interactive Viewer

Help Viewer

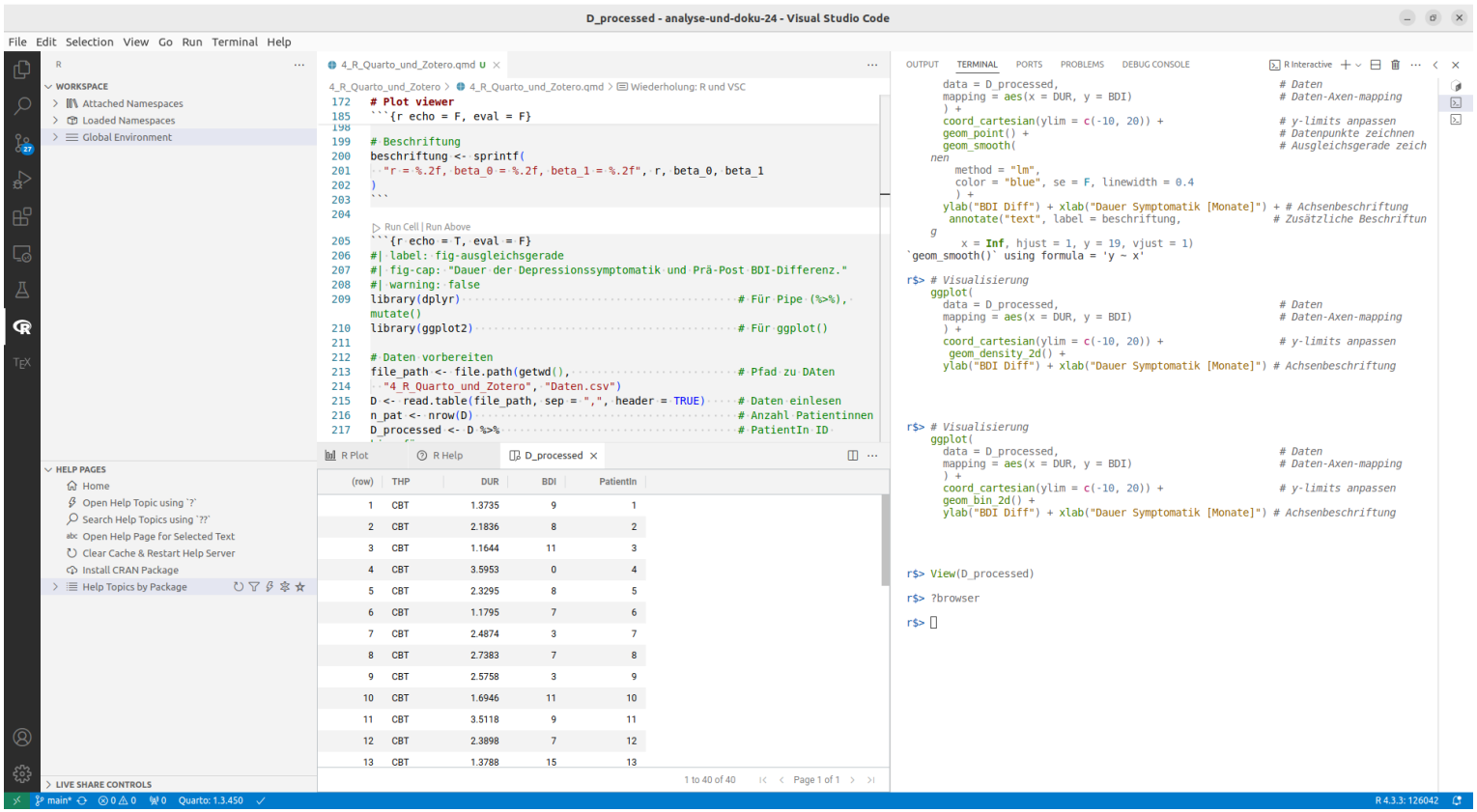


Mit dem Befehl `?funktionsname` oder über HELP PAGES öffnen.

VS Code Wiki - Interactive viewers

Wiederholung: R workspace und Interactive Viewer

Table Viewer

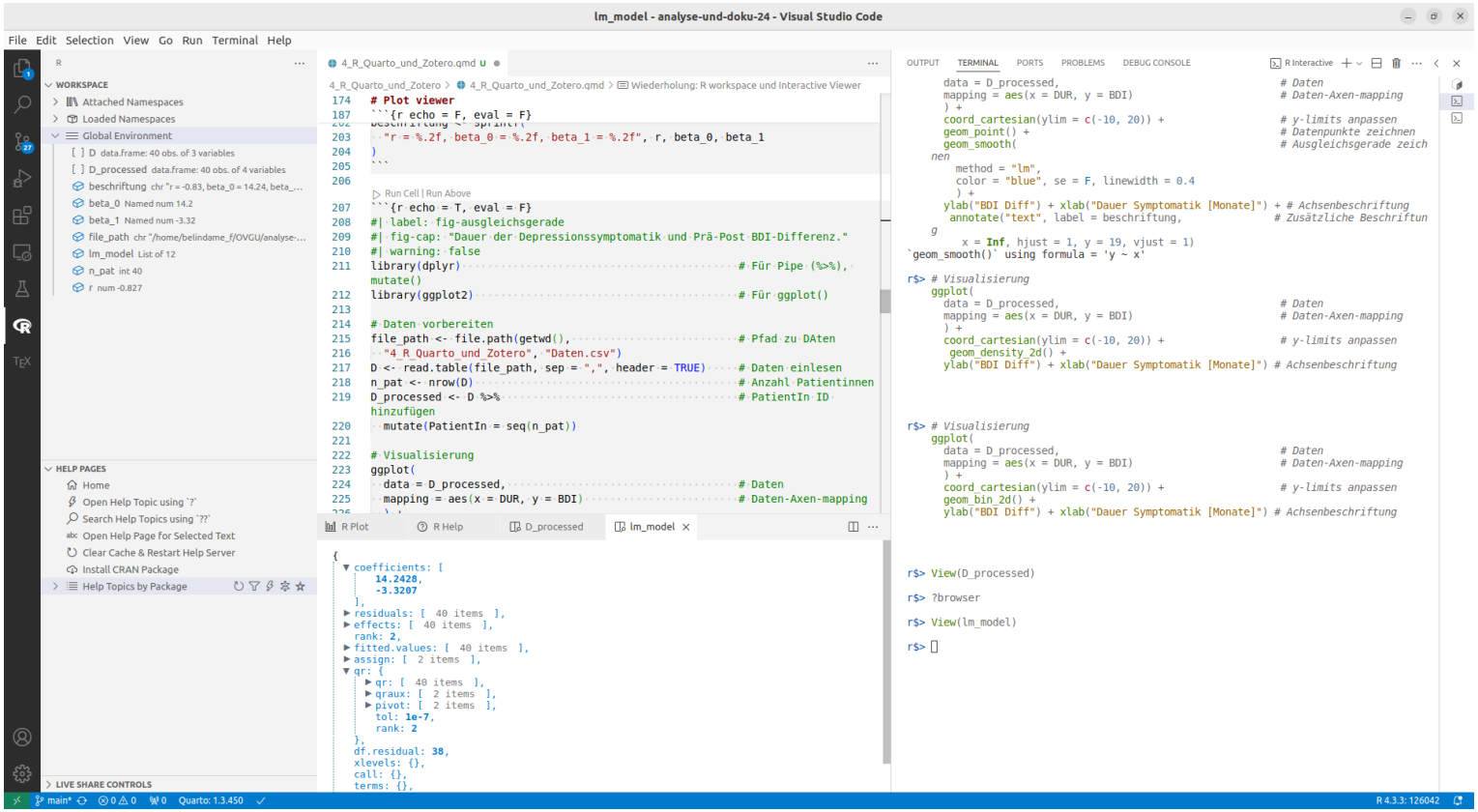


Mit dem Befehl `View()` oder im R WORKSPACE → Global Environment über das View Symbol  neben entsprechendem Objekt

