Tagebuchstudien und Mehrebenenmodelle

Dr. Francisco Wilhelm

2025-02-01

Table of contents

1 Kapitel 1: Einleitung

2 Einleitung

Dieses Dokument enthält Anleitungen und Übungen zur Analyse von Daten aus Tagebuchstudien mittels Mehrebenenmodellen. Es ist im Rahmen des Seminars zu Tagebuchstudien in Psychologie an der Uni Bern entstanden.

2.1 Voraussetzungen

Materialien:

- RStudio (erstellt RStudio 2024.12.0, Build 467)
- R (erstellt mit R 4.4.2)
- Installation von Paketen mit dem Code unten.

Kenntnisse:

- Data Wrangling in R, Umgang mit R-Notebooks, s. Anhang: Data Wrangling, sowie https://methodenlehre.github.io/einfuehrung-in-R/
- Statistisches Wissen zu linearen Modellen (Regressionen) und Testkonstruktion (Likert-Skalen, Reliabilitätsanalyse)

2.1.1 Installation von R-Paketen, die im Kurs verwendet werden

Der Code unten installiert, falls noch nicht vorhanden, den "pacman" Paketmanager und danach alle R-Pakete, die für den Kurs benötigt werden.

```
if (!require("pacman")) install.packages("pacman")

pacman::p_load(devtools)
if (!require("multilevelmediation")) devtools::install_github("falkcarl/multilevelmediation")
if (!require("franzpak")) devtools::install_github("franciscowilhelm/franzpak")

pacman::p_load(lmerTest, haven, brms, psych,
               sjmisc, sjlabelled, sjPlot, writexl, broom.mixed, qgraph,
               tidyverse, multilevelTools, parameters, devtools,
               multilevelmediation, reghelper)
```

3 Kapitel 2: Datenaufbereitung

3.1 Pakete installieren und laden

```
if (!require("pacman")) install.packages("pacman")
```

```
Lade nötiges Paket: pacman
```

```
pacman::p_load(haven, psych,
               sjmisc, sjPlot, writexl,
               tidyverse, multilevelTools)
```

3.2 Daten einlesen

Mit `load()` können wir `.RData` Dateien einlesen (für weitergehende Infos s. auch Einführung in R, Kapitel 3.2.4.)

```
load("../data/df_cfa_wide.RData")
```

3.3 Daten ansehen

Der Datensatz hat 131 Spalten (`ncol()`) und 100 Zeilen (`nrow()`) (eine pro Person).

Wie wir mit `names()` sehen, gibt es die Variablen id für die Personidentifikation (jede Person hat ihre eigene Nummer), a1-a5, b1-b5, und c1-c3. Da jeder Tag (1-10 Tage) von jeder Variable seine eigene Spalte bekommt (t1-t10) gibt es viele Spalten und wir nennen das Datenformat daher breites Datenformat (*wide* format).

a,b und c bilden jeweils eine Skala mit 5 bzw. bei c 3 Indikatoren.

