# R Studio vs. SPSS :: **Datenauswertung**

## Nützliche Funktionen in R:

1. Zu Beginn des Codes können Sie die Funktion `rm(list = ls())` ausführen. Dieser Befehl löscht alles, was aktuell in der Environment ist. Sie können mithilfe von `rm(data)` auch einen ganzen Datensatz (oder mehrere) entfernen.
2. Folgende Funktion zeigt Ihnen alle Objekte (Datensätze, Variablen), die Sie aktuell in der Environment haben, an: `ls()`
3. Mit folgenden Funktionen können Sie Hilfe zu bestimmten Funktionen und Outputs bekommen. Es öffnet sich ein neues Ausgabefenster mit einer Beschreibung: `help(Funktion)` **oder:** `?Funktion`
4. Mit `example(Funktion)` können Sie sich einen Beispiel-Code für eine bestimmte Funktion ausgeben lassen, wodurch Sie diese besser verstehen können.
5. Wenn der Output zu groß wird, können Sie auch `View()` nutzen. Schreiben Sie dazu einfach `View(data %>% Befehl)`. Dann öffnet sich ein Fenster, in dem Ihnen der Output schöner und übersichtlicher dargestellt wird.
6. Ergebnisse in einem neuen R-Objekt abspeichern mit `ergebnisse <- data %>%...` und dann bspw. mit `View(ergebnisse)` erneut aufrufen oder für weitere Berechnungen nutzen.

## NAs / Fehlende Werte / Missing Values bei der Berechnung ausschließen:

**SPSS:** Werte filtern: Daten → Fälle auswählen → Falls Bedingung zutrifft → ~ MISSING() → Nicht ausgewählte Fälle filtern
**R:** Möchten Sie bei einer Berechnung keine NAs berücksichtigen, dann filtern Sie zuvor die NAs raus:
`data %>% filter(!is.na(variable)) %>% …`

## Streuungs- und Lagemaße:

1. **Für metrische Variablen:** (Package: tidycomm) `data %>% describe(variable)`
→ **Wichtig:** Den Interquartilsabstand berechnet R hier nicht. Durch den folgenden Befehl können Sie diesen aber mitberechnen lassen: (zusätzlich: Package: dplyr)
`data %>% describe(variable) %>% mutate(IQR = Q75 – Q25)`
Dasselbe mit der Varianz: Fügen Sie dafür einfach folgenden Befehl hinzu: `%>% mutate(Variance = SD^2)`
→ **Tipp:** (Package: tidycomm) Mit `?describe()` wird Ihnen die Ausgabe genauer erklärt.

2. **Für nominal– und ordinalskalierte Variablen:** (Package: tidycomm) `data %>% describe_cat(variable)`

## Gruppen erstellen & Variablen umkodieren:

**SPSS**: Transformieren → Umkodieren in andere Variable → Alte und neue Werte
**R:**
1. **Nominal- oder ordinalskalierte Variablen umkodieren:** (Package: dplyr, tidycomm)
   a. String-Werte: `data <- data %>% recode_cat_scale(variable, assign = c("alter Wert1" = "neuer Wert1", "alter Wert2" = "neuer Wert2"), overwrite = TRUE)`
   b. Integers (Zahlenwerte): `data <- data %>% recode_cat_scale(variable, assign = c('Zahl1'= "Bezeichnung1", 'Zahl2' = "Bezeichnung2"), overwrite = TRUE)`

→ Wenn "alle anderen Werte" als "other" (in SPSS: ELSE) betitelt werden sollen: `data <- data %>% recode_cat_scale(variable, assign = c("alter Wert1" = "neuer Wert1", "alter Wert2" = "neuer Wert2"), other = "other")`

2. **Intervall- bzw. verhältnisskalierte Skala in eine nominal- oder ordinalskalierte Skala umkodieren und Gruppen erstellen:** (Package: tidycomm) `data <- data %>% categorize_scale(variable, lower_end = Minimum der alten Skala, upper_end = Maximum der alten Skala, breaks = c(Werte, nach denen eine neue Gruppe beginnt), labels = c("Name Gruppe1", "Name Gruppe2"))`