VS Code Wiki - Interactive viewers

R workspace und Interactive Viewer

List Viewer

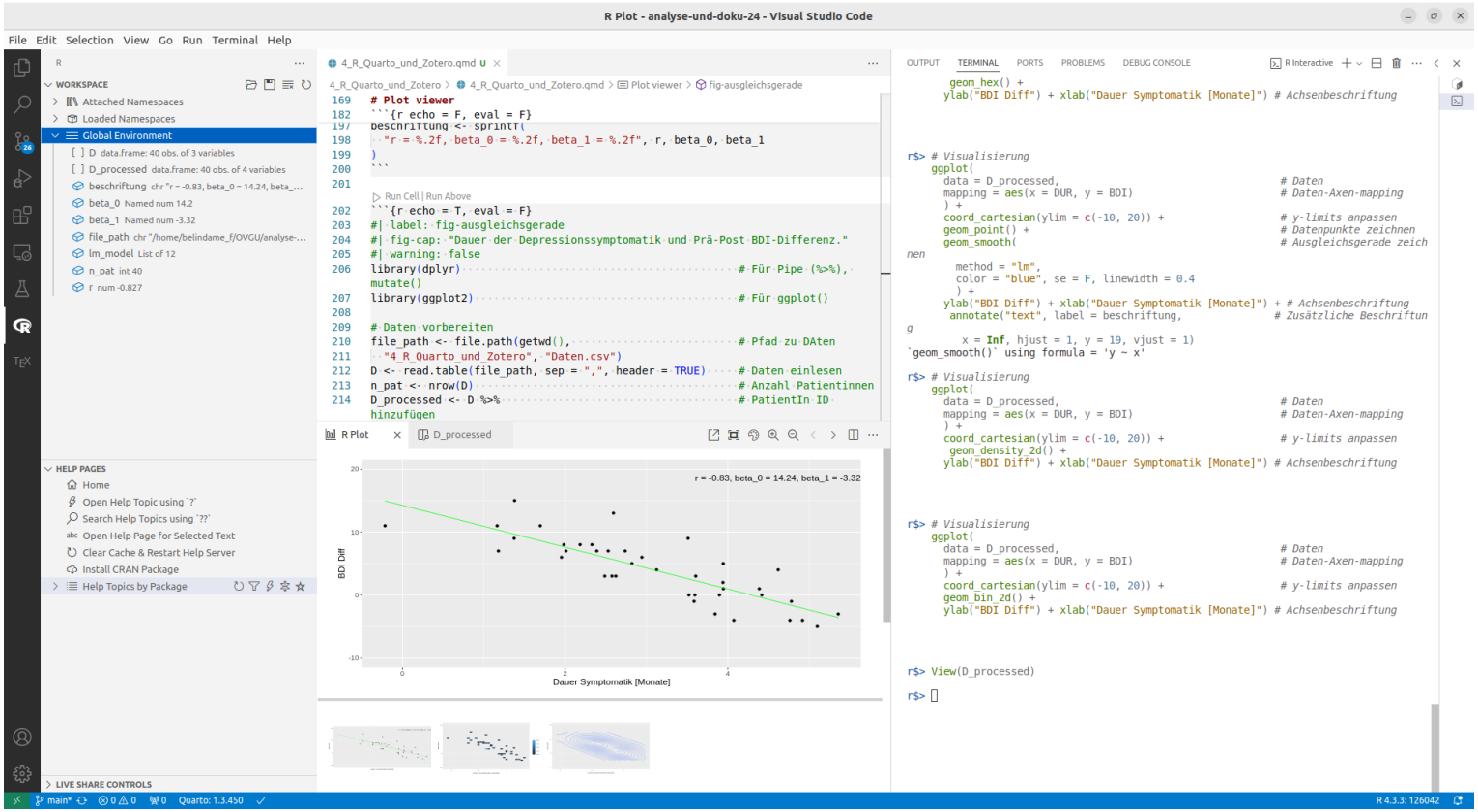


Mit dem Befehl `View()` oder im R WORKSPACE → Global Environment über das View Symbol neben entsprechendem Objekt

VS Code Wiki - Interactive viewers

R workspace und Interactive Viewer

Plot Viewer



Das `R Paket` [httpgd](#) erleichtert die Ansicht erstellter Grafiken.

Nach Installation über r.plot.useHttpgd in VS Code Einstellungen freischalten. Grafiken öffnen nach Ausführen eines plot Befehls automatisch im Plot Viewer.

VS Code Wiki - Interactive viewers

Debugging mit `browser()`

Die `R base` Funktion `browser()` erlaubt das Pausieren der Exekution eines Skripts und Inspektion der aktuellen *environment*.

Beispiel

```
# Beispiel 1
erste_variable <- 1
zweite_variable <- 3
ergebnis <- c()
print(ergebnis)
browser()                                    # Pausiert Skript
ergebnis <- erste_variable + zweite_variable
#browser()

# Beispiel 2
for (i in 1:5) {
  print(i + 2)
  #browser()
}
```

Über das Argument `expr` kann auch eine Bedingung als boolesche Operation spezifiziert werden.

Mit `Enter` wird die Exekution fortgeführt.

Mit `Q` wird der browser beendet.

Motivation

Programmiercode wird streng sequentiell Befehl für Befehl ausgeführt.

Manchmal möchten wir von dieser rein sequentiellen Befehlsreiehenfolge abweichen.

Die prinzipiellen Werkzeuge dafür sind **Kontrollstrukturen**. Dazu gehören `if`-statements, `switch`-statements und Schleifen mit `for`, `while` oder `repeat`.

Kontrollstrukturen: `if`-statements

if-statements

```
if (Bedingung) {
  TrueAktion        # Befehl, der ausgeführt wird, falls Bedingung TRUE ist
}
```

- Wenn `Bedingung` `TRUE` ist, wird `TrueAktion` ausgführt.
- Wenn `Bedingung` `FALSE` ist, wird `TrueAktion` nicht ausgeführt.

if-else-statements

```
if (Bedingung) {
  TrueAktion        # Befehl, der ausgeführt wird, falls Bedingung TRUE ist
} else {
  FalseAktion       # Befehl, der ausgeführt wird, falls Bedingung FALSE ist
}
```

- Wenn `Bedingung` `TRUE` ist, wird `TrueAktion` ausgführt.
- Wenn `Bedingung` `FALSE` ist, wird `FalseAktion` ausgeführt.

Kostrollstrukturen: `if`-statements

Beispiele

```
x <- 3
if (x > 0) {
  print("x ist größer als 0")
}
```

```
[1] "x ist größer als 0"
```

```
y <- -3
if (y > 0){
  print("y ist größer als 0")
} else{
  print("y ist nicht größer als 0")
}
```

```
[1] "y ist nicht größer als 0"
```

Wiederholung: Logischer Operatoren

- Die Boolesche Algebra und R kennen zwei *logische Werte*: TRUE und FALSE
- Bei Auswertung von Relationsoperatoren ergeben sich logische Werte

Relationsoperator	Bedeutung
$==$	Gleich
$!=$	Ungleich
$<$, $>$	Kleiner, Größer
$<=$, $>=$	Kleiner gleich, Größer gleich
$\vert$	ODER
$\&$	UND

- $<, <=, >, >=$ werden zumeist auf numerische Werte angewendet.
- $==, !=$ werden zumeist auf beliebige Datenstrukturen angewendet.
- $\vert$ und $\&$ werden zumeist auf logische Werte angewendet.
- $\vert$ implementiert das inklusive *oder*. Die Funktion xor() implementiert das exklusive ODER.

