



Analyse und Dokumentation

BSc Psychologie SoSe 2024

Belinda Fleischmann

Inhalte basieren teilweise auf Design, Analyse, Dokumentation von Dirk Ostwald, lizenziert unter CC BY-NC-SA 4.0

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 Varianzanalyse	BF
03.07.24	Präsentationen	Multiple 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

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

Visual Studio Code

Docs

Updates

Blog

API

Extensions

FAQ

Learn

Search Docs

Download

OVERVIEW

SETUP

GET STARTED

USER GUIDE

SOURCE CONTROL

TERMINAL

GITHUB COPILOT

LANGUAGES

OVERVIEW

JAVASCRIPT

JSON

HTML

CSS, SCSS and LESS

Typescript

Markdown

PowerShell

C++

Java

R#P#P

Python

Julia

R

Ruby

Post

SQL

T-SQL

C#

.NET

Polyscript

NODEJS / JAVASCRIPT

TYPESCRIPT

PYTHON

JAVA

C++

C#

DOCKER

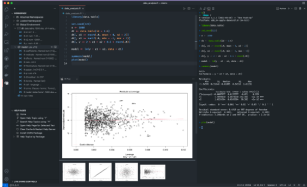
DATA SCIENCE

AI/ML

R in Visual Studio Code

The **R programming language** is a dynamic language built for statistical computing and graphics. It is commonly used in statistical analysis, scientific computing, machine learning, and data visualization.

The **R extension** for Visual Studio Code supports extended syntax highlighting, code completion, linting, formatting, interacting with R terminals, viewing data, plots, workspace variables, help pages, managing packages and working with **R Markdown** documents.



IN THIS ARTICLE

- Getting started
- Running R code
- Code completion (IntelliSense)
- Linting
- Workspace viewer
- Debugging
- Next steps

[Subscribe](#)

[Ask questions](#)

[Follow @code](#)

[Request features](#)

[Report issues](#)

[Watch videos](#)

Getting started

1. **Install R** ($\geq 3.4.0$) for your platform. For Windows users, it is recommended to check **Save version number in registry** during installation so that the R extension can find the R executable automatically.
2. **Install `languageserver`** in R.

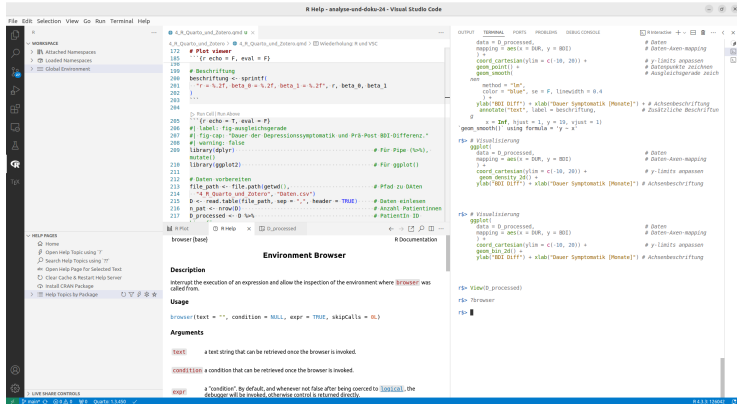
```
install.packages("languageserver")
```
3. **Install the R extension for Visual Studio Code.**
4. **Create an R file and start coding.**

To enhance the experience of using R in VS Code, the following software and packages are recommended:

VS Code Website

Wiederholung: R workspace und Interactive Viewer

Help Viewer



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

VS Code Wiki - Interactive viewers

Wiederholung: R workspace und Interactive Viewer

Table Viewer

The screenshot displays the Visual Studio Code interface with an R script open in the editor. The script defines a function `plot_viewer` and processes data from a CSV file. The output of the script is shown in the Table Viewer, which displays a table with 5 columns: (row), THP, DMR, BDI, and Patienten ID. The table contains 13 rows of data.

