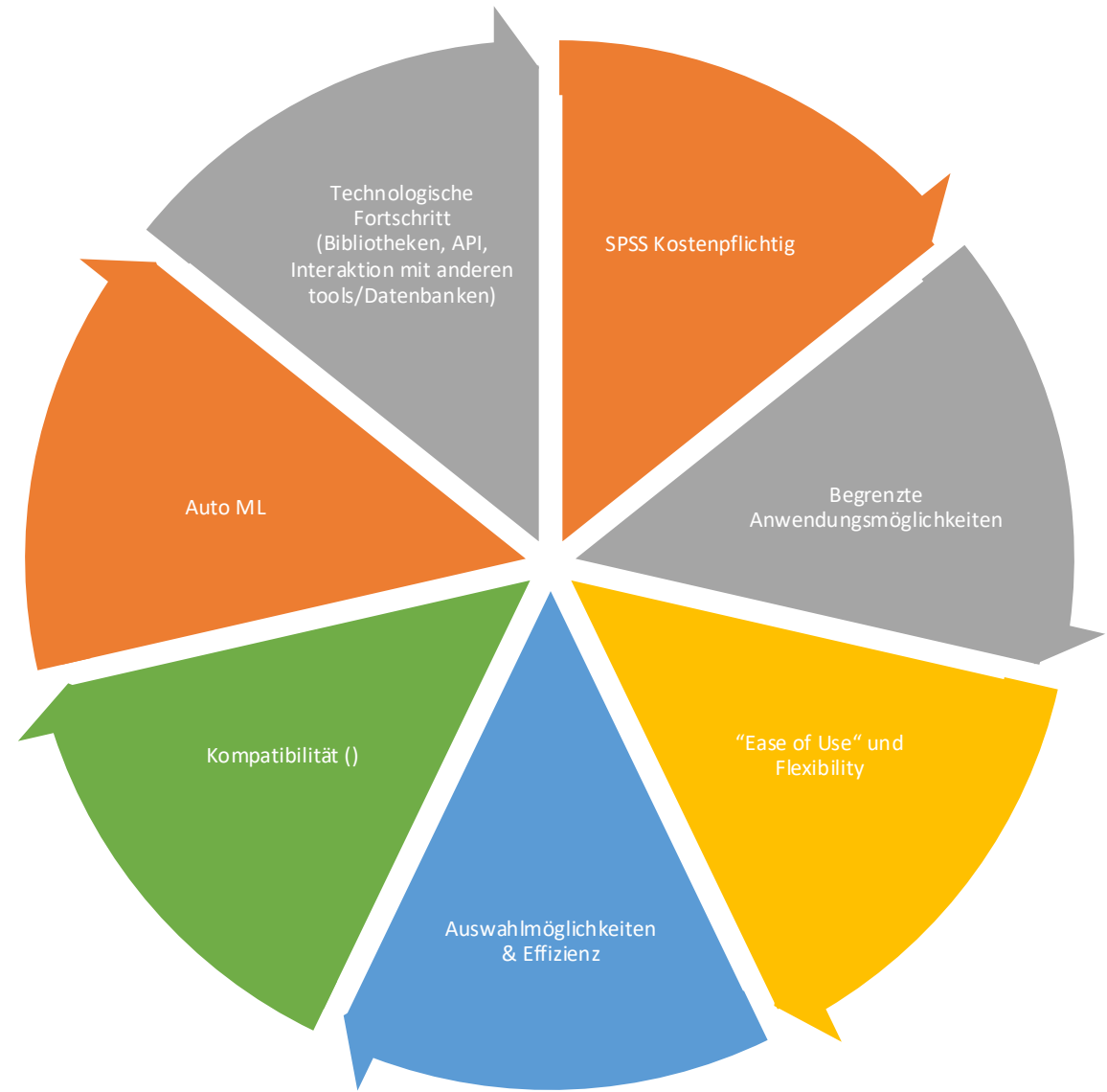


A blurred background image of a business meeting. Several people in professional attire (suits, blouses) are gathered around a table. One person is holding a tablet displaying a document with charts and text. Another person is holding a smartphone. There are white coffee cups on the table. The overall atmosphere is professional and collaborative.