## Datensatz nach Variable(n) gruppieren/ gruppenweise auswerten:

**SPSS**: Daten → Datei aufteilen → Ausgabe nach Gruppen aufteilen → Dann: gewünschten Befehl ausführen
**R:** (Package: dplyr) `data %>% group_by(variable) %>% ...`
→ Diese Funktion ist für weitere Berechnungen/ Funktionen wie `tab_frequencies()` nützlich.

## Häufigkeiten:

**SPSS**: Analysieren → Deskriptive Statistiken → Häufigkeiten → Statistiken
**R:** (Package: tidycomm) `data %>% tab_frequencies(variable)`

## Fälle und Variablen auswählen:

**SPSS**: Daten → Fälle auswählen → Falls Bedingung zutrifft (Ausgabe: Nicht ausgewählte Fälle filtern)
**R:** (Package: dplyr)
1. **Ganze Variablen auswählen:** `data %>% select(variable)`
Nützliche Funktionen, die Sie in `select()` einbetten können:
```
select(starts_with("String"))
       ends_with("String")
       contains("String")
       last_col()
       where(is.character)
             is.numeric
             is.integer
             is.double
             is.logical
             is.factor
```

2. **Bestimmte Fälle/Werte einer Variable auswählen:** `data %>% filter(variable1 == Wert & variable2 == "String")`
**Tipp:** (Package: stringr) Wenn Sie alle Fälle mit einem bestimmten String ausgegeben haben möchten, können Sie auch folgende Funktion verwenden: `str_detect(variable, "String, der enthalten ist")`
Da `str_detect()` sensibel bzgl. Groß-/ Kleinschreibung ist, kann man bei Zweifeln, ob der String groß/kleingeschrieben wird, auch `[]` nutzen. Bspw. `"[sS]tring"`

**Tipp:** Statt bei vielen Bedingungen wiederholt „ | " zu nutzen, können Sie auch `c(…, …, …)` nutzen (bspw. `select(c("variable1", "variable2", "variable3"))`)
**Wichtig:** Während in SPSS die Fälle ausgewählt werden, die man ausschließen möchte, werden in R mit `filter()` und `select()` die Fälle ausgewählt, die man behalten möchte!

## Fälle sortieren:

**SPSS**: Daten → Fälle sortieren → (Sortierreihenfolge) Auf/Absteigend
**R:** (Package: dplyr)
1. **Aufsteigend:** `data %>% arrange(variable)`
2. **Absteigend:** `data %>% arrange(-variable)` **oder** `data %>% desc(variable)`

## Reliabilitätsanalyse:

### 1. Reliabilitätsanalyse bei Items (Cronbachs Alpha):

**SPSS:** Analysieren → Metrisch → Reliabilitätsanalyse (Items auswählen) → Statistiken… (gewünschte Optionen wie „Deskriptive Statistiken für… Skala, wenn Items gelöscht" auswählen) → Weiter → OK

**R:** (Package: tidycomm)

1.1 Index erstellen und abspeichern:

a. Mittelwertindex: `data <- data %>% add_index(index_einstellung, variable1, variable2, variable3)`
→ Der Index wird hier „index_einstellung" genannt und wurde aus Variable1, Variable2 und Variable3 erstellt.

b. Summenindex: `data <- data %>% add_index(index_einstellung, variable1, variable2, variable3, type = "sum")`

1.2 Interne Konsistenz/Reliabilität berechnen (Output: Cronbachs Alpha, Index-Beschreibung, deskriptive Statistiken über den Index): `data %>% get_reliability(index_einstellung)`
→ Wenn Sie nur `data %>% get_reliability()` schreiben, dann wird Ihnen die Reliabilität für alle Indizes aus Ihrem Datensatz berechnet, die Sie mit `add_index()` berechnet haben.

