Cheatsheet Einführung in R

Statistik II - FS 2024

1. Die R Sprache

Arithmetische Operatoren

```
+               # Addition
-               # Subtraktion
*               # Multiplikation
/               # Division
^ oder **       # Potenz
x %% y          # Modulo (x mod y)
                # 5 %% 2 == 1
x %/% y         # Ganzzahlige Teilung:
                # 5 %/% 2 == 2
```

Logische Operatoren und Funktionen

```
<               # Kleiner
<=              # Kleiner gleich
>               # Grösser
>=              # Grösser gleich
==              # Gleich (Äquivalenz)
!=              # Ungleich
!x              # Nicht x (Verneinung)
x | y           # x ODER y
x & y           # x UND y
xor(x, y)       # Exklusiv ODER (entweder
                # in x oder y, nicht beide)
isTRUE(x)       # testet ob x wahr ist
```

Weiterer nützlicher Operator

```
|>              # Pipe-Operator
```

`|>` gibt den Output einer Funktion an die nächste Funktion *weiter* (default: als erstes Argument).

Numerische Funktionen

```
abs(x)          # Betrag
sqrt(x)         # Quadratwurzel
ceiling(x)      # Aufrunden:
                # ceiling(3.475) ist 4
floor(x)        # Abrunden: floor(3.475)
                # ist 3
round(x,        #
      digits = n) # Runden: round(3.475,
                  # digits = 2) ist 3.48
log(x)          # Natürlicher Logarithmus
log10(x)        # Logarithmus zur Basis 10
exp(x)          # Exponentialfunktion: e^x
sum(x)          # Summe
cumsum(x)       # kumulierte Summe
rank(x)         # (Mittel-)Ränge
```

Statistische Funktionen

```
mean(x,     # Arithmetisches Mittel
     trim) # delta-getrimmtes Mittel
median(x)  # Median
var(x)     # Stichproben-Varianz
sd(x)      # /-Standardabweichung
quantile(x, probs = c(0.25, 0.75), type = 2)
     # allg.: Quantile; hier: Quartile
IQR(x, type = 2)  # Interquartilsabstand
min(x)     # Minimum
max(x)     # Maximum
range(x)   # Streubereich
tb <- table(x, y)
           # Kreuztabelle (x = Zeilen,
           # y = Spalten) mit Zuweisung
proportions(tb) # rel. Häufigkeiten
                # (gemeinsame Anteile)
proportions(tb, margin)
     # margin = 1: zeilenbedingte Anteile
     # margin = 2: spaltenbedingte Anteile
# Zentrieren und Standardisieren:
scale(x, center = TRUE, scale = TRUE)
  # center: zentrieren
  # scale: durch SD teilen
# Ziehen mit/ohne Zurücklegen:
sample(x, size, replace = FALSE, prob)
  # prob: Vektor mit Wahrscheinlichkeiten
  # Mit prob gewichtet ziehen.
```

Wenn `na.rm = TRUE` werden `NA`s entfernt. Ist `na.rm = FALSE` und enthält der zu evaluierende Vektor `NA`s, können manche Funktionen nicht berechnet werden.

Funktionen zur Erstellung eines Vektors

```
c() # kreiert einen Vektor
seq(from, to, by) # Generiert eine Sequenz
x:y # Colon Operator: generiert reguläre
    # Sequenz (Einerschritte von x bis y)
rep(x, times, each) # Wiederholt x
    # times: Sequenz wird n-mal wiederholt
    # each: jedes Element wird n-mal wiederholt
```

Weitere nützliche Funktionen

```
head(x, n = 6) # Zeigt n erste Elemente
tail(x, n = 6) # Zeigt n letzte Elemente
glimpse(x)     # Zeigt Variablennamen,
               # -typen und erste Werte
set.seed(123) # "Startpunkt" für Random-
               # numbers fixieren (z.B. 123)
```

`set.seed()` wird benutzt, um beim nächsten Durchlauf die gleichen (Pseudo-)Zufallszahlen zu erhalten (und so bei Funktionen mit Zufallsgenerator wie z.B. `sample()` der gleiche Output generiert wird).