# Einführung in R

Dr. Houssam Jedidi

# Motivation



# Was ist R

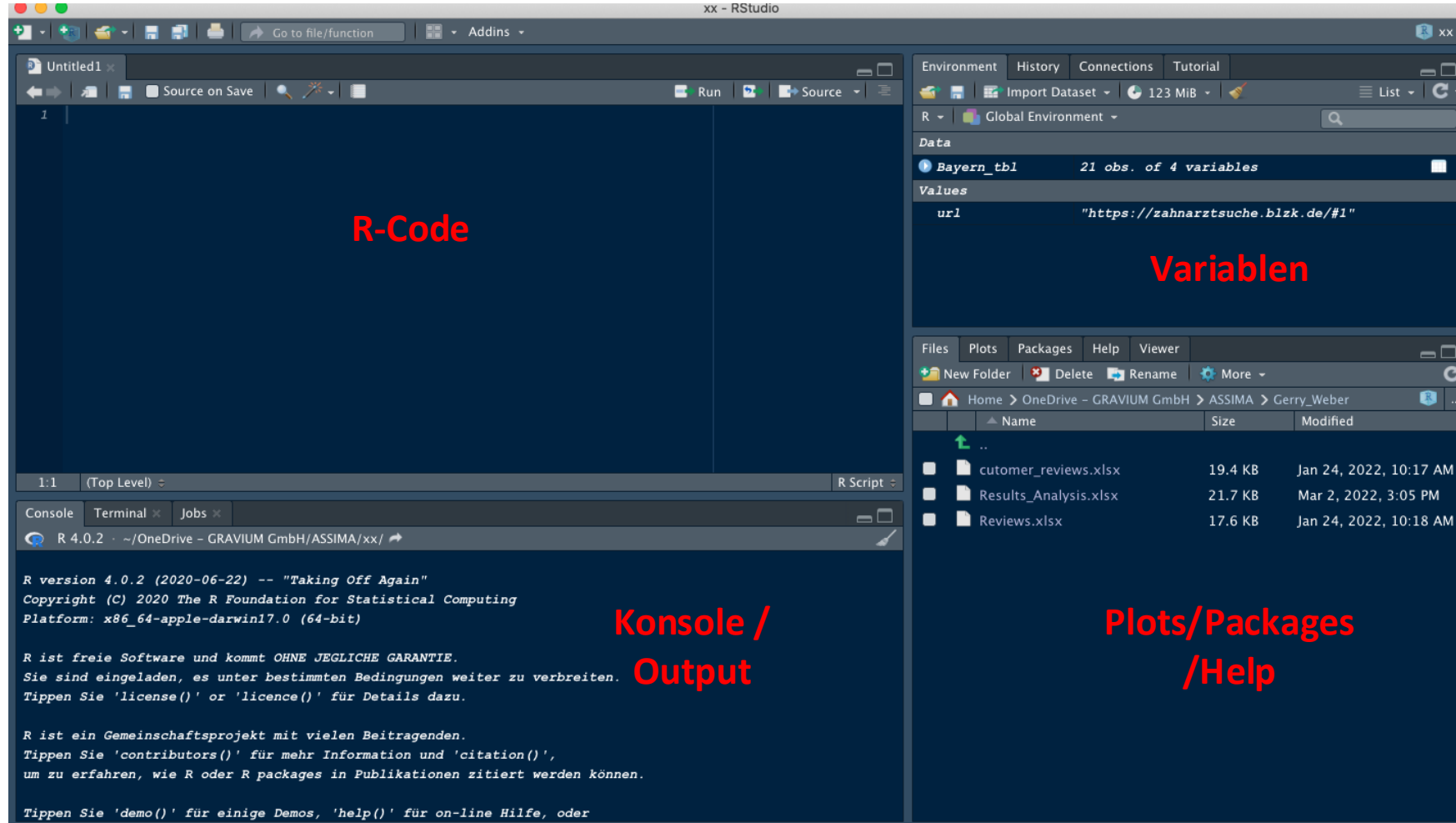
- Programmiersprache als auch eine Statistikumgebung
- Open source
- Kostenlos
- Name R: **R**oss Ihaka und **R**obert Gentleman

# R- Installation

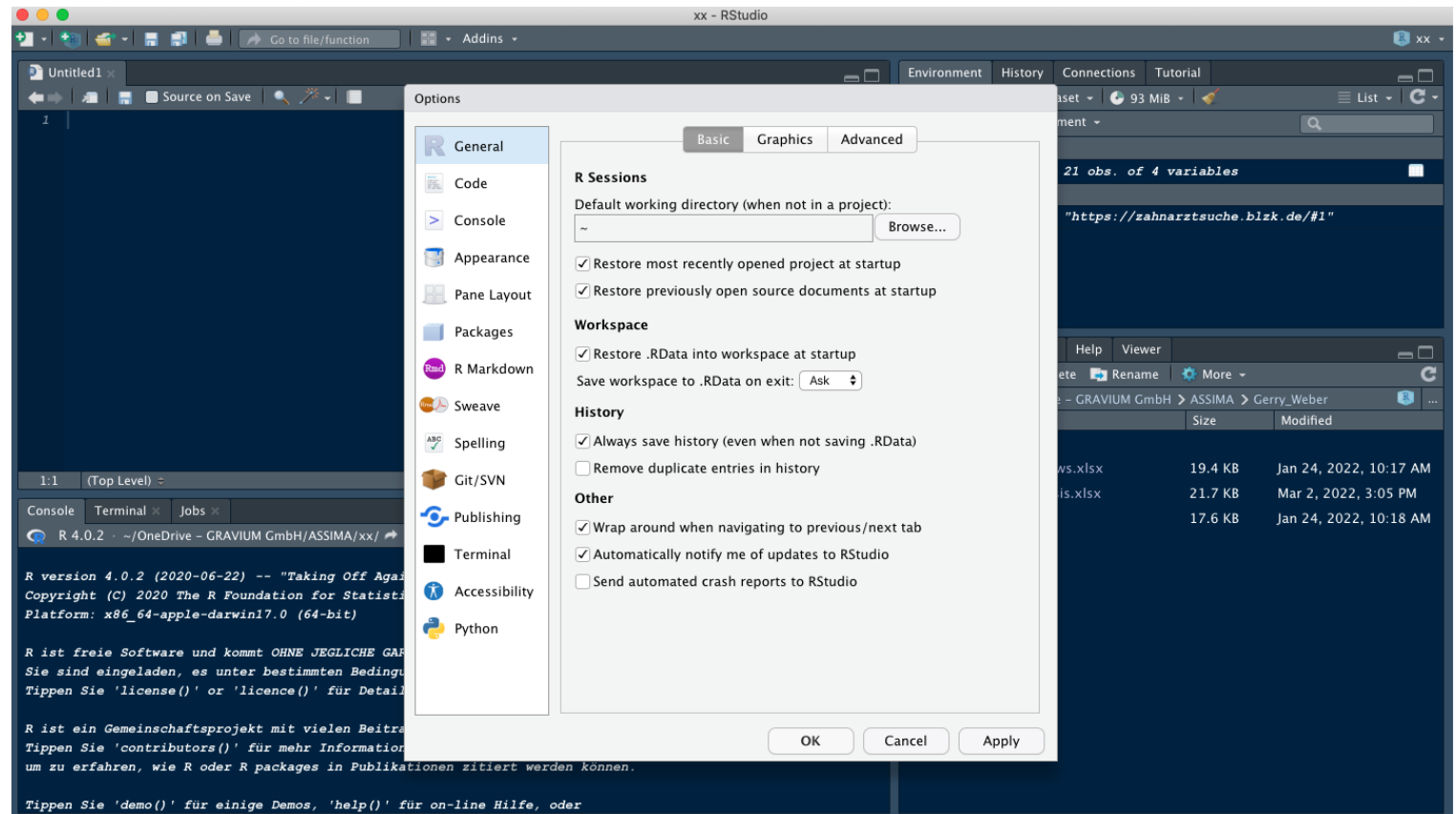
R und Rstudio:

<https://www.rstudio.com/products/rstudio/download/>

# R-studio



# Global options



# Projekte

Packages stellen zusätzliche Funktionen zur Verfügung, welche nicht in der Basisausstattung von R (base R) enthalten sind. Wir installieren zunächst eine Sammlung von Packages zur Datenmanipulation (tidyr, dplyr)


Packages beinhalten Bibliotheken, die zahlreiche Funktionalitäten erlauben.