Mit `head()` können wir einen Blick in die Daten werfen.

```r
ncol(df_cfa_wide)
```

```
[1] 131
```

```r
nrow(df_cfa_wide)
```

```
[1] 100
```

```r
names(df_cfa_wide)
```

```
  [1] "id"     "a1_t1"  "a1_t2"  "a1_t3"  "a1_t4"  "a1_t5"  "a1_t6"  "a1_t7"
  [9] "a1_t8"  "a1_t9"  "a1_t10" "a2_t1"  "a2_t2"  "a2_t3"  "a2_t4"  "a2_t5"
 [17] "a2_t6"  "a2_t7"  "a2_t8"  "a2_t9"  "a2_t10" "a3_t1"  "a3_t2"  "a3_t3"
 [25] "a3_t4"  "a3_t5"  "a3_t6"  "a3_t7"  "a3_t8"  "a3_t9"  "a3_t10" "a4_t1"
 [33] "a4_t2"  "a4_t3"  "a4_t4"  "a4_t5"  "a4_t6"  "a4_t7"  "a4_t8"  "a4_t9"
 [41] "a4_t10" "a5_t1"  "a5_t2"  "a5_t3"  "a5_t4"  "a5_t5"  "a5_t6"  "a5_t7"
 [49] "a5_t8"  "a5_t9"  "a5_t10" "b1_t1"  "b1_t2"  "b1_t3"  "b1_t4"  "b1_t5"
 [57] "b1_t6"  "b1_t7"  "b1_t8"  "b1_t9"  "b1_t10" "b2_t1"  "b2_t2"  "b2_t3"
 [65] "b2_t4"  "b2_t5"  "b2_t6"  "b2_t7"  "b2_t8"  "b2_t9"  "b2_t10" "b3_t1"
 [73] "b3_t2"  "b3_t3"  "b3_t4"  "b3_t5"  "b3_t6"  "b3_t7"  "b3_t8"  "b3_t9"
 [81] "b3_t10" "b4_t1"  "b4_t2"  "b4_t3"  "b4_t4"  "b4_t5"  "b4_t6"  "b4_t7"
 [89] "b4_t8"  "b4_t9"  "b4_t10" "b5_t1"  "b5_t2"  "b5_t3"  "b5_t4"  "b5_t5"
 [97] "b5_t6"  "b5_t7"  "b5_t8"  "b5_t9"  "b5_t10" "c1_t1"  "c1_t2"  "c1_t3"
[105] "c1_t4"  "c1_t5"  "c1_t6"  "c1_t7"  "c1_t8"  "c1_t9"  "c1_t10" "c2_t1"
[113] "c2_t2"  "c2_t3"  "c2_t4"  "c2_t5"  "c2_t6"  "c2_t7"  "c2_t8"  "c2_t9"
[121] "c2_t10" "c3_t1"  "c3_t2"  "c3_t3"  "c3_t4"  "c3_t5"  "c3_t6"  "c3_t7"
[129] "c3_t8"  "c3_t9"  "c3_t10"
```

```r
head(df_cfa_wide)
```

```
# A tibble: 6 x 131
     id a1_t1  a1_t2 a1_t3 a1_t4 a1_t5 a1_t6 a1_t7 a1_t8  a1_t9 a1_t10 a2_t1
  <int> <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>  <dbl> <dbl>
1     1  3.34  2.15   1.61 3.35   3.32  4.17  3.65  1.93  5.62    4.95  3.38
2     2  3.71  2.38   1.85 4.63   2.48  2.23  4.49  4.01  2.43    2.60  4.00
3     3  5.18 -0.128  2.80 0.303  4.31  2.34  3.58  1.82 -0.306   1.61  2.96
4     4  4.71  3.60   2.95 2.10   2.78  4.42  1.36  3.85  3.71    2.28  4.04
5     5  3.95  3.33   5.41 3.72   5.80  3.09  4.01  5.88  3.18    4.93  3.32
6     6  2.27  2.29   1.38 2.17   2.10  3.31  2.17  2.60  2.29    1.58  2.79
```

```
# i 119 more variables: a2_t2 <dbl>, a2_t3 <dbl>, a2_t4 <dbl>, a2_t5 <dbl>,
#   a2_t6 <dbl>, a2_t7 <dbl>, a2_t8 <dbl>, a2_t9 <dbl>, a2_t10 <dbl>,
#   a3_t1 <dbl>, a3_t2 <dbl>, a3_t3 <dbl>, a3_t4 <dbl>, a3_t5 <dbl>,
#   a3_t6 <dbl>, a3_t7 <dbl>, a3_t8 <dbl>, a3_t9 <dbl>, a3_t10 <dbl>,
#   a4_t1 <dbl>, a4_t2 <dbl>, a4_t3 <dbl>, a4_t4 <dbl>, a4_t5 <dbl>,
#   a4_t6 <dbl>, a4_t7 <dbl>, a4_t8 <dbl>, a4_t9 <dbl>, a4_t10 <dbl>,
#   a5_t1 <dbl>, a5_t2 <dbl>, a5_t3 <dbl>, a5_t4 <dbl>, a5_t5 <dbl>, ...
```

3.4 Daten transformieren: Langformat

Als erstes transformieren wir die Daten vom Breit- ins Langformat, so dass jede Messung (Tag 1-Tag 10) eine eigene Zeile bekommt. Diese Variable nennen wir "time". Im ersten Schritt machen wir mit `pivot_longer()` den Datensatz seeehr lang, es bekommt nämlich jede Messung von jeder Variable ihre eigenen Zeile. Wir machen den Datensatz dann im zweiten Schritt mit `pivot_wider()` wieder etwas breiter mit dem Ziel, eine Zeile pro Person und Tag zu bekommen, und jeweils eine Spalte pro Item.