(row)	THP	DMR	BDI	Patienten ID
1	OST	1.9755	9	1
2	OST	2.1606	8	2
3	OST	1.1644	11	3
4	OST	3.9953	0	4
5	OST	2.3256	8	5
6	OST	1.1795	7	6
7	OST	2.4874	3	7
8	OST	2.7383	7	8
9	OST	2.5758	3	9
10	OST	1.6946	11	10
11	OST	3.5118	9	11
12	OST	2.3898	7	12
13	OST	1.3788	15	13

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

[VS Code Wiki - Interactive viewers](#)

R workspace and Interactive Viewer

List Viewer

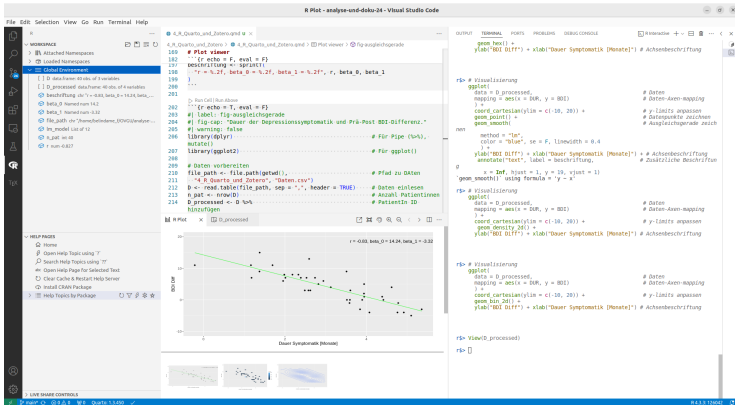
The screenshot displays the VS Code interface with the R workspace and interactive viewer. The left sidebar shows the 'R' workspace with a list of objects: 'D', 'D_processed', 'beta_0', 'beta_1', 'beta_2', 'beta_3', 'beta_4', 'beta_5', 'beta_6', 'beta_7', 'beta_8', 'beta_9', 'beta_10', 'beta_11', 'beta_12', 'beta_13', 'beta_14', 'beta_15', 'beta_16', 'beta_17', 'beta_18', 'beta_19', 'beta_20', 'beta_21', 'beta_22', 'beta_23', 'beta_24', 'beta_25', 'beta_26', 'beta_27', 'beta_28', 'beta_29', 'beta_30', 'beta_31', 'beta_32', 'beta_33', 'beta_34', 'beta_35', 'beta_36', 'beta_37', 'beta_38', 'beta_39', 'beta_40', 'beta_41', 'beta_42', 'beta_43', 'beta_44', 'beta_45', 'beta_46', 'beta_47', 'beta_48', 'beta_49', 'beta_50', 'beta_51', 'beta_52', 'beta_53', 'beta_54', 'beta_55', 'beta_56', 'beta_57', 'beta_58', 'beta_59', 'beta_60', 'beta_61', 'beta_62', 'beta_63', 'beta_64', 'beta_65', 'beta_66', 'beta_67', 'beta_68', 'beta_69', 'beta_70', 'beta_71', 'beta_72', 'beta_73', 'beta_74', 'beta_75', 'beta_76', 'beta_77', 'beta_78', 'beta_79', 'beta_80', 'beta_81', 'beta_82', 'beta_83', 'beta_84', 'beta_85', 'beta_86', 'beta_87', 'beta_88', 'beta_89', 'beta_90', 'beta_91', 'beta_92', 'beta_93', 'beta_94', 'beta_95', 'beta_96', 'beta_97', 'beta_98', 'beta_99', 'beta_100'. The main editor shows the R script 'lm_model - analyse-und-doku-24 - Visual Studio Code'. The script includes comments and code for plotting the results of a linear model. The right sidebar shows the 'OUTPUT' and 'TERMINAL' tabs, with the 'TERMINAL' tab displaying the output of the R script, including the plot of the linear model.

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

[VS Code Wiki - Interactive viewers](#)

R workspace and Interactive Viewer

Plot Viewer



Das R Paket [htpgd](#) 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()
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 Befehlsreihenfolge abweichen.

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

if-statements

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

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

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
	ODER
&	UND

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

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"
```

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"
```

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**

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

pipes

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 tidy cheat sheet for list-column workflow.

```
starwars >-> rowwise() >-> mutate(film_count = length(film))
```

ungroup() (x, ...) Returns ungrouped copy of table.
mtcars >-> mtcars >-> group_by(cyl) >-> ungroup(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 = 3)
```

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) i.e. mpg:cyl
ends_with(match) all_of(s)any_of(s, ..., vars) i.e. log
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))
```

```
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).