Weitere Packages werden wir installieren, wenn wir sie benötigen

# Projekte

New Project

[Back](#) **Create New Project**



Directory name:

Create project as subdirectory of:  
 [Browse...](#)

☐ Create a git repository

☐ Use packrat with this project

☐ Open in new session

[Create Project](#) [Cancel](#)



# Arithmetische Operatoren

Operator	Bedeutung
+	Addition
-	Substraktion
*	Multiplikation
/	Division
^ oder **	Potenz
x %*% y	Matrixmultiplikation
X %% y	Modulo
X %/% y	Ganzzeilige Teilung

# Logische Operatoren

Operator	Bedeutung
<	kleiner
<=	Kleiner gleich
>	größer
>=	Größer gleich
==	gleich
!=	ungleich
!x	Nicht x
x   y	x oder y
x & y	x und y
Xor(x,y)	Entweder in x oder in y aber nicht in beiden

# Statistische Funktionen

Funktion	Bedeutung
Mean(x, na.rm= False)	Mittelwert
Sd(x)	Standardabweichung
Var(x)	Varianz
Median(x)	Median
Quantile(x, probs)	Quantile von x, probs ist der Vektor mit Wahrscheinlichkeiten
Sum(x)	Summe
Min(x)	Minimalwert
Max(x)	Maximalwert
Range(x)	Intervall: min _max

# Hilfreiche Befehle

Befehl	Bedeutung
C()	Vektor erstellen
Seq(from, to, by)	Generiert eine Sequenz (Zahlen, Buchstaben, etc.)
Rep(x, times, each)	Wiederhole (x, n-mal, jedes element m-mal)
Head(x, n= 10)	Zeigt die ersten 10 Elemente
Tail(x, n=5)	Zeigt die letzten 5 Elemente

# Erste Befehle

```
c(10, 20, 30, 40, 50, 60)
```

```
mean(c(1,2,3,4,5,6,7,8,9,10))
```

```
mean(c(1,2,3,NA,5,6,7,8,9,10), na.rm=False)
```

```
mean(c(1,2,3,NA,5,6,7,8,9,10), na.rm=False)
```

```
sd((1,2,3,4,5,6))
```

```
sum(c(12,13,14,15,1))
```

```
min(c(231,124,212,312,901,120))
```

```
range(c(24,31,15,22,77,12,101))
```

```
seq(from=12, to=139, by=7)
```

```
rep(1:28,times=2, each=2)
```

```
scale(c(1, 2, 3, 4, 5, 6), center = TRUE, scale = FALSE/False)
```

```
# ziehen mit Zurücklegen:
```

```
sample(c(1, 2, 3, 4, 5, 6), size = 1, replace = TRUE)
```

```
# ziehen mit Zurücklegen, ungleich gewichtet:
```

```
Sample(c(1,2,3,4,5,6), size=12, replace=TRUE, prob=c(4/12, 1/12, 1/12, 2/12, 2/12, 2/12 ))
```

# Übung

1. Erzeugen sie eine Sequenz von 0 bis 100 in 5-er Schritten
2. Berechnen Sie den Mittelwert des Vektors [5,6,7,3,4,2]
3. Geben Sie die Spannweite  $x_{\min}$   $x_{\max}$  aus
4. Berechnen Sie die Summe des Vektors
5. Zentrieren Sie diesen Vektor
6. Generieren Sie einen Vektor, der aus 100 Wiederholungen der Zahl 3 besteht.
7. Simulieren Sie einen Münzwurf mit der Funktion `sample()`. Tipp: nehmen Sie für Kopf 1 und für Zahl 0. Simulieren Sie 100 Münzwürfe
8. Simulieren Sie einen `Trick_Münze` mit `p #0`
9. Generieren Sie einen Vektor, das aus 15 Wiederholungen der Zahl 9 besteht

# Lösung

1. `seq(from = 1, to = 100, by = 5)`
2. `mean(C(5,6,7,3,4,2))`
3. `range(C(5,6,7,3,4,2))`
4. `sum(C(5,6,7,3,4,2))`
5. `scale(C(5,6,7,3,4,2), center=True, scale=True)`

[# wenn center = TRUE: zentrieren  
# wenn scale = TRUE: durch SD teilen]

6. `rep(3, times = 100)`
7. `sample(c(1, 2), size = 100, replace = TRUE)`
8. `outcomes <- c(0, 1)`  
`probabilities <- c(0.4, 0.6)`  
`coin_toss <- sample(outcomes, 1, prob = probabilities)`
9. `mein_vektor <- rep(9, times = 15)`

# Variable definieren

 `<-` ist hier ein spezieller Zuweisungspfeil

Abkürzung: ALT + -

Beispiel:

```
meine_Var <- 14
```

Variablenamen:

Kombination Buchstaben & Zahlen

keine Leerzeichen

Bedeutungsvoll: Gruppe\_1<-

```
meine_Ausgaben<-
```

```
x_sd<-
```

unmöglich wäre: x mittelwert / sd von x / ...



# Datentypen

## 1. Numerischer Vektor:

als integer (ganze Zahlen) oder/und double (reelle Zahlen)

## 2- Character Vektor:

besteht aus Zeichen / Buchstaben

hilfreiche Funktionen:

`typeof()` – was ist das?