1.3 Wenn Item weggelassen… ("Reliability if an item is dropped"): (Package: dplyr, psych) `data %>% select(c(variablen, für, die, Sie, Cronbachs, Alpha, berechnen, möchten)) %>% alpha(., check.keys = TRUE)` Das Argument `check.keys = TRUE` gibt Ihnen weiterhin unter "Item statistics" an, ob das Item invers kodiert ist. Zu sehen ist das im Output, wenn hinter der Variable ein Minus steht:

```
Item statistics
             n raw.r std.r r.cor r.drop mean  sd
EK01_01- 878  0.88 -0.82 -0.87   0.85  4.1 1.4
EK01_02  878  0.90  0.90  0.89   0.88  4.2 1.3
EK01_03  878  0.92  0.91  0.91   0.90  4.4 1.3
EK01_04  878  0.88  0.87  0.86   0.85  4.0 1.3
EK01_05  878  0.89  0.88  0.87   0.86  3.8 1.5
EK01_06  878  0.91  0.90  0.90   0.89  4.2 1.3
EK01_07  878  0.76  0.77  0.73   0.72  3.4 1.2
EK01_08  878  0.95  0.94  0.94   0.93  4.1 1.5
EK01_09  878  0.92  0.92  0.91   0.90  4.3 1.3
EK01_10  878  0.90  0.91  0.90   0.89  4.4 1.2
EK01_11  878  0.89  0.90  0.89   0.87  4.4 1.3
EK01_12  878  0.83  0.84  0.81   0.80  4.0 1.3
```

→ Mit dem `reverse_scale()` Befehl aus dem tidycomm-Package können Sie die Skala invertieren (sh. Cheat Sheet *R Studio vs. SPSS :: Datenaufbereitung*)

### 2. Interrater-Reliabilität/Intercoderreliabilität:

**SPSS:** Cohens Kappa: Analysieren → Deskriptive Statistiken → Kreuztabellen… → Variablen einfügen → Statistiken… („Kappa" ankreuzen) → Weiter → Zellen… (Prozentwerte: „Zeilenweise" und „Spaltenweise" ankreuzen) → Weiter → OK

Dann Fleiss' Kappa: Analysieren → Metrisch → Reliabilitätsanalyse (Variablen in Bewertungen einfügen) → Statistiken… (Unter „Bewerterübergreifende Übereinstimmung: Fleiss' Kappa" „Übereinstimmung bei einzelnen Kategorien anzeigen" anklicken) → Weiter → OK

**R:** (Package: tidycomm) `data %>% test_icr(variable_mit_einheitenbezeichnung, variable_mit_codierendenkennung)`

Möchten Sie für alle Variablen die Intercoderreliabilität berechnen, schreiben Sie einfach `data %>% test_icr()`

→ Der Output gibt die Übereinstimmung in Prozent, den Reliabilitätskoeffizienten nach Holsti-Schätzung (paarweise Mittelwertsübereinstimmung) und Krippendorffs Alpha an. Für zusätzliche Berechnung(en):

```
data %>% test_icr(variable_mit_einheitenbezeichnung, variable_mit_codierendenkennung,
  fleiss_kappa = TRUE      ) # Fleiss Kappa
  lotus = TRUE               # Lotus
  cohens_kappa = TRUE        # Cohen's Kappa
  brennan_prediger = TRUE    # Brennen & Prediger's Kappa
  s_lotus = TRUE             # S-Lotus
```

## Chi-Quadrat-Test: χ 2

**SPSS:** Analysieren → Deskriptive Statistiken → Kreuztabellen → Zeile(n) und Spalten definieren → Statistiken…: Haken bei „Chi-Quadrat" → Weiter → Zellen…: Unter „Häufigkeiten" Haken bei „Beobachtet" und „Erwartet" und unter „Prozentwerte" Haken bei „Zeilenweise", „Spaltenweise" oder „Gesamtsumme".