```
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)
```



Datenvorverarbeitung mit dplyr

```
print(getwd())
```

```
[1] "/home/belindame_f/OVGU/analyse-und-doku-24/4_R_Quarto_und_Zotero"
```

```
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).

```
library(dplyr)
n <- nrow(D)
D_processed <- D %>%
  mutate(ID = seq(n)) %>%
  mutate(Summe = Variable_1 + Variable_2 + Variable_3)
```

Anzahl Beobachtungen
D wird an nächste Funktion übergeben
ID-Spalte hinzufügen
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

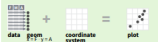
Plotten mit ggplot2

Data visualization with ggplot2 : : CHEATSHEET

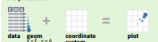


Basics

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.



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



Complete the template below to build a graph.

```
ggplot (data = DATA) +  
  GEOM FUNCTION (aes(mapping = MAPPLING)) +  
  SCALE (COORD, position = POSITION) +  
  COORDINATE FUNCTION (COORD) +  
  THEME FUNCTION (THEME)
```

ggplot(data = mpg, aes(x = cty, y = hwy)) Begins a plot that you finish by adding layers. 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", "RRRRGGGG")
linetype - integer or string (0 = "blank", 1 = "solid", 2 = "dashed", 3 = "dotted", 4 = "longdash", 5 = "longdash", 6 = "twodash")

size - integer (line width in mm)

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



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(aes(x = x, y = y))

a = **geom_blank()** and **a** = **expand_limits()** ensure limits include values across all plots.

b = **geom_curve()**(aes(x = x, y = y, curvature = 1)) - x, y, end, y, yend, alpha, angle, color, curvature, linetype, size

a = **geom_path()**(aes(x = x, y = y, linetype = "solid", linetype = "round", linetype = 2))
linetype = "solid", linetype = 2

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

b = **geom_rect()**(aes(xmin = long, ymin = lat, xmax = long + 2, ymax = lat + 2)) - xmin, ymin, ymax, xmin, 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(intercept = lat))

b = **geom_vline()**(aes(intercept = lat))

b = **geom_segment()**(aes(xend = lat + 1, xend = long + 1))
b = **geom_spoke()**(aes(angle = 1:155, radius = 2))

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, fill, group, linetype, size

c = **geom_histogram()**(binwidth = 3)
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(class))

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, nudje, x = 1, nudje, y = 2)) - x, label, alpha, angle, color, family, fontface, hjust, linetype, 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()**(size = "thin")
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, nudje, x = 1, nudje, y = 2)) - x, y, label, alpha, angle, color, family, fontface, hjust, linetype, size, vjust

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, xmin, alpha, color, fill, group, linetype, shape, size, weight

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

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

both discrete

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

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

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

THREE VARIABLES

h <- ggplot(diamonds, aes(carat, log2(carat), fill))
h = **geom_point()**(aes(long, lat))

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

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

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 = "bw")
x, y, alpha, color, group, linetype, size

visualizing error

j <- ggplot(mpg, aes(class, hwy))
j = **geom_crosstab()**(x = x, y = y, ymax, ymin, alpha, color, fill, group, linetype, size)

j = **geom_errorbar()**(x = y, ymax, ymin, alpha, color, group, linetype, size, width)

j = **geom_errorbarh()**(x = y, ymax, ymin, alpha, color, group, linetype, size, width)

j = **geom_linerange()**
x, y, ymin, ymax, alpha, color, fill, group, linetype, size

j = **geom_pointrange()**(x = y, ymin, ymax, alpha, color, fill, group, linetype, size)

maps

k <- data.frame(murder = USArrests\$Murder, state = tolower(rownames(USArrests)))

k = **geom_map()**(map_id = state, map = map)

k = **geom_map()**(map_id = state, map = map)

k = **geom_map()**(map_id = state, map = map)

k = **geom_map()**(map_id = state, map = map)

k = **geom_map()**(map_id = state, map = map)

k = **geom_map()**(map_id = state, map = map)

k = **geom_map()**(map_id = state, map = map)