`length()` – wieviele Elemente

`attributes()` - Metadaten

## 3. Logical Vektor:

True / False oder NA

# Numerischer Vektor

```
test_vek<- c(13,12.55, 6.75,10.60, 3.3,21)
```

```
typeof(test_vek)
```

```
length(test_vek)
```

```
test_vek[1]
```

```
test_vek[-2]
```

```
test_vek[1,4]
```

```
test_vek[2:5]
```

```
test_vek[-c(1,2)]
```

```
test_vek[-c(1:4)]
```

# Matrizen

R ist auf Vektoren basiert

Eine Matrix ist ein Vektor mit dimension >1

```
x<- 1:8
```

```
dim(x)<- c(2,4)    → 2 Zeilen & 4 Spalten
```

```
m_1<- matrix(x<-1:8, nrow=2, ncol=4, byrow=FALSE)
```

```
m_2<- matrix(x<-1:8, nrow=2, ncol=4, byrow=FALSE)
```

```
m_transponiert<- t(m_2)
```

# Matrizen -wichtige Befehle

`cbind`

```
x_1<- 1:3
```

```
x_2<- 6:8
```

```
m_1<- cbind(x_1,x_2)
```

`rbind`

```
x_1<- 1:3
```

```
x_2<- 6:8
```

```
m_2<- rbind(x_1,x_2)
```

`is.na()`

```
Bsp_1<- c(12,23,13,11,5,NA,NA,0,7)
```

```
is.na(Bsp_1)
```

# Charakter Vektor

```
my_text<- c(„Liebe Grüße“,“Guten Tag“, “Hallo“, “Guten appetit“)
```

```
my_text
```

```
typeof(my_text)
```

```
length(my_text)
```

# Charakter Vektor – wichtige Befehle

```
Vorname<-“Dein Vorname“
```

```
Nachname<-“Dein Nachname“
```

```
Grüße<- paste(„Hallo, mein Name ist“, Vorname,Nachname, sep=“_“)
```

```
print(Grüße)
```

# Logischer Vektor\_ BSP

`set.seed(123)` → Reproduzierbarkeit

`x<- rnorm(24)` → 24 Zufallsvariablen, die normal verteilt sind

`x>0` → logischer Vektor für Positivität der Elemente

`x[x>0]` oder `index<- x>0` → Indizierung

- Suche nach Elementen, welche eine Standardabweichung über den Mittelwert haben:

`mean_x<- mean(x)`

`sd_x<- sd(x)`

`x>mean_x+sd_x` → logical

`x[x>(mean_x+sd_x)]` Alternativ `x[x>(mean(x)+sd(x))]`

`which(x > (mean(x) + sd(x)))` → Elemente

`x[which(x > (mean(x) + sd(x)))]` → Werte

# Faktoren

```
Geschlecht<-c(„männlich“,“weiblich“, „männlich“, „männlich“,  
„männlich“, „männlich“, “weiblich“)
```

```
typeof(Geschlecht)
```

```
Geschlecht<- as.factor(Geschlecht, levels=c(„männlich“, “weiblich“))
```

```
typeof(Geschlecht)
```

```
class(Geschlecht)
```

```
attributes(Geschlecht)
```



# Liste

Vektoren bestehen aus Elementen desselben Typs

Listen können heterogene Elemente haben

```
my_first_list<- list(1:3,"F",c(FALSE,TRUE,TRUE),c(2.2,2.4,2.5))  
my_first_list[1]
```

Erstellung einer „named“ Liste

```
my_named_list<- list(my_int=1:3,  
                     my_char=„F“,  
                     my_log= c(FALSE,TRUE,TRUE),  
                     my_double=c(2.2,2.4,2.5))
```

# Dataframe

wichtigste Datenformat

Datensätze sind als Dataframes repräsentiert ~ SPSS

2Dimensional: Spalten und Zeilen (Gleiche Länge)

Moderne Bezeichnung in R Tibble

# Dataframe-Bsp.

```
library(dplyr)
```

```
library(tibble)
```

```
my_df<- data.frame(Geschlecht=factor(c(„männlich“,„männlich“, „weiblich“,„männlich)),  
  Alter=c(37,25,20,19),
```

```
  Beruf=c(„Arzt“, „ CTO“, „NA“, „Student“))
```

```
my_df<- tibble(Geschlecht=factor(c(„männlich“,„männlich“, „weiblich“,„männlich)),  
  Alter=c(37,25,20,19),
```

```
  Beruf=c(„Arzt“, „ CTO“, „NA“, „Student“))
```

```
attributes(my_df)
```

```
dim(my_df)
```

```
my_df$Beruf / my_df[3] / my_df[, Beruf]
```

```
my_df[2,2]
```

# Übungen

```
x<- rnorm(20,mean=2, sd=0.5)
```

1. Vektor x auf 0 Dezimalstellen runden
2. Vektor x auf 3 Dezimalstellen runden

```
round(x = x, digits = 0)
```

```
Meine_Zahl<-7.98432
```

1. Meine\_Zahl auf die nächste natürliche Zahl auf-/abrunden

```
ceiling(meine_Zahl)
```

```
floor(meine_Zahl)
```

# Übungen

1. Berechnen Sie den Mittelwert der Variable Alter (aus dem Dataframe bsp.)
2. Deskriptive Statistik ausgeben (summary())
3. Kombinieren Sie die beiden Matrizen M\_1 und M\_2 in M\_3  
M\_1 <- matrix(rnorm(48, mean = 110, sd = 5), ncol = 4)  
M\_2 <- matrix(rnorm(48, mean = 100, sd = 10), ncol = 4)
4. wählen Sie aus M\_3 alle Elemente aus M\_2 (matrixsubsetting / which)
5. wählen Sie aus M\_3 die ersten 2 Spalten und die ersten 6 Zeilen aus. gespeichert wird alles in M\_4

# Übungen

6. Nutzen Sie den Befehl „paste“ um aus den folgenden 3 Variablen, eine neue Variablenidentität zu generieren:

```
Initialen<-c(„CR“, „LM “, „KB“, „RL“)
```

```
Verein<-c(„ManU“, „PSG“, „RM“, „FCB“)
```

```
Trikot<-c(„7“, „30“, „9“, „9“)
```

# Umgang mit fehlenden Werten

```
X <- c(22,3,7,NA,NA,67)
```

1.
  - a. `X[!is.na(X)]`
  - b. `X[is.na(X)]`
  - c. `X[X==NA]= 0`
2. `Y = c(1,3,12,NA,33,7,NA,21)`, welcher R-befehl erlaubt es alle NA durch 11 zu ersetzen?
  - a. `Y[Y==NA]= 11`
  - b. `Y[is.na(Y)]= 11`
  - c. `Y[Y==11] = NA`