Beispiele für c(), seq(), rep(), sample()

```
c(1, 2, 3, 4, 5)
```

```
## [1] 1 2 3 4 5
```

```
# äquivalent zu
seq(from = 1, to = 5) # oder 1:5
```

```
## [1] 1 2 3 4 5
```

```
# andere Schrittgrösse
seq(from = 1, to = 10, by = 2)
```

```
## [1] 1 3 5 7 9
```

```
# rep():
my_vector <- 1:4
rep(my_vector, times = 2)
```

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

```
rep(my_vector, each = 2)
```

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

```
rep(my_vector, times = 2, each = 2)
```

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

```
# sample():
set.seed(123)
sample(my_vector, size = 3, replace = FALSE)
```

```
## [1] 3 4 1
```

```
sample(my_vector, size = 8, replace = TRUE)
```

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

```
sample(my_vector, size = 8,
       replace = TRUE,
       prob = c(0.1, 0.1, 0.1, 0.7))
```

```
## [1] 4 4 4 4 2 4 4 4
```

Datentypen

- **numeric vectors**: Numerische Vektoren werden weiter in `integer` (ganze Zahlen) und `double` (reelle Zahlen) unterteilt.
- **character vectors**: bestehen aus Zeichen, werden von Anführungszeichen umgeben (entweder `'` oder `"`), z.B. `'Wort'` oder `"Wort"`.
- **logical vectors**: können nur 3 Werte annehmen: `TRUE`, `FALSE` oder `NA`.

Vektoren haben 3 Eigenschaften:

- Typ: `typeof()`: Welche Art Vektor ist es?
- Länge: `length()`: Wie viele Elemente?
- Attribute: `attributes()`: Zusätzliche Information (Metadaten)

```
# x ist ein Vektor:
x <- 1:8
# x als Matrix (2 Zeilen und 4 Spalten):
dim(x) <- c(2, 4)
# oder direkt als Matrix:
```

```
m1 <- matrix(x, nrow = 2, ncol = 4,
             byrow = FALSE)
m1
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    5    7
## [2,]    2    4    6    8
```

```
m2 <- matrix(x, nrow = 2, ncol = 4,
             byrow = TRUE)
m2
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
```

```
m1_transponiert <- t(m1)
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6
## [4,]    7    8
```

Matrizen (*und Data Frames*) können mit `[ ]` indiziert werden. Subsetting-Logik erfolgt nach einem festen Prinzip mit `[zeilennummer, spaltennummer]`

```
# Alle Zeilen, Spalte 1:
m1[, 1]
```

```
## [1] 1 2
```

```
# Zeile 2, alle Spalten:
m1[2, ]
```

```
## [1] 2 4 6 8
```

```r
# Zeile 1, Spalten 2 bis 3:
m1[1, 2:3]
```

```
## [1] 3 5
```

Character vectors können mit einer Funktion zusammengefügt werden.

```r
paste(LETTERS[1:3], letters[24:26],
      c("eins", "2", "dr3i"), sep = "_")
```

```
## [1] "A_x_eins" "B_y_2"    "C_z_dr3i"
```

Logische Vektoren können zum indizieren (und subsetting) benutzt werden.

```r
set.seed(123)
# 20 normalverteilte Zufallszahlen:
x <- rnorm(20)
index <- x > 1
x[index]
```

```
## [1] 1.558708 1.715065 1.224082 1.786913
```

```r
# oder:
x[x > 1]
```

```
## [1] 1.558708 1.715065 1.224082 1.786913
```

Ein **Faktor** ist ein Vektor von natürlichen Zahlen *(integer vector)*, der mit zusätzlicher Information (Metadaten) versehen ist. Diese `attributes` sind die Objektklasse `factor` und die Faktorstufen `levels`.

```r
# geschlecht ist ein character vector:
geschlecht <- c("male", "female",
                "male", "male", "female")
# Umwandeln in einen Faktor:
geschlecht <- factor(geschlecht,
              levels = c("female", "male"))
```

Mit `relevel()` kann die Referenzkategorie bestimmt werden.

```r
# Resultat muss wieder zugewiesen werden:
geschlecht <- relevel(geschlecht,
                      ref = "male")
```

Während Vektoren aus Elementen desselben Typs bestehen, können **Listen** und **Data Frames** aus heterogenen Elementen zusammengesetzt werden.