**R:** (Package: tidycomm)

a. Voraussetzungen prüfen: `data %>% tab_frequencies(variable1, variable2)`

b. Ausführung: `data %>% crosstab(unabhängige_variable, abhängige_variable(n), chi_square = TRUE)`

→ Berechnet wird der Chi-Quadrat-Test und Cramer's V

→ **Tipp:** Wenn Sie auch die Prozentzahlen angezeigt bekommen möchten, hängen Sie am Ende innerhalb der Klammer noch `percentages = TRUE` an.

→ **Tipp:** Wenn Sie auch die Zeilensumme ausgegeben haben möchten, hängen Sie am Ende innerhalb der Klammer noch `add_total = TRUE` an.

## Regression:

### 1. Einfache lineare Regression:

**SPSS:** Analysieren → Regression → Linear… → Abhängige und unabhängige Variable definieren

Dann: Visualisieren: Grafik → Diagrammerstellung… → Galerie: Auswählen aus: Streu-/Punktdiagramm → Streudiagramm auswählen → y: die zu erklärende Variable; x: die erklärende Variable → OK → Doppelklick auf das Diagramm → Anpassungslinie bei Gesamtsumme hinzufügen → unter Eigenschaften: Anpassungsmethode „Linear" anklicken

**R:** (Package: tidycomm)

Voraussetzungen prüfen und ausführen: `data %>% regress(abhängige_variable, unabhängige_variable, check_independenterrors = TRUE, check_multicollinearity = TRUE, check_homoscedasticity = TRUE)`

### 2. Multiple lineare Regression:

**SPSS:** Analysieren → Regression → Linear… → Abhängige und unabhängige Variablen definieren

Dann: Visualisieren: Grafik → Diagrammerstellung… → Galerie: Auswählen aus: Streu-/Punktdiagramm → Streudiagramm-Matrix auswählen

**R:** (Package: tidycomm)

Voraussetzungen prüfen und ausführen: `data %>% regress(abhängige_variable, unabhängige_variable1, unabhängige_variable2, check_independenterrors = TRUE, check_multicollinearity = TRUE, check_homoscedasticity = TRUE)`

### 3. Visualisierung der Regressionen:

**R:** (Package: tidycomm) `data %>% regress(abhängige_variable, unabhängige_variable(n)) %>% visualize()` → Es wird die automatische Grafik von Tidycomm erstellt.

**oder:**

```
data %>% regress(abhängige_variable, unabhängige_variable(n)) %>%
visualize(which = "correlogram") # Korrelogramm
          which = "resfit"       # Residuals vs Leverage Diagramm
          which = "pp"           # Normal probability-probability plot
          which = "scaloc"       # skaliertes Standortdiagramm
          which = "reslev"       # Residuals vs Leverage Diagramm
```

## Korrelation:

### 1. Pearson-Korrelation:

**SPSS:** Analysieren → Korrelation → Bivariat… → metrische Variablen definieren → Haken unter Korrelationskoeffizient: „Pearson"

**R:** (Package: tidycomm) `data %>% correlate(variable1, variable2, method = "pearson")`

**Hinweis:** Sie können auch mehr als zwei Variablen miteinander korrelieren lassen, wobei alle in zweier-Paaren korreliert werden. Fügen Sie dafür einfach noch weitere Variablen in der Klammer hinzu. Möchten Sie diese Variablen mit einer speziellen Variable korrelieren, dann nutzen Sie folgenden Befehl: `data %>% correlate(variable1, variable2, with = variable_mit_der_alle_korreliert_werden_sollen)`

**Hinweis:** Mit `data %>% correlate()` werden alle Variablen aus dem Datensatz miteinander korreliert, die geeignet sind.