# Umgang mit fehlenden Werten

1. `X = c(34,33,65,37,89,NA,43,NA,11,NA,23,NA)`, Anzahl der Vorkommnisse von NA in X?
  - a. `sum(X==NA)`
  - b. `sum(X == NA, is.na(X))`
  - c. `sum(is.na(X))`
2. Orange Datensatz durch den Befehl `data(Orange)`, dann wenn `Alter=118`, durch NA ersetzen.
3. `A <- c (33, 21, 12, NA, 7, 8)`, Berechne den Mittelwert von A ohne NA



# Datensatz aus R-packages- Faktoren

1. Gapminder-Datensatz aus dem Gapminder-Paket, sowie forcats laden. Überprüft bitte die Niveaus der Kontinentalfaktor-Variablen und ihre Häufigkeit in den Daten

```
library(gapminder)
```

```
library(forcats)
```

```
gp <- gapminder
```

```
fct_count(gp$continent)
```

# Datensatz aus R-packages- Faktoren

2. Achtung, ein Kontinent, die Antarktis, fehlt – fügt ihn als letzte von sechs Ebenen hinzu.

```
gp$continent <- fct_expand(gp$continent, "Antarctica")  
fct_count(gp$continent)
```

3. Wir haben uns umentschieden und wollen doch die Antarktis löschen.

```
gp$continent <- fct_drop(gp$continent)  
fct_count(gp$continent)
```

# Datensatz aus R-packages- Faktoren

4. wir waren doch ungenau bei den Kontinenten und wollen daher folgende Länder als Süd\_am kennzeichnen ("Argentina", "Bolivia", "Brazil", "Chile", "Colombia", "Ecuador", "Paraguay", "Peru", "Uruguay", "Venezuela")

```
süd_Am <- c("Argentina", "Bolivia", "Brazil", "Chile", "Colombia", "Ecuador", "Paraguay", "Peru", "Uruguay", "Venezuela")
gp$continent <- fct_expand(gp$continent, "South America", "North America")
gp$continent[gp$country %in% süd_Am ] <- "South America"
gp$continent[gp$continent == "Americas"] <- "North America"
gp$continent <- fct_drop(gp$continent)
fct_count(gp$continent)
```

# Datensatz aus R-packages- Faktoren

5. wir wollen die Ebenen des Kontinentalfaktors in alphabetischer Reihenfolge anordnen.

```
gp$continent <- fct_relevel(gp$continent, sort(levels(gp$continent)))  
fct_count(gp$continent)
```

6. `gender <- c("f", "m ", "male ", "male", "female", "FEMALE", "Male", "f", "m")`. Wir wollen den folgenden unübersichtlichen Vektor in einen Faktor mit zwei Stufen: Male/Female umwandeln

```
gender <- as_factor(gender)  
gender <- fct_collapse(  
  gender, Female = c("f", "female", "FEMALE"),  
  Male = c("m ", "m", "male ", "male", "Male") )  
fct_count(gender)
```

# Strings- Zeichenketten

```
addresses <- c("14 Pine Street, Los Angeles", "152 Redwood Street, Seattle", "8  
Washington Boulevard, New York")
```

```
products <- c("TV ", " laptop", "portable charger", "Wireless Keybord", "  
HeadPhones ")
```

```
long_sentences <- stringr::sentences[1:10]
```

```
field_names <- c("order_number", "order_date", "customer_email",  
"product_title", "amount")
```

```
employee_skills <- c("John Bale (Beginner)", "Rita Murphy (Pro)", "Chris White  
(Pro)", "Sarah Reid (Medium)")
```

# Strings- Zeichenketten

1. Adressvektor normalisieren, indem die Großbuchstaben durch Kleinbuchstaben ersetzt sind.

`str_to_lower(string = addresses)`

2. Den numerischen Teil des Adressvektors ziehen

`str_extract(string = addresses, pattern = "[:digit:]+")`

3. Adressvektor in zwei Teile: Adresse und Ort

`str_split(string = addresses, pattern = ", ", simplify = T)`

# Strings- Zeichenketten

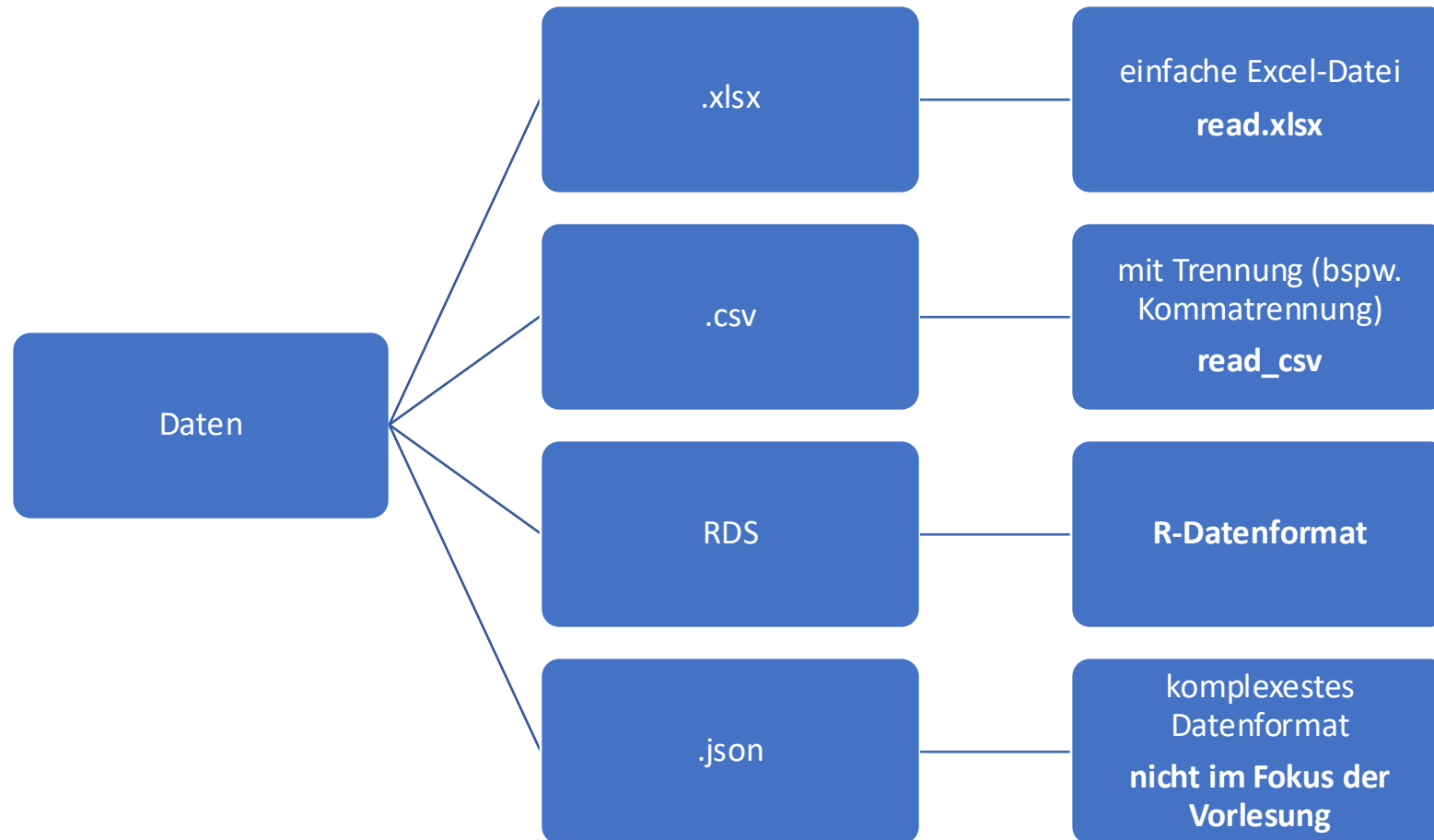
4. Den Adressvektor in drei Teile zerlegen: Hausnummer, Straße und Ort. Das Ergebnis sollte eine Matrix sein.

```
str_split(string = addresses, pattern = "(?<=[:digit:]) |, ", simplify = T)
```