Kostrollstrukturen: `if`-statements

Beispiele

```
x <- 3
y <- 2

# Logisches UND/ODER
if (x > 0 | y > 0) {
  print("beide, oder eine der beiden Variablen sind größer 0")
} else{
  print("Keine der Variablen ist größer 0")
}
```

```
[1] "beide, oder eine der beiden Variablen sind größer 0"
```

```
# Logisches UND
if (x > 0 & y > 0) {
  print("x und y sind größer 0")
} else{
  print("Es sind nicht beide Variablen x und y größer 0, aber vielleicht eine der beiden")
}
```

```
[1] "x und y sind größer 0"
```

```
# Exklusives ODER
if (xor(x > 0, y > 0)){
  print("Genau eine der 2 Variablen x und y ist größer 0")
} else{
  print("Es sind entweder keine der Variablen x und y oder beide größer 0")
}
```

```
[1] "Es sind entweder keine der Variablen x und y oder beide größer 0"
```

Kontrollstrukturen: `switch`-statements

Motivation

Kombinierte `if-else` -statements können leicht unübersichtlich werden.

```
x <- 2
if (x == 1){
  print("Aktion 1")
} else if(x == 2){
  print("Aktion 2")
} else if(x == 3){
  print("Aktion 3")
} else if(x == 4){
  print("Aktion 4")
}
```

```
[1] "Aktion 2"
```

switch-statement mit Integer

```
x <- 2
switch(
  x,                          # switch Variable
  print("Aktion 1"),          # 1. Aktion
  print("Aktion 2"),          # 2. Aktion
  print("Aktion 3"),          # 3. Aktion
  print("Aktion 4")           # 4. Aktion
)
```

```
[1] "Aktion 2"
```

Kontrollstrukturen: `switch`-statements

switch-statement mit Character

```
x <- "a"
switch(
  x,                              # switch Variable
  a = print("Aktion 1"),          # 1. Aktion
  b = print("Aktion 2"),          # 2. Aktion
  c = print("Aktion 3"),          # 3. Aktion
  d = print("Aktion 4")           # 4. Aktion
)
```

```
[1] "Aktion 1"
```

Kontrollstrukturen: `for`-Schleifen

for-Schleifen

```
for (item in sequenz){
  zu_wiederholende_Aktion          # Aktion, die wiederholt werden soll
}
```

Beispiel

```
for (i in 1:3) {
  print(i)                         # Aktion, die wiederholt werden soll
}
```

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

while-Schleifen

while-Schleifen iterieren Codeabschnitte basierend auf einer Bedingung.

```
while (Bedingung) {
  TrueAktion            # TrueAktion wird ausgeführt, solange Condition == TRUE
}
```

Beispiel

```
i <- 5
while (i < 11) {
  print(i)
  i <- i + 1
}
```

```
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

repeat-Schleifen

repeat-loops wiederholen Codeabschnitte bis zu einem 'break' Befehl

```
repeat {
  TrueAktion            # Aktion wird ausgeführt, bis ein break Befehl evaluiert wird
}
```

Beispiel

```
i <- 1
repeat {
  print(i)
  i <- i + 1
  if (i == 5) {
    break
  }
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
```

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.

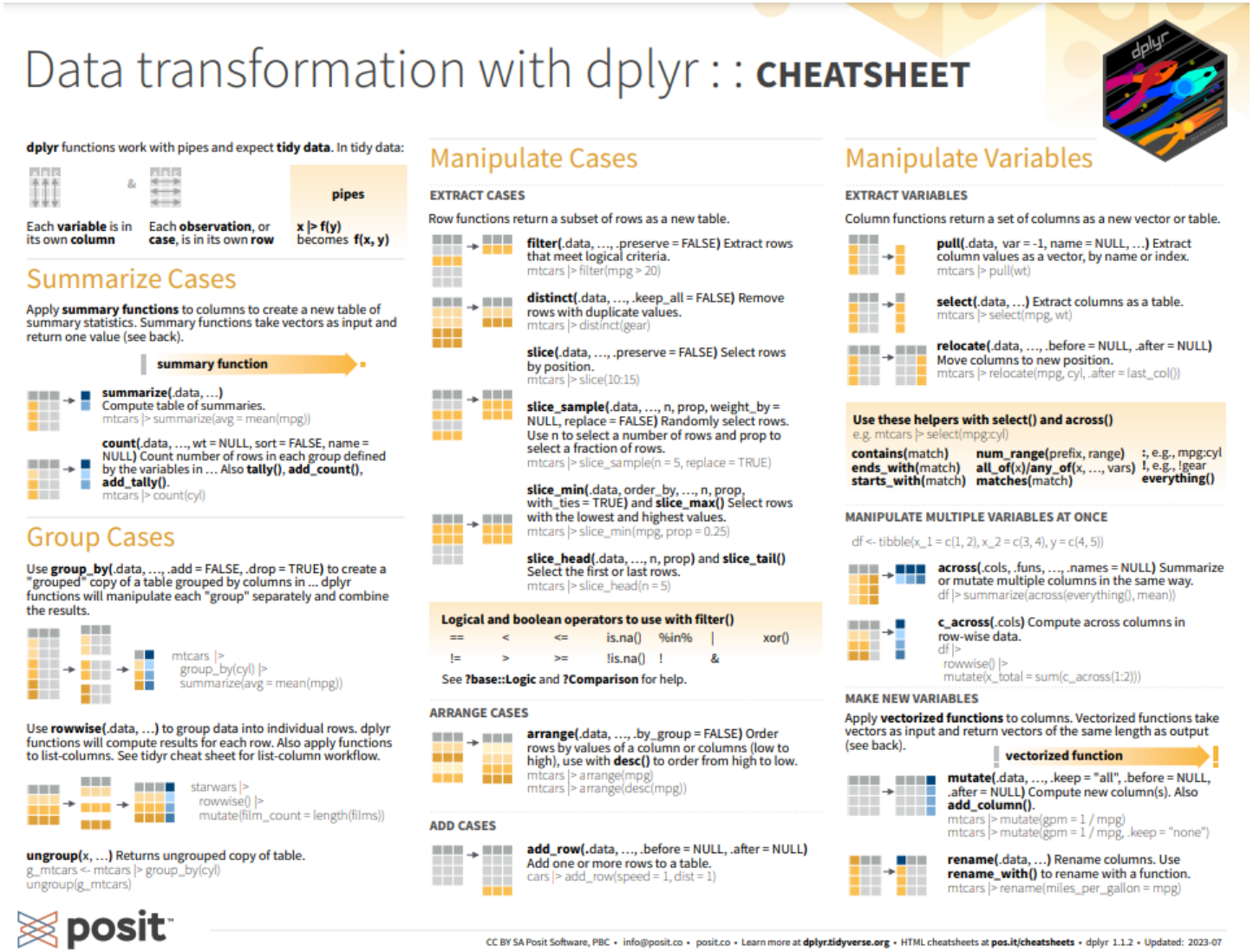
Install the complete tidyverse with:

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

Learn the tidyverse

Tidyverse

Cheat Sheets



Data transformation with dplyr : : CHEATSHEET

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column** & Each **observation**, or **case**, is in its own **row**

pipes

x |> f(y) becomes **f(x, y)**

Summarize Cases

Apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarize(.data, …)
Compute table of summaries.
mtcars |> summarize(avg = mean(mpg))

count(.data, …, wt = NULL, sort = FALSE, name = NULL) Count number of rows in each group defined by the variables in … Also **tally()**, **add_count()**, **add_tally()**.
mtcars |> count(cyl)

Group Cases

Use **group_by**(.data, …, .add = FALSE, .drop = TRUE) to create a "grouped" copy of a table grouped by columns in ... dplyr functions will manipulate each "group" separately and combine the results.

```
mtcars |>
  group_by(cyl) |>
  summarize(avg = mean(mpg))
```

Use **rowwise**(.data, …) to group data into individual rows. dplyr functions will compute results for each row. Also apply functions to list-columns. See tidyr cheat sheet for list-column workflow.

```
starwars |>
  rowwise() |>
  mutate(film_count = length(films))
```

ungroup(x, …) Returns ungrouped copy of table.

```
g_mtcars <- mtcars |> group_by(cyl)
ungroup(g_mtcars)
```

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, …, .preserve = FALSE) Extract rows that meet logical criteria.
mtcars |> filter(mpg > 20)

distinct(.data, …, .keep_all = FALSE) Remove rows with duplicate values.
mtcars |> distinct(gear)

slice(.data, …, .preserve = FALSE) Select rows by position.
mtcars |> slice(10:15)

slice_sample(.data, …, n, prop, weight_by = NULL, replace = FALSE) Randomly select rows. Use n to select a number of rows and prop to select a fraction of rows.
mtcars |> slice_sample(n = 5, replace = TRUE)

slice_min(.data, order_by, …, n, prop, with_ties = TRUE) and **slice_max()** Select rows with the lowest and highest values.
mtcars |> slice_min(mpg, prop = 0.25)

slice_head(.data, …, n, prop) and **slice_tail()** Select the first or last rows.
mtcars |> slice_head(n = 5)

Logical and boolean operators to use with filter()

==	<	<=	is.na()	%in%	|	xor()
!=	>	>=	!is.na()	!	&	

See **?base::Logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …, .by_group = FALSE) Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
mtcars |> arrange(mpg)
mtcars |> arrange(desc(mpg))

ADD CASES

add_row(.data, …, .before = NULL, .after = NULL) Add one or more rows to a table.
cars |> add_row(speed = 1, dist = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1, name = NULL, …) Extract column values as a vector, by name or index.
mtcars |> pull(wt)

select(.data, …) Extract columns as a table.
mtcars |> select(mpg, wt)

relocate(.data, …, .before = NULL, .after = NULL) Move columns to new position.
mtcars |> relocate(mpg, cyl, .after = last_col())

Use these helpers with select() and across()
e.g. mtcars |> select(mpg:cyl)

contains(match) **num_range**(prefix, range) **:**, e.g., mpg:cyl
ends_with(match) **all_of**(x)/**any_of**(x, …, vars) **!**, e.g., !gear
starts_with(match) **matches**(match) **everything()**

MANIPULATE MULTIPLE VARIABLES AT ONCE