**Hinweis:** Auch in R können Sie eine Korrelationsmatrix erstellen: `data %>% correlate(…) %>% to_correlation_matrix()`

### 2. Kendall:

**SPSS:** Analysieren → Korrelation → Bivariat… → metrische Variablen definieren → Haken unter „Korrelationskoeffizient: Kendall-Tau-b"

**R:** (Package: tidycomm) `data %>% correlate(variable1, variable2, method = "kendall")`

### 3. Spearman Rho:

**SPSS:** Analysieren → Korrelation → Bivariat… → metrische Variablen definieren → Haken unter „Korrelationskoeffizient: Spearman"

**R:** (Package: tidycomm) `data %>% correlate(variable1, variable2, method = "spearman")`

### 4. Partielle Korrelation:

**SPSS:** Analysieren → Korrelation → Partiell… → metrische Variablen definieren und die Kontrollvariable definieren → Optionen…: Haken unter Statistiken „Korrelationen nullter Ordnung"

**R:** (Package: tidycomm) `data %>% correlate(variable1, variable2, partial = drittvariable)`

### 5. Visualisieren:

**SPSS:** Grafik → Diagrammerstellung… → Galerie: Auswählen aus: Streu-/Punktdiagramm → Streudiagramm auswählen → Variablen definieren

**R:** (Package: tidycomm) Hängen Sie hinter jeden der vorherigen Korrelationsbefehle `%>% visualize()` an. Es wird die automatische Grafik von Tidycomm erstellt.

## T-Test:

### 1. Einfacher T-Test:

**SPSS:** Analysieren → Mittelwerte und Proportionen vergleichen → t-Test bei einer Stichprobe… → Testvariable definieren → Testwert eingeben → Haken unter „Effektgrößen schätzen"

**R:** (Package: tidycomm)

a. Voraussetzungen prüfen: `data %>% describe(abhängige_variable)`
b. Ausführung: `data %>% t_test(variable, mu = Wert)` Der Wert aus der Grundgesamtheit ist **mu**.
c. Ausgabe: **Wichtig:** Tidycomm führt immer eine zweiseitige Testung durch! Bei einseitiger Testung müssen Sie p manuell halbieren **oder** Sie speichern den p-Wert in einem R-Objekt (p) und nutzen den Befehl: `p / 2`

### 2. Unabhängiger T-Test:

**SPSS:** Analysieren → Mittelwerte und Proportionen vergleichen → t-Test bei unabhängigen Stichproben… → Test- und Gruppierungsvariable definieren

**R:** (Package: dplyr, tidycomm)

a. Voraussetzungen prüfen: `data %>% dplyr::group_by(unabhängige_variable) %>% describe(abhängige_variable)` (**Hinweis:** Der Levene-Test wird erst mit dem T-Test durchgeführt)

→ Der Output ist zu groß für die Konsole:

```
# A tibble: 3 x 16
# Groups:   klima_groups [3]
  klima_groups Variable     N Missing     M    SD   Min   Q25   Mdn   Q75   Max Range CI_95_LL
  <fct>        <chr>    <int>   <int> <db> <db> <db> <db> <db> <db> <db> <db>    <db>
1 Klimaaktivi… tpe_sch…   430       0  3.78  2.62 -4.5   2     3.75  5.75 10    14.5     3.54
2 Klimaskepti… tpe_sch…   158       0 -0.045 1.86 -5    -1    0     0.5   6.75 11.8    -0.338
3 Mitte        tpe_sch…   290       0  2.50  2.57 -3.25  0.75  2.25  4.25 10    13.2     2.20
# 3 more variables: CI_95_UL <db>, Skewness <db>, Kurtosis <db>
```

**View(data %>% dplyr::group_by(unabhängige_variable) %>% describe(abhängige_variable))**

| | klima_groups | Variable | N | Missing | M | SD | Min | Q25 | Mdn | Q75 | Max | Range | CI_95_LL | CI_95_UL | Skewness | Kurtosis |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Klimaaktivist*innen | tpe_schaden | 430 | 0 | 3.78488372 | 2,616283 | -4.50 | 2.00 | 3.75 | 5.75 | 10.00 | 14.50 | 3.5368987 | 4.0328688 | -0.2709141 | 2.864087 |
| 2 | Klimaskeptiker*innen | tpe_schaden | 158 | 0 | -0.04588608 | 1.858626 | -5.00 | -1.00 | 0.00 | 0.50 | 6.75 | 11.75 | -0.3379461 | 0.2461739 | 0.8335938 | 4.987028 |
| 3 | Mitte | tpe_schaden | 290 | 0 | 2.49827586 | 2.571264 | -3.25 | 0.75 | 2.25 | 4.25 | 10.00 | 13.25 | 2.2010968 | 2.7954550 | 0.3278140 | 2.729924 |