5. Bereiten Sie die `field_names` für die Anzeige vor, indem Sie alle Unterstriche durch Leerzeichen ersetzen und sie in Großbuchstaben umwandeln

```
str_to_title(string = str_replace_all(string = field_names, pattern = "_", replacement = " "))
```

# Daten einlesen





# Daten transformieren

## 1-Tidy Daten:

- Package: tidyr / dplyr
- gehören zu tidyverse → stellt Kompatible Packages, die eine „Grammatik“ der Datenverarbeitung zur Verfügung
- Tidyr: Dieses Package verwenden wir, um Datensätze von wide zu long zu transformieren, und natürlich auch von long zu wide.
- dplyr: Dieses Packages stellt eine Sammlung von Funktionen zur Verfügung, um Daten zu manipulieren: Fälle/Variablen auswählen, Daten zusammenfassen, neue Variablen kreieren, neue Datensätze erstellen.

# Daten transformieren

## 2-Der Pipe Operator %:

- Wir haben schon festgestellt, dass Code schnell unübersichtlich werden kann, wenn wir eine Sequenz von Operationen ausführen. Dies führt zu verschachtelten Funktionsaufrufen.
- Beispiel: Wir haben einen numerischen Vektor von Messwerten (an einer Stichprobe der Grösse  $n = 10$  erhoben oder wie hier zu Übungszwecken durch einen (Pseudo-)Zufallsgenerator generiert) und wollen diese zuerst zentrieren, dann die Standardabweichung berechnen, und anschließend noch auf zwei Nachkommastellen runden.

# Daten transformieren

## 2-Der Pipe Operator %:

```
set.seed(1283)
stichprobe <- rnorm(10, 24, 5)
stichprobe
#> [1] 24.74984 21.91726 23.98551 19.63019 23.96428 22.83092 18.86240
#> [8] 19.08125 23.76589 21.88846
```

- Die gewünschte Berechnung der gerundeten Standardabweichung der zentrierten Werte können wir als verschachtelte Funktionsaufrufe durchführen:

```
round(sd(scale(stichprobe,
               center = TRUE,
               scale = FALSE)),
      digits = 2)
#> [1] 2.19
```

# Daten transformieren

## 2-Der Pipe Operator %

```
stichprobe_z <- scale(stichprobe, center = TRUE,  
                      scale = FALSE)  
  
sd_stichprobe_z <- sd(stichprobe_z)  
sd_stichprobe_z_gerundet <- round(sd_stichprobe_z,  
                                  digits = 2)  
  
sd_stichprobe_z_gerundet  
#> [1] 2.19
```

Es gibt nun aber eine sehr elegante Methode, um Funktionen nacheinander aufzurufen, ohne diese Funktionen ineinander verschachtelt schreiben zu müssen: wir benutzen dafür den pipe Operator. Dieser wird vom Package dplyr zur Verfügung gestellt und sieht so aus:

# Daten transformieren

## 2-Der Pipe Operator %

```
library(dplyr)
stichprobe %>%
  scale(center = TRUE, scale = FALSE) %>%
  sd() %>%
  round(digits = 2)
#> [1] 2.19
```

- 1) Wir beginnen mit dem Objekt stichprobe und übergeben es mit %>% als Argument an die Funktion scale()
- 2) Wir wenden scale(), mit den zusätzlichen Argumenten center = TRUE, scale = FALSE darauf an, und übergeben den Output als Argument an die Funktion sd()
- 3) Wir wenden sd() an (ohne weitere Argumente) und reichen den Output als Argument weiter an round()
- 4) round(), mit dem weiteren Argument digits = 2, wird ausgeführt. Da kein weiterer pipe folgt, wird der Output in die Konsole geschrieben.