Plotten mit ggplot

Beispieldatensatz

```
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

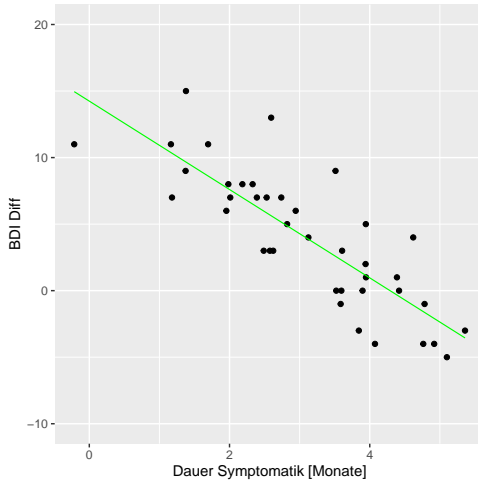
Plotten mit ggplot

```
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 = "green", se = F, 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 ggplot



R Tools

Quarto

Zotero

Welcome to Quarto

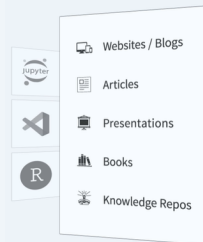
An open-source scientific and technical publishing system

- 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

Get Started

Tutorial: Hello, Quarto
Tutorial: Computations
Tutorial: Authoring

Get Started

Install Quarto, then check out the tutorials to learn the basics.

Step 1

Install Quarto

Find your operating system in the table below

Platform	Download	Size	SHA-256
Ubuntu 18+/Debian 10+	quarto-1.4.554-linux-amd64.deb	111.82 MB	7b57d62
Linux x86 Tarball	quarto-1.4.554-linux-amd64.tar.gz	113.04 MB	f0d1283f
Linux Arm64	quarto-1.4.554-linux-arm64.deb	112.52 MB	4201e1b
Linux Arm64 Tarball	quarto-1.4.554-linux-arm64.tar.gz	113.6 MB	43c788d
RHEL 7 Tarball	quarto-1.4.554-linux-rhel7-amd64.tar.gz	113.4 MB	7d5264b
Mac OS	quarto-1.4.554-macos.pkg	186.2 MB	ab6a44c
Windows	quarto-1.4.554-win.msi	108.89 MB	f6d281d
Release notes and more downloads...			

Step 2

Choose your tool and get started



Quarto Website Installation

Quarto und VS Code

quarto

Overview

Get Started

Guide

Extensions

Reference

Gallery

Blog

Help

Get Started

Tutorial: Hello, Quarto

Tutorial: Computations

Tutorial: Authoring

Tutorial: Hello, Quarto

Choose your tool

VS Code

Jupyter

RStudio

Neovim

Editor

Overview

In this tutorial we'll show you how to use Quarto with VS Code. Before getting started, you should install the [Quarto VS Code Extension](#), which includes many tools that enhance working with Quarto, including:

- Integrated render and preview for Quarto documents.
- Syntax highlighting for markdown and embedded languages
- Completion and diagnostics for YAML options
- Completion for embedded languages (e.g. Python, R, Julia, etc.)
- Commands and key-bindings for running cells and selected lines.

You can install the Quarto extension from within the **Extensions** tab in VS Code, from the [Extension Marketplace](#), the [Open VSX Registry](#) or directly from a [VSX extension file](#).

Note

This tutorial focuses on editing plain text Quarto `.qmd` files in VS Code. Depending on your preferences and the task at hand there are two other editing modes available for Quarto documents: the [Visual Editor](#) and the [Notebook Editor](#). For the purposes of learning we recommend you work through this tutorial using the VS Code text editor, then after you've mastered the basics explore using the other editing modes.