```
df <- tibble(x_1 = c(1, 2), x_2 = c(3, 4), y = c(4, 5))
```

across(.cols, .funs, …, .names = NULL) Summarize or mutate multiple columns in the same way.
df |> summarize(across(everything(), mean))

c_across(.cols) Compute across columns in row-wise data.
df |>
rowwise() |>
mutate(x_total = sum(c_across(1:2)))

MAKE NEW VARIABLES

Apply **vectorized functions** to columns. Vectorized functions take vectors as input and return vectors of the same length as output (see back).

vectorized function

mutate(.data, …, .keep = "all", .before = NULL, .after = NULL) Compute new column(s). Also **add_column()**.
mtcars |> mutate(gpm = 1 / mpg)
mtcars |> mutate(gpm = 1 / mpg, .keep = "none")

rename(.data, …) Rename columns. Use **rename_with()** to rename with a function.
mtcars |> rename(miles_per_gallon = mpg)

posit

dplyr Cheat Sheet

Datenvorverarbeitung mit `dplyr`

```r
D <- read.table("Daten_1.csv", sep = ",", header = TRUE)    # Daten einlesen
```

Variable_1	Variable_2	Variable_3
34.87	34.61	33.56
32.16	22.89	15.75
33.95	31.82	28.83
28.78	25.91	20.04
30.13	26.83	22.00
30.50	26.50	24.42
32.48	26.92	22.96
31.66	31.84	28.83
32.76	33.00	33.28
31.60	26.77	21.21
32.44	28.55	28.63
29.48	25.33	24.19
31.24	28.97	25.18
34.33	31.31	28.22
31.56	27.11	22.92
31.87	30.95	30.30
27.07	21.94	17.60
29.36	25.41	19.32
36.07	33.56	33.41
33.03	28.81	26.58
33.12	32.20	29.44

Datenvorverarbeitung mit `dplyr`

Der Pipe operater %>% oder |> ermöglicht es, Funktionen in einer Reihe nacheinander auszuführen.

Mit der R-Funktion `mutate()` können wir neue Spalten erzeugen (auch als Funktionen bestehender Spalten).

```r
library(dplyr)
n <- nrow(D)                                                    # Anzahl Beobachtungen
D_processed <- D %>%                                            # D wird an nächste Funktion übergeben
  mutate(ID = seq(n)) %>%                                       # ID-Spalte hinzufügen
  mutate(Summe = Variable_1 + Variable_2 + Variable_3)          # Summen-Spalte hinzufügen
```

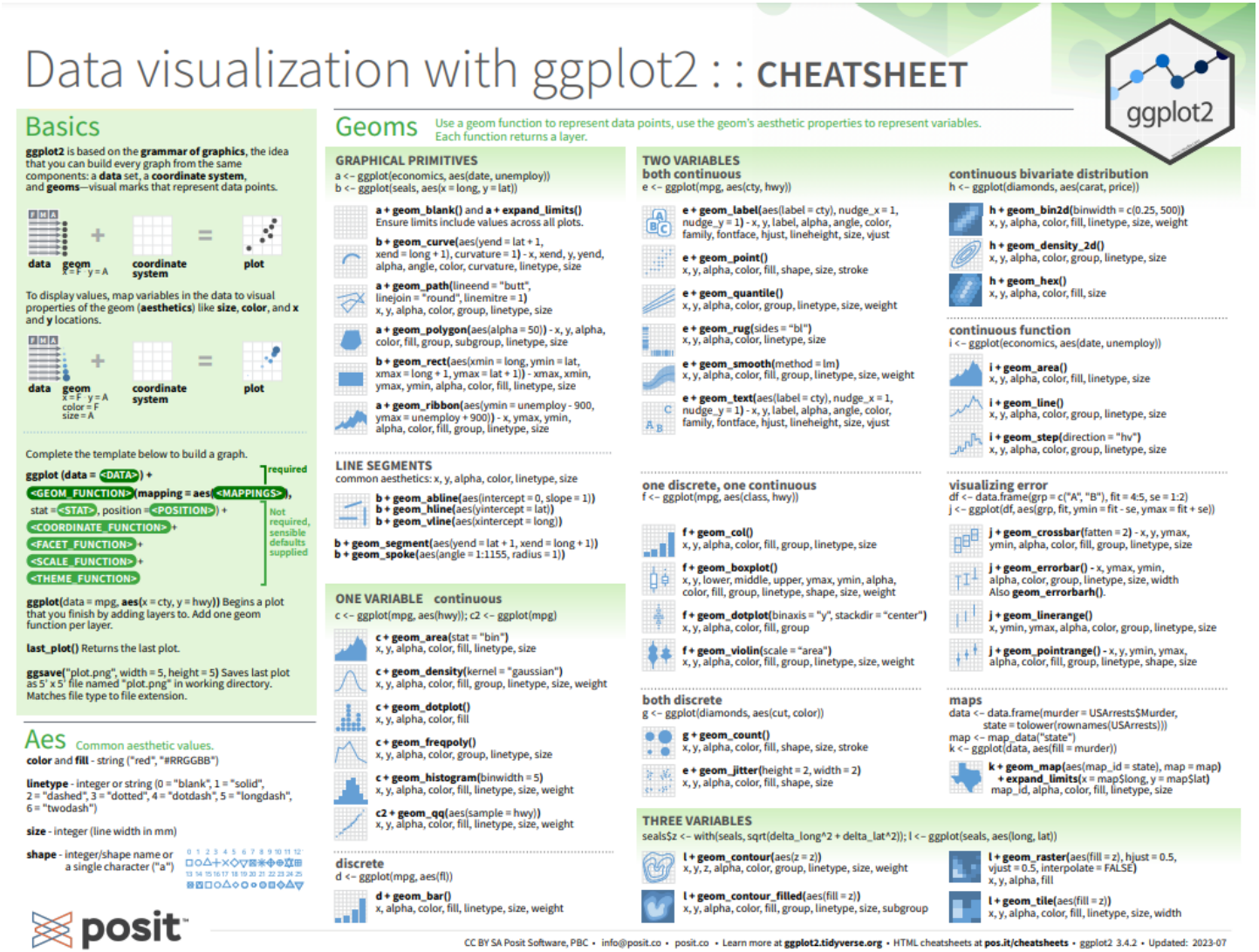
Variable_1	Variable_2	Variable_3	ID	Summe
34.87	34.61	33.56	1	103.04
32.16	22.89	15.75	2	70.79
33.95	31.82	28.83	3	94.60
28.78	25.91	20.04	4	74.74
30.13	26.83	22.00	5	78.96
30.50	26.50	24.42	6	81.42
32.48	26.92	22.96	7	82.37
31.66	31.84	28.83	8	92.34
32.76	33.00	33.28	9	99.05
31.60	26.77	21.21	10	79.58
32.44	28.55	28.63	11	89.62
29.48	25.33	24.19	12	79.00
31.24	28.97	25.18	13	85.40
34.33	31.31	28.22	14	93.86
31.56	27.11	22.92	15	81.59
31.87	30.95	30.30	16	93.12
27.07	21.94	17.60	17	66.61
29.36	25.41	19.32	18	74.09
36.07	33.56	33.41	19	103.04
33.03	28.81	26.58	20	88.42
33.12	32.20	29.44	21	94.75

Datenvorverarbeitung mit `dplyr`

Mit der R-Funktion `filter()` können wir Zeilen gemäß bestimmten Bedingungen auswählen.