Die Funktionsweise von pivot_longer() und pivot_wider() ist in der Einführung zu R, Kapitel 4.3 ausführlicher beschrieben.

```
df_cfa_superlong <- df_cfa_wide |>
  pivot_longer(
    cols = -id, # All columns except id
    names_to = c("variable", "time"), # Namensgebende Spalten sollen heissen: variable, time
    names_sep = "_t"
    # Namensgebende Spalten anhand "_t_ separieren (z.B. "a1_t1" --> gespalten in "a1" und "
  )

head(df_cfa_superlong)
```

```
# A tibble: 6 x 4
     id variable time  value
  <int> <chr>    <chr> <dbl>
1     1 a1       1      3.34
2     1 a1       2      2.15
3     1 a1       3      1.61
4     1 a1       4      3.35
5     1 a1       5      3.32
6     1 a1       6      4.17
```

```r
df_cfa_long <- df_cfa_superlong |>
  pivot_wider(names_from = variable, #namen aus variable
              values_from = value) |>  # werte aus "value" (Standardname)
  mutate(time = as.numeric(time)) # time ist eine Zahl von 1-10, wurde aber zuvor als Charact

head(df_cfa_long)
```

```
# A tibble: 6 x 15
     id  time    a1    a2    a3    a4    a5    b1       b2      b3     b4    b5
  <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>    <dbl>   <dbl>  <dbl> <dbl>
1     1     1  3.34  3.38  3.22  4.40  3.54 3.37   1.70    0.584    1.58   2.85
2     1     2  2.15  5.11  5.68  2.77  2.39 2.49   0.952   2.10     0.331  2.55
3     1     3  1.61  3.00  2.50  1.90  2.81 0.992 -0.686   0.00315 -0.518  1.96
4     1     4  3.35  3.85  4.97  2.61  3.23 2.25   0.00412 0.711    2.19   2.03
5     1     5  3.32  3.69  4.70  3.48  4.21 1.95   0.676   0.922    0.285  2.36
6     1     6  4.17  5.66  3.27  4.19  3.57 1.54  -0.373   1.30     1.69   2.54
# i 3 more variables: c1 <dbl>, c2 <dbl>, c3 <dbl>
```

Das Ergebnis:

- eine Zeile pro Person (id) und Messung (time). Die Spalten id und time identifizieren also für jede Zeile, von welcher Person und welchem Tag die Werte in den folgenden Spalten kommen.
- eine Spalte pro Variable (a1-a5, b1-b5, c1-c3)

3.5 Daten transformieren: Skalenscores erstellen

Als nächstes können wir mittels `summarise()` die Skalenscores erstellen. Wir verwenden eine simple Form der Skalenerstellung bei dem der Mittelwert aller vorhandenen Items einer Skala `rowMeans(...)` verwendet wird.

```r
df_cfa_long_scores <- df_cfa_long |> group_by(id, time) |>
  summarise(
    a = rowMeans(across(starts_with("a")), na.rm = TRUE),
    b = rowMeans(across(starts_with("b")), na.rm = TRUE),
    c = rowMeans(across(starts_with("c")), na.rm = TRUE),
    .groups = "drop" # group_by() wieder aufheben für den finalen Datensatz
  )
```

(Best-practice ist es bei fehlenden Daten genau hinzuschauen und nur dann einen Skalenwert zu erstellen, wenn die Personen zu einen gewissen Prozentsatz aller Items eine Antwort geben (etwa 2/3). Eine solche Funktion könnten wir programmieren, lassen es aber für das Beispiel weg.)

3.6 Daten transformieren: Zentrierung

Für die spätere Verwendung zerlegen wir die Rohvariablen mittels person-mean Zentrierung. Wir zentrieren wir die Skalenvariablen, die täglich gemessen werden (aber nicht Baseline-Variablen), mittels `de_mean()`. `de_mean()` nimmt als Argumente (a) mit Komma getrennte Namen der Variablen, die wir zentrieren wollen (mehrere auf einmal ist