# Daten transformieren

## 2-Der Pipe Operator %

```
x %>% f() %>% g() %>% h()
```

```
# oder
```

```
x %>%  
  f() %>%  
  g() %>%  
  h()
```

```
# ist äquivalent zu
```

```
h(g(f(x)))
```

# Daten transformieren

## 3- Reshaping

- `Gather()`

Wir benutzen `gather()`, wenn wir einen wide Datensatz zu einem long Datensatz konvertieren wollen; d.h. `gather()` wird dazu verwendet, mehrere Spalten, welche evtl. Stufen eines Faktors repräsentieren könnten, zu einer Spalte zusammenzufügen, welche den Faktor selber repräsentiert. Die Werte in den ursprünglichen Variablen werden in einer Werte-Variable zusammengefasst.

```
gather(data, key, value, column1, column2, ...)  
  
# oder mit %>%  
  
data %>%  
  gather(key, value, column1, column2, ...)
```

# Daten transformieren

## 3- Reshaping (gather)

```
bsp_wide <- data_frame(  
  ID = factor(as.character(1:5)),  
  A = floor(rnorm(5, 70, 1)),  
  B = floor(rnorm(5, 50, 2)))
```

```
bsp_wide  
#> # A tibble: 5 x 3  
#>   ID      A      B  
#>   <fct> <dbl> <dbl>  
#> 1 1      71     50  
#> 2 2      71     48  
#> 3 3      70     52  
#> 4 4      69     49  
#> 5 5      69     50
```

```
# Nicht vergessen, den Output einer neuen  
# Variablen zuzuweisen  
library(tidyr)  
bsp_long <- bsp_wide %>%  
  gather(key = Bedingung, value = Score, A, B, -ID)
```

```
bsp_long  
#> # A tibble: 10 x 3  
#>   ID Bedingung Score  
#>   <fct> <chr>    <dbl>  
#> 1 1      A      71  
#> 2 2      A      71  
#> 3 3      A      70  
#> 4 4      A      69  
#> 5 5      A      69  
#> 6 1      B      50  
#> # ... with 4 more rows
```



# Daten transformieren

## 3- Reshaping (gather)

```
#> # A tibble: 100 x 5
#>   Vpnr  Gruppe      Pretest Posttest Difference_PrePost
#>   <fct> <fct>      <dbl>    <dbl>          <dbl>
#> 1 1      Kontrollgruppe  4.29     3.21          1.08
#> 2 2      Kontrollgruppe  6.18     5.99          0.190
#> 3 3      Kontrollgruppe  3.93     4.17         -0.239
#> 4 4      Kontrollgruppe  5.06     4.76          0.295
#> 5 5      Kontrollgruppe  6.45     5.64          0.814
#> 6 6      Kontrollgruppe  4.49     4.67         -0.180
#> # ... with 94 more rows
```

```
Therapy_long <- Therapy %>%
  gather(key = messzeitpunkt,
         value = rating,
         Pretest, Posttest, -Vpnr, -Gruppe)
```

```
# messzeitpunkt muss ein Faktor sein
```

```
Therapy_long$messzeitpunkt <- factor(Therapy_long$messzeitpunkt,
                                     levels = c("Pretest", "Posttest"))
```

Therapy\_long

```
#> # A tibble: 200 x 4
#>   Vpnr  Gruppe      messzeitpunkt rating
#>   <fct> <fct>      <fct>          <dbl>
#> 1 1      Kontrollgruppe Pretest          4.29
#> 2 2      Kontrollgruppe Pretest          6.18
#> 3 3      Kontrollgruppe Pretest          3.93
#> 4 4      Kontrollgruppe Pretest          5.06
#> 5 5      Kontrollgruppe Pretest          6.45
#> 6 6      Kontrollgruppe Pretest          4.49
#> # ... with 194 more rows
```

# Daten transformieren

## 3- Reshaping (spread)

- `spread()` ist quasi das Gegenteil von `gather()`. Diese Funktion nimmt einen Faktor und eine Messvariable und “verteilt” die Werte der Messvariable über neue Spalten, welche die Stufen des Faktors repräsentieren.
- Dies bedeutet, dass wir `spread()` verwenden, wenn wir aus einem long Datensatz einen wide Datensatz machen wollen.

```
spread(data, key, value)
```

```
# oder
```

```
data %>%
```

```
  spread(key, value)
```

```
bsp_long
#> # A tibble: 10 x 3
#>   ID      Bedingung Score
#>   <fct> <fct>      <dbl>
#> 1 1      A          71
#> 2 2      A          71
#> 3 3      A          70
#> 4 4      A          69
#> 5 5      A          69
#> 6 1      B          50
#> # ... with 4 more rows
```

```
bsp_wide_2 <- bsp_long %>%
```

```
  spread(key = Bedingung, value = Score)
```

```
bsp_wide_2
```

```
#> # A tibble: 5 x 3
#>   ID      A      B
#>   <fct> <dbl> <dbl>
#> 1 1      71    50
#> 2 2      71    48
#> 3 3      70    52
#> 4 4      69    49
#> 5 5      69    50
```

# Daten transformieren

## 3- Reshaping (spread)

```
Therapy_long
#> # A tibble: 200 x 4
#>   Vpnr  Gruppe      messzeitpunkt rating
#>   <fct> <fct>      <fct>          <dbl>
#> 1 1      Kontrollgruppe Pretest      4.29
#> 2 2      Kontrollgruppe Pretest      6.18
#> 3 3      Kontrollgruppe Pretest      3.93
#> 4 4      Kontrollgruppe Pretest      5.06
#> 5 5      Kontrollgruppe Pretest      6.45
#> 6 6      Kontrollgruppe Pretest      4.49
#> # ... with 194 more rows
```

```
Therapy_wide <- Therapy_long %>%
  spread(key = messzeitpunkt, value = rating)
```

```
Therapy_wide
#> # A tibble: 100 x 4
#>   Vpnr  Gruppe      Pretest Posttest
#>   <fct> <fct>      <dbl>    <dbl>
#> 1 1      Kontrollgruppe 4.29     3.21
#> 2 2      Kontrollgruppe 6.18     5.99
#> 3 3      Kontrollgruppe 3.93     4.17
#> 4 4      Kontrollgruppe 5.06     4.76
#> 5 5      Kontrollgruppe 6.45     5.64
#> 6 6      Kontrollgruppe 4.49     4.67
#> # ... with 94 more rows
```

# Daten manipulieren

## 1- select()

```
# df steht für data frame  
select(df, variable1, variable2, -variable3)
```

```
# oder
```

```
df %>% select(variable1, variable2, -variable3)
```

```
# nur ID
```

```
bsp_long %>% select(ID)
```

```
# Bedingung und Score  
bsp_long %>% select(Bedingung, Score)  
#> # A tibble: 10 x 2  
#>   Bedingung Score  
#>   <fct>      <dbl>  
#> 1 A          71  
#> 2 A          71  
#> 3 A          70  
#> 4 A          69  
#> 5 A          69  
#> 6 B          50  
#> # ... with 4 more rows
```

```
# oder
```

```
bsp_long %>% select(-ID)  
#> # A tibble: 10 x 2  
#>   Bedingung Score  
#>   <fct>      <dbl>  
#> 1 A          71  
#> 2 A          71  
#> 3 A          70  
#> 4 A          69  
#> 5 A          69  
#> 6 B          50  
#> # ... with 4 more rows
```