b. Ausführung: `data %>% t_test(unabhängige_variable, abhängige_variable)` (**Tipp:** Wenn Ihre UV mehr als 2-stufig ist (bspw. UV = Gruppen der Klimaskeptiker*innen, Mitte, Klimaktivist*innen), dann können Sie auch speziell mit dem Argument **levels =** spezielle Ausprägungen definieren, die Sie testen möchten: `data %>% t_test(unabhängige_variable, abhängige_variable, levels = c("Ausprägung1 der UV", "Ausprägung2 der UV"))`

→ Output (mit **View()** um den Befehl):

| | Variable | M_Klimaaktiv | SD_Klimaaktiv | M_Klimaskept | SD_Klimaskept | Delta_M | t | df | p | d | Levene_p | var_equal |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | tpe_schaden | 3.785 | 2.616 | -[31m0-[39m+[31m.+[39m+[31m0+[39m+[31m46-[39m | 1.859 | 3.831 | 19.708 | 393 | 0.000 | 1.572 | 0 | FALSE |

Von links nach rechts: Mittelwert und Standardabweichung für beide UVs, Mittelwertsdifferenz, t-Wert, Freiheitsgrade, p-Wert (**Wichtig:** immer zweiseitig! Bei einseitiger Testung müssen Sie p halbieren), Cohens d, p-Wert des Levene-Tests (**Wichtig:** Wenn signifikant, dann wird automatisch eine Korrektur mit dem Welch-Test durchgeführt. In diesem Beispiel sehen wir also die Ergebnisse des Welch-Tests), var_equal = Varianzhomogenität (**TRUE** = Varianzhomogenität, **FALSE** = Varianzheterogenität)

### 3. Abhängiger T-Test:

**SPSS:** Analysieren → Mittelwerte und Proportionen vergleichen → t-Test bei Stichproben mit paarigen Werten… → Paarige Variablen definieren

**R:** (Package: dplyr, tidycomm)

a. Voraussetzungen prüfen: `data %>% dplyr::group_by(unabhängige_variable) %>% describe(abhängige_variable)`
b. Ausführung: `data %>% t_test(unabhängige_variable, abhängige_variable, paired = TRUE, case_var = variable)` Bei **case_var =** geben Sie die Variable an, die R zeigt, dass Person a in Datensatz1 Person b aus Datensatz2 ist. Das kann bspw. eine case_id sein.
c. Ausgabe: sh. T-Test für unabhängigen Stichproben. **Wichtig:** Auch hier ist der p-Wert immer zweiseitig und muss bei einseitiger Testung halbiert werden.

→ **Hinweis:** Wenn das Format des Datensatzes geändert werden muss (Wide-Format in Long-Format): sh. Cheat Sheet *R Studio vs. SPSS :: Datenaufbereitung*

### 4. T-Test Visualisieren:

**SPSS:** Grafik → Diagrammerstellung… → Galerie: Auswählen aus: Balken → Einfache Balken auswählen → Variablen definieren

**R:** (Package: tidycomm) Hängen Sie am Ende des Befehls noch `%>% visualize()` an. Es wird die automatische Grafik von Tidycomm erstellt.