```
D_selected <- D_processed %>%
  filter(ID %in% 1:10) %>%                    # Auswahl der IDs 1-10
  filter(Summe > 90)                          # Selektion der Beobachtungen mit Summe > 90
```

Variable_1	Variable_2	Variable_3	ID	Summe
34.87	34.61	33.56	1	103.04
33.95	31.82	28.83	3	94.60
31.66	31.84	28.83	8	92.34
32.76	33.00	33.28	9	99.05



Data visualization with ggplot2 : : CHEATSHEET

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and **geoms**—visual marks that represent data points.

data + geom (x = F, y = A) = coordinate system + plot

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.

data + geom (x = F, y = A, color = F, size = A) = coordinate system + plot

Complete the template below to build a graph.

required

```
ggplot(data = <DATA>) +
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),
  stat = <STAT>, position = <POSITION>) +
<COORDINATE_FUNCTION> +
<FACET_FUNCTION> +
<SCALE_FUNCTION> +
<THEME_FUNCTION>
```

Not required, sensible defaults supplied

ggplot(data = mpg, **aes(**x = cty, y = hwy**))** Begins a plot that you finish by adding layers to. Add one geom function per layer.

last_plot() Returns the last plot.

ggsave("plot.png", width = 5, height = 5**)** Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Aes

Common aesthetic values.

color and **fill** - string ("red", "#RRGGBB")

linetype - integer or string (0 = "blank", 1 = "solid", 2 = "dashed", 3 = "dotted", 4 = "dotdash", 5 = "longdash", 6 = "twodash")

size - integer (line width in mm)

shape - integer/shape name or a single character ("a")

0 1 2 3 4 5 6 7 8 9 10 11 12
13 14 15 16 17 18 19 20 21 22 23 24 25

posit

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))

a + geom_blank() and **a + expand_limits()** Ensure limits include values across all plots.

b + geom_curve(aes(yend = lat + 1, xend = long + 1), curvature = 1**)** - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend = "butt", linejoin = "round", linemitre = 1**)** x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(alpha = 50)**)** - x, y, alpha, color, fill, group, subgroup, linetype, size

b + geom_rect(aes(xmin = long, ymin = lat, xmax = long + 1, ymax = lat + 1)**)** - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin = unemploy - 900, ymax = unemploy + 900)**)** - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept = 0, slope = 1)**)**
b + geom_hline(aes(yintercept = lat)**)**
b + geom_vline(aes(xintercept = long)**)**

b + geom_segment(aes(yend = lat + 1, xend = long + 1)**)**
b + geom_spoke(aes(angle = 1:1155, radius = 1)**)**

ONE VARIABLE continuous

c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

c + geom_area(stat = "bin"**)** x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian"**)** x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot() x, y, alpha, color, fill

c + geom_freqpoly() x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5**)** x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy)**)** x, y, alpha, color, fill, linetype, size, weight

discrete

d <- ggplot(mpg, aes(fl))

d + geom_bar() x, alpha, color, fill, linetype, size, weight

TWO VARIABLES

both continuous

e <- ggplot(mpg, aes(cty, hwy))

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1**)** - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_point() x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile() x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl"**)** x, y, alpha, color, linetype, size

e + geom_smooth(method = lm**)** x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1**)** - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

continuous bivariate distribution

h <- ggplot(diamonds, aes(carat, price))

h + geom_bin2d(binwidth = c(0.25, 500)**)** x, y, alpha, color, fill, linetype, size, weight

h + geom_density_2d() x, y, alpha, color, group, linetype, size

h + geom_hex() x, y, alpha, color, fill, size

continuous function

i <- ggplot(economics, aes(date, unemploy))

i + geom_area() x, y, alpha, color, fill, linetype, size

i + geom_line() x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv"**)** x, y, alpha, color, group, linetype, size

one discrete, one continuous

f <- ggplot(mpg, aes(class, hwy))

f + geom_col() x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot() x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center"**)** x, y, alpha, color, fill, group

f + geom_violin(scale = "area"**)** x, y, alpha, color, fill, group, linetype, size, weight

visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se))

j + geom_crossbar(fatten = 2**)** - x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar() - x, ymax, ymin, alpha, color, group, linetype, size, width Also **geom_errorbarh()**.

j + geom_linerange() x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange() - x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

both discrete

g <- ggplot(diamonds, aes(cut, color))

g + geom_count() x, y, alpha, color, fill, shape, size, stroke

e + geom_jitter(height = 2, width = 2**)** x, y, alpha, color, fill, shape, size

maps

data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))

k + geom_map(aes(map_id = state), map = map**)** **+ expand_limits(**x = map$long, y = map$lat**)** map_id, alpha, color, fill, linetype, size

THREE VARIABLES

seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2)); l <- ggplot(seals, aes(long, lat))

l + geom_contour(aes(z = z)**)** x, y, z, alpha, color, group, linetype, size, weight

l + geom_contour_filled(aes(fill = z)**)** x, y, alpha, color, fill, group, linetype, size, subgroup

l + geom_raster(aes(fill = z), hjust = 0.5, vjust = 0.5, interpolate = FALSE**)** x, y, alpha, fill

l + geom_tile(aes(fill = z)**)** x, y, alpha, color, fill, linetype, size, width

ggplot2 Cheat Sheet

Beispieldatensatz

```r
library(dplyr)                                          # Für Pipe (%>%), mutate()

# Daten vorbereiten
D <- read.table("Daten_2.csv", sep = ",", header = TRUE)     # Daten einlesen
n_pat <- nrow(D)                                        # Anzahl Patientinnen
D_processed <- D %>%                                    # PatientIn ID hinzufügen
  mutate(PatientIn = seq(n_pat))
```

Die ersten 12 Zeilen des Dataframes:

DUR	BDI	PatientIn
1.37	9	1
2.18	8	2
1.16	11	3
3.60	0	4
2.33	8	5
1.18	7	6
2.49	3	7
2.74	7	8
2.58	3	9
1.69	11	10
3.51	9	11
2.39	7	12

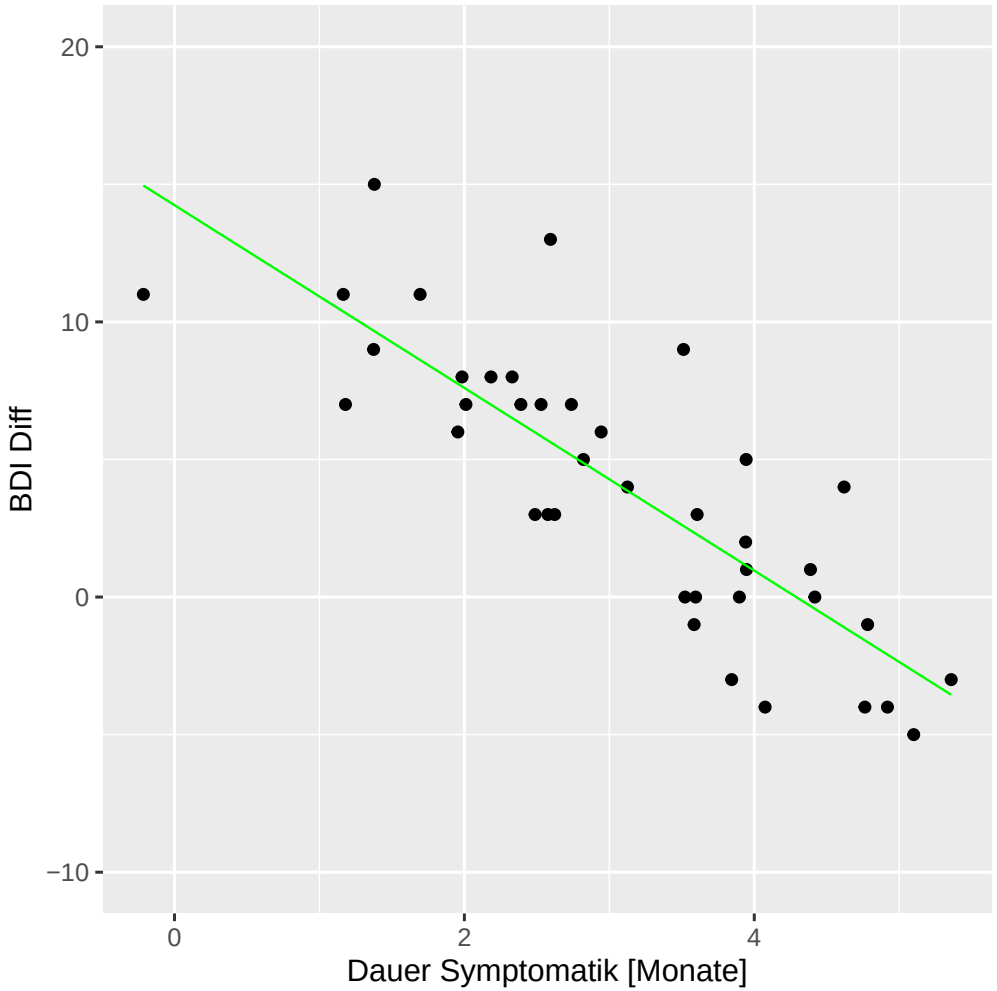
Plotten mit ggplot2