# Daten manipulieren

## 2- select(starts\_with / ends\_with / Contains)

```
# einschliessen
Therapy_long %>% select(starts_with("Gr"))
#> # A tibble: 200 x 1
#>   Gruppe
#>   <fct>
#> 1 Kontrollgruppe
#> 2 Kontrollgruppe
#> 3 Kontrollgruppe
#> 4 Kontrollgruppe
#> 5 Kontrollgruppe
#> 6 Kontrollgruppe
#> # ... with 194 more rows
```

```
Therapy_long %>% select(ends_with("ng"))
#> # A tibble: 200 x 1
#>   rating
#>   <dbl>
#> 1  4.29
#> 2  6.18
#> 3  3.93
#> 4  5.06
#> 5  6.45
#> 6  4.49
#> # ... with 194 more rows
```

# Daten manipulieren

## 2- select(starts\_with / ends\_with / Contains)

```
# einschliessen
Therapy_long %>% select(starts_with("Gr"))
#> # A tibble: 200 x 1
#>   Gruppe
#>   <fct>
#> 1 Kontrollgruppe
#> 2 Kontrollgruppe
#> 3 Kontrollgruppe
#> 4 Kontrollgruppe
#> 5 Kontrollgruppe
#> 6 Kontrollgruppe
#> # ... with 194 more rows
```

```
Therapy_long %>% select(ends_with("ng"))
#> # A tibble: 200 x 1
#>   rating
#>   <dbl>
#> 1    4.29
#> 2    6.18
#> 3    3.93
#> 4    5.06
#> 5    6.45
#> 6    4.49
#> # ... with 194 more rows
```

# Daten manipulieren

## 3- Variablen umbenennen (rename)

```
df %>% rename(neuer_name = alter_name)
```

## 2- Beobachtungen auswählen (Filter)

```
df %>% filter(variable1 < WERT1 & variable2 == WERT2)
```

## 3- Beobachtungen auswählen sortieren (arrange)

Mit der arrange() Funktion können wir Beobachtungen sortieren, entweder in aufsteigender oder in ab- steigender Reihenfolge.

## 4- Neue Variablen erstellen (mutate)

```
df %>% mutate(neue_variable_1 = FORMEL_1,  
              neue_variable_2 = FORMEL_2)
```

# Daten manipulieren

## 3- Variablen gruppieren (group\_by)

- Nun ist es oft der Fall, dass wir bestimmte Operationen nicht auf den ganzen Datensatz anwenden wollen, sondern nur auf Subgruppen, welche durch Faktorstufen definiert sind. Dafür gibt es die Funktion `group_by()` - diese teilt den Datensatz anhand einer Gruppierungsvariable, wendet eine Funktion auf jeden Teil an, und setzt den Datensatz danach wieder zusammen (split-apply-combine). `group_by()` wird deshalb meistens in Kombination mit anderen Funktionen verwendet

```
df <- group_by(gruppierung_1, gruppierung_2, gruppierung_3)
```



# Daten manipulieren

## 3- Variablen zusammenfassen (summarize)

- Mit `summarize()` oder `summarise()` können wir Variablen zusammenfassen und deskriptive Kennzahlen berechnen. Im Gegensatz zu `mutate()` gibt `summarize()` nicht einen Wert für jede Beobachtung als Output, sondern einen Wert für jede Gruppe.

```
# Gruppenmittelwerte pro Messzeitpunkt
Therapy_long %>%
  group_by(Gruppe, messzeitpunkt) %>%
  summarize(mean_rating = mean(rating))

#> # A tibble: 4 x 3
#> # Groups:   Gruppe [2]
#>   Gruppe      messzeitpunkt mean_rating
#>   <fct>      <fct>          <dbl>
#> 1 Kontrollgruppe Pretest          5.06
#> 2 Kontrollgruppe Posttest         4.65
#> 3 Therapiegruppe Pretest          4.82
#> 4 Therapiegruppe Posttest         4.23
```

# Übung

1. Packages und entsprechende Bibliotheken runterladen:  
dplyr, tidyverse, magrittr, readr, stringr
2. Datensatz benzfinal.csv hochladen
3. Datensatz mit „skim“ (aus skimr) untersuchen
4. Datensatzvollständigkeit mit „plotmissing“ (aus DataExplorer) überprüfen
5. Extrahieren Sie alle Mercedes E-klasse und speichern Sie den Datensatz unter E\_klasse\_df
6. Extrahieren Sie alle Autos, die teurer als 170.000\$ und speichern Sie den Datensatz unter Luxus\_autos\_df
7. Extrahieren Sie und speichern Sie den Datensatz unter Luxus\_autos\_df
8. Nutzen Sie die Function „Case\_when“ um alle Autos, die über 400Ps haben und teurer als 150.000\$ zu filtern. Speichern Sie die Menge unter Luxus\_sport\_df

# Übung

9. Nehmen Sie den Datensatz `Luxus_sport_df` und:
  - Dieser Datensatz muss zuerst ins long Format konvertiert werden.  
Konvertieren Sie Manufacturer falls nötig zu faktoren.
10. Berechnen sie den Mittleren Preis und die Standardabweichung für jeden Hersteller/Modell  
(Hinweis: `group_by`)

# Übung

11. Nutzen Sie folgende Funktion um:

- Preisentwicklung BMW vs. Mercedes zu visualisieren
- Preisentwicklung für Gebrauchtwagen BMW vs. Mercedes zu visualisieren

```
benzfinal %>%  
  filter(state==0) %>% ggplot(aes(x = Year, y = Price,  
    color = Manufacturer,  
    group = Manufacturer,  
    linetype = Manufacturer)) + geom_point(size = 4) +  
  geom_line(size = 2) +  
  labs(x = "Baujahr (Jahre)",  
    y = "Price",  
    title = "Preisentwicklung Gebrauchtwagen") + theme_classic()
```