## Varianzanalyse (ANOVA):

### 1. Einfaktorielle Varianzanalyse (ANOVA):

**SPSS:** Analysieren → Allgemeines lineares Modell → Univariat → Abhängige Variable und festen Faktor definieren → Optionen…: Haken unter „Deskriptive Statistiken, Homogenitätstests und Schätzungen der Effektgröße“ → Weiter → Diagramme… → Faktor als Horizontale Achse definieren → Hinzufügen → Weiter → Post hoc… → Faktor unter „Post-hoc Tests für:“ definieren → Haken unter „Varianzgleichheit angenommen…“ bei „Bonferroni“ setzen

**R:** (Package: dplyr, tidycomm)

a. Voraussetzungen prüfen: `data %>% dplyr::group_by(unabhängige_variable) %>% describe(abhängige_variable)`
b. Ausführung: `data %>% unianova(unabhängige_variable, abhängige_variable, post_hoc = TRUE)`

→ **Tipp:** Wenn Sie die deskriptive Statistik erneut ausgegeben bekommen möchten, dann nutzen Sie am Ende in der Klammer noch `descriptives = TRUE`.

→ **Wichtig:** Auch hier ist der p-Wert immer zweiseitig und muss bei einseitiger Messung halbiert werden.

c. Visualisierung: Hängen Sie hinter der Klammer noch den Befehl `%>% visualize()` an. Es wird die automatische Grafik von Tidycomm erstellt.

### 2. Zweifaktorielle Varianzanalyse (ANOVA):

**SPSS:** Analysieren → Allgemeines lineares Modell → Univariat → Abhängige Variable und feste Faktoren definieren → Optionen…: Haken unter „Deskriptive Statistiken, Homogenitätstests und Schätzungen der Effektgröße“ → Weiter → Diagramme… → Faktoren als Horizontale Achse und Separate Linien definieren → Hinzufügen → Weiter → Post hoc… → Faktoren unter „Post-hoc Tests für:“ definieren → Haken unter „Varianzgleichheit angenommen…“ bei „Bonferroni“ setzen

**R:** (Package: tidyverse, psych, car)

a. Vorbereitung - Faktorisierung der Faktor-Variablen: `data <- data %>% mutate(variable = as.factor(variable))`
b. Voraussetzungen prüfen: `describeBy(abhängige_variable ~ unabhängige_variable1 + unabhängige_variable2, data = data)` (**Wichtig:** Sie schreiben kein `data %>%` vor den Befehl!)
   Sie können auch einen Boxplot erstellen: `boxplot(abhängige_variable ~ unabhängige_variable1 + unabhängige_variable2, data = data)` (**Wichtig:** Sie schreiben kein `data %>%` vor den Befehl!)
   Homogenität der Varianzen: `leveneTest(abhängige_variable ~ unabhängige_variable1 * unabhängige_variable2, data = data)` (**Wichtig:** Sie schreiben kein `data %>%` vor den Befehl!)
   → Ausgegeben wird Ihnen der p-Wert unter „Pr(>F)“. Wenn nicht signifikant:
   Normalverteilung der Residuen: `anova <- aov(abhängige_variable ~ unabhängige_variable1 + unabhängige_variable2, data = data)`
   Wenn man unterstellt, dass die UVs nicht unabhängig voneinander sind: `anova <- aov(abhängige_variable ~ unabhängige_variable1 + unabhängige_variable2 + unabhängige_variable1 * unabhängige_variable2, data = data)` (**Wichtig:** Sie schreiben bei beiden Varianten kein `data %>%` vor den Befehl!)
   Q-Q-Diagramm: `plot(anova, 2)` (2 bedeutet hier: es wird ein Q-Q-Plot für anova ausgegeben) (**Wichtig:** Sie schreiben kein `data %>%` vor den Befehl!)
c. Ausführung: `summary(anova)` (**Wichtig:** Sie schreiben kein `data %>%` vor den Befehl!)
d. Post Hoc-Test: `TukeyHSD(anova)` Für die Hauptfaktoren wird jeweils ein Post-Hoc Test gerechnet. Und ggf. der Interaktionseffekt (Output: Differenz; Konfidenzintervall; p-Wert) (**Wichtig:** Sie schreiben kein `data %>%` vor den Befehl!).