Data Frames werden in R mit der Funktion `data.frame()` definiert, oft wird aber statdessen die `tidyverse`-Funktion `tibble()` verwendet.

```r
library(tidyverse)
df <- tibble(
    geschlecht = factor(
        c("male", "female","male",
          "male","female")),
        alter = c(22, 45, 33, 27, 30))
df
```

```
## # A tibble: 5 x 2
##   geschlecht alter
##   <fct>      <dbl>
## 1 male          22
## 2 female        45
## 3 male          33
## 4 male          27
## 5 female        30
```

```r
# Spaltennamen (Variablen) auswählen:
    df$geschlecht
    df$alter
    df["geschlecht"]
    df["alter"]
# Spalten nach Position auswählen:
    df[1]
    df[, 1]
    df[2]
    df[, 2]
# Die ersten drei Zeilen, alle Spalten:
    df[1:3, ]
# Der erste Wert der Variablen "alter":
    df$alter[1]
    df[1, 2]
    df[1, "alter"]
```

2. Daten importieren

Es gibt in RStudio zwei Möglichkeiten, Datensätze zu importieren:

1) Mit Funktionsaufrufen: `read_csv()`, `read_delim()`, `read_sav()`, `read_excel()`.
2) Mit dem GUI: Das Menu kann entweder via 'File > Import Dataset' oder im Environment-Bereich aufgerufen werden.

```r
library(readr)
zufrieden <- read_csv("data/zufrieden.csv")
library(haven)
zufrieden <- read_sav("data/zufrieden.sav")
```

Nach dem Import sollte man die Gruppenvariablen im Datensatz zu Faktoren konvertieren:

```r
# Bei Import aus einer .csv-Datei:
zufrieden$zeitpunkt <-
    factor(zufrieden$zeitpunkt)
# Bei Import aus einer SPSS-Datei:
zufrieden$zeitpunkt <-
    as_factor(zufrieden$zeitpunkt,
              levels = "default")
```

Speichern eines Datensatzes als .csv-Datei:

```
write_csv(x = zufrieden,
          file = "data/zufrieden.csv")
```

3. Daten transformieren

```
library(tidyverse)
# Fehlende Werte löschen mit drop_na()
Therapy_wide |>
    drop_na()
# Wide-to-long Transformation: pivot_longer()
Therapy_long <- Therapy_wide |>
    pivot_longer(cols = c(Pretest, Posttest),
           names_to = "Zeit",
           values_to = "Punkte")
# Long-to-wide Transformation: pivot_wider()
Therapy_wide <- Therapy_long |>
    pivot_wider(names_from = Zeit,
                values_from = Punkte)

# Variablen umbenennen mit rename()
# (hier: "Vpnr" umbenennen in "ID")
Therapy_wide |>
    rename(ID = Vpnr)

# Variablen auswählen mit select()
Therapy_wide |>
    select(ID, Pretest, !Gruppe)
# Helper functions für select (Beispiele)
Therapy_wide |>
    select(starts_with("Gr"),
           ends_with("test"),
           !contains("u"),
           num_range("swk", 1:36))
# Nach Variablentyp auswählen mit where()
Therapy_wide |>
    select(where(is.numeric),
           where(is.factor))

# Variabeln umsortieren mit relocate()
Therapy_wide |>
  relocate(Gruppe, .before = Posttest)

# Beobachtungen auswählen mit filter()
Therapy_long |>
    filter(Zeitpunkt == "Pretest",
           Punkte >= mean(Punkte))

# Beobachtungen sortieren mit arrange()
# aufsteigend
Therapy_wide |>
    arrange(ID)
# absteigend
Therapy_wide |>
    arrange(desc(ID))

# Neue Variable berechnen/Variable
# verändern mit mutate():
 # Neue Variable mit Punkte in %
Therapy_long |>
    mutate(Punkte_percent = Punkte / 7 * 100)
 # Mittelwert pro Person über mehrere
 # Variablen bilden mit rowwise()
Therapy_wide |>
    rowwise() |>
    mutate(mean_PrePost =
               mean(c(Pretest, Posttest),
                    na.rm = TRUE))
 # Variablen rekodieren mit case_when()
 # z.B. Punkte grösser/kleiner als Mittelw.
Therapy_long |>
    mutate(Punkte = case_when(
          Punkte >= mean(Punkte) ~ "hoch",
          Punkte < mean(Punkte) ~ "niedrig"))
 # Faktorstufen rekodieren mit fct_recode()
Therapy_long |>
    mutate(Gruppe = fct_recode(Gruppe,
              control = "Kontrollgruppe",
              treatment = "Therapiegruppe"))

# Mehrere Variablen berechnen/verändern
# mit mutate() und across()
 # Gruppe und Zeit in Faktoren konvertieren
 # (Funktion hat keine weiteren Argumente)
Therapy_long |>
    mutate(across(.cols = c(Gruppe, Zeit),
                  .fns = as.factor))
 # Alle numerischen Variablen z-standardi-
 # sieren (Argumentnamen weggelassen)
Therapy_wide |>
    mutate(across(where(is.numeric), scale))
 # Pretest und Posttest in % umrechnen und
 # mit Suffix _percent abspeichern
 # (anonymous function wird benötigt)
Therapy_long |>
  mutate(across(.cols = c(Pretest, Posttest),
                .fns = \(x) x / 7 * 100),
                .names = "{.col}_percent")

# Daten gruppieren mit group_by() und dann
# gruppiert zusammenfassen mit summarize()
# (hier: Gruppenmittelwerte pro Zeitpunkt)
Therapy_long |>
    group_by(zeitpunkt) |>
    summarize(mean_rating = mean(rating))
```