```
library(ggplot2)                                          # Für ggplot()

# Visualisierung
ggplot(
  data = D_processed,                                     # Daten
  mapping = aes(x = DUR, y = BDI)                         # Daten-Axen-mapping
  ) +
  coord_cartesian(ylim = c(-10, 20)) +                    # y-limits anpassen
  geom_point() +                                          # Datenpunkte zeichnen
  geom_smooth(                                            # Ausgleichsgerade zeichnen
    method = "lm",
    color = "blue", se = , linewidth = 0.4
    ) +
  ylab("BDI Diff") + xlab("Dauer Symptomatik [Monate]")  # Achsenbeschriftung
graphics.off()                                            # Schließt browser

ggsave(                                                   # Abbildung speichern
  filename = "ggplot_beispiel.pdf",
  height = 5, width = 5
)
```

Plotten mit ggplot2



R Tools

Quarto

Zotero

Quarto

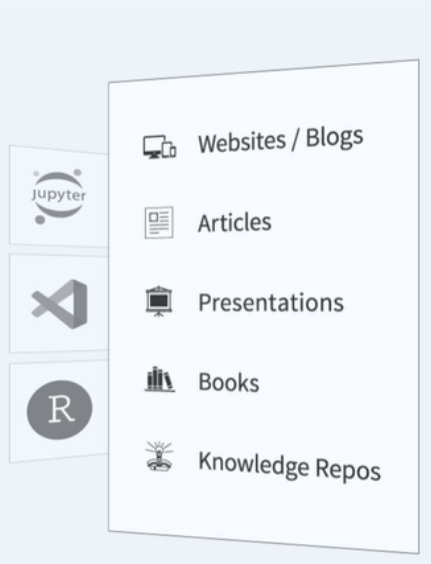
Welcome to Quarto

An open-source scientific and technical publishing system

- Author using Jupyter notebooks or with plain text markdown in your favorite editor.
- Create dynamic content with Python, R, Julia, and Observable.
- Publish reproducible, production quality articles, presentations, websites, blogs, and books in HTML, PDF, MS Word, ePub, and more.
- Share knowledge and insights organization-wide by publishing to Posit Connect, Confluence, or other publishing systems.
- Write using Pandoc markdown, including equations, citations, crossrefs, figure panels, callouts, advanced layout, and more.

Analyze. Share. Reproduce. You have a story to tell with data—tell it with Quarto.

Get Started Guide

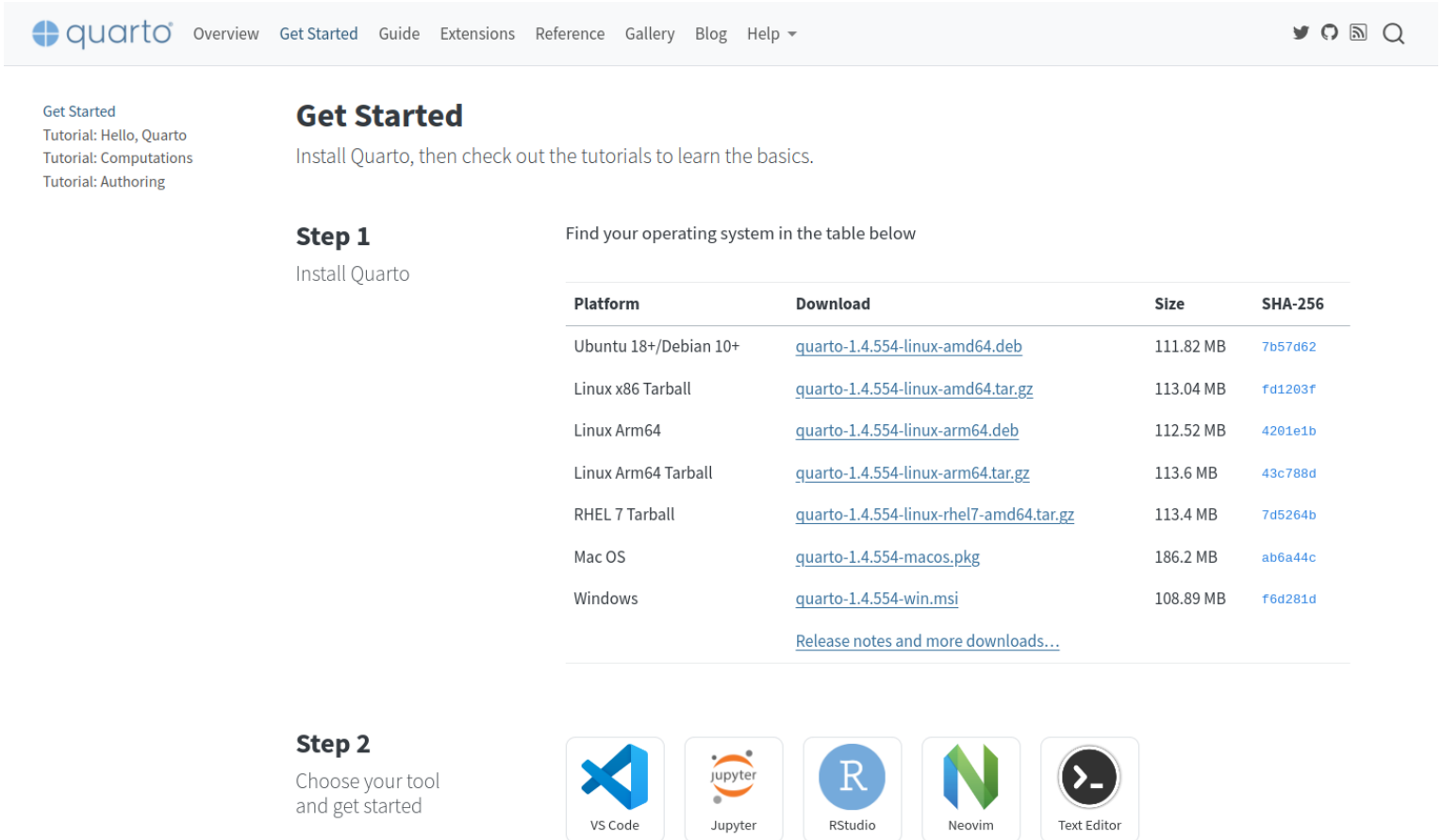


Quarto Website

Was ist Quarto?

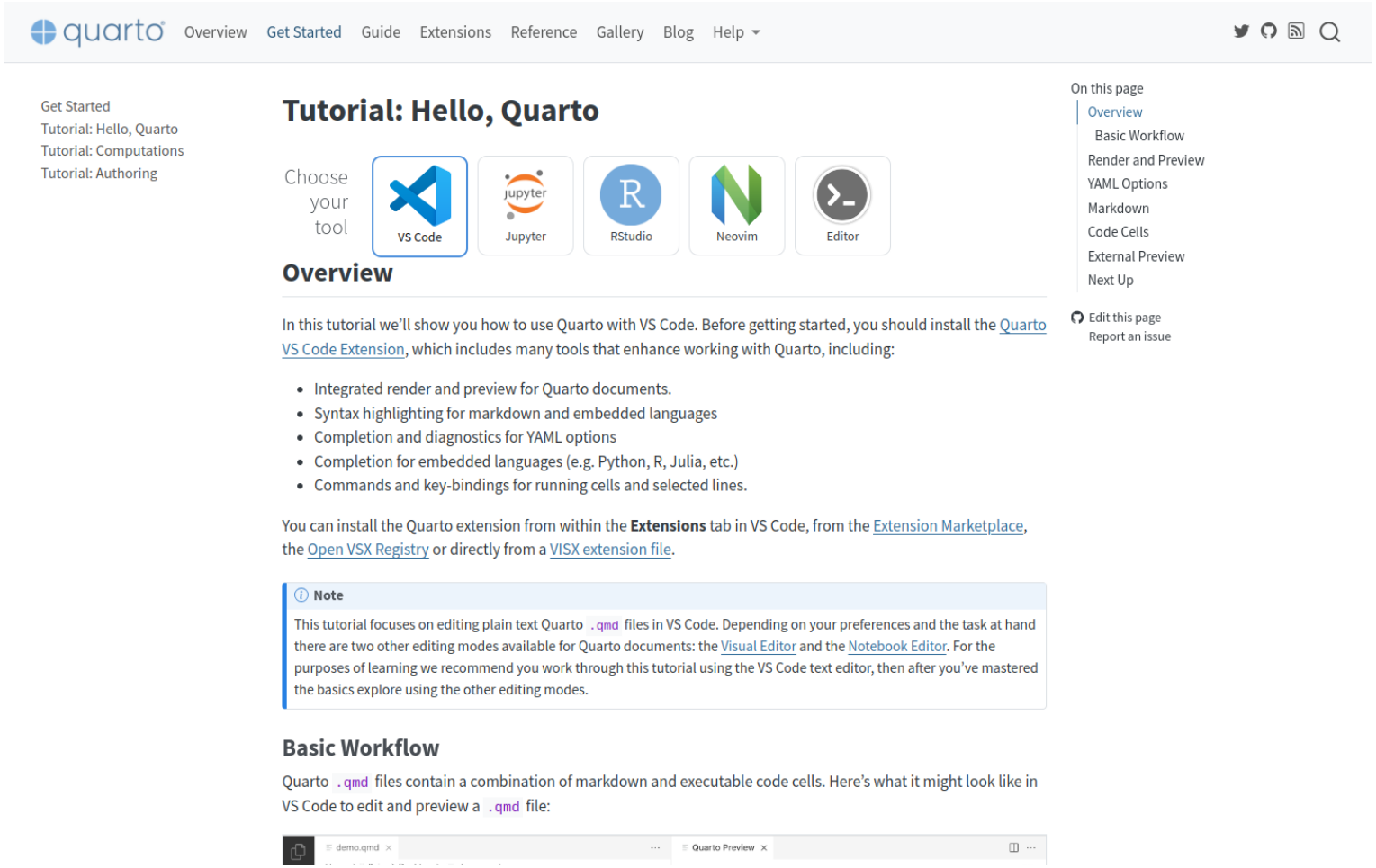
- Ein seit 2022 verfügbares freies wissenschaftlich-technisches Publikationssystem
- Eine Weiterentwicklung von RMarkdown und RBookdown durch Posit
- RMarkdown/RBookdown sind RStudio Adaptationen von Markdown und Jupyter Notebooks
- Allgemeines Ziel ist hier die einfache Integration von ausführbarem Programmiercode in ein ansprechendes Text-, Tabellen- und Abbildungslayout für Web- und Printdokumente.
- Quarto nutzt Markdown und Latex für Layoutprozesse.
- Quarto nutzt Pandoc für multiple Outputformate (.html, .docx, .pdf, etc.)
- Quarto läuft smoother und schneller als RMarkdown und RBookdown.

Installation von Quarto



Quarto Website Installation

Quarto und VS Code



Quarto Website Tutorial: VS Code

Markdown

- Eine Markup Language (Auszeichnungssprache) zur Erzeugung formatierten Texts
- Eine HTML Alternative zur Erstellung von Webseiten etc. mithilfe einfacher Texteditoren
- Von John Gruber und Aaron Swartz 2004 mit dem Ziel hoher Lesbarkeit entwickelt

Text using Markdown syntax	Corresponding HTML produced by a Markdown processor	Text viewed in a browser
`Heading` `=======` `Sub-heading` `----------` `# Alternative heading` `## Alternative sub-heading` `Paragraphs are separated` `by a blank line.` `Two spaces at the end of a line` `produce a line break.`	`<h1>Heading</h1>` `<h2>Sub-heading</h2>` `<h1>Alternative heading</h1>` `<h2>Alternative sub-heading</h2>` `<p>Paragraphs are separated` `by a blank line.</p>` `<p>Two spaces at the end of a` `line ` `produce a line break.</p>`	Heading Sub-heading Alternative heading Alternative sub-heading Paragraphs are separated by a blank line. Two spaces at the end of a line produce a line break.

`Text attributes _italic_, **bold**,`
`` `monospace` ``.
 	
`Horizontal rule:`
 	
`---`
	`<p>Text attributes <em>italic</em>,`
	`<strong>bold</strong>,`
	`<code>monospace</code>.</p>`
	
	`<p>Horizontal rule:</p>`
	
	`<hr />`
		Text attributes *italic*, **bold**, `monospace`.
		Horizontal rule:

LaTeX

- Ein Softwarepaket zur Vereinfachung von TeX
- TeX ist ein von Donald E. Knuth ab 1977 entwickeltes Textsatzsystem mit Makrosprache
- LaTeX wurde von Leslie Lamport Anfang 1984 entwickelt
- LaTeX ist insbesondere für mathematische Berichte und Präsentationen (Beamer) nützlich