Basic Workflow

Quarto `.qmd` files contain a combination of markdown and executable code cells. Here's what it might look like in VS Code to edit and preview a `.qmd` file:

document x

Quarto Preview x

On this page

Overview

Basic Workflow

Render and Preview

YAML Options

Markdown

Code Cells

External Preview

Next Up

Edit this page

Report an issue

Quarto Website Tutorial: VS Code

Analyse und Dokumentation | © 2024 Belinda Fleischmann CC BY 4.0 | Folie 36

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
<p>Heading *****</p> <p>Sub-heading -----</p> <p># Alternative heading</p> <p>## Alternative sub-heading</p> <p>Paragraphs are separated by a blank line.</p> <p>Two spaces at the end of a line produce a line break.</p>	<pre><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></pre>	<p>Heading</p> <p>Sub-heading</p> <p>Alternative heading</p> <p>Alternative sub-heading</p> <p>Paragraphs are separated by a blank line.</p> <p>Two spaces at the end of a line produce a line break.</p>
<p>Text attributes <i>_italic_</i>, **bold**, <code>'monospace'</code>.</p> <p>Horizontal rule:</p> <p>---</p>	<pre><p>Text attributes italic, bold, <code>monospace</code>.</p> <p>Horizontal rule:</p> <hr /></pre>	<p>Text attributes <i>italic</i>, bold, <code>monospace</code>.</p> <p>Horizontal rule:</p> <hr/>

- 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 \text{ mit } \varepsilon \sim N(0_n, \sigma^2 I_n)
\end{equation}
das ALM. Dann gilt
\begin{equation}
\upsilon \sim N(\mu, \sigma^2 I_n) \text{ mit } \mu := X\beta \in \mathbb{R}^n.
\end{equation}
\end{theorem}
```




Theorem (Datenverteilung des Allgemeinen Linearen Modells)





Es sei

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

das ALM. Dann gilt

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

 Overview Get Started Guide Extensions Reference Gallery Blog Help



Guide

Authoring

Computations

Tools

Documents

Presentations

Dashboards

Websites

Books

Manuscripts

Interactivity

Publishing

Projects

Advanced

Guide

Comprehensive guide to using Quarto. If you are just starting out, you may want to explore the tutorials to learn the basics.

Authoring
Create content with markdown
Markdown Basics
Figures
Tables
Diagrams
Citations & Footnotes
Cross References
Article Layout

Computations
Execute code and display its output
Using Python
Using R
Using Julia
Using Observable
Execution Options
Parameters

Tools
Use your favorite tools with Quarto
JupyterLab
RStudio IDE
VS Code
Neovim
Text Editors
Visual Editor

Documents
Generate output in many formats
HTML
PDF
MS Word
Typst
Markdown
All Formats

Presentations
Present code and technical content
Presentation Basics
Reveal.js (HTML)
PowerPoint (Office)
Beamer (PDF)

Dashboards
Publish data with dashboards
Dashboard Basics
Layout
Data Display
Interactivity
Deployment

Websites
Create websites and blogs
Creating a Website
Website Navigation
Creating a Blog
Website Search
Website Listings

Books
Create books and manuscripts
Creating a Book
Book Structure
Book Crossrefs
Customizing Output

Manuscripts
Write and publish notebook-first scholarly articles
Getting Started
Authoring Manuscripts
Publishing Manuscripts
Using Manuscripts

Interactivity
Engage readers with interactivity
Overview
Observable JS
Shiny
Widgets
Component Layout

Publishing
Publishing documents and sites
Publishing Basics
Quarto Pub
GitHub Pages
Posit Connect
Posit Cloud
Netlify
Confluence
Other Services

Projects
Scale up your work with projects
Project Basics
Managing Execution
Project Profiles
Environment Variables
Project Scripts
Virtual Environments

Quarto Website Guide

Analyse und Dokumentation | © 2024 Belinda Fleischmann CC BY 4.0 | Folie 39

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.

{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  $\epsilon \sim N(\mu, \sigma^2 I_n)$  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.

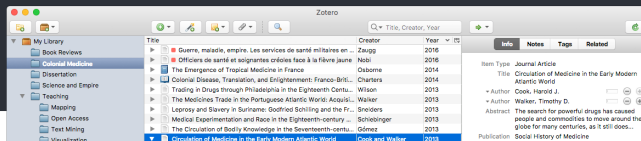
Your personal research assistant

Zotero is a free, easy-to-use tool to help you collect, organize, annotate, cite, and share research.

Download

Available for Mac, Windows, Linux, and iOS

Just need to create a quick bibliography? Try [ZoteroBib](#)



[Zotero Website](#)

[Zotero Documentation](#)