4. Grafiken

Schritt 1: Plot-Objekt mit der Funktion `ggplot()` erstellen.

Schritt 2: "aesthetic mappings", Variablen X- und Y-Achse zuweisen, Gruppierungsargumente mit `aes()`.

Schritt 3: "Layers" hinzufügen mit `geom_` + weitere Argumente

```
p <- stress |>
  ggplot(mapping =
           aes(x = geschlecht,
               y = stress_psychisch,
               color = geschlecht,
               fill = geschlecht))
p + "verschiedene geoms""
```

Mögliche geoms z.B:

```
geom_point(size = 3)
geom_jitter(width = 0.2, alpha = 0.6)
  # Mappings können auch in den Layers
  # definiert werden (hier z.B. ohne
  # fill wegen Boxplot)
geom_boxplot(aes(color = geschlecht))
geom_violin()
geom_bar(fill = "lightblue",
         color = "black")
geom_histogram(position = "dodge",
               binwidth = 0.5)
    # anstelle von dodge (nebeneinander)
    # identity (überlappend -> mit alpha
    #           Transparenz bestimmen)
    # default ist aufeinander (stacked)
geom_line(linetype = "dashed")
    # default ist solid (durchgezogen)
```

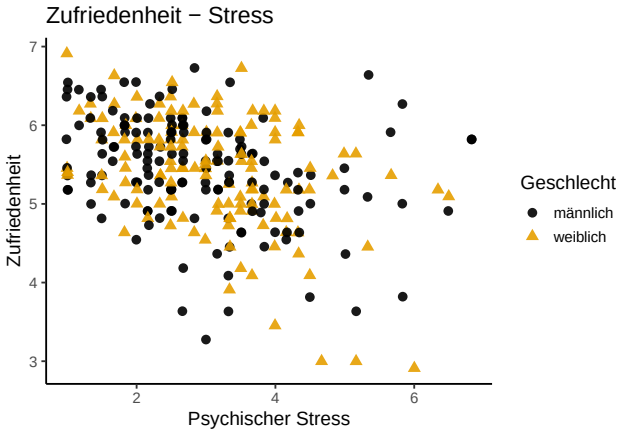
Mit `facet_wrap(~ factor)` erstellen wir eine Grafik für jede Kategorie der Gruppierungsvariable (Faktor).

Mit `facet_grid(factor1 ~ factor2)` werden die Stufen von `factor1` in den Zeilen dargestellt, die Stufen von `factor2` in den Spalten.

Komplexes Plot-Beispiel:

```
library(ggplot2)
# Farbpalette:
palette <- c("#000000", "#E69F00")
# Erstellen des Plot-Objekts
p <- stress |>
  ggplot(mapping =
           aes(x = stress_psychisch,
               y = leben_gesamt,
               color = geschlecht,
               shape = geschlecht))
# Hinzufügen von geoms + Beschriftungen und
# speichern als my_plot für ggsave() (s.u.)
my_plot <- p +
  geom_jitter(size = 3, alpha = 0.9) +
  scale_colour_manual(values = palette) +
  theme_classic(base_size = 14) +
  ggtitle("Zufriedenheit - Stress") +
  xlab("Psychischer Stress")+
  ylab("Zufriedenheit") +
# Titel für beide Legenden ist "Geschlecht"
  labs(color = "Geschlecht",
       shape = "Geschlecht")

my_plot
```




```
# Plot speichern:
ggsave(filename = "my_plot.png",
       plot = my_plot)
```

5. ANOVA-Modelle mit `afex`

Für alle ANOVA-Modelle verwenden wir `aov_4()` aus `afex` zusammen mit `emmeans()`, `pairs()`, `contrast()` und `joint_test()` aus `emmeans` (für Einzelvergleiche/Kontraste/bedingte Haupteffekte).