```
\footnotesize
\begin{theorem}[Datenverteilung des Allgemeinen Linearen Modells]
\justifying
\normalfont
Es sei
\begin{equation}
\upsilon = X\beta + \varepsilon \mbox{ mit } \varepsilon \sim N(0_n,\sigma^2I_n)
\end{equation}
das ALM. Dann gilt
\begin{equation}
\upsilon \sim N(\mu,\sigma^2I_n) \mbox{ mit } \mu := X\beta \in \mathbb{R}^n.
\end{equation}
\end{theorem}
```

Theorem (Datenverteilung des Allgemeinen Linearen Modells)

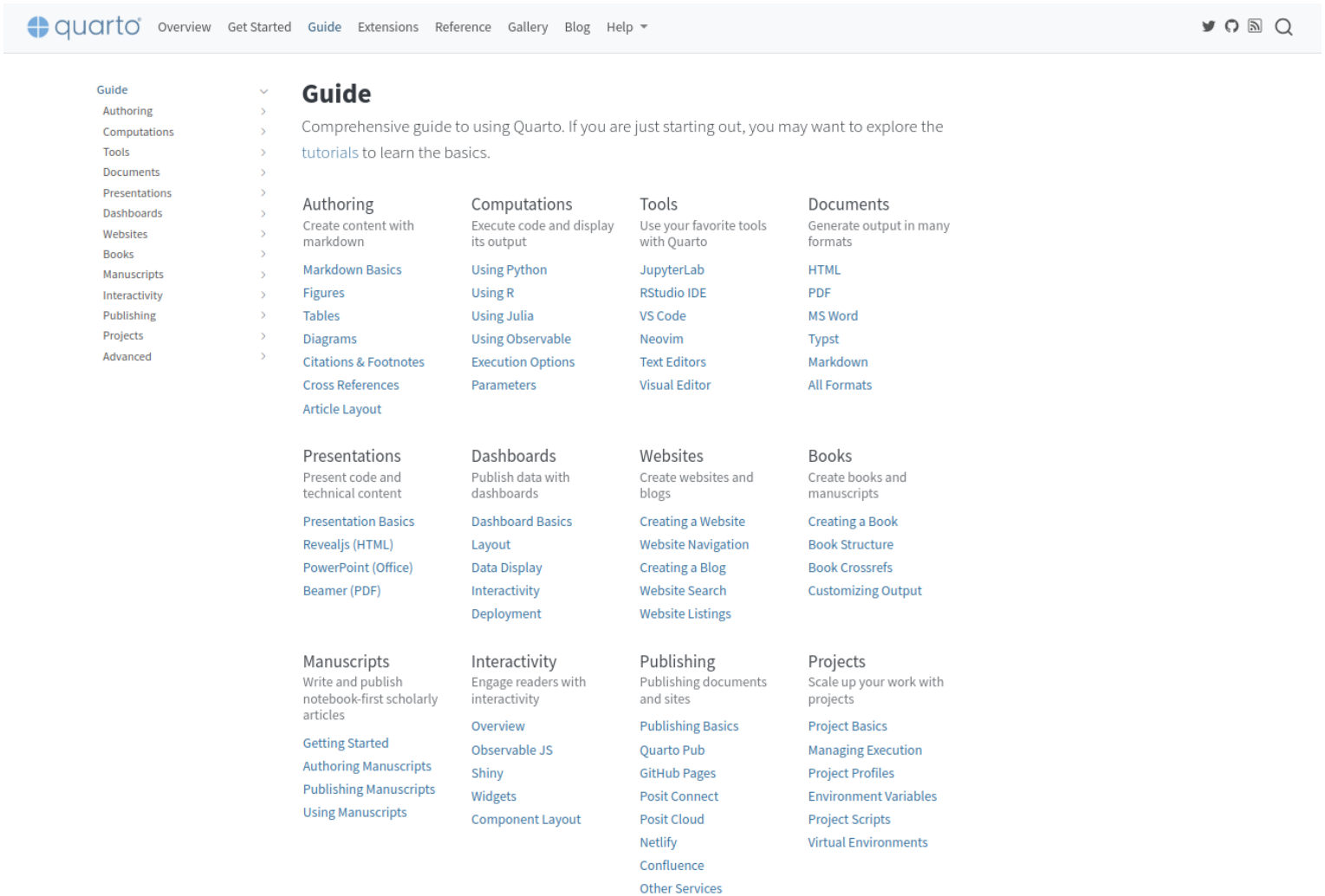
Es sei

$$\upsilon = X\beta + \varepsilon \text{ mit } \varepsilon \sim N(0_n, \sigma^2 I_n) \tag{7}$$

das ALM. Dann gilt

$$\upsilon \sim N(\mu, \sigma^2 I_n) \text{ mit } \mu := X\beta \in \mathbb{R}^n. \tag{8}$$

Quarto Guide



Quarto Website Guide

Quarto Beispiel