`aov_4()` benötigt eine `ID`-Variable im Datensatz. `ID` muss bei Modellen mit MW eine Variable sein, die die Messwiederholung berücksichtigt (wo also jeder Wert von `ID` entsprechend der Anzahl messwiederholter Faktorstufen wiederholt vorkommt, vgl. Datenbeispiele in den Übungen).

```
library(afex)
library(emmeans)
```

Einfaktorielle ANOVA ohne MW

```
# ANOVA berechnen und Ergebnis zuweisen
anova1 <- aov_4(av ~ faktor1 + (1 | ID),
                data = df)
summary(anova1) # Zusammenfassung F-Test
anova1$Anova    # ANOVA-Table

# Plot der Mittelwerte (inkl. Fehlerbalken
# mit Konfidenzintervallen)
afex_plot(object = anova1, x = "faktor1")
```

Post-hoc Einzelvergleiche & Kontraste

```
# Erstellung eines emmeans-Objekts auf Basis
# des Ergebnisobjekts der einfakt. ANOVA
result <- emmeans(object = anova1,
                  specs = ~ faktor1)

# ALLE MÖGLICHEN Einzelvergleiche
 # Multiple t-Tests ohne Adjustierung
pairs(x = result, adjust = "none")
 # mit Tukey-Adjustierung (default)
pairs(x = result, adjust = "tukey")

# AUSGEWÄHLTE Einzelvergleiche
 # Vergleiche müssen als Kontraste definiert
 # und als Listenobjekt abgespeichert werden
vgl <- list("gr1 - gr2" = c(1, -1, 0),
            "gr2 - gr3" = c(0, 1, -1))
 # Listenobjekt als method-Argument in:
contrast(object = result,
         method = vgl,
         adjust = "sidak")
 # oder mit adjust = c("bonferroni", "holm")

# Kontrastanalyse
 # Vergleich gr2 mit Durchschnitt gr1 & gr3
ktr <- list("gr2 - rest" = c(-0.5, 1, -0.5))
 # Listenobjekt als method-Argument in:
contrast(object = result,
         method = ktr)
```

Zweifaktorielle ANOVA ohne MW

```
anova2 <- aov_4(av ~ faktor1 * faktor2 +
                  (1 | ID), data = df)
summary(anova2) # Zusammenfassung F-Tests
anova2$Anova    # ANOVA-Table

# Plot der Mittelwerte
afex_plot(object = anova2,
          x = "faktor1",     # auf X-Achse
          trace = "faktor2") # sep. Linien

# Bedingte Haupteffekte
 # für faktor1 (getrennt für Stufen faktor2)
joint_tests(object = anova2, by = "faktor2")
 # für faktor2 (getrennt für Stufen faktor1)
joint_tests(object = anova2, by = "faktor1")
```

Einfaktorielle ANOVA mit MW

```
anova1MW <- aov_4(av ~ 1 + (mw_faktor1 | ID),
                  data = df)
summary(anova1MW) # Zusammenfassung F-Test,
# Mauchly-Test und GG-/HF-korrigierte Tests

# Plot der Mittelwerte (mit CIs basierend
# auf SEs für messwiederholte Bedingungen)
afex_plot(object = anova1MW,
          x = "mw_faktor1",
          error = "within")

# Post-hoc-Einzelvergleiche und Kontraste
 # Methoden für ALLE und AUSGEWÄHLTE Einzel-
 # vergleiche s.o. Post-hoc ANOVA ohne MW

 # Polynomiale Trendkontraste
resultMW <- emmeans(object = anova1MW,
                    specs = ~ mw_faktor1)
contrast(object = resultMW,
         method = "poly",
         adjust = "sidak")
 # oder mit adjust = c("bonferroni", "holm")
```

Zweifaktorielle ANOVA mit MW

```
# Zweifaktorielle ANOVA mit vollständiger MW
anova2MW <- aov_4(av ~ 1 + (mw_faktor1 *
                            mw_faktor2 | ID),
                  data = df)
summary(anova2MW) # Zusammenfassung F-Tests

# Zweifaktorielle ANOVA mit MW auf Faktor 2
anova2MIXED <- aov_4(av ~ faktor1 +
                       (mw_faktor2 | ID),
                     data = df)
summary(anova2MIXED) # Zusammenfass. F-Tests

# wegen 2 x 2-Design Ergebnisse hier ohne
# Sphärizitätskorrektur (beide Varianten)
```