```
---
title: "Quarto Demonstration"
author: "Belinda Fleischmann"
date: today
format: pdf
---

# Überschrift zu Kapitel 1.

Hier steht der Text für Kapitel 1. Darin könnte auch eine Abbildung enthalten sein.

![](../Abbildungen/otto.png){width="10%"}

## Überschrift zum Unterkapitel 1.1

Hier steht der Text für Unterkapitel 1.1. Manche Worte möchte ich **fett** und manche Worte _kursiv_, und Befehle
in `monospace` schreiben. Mögliche Farben möchte ich mit Stichpunkten auflisten.

* \textcolor{blue}{blau}
* \textcolor{green}{grün}
* \textcolor{red}{rot}
* \textcolor{gray}{grau}

Wenn wir mathematische Ausdrücke mit Dollarzeichen umrahmen, werden sie mithilfe von \LaTeX formatiert.
So können wir z.B. die Verteilung eines Zufallsvektors formal mit $\upsilon \sim N(\mu, \sigma^2 I_n)$ mit
$\mu := X\beta \in \mathbb{R}^n$ aufschreiben.
```

Quarto Demonstration

Belinda Fleischmann

2024-05-02

Überschrift zu Kapitel 1.

Hier steht der Text für Kapitel 1. Darin könnte auch eine Abbildung enthalten sein.

Überschrift zum Unterkapitel 1.1

Hier steht der Text für Unterkapitel 1.1. Manche Worte möchte ich **fett** und manche Worte *kursiv*. und Befehle in `monospace` schreiben. Mögliche Farben möchte ich mit Stichpunkten auflisten.

- blau
- grün
- rot
- grau

Wenn wir mathematische Ausdrücke mit Dollarzeichen umrahmen, werden sie mithilfe von LaTeX formatiert. So können wir z.B. die Verteilung eines Zufallsvektors formal mit $v \sim N(\mu, \sigma^2 I_n)$ mit $\mu := X\beta \in \mathbb{R}^n$ aufschreiben.

Beispielbericht

Beispielpräsentation

R Tools

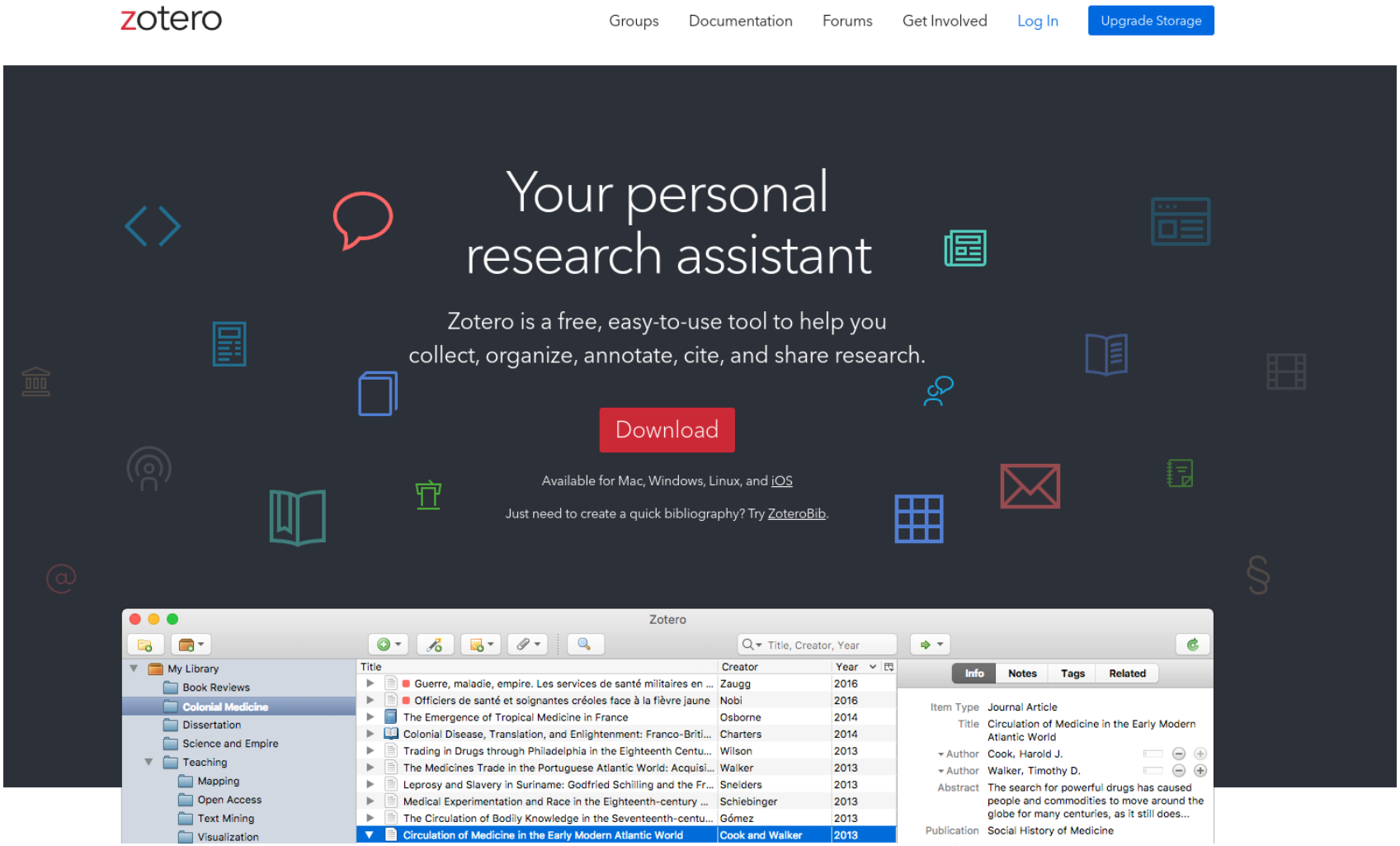
Quarto

Zotero

Was ist ein Reference Manager?

- Reference Manager sind Literaturverwaltungsprogramme
- Reference Manager unterstützen Zitationen und das Erstellen von Literaturverzeichnissen
- Zitierstile können automatisch auf bestimmte Spezifikationen (z.B. APA) eingestellt werden
- Reference Manager dienen auch als digitale Bibliotheken
- Kommerzielle Reference Manager sind z.B. EndNote, Citavi, Mendeley und Papers
- Kostenlose/Freemium Reference Manager sind z.B. JabRef und Zotero
- Eine Integration in Quarto erlaubt z.B. der Export der eigenen Library in das BibTex Format.

Zotero



Zotero Website

Zotero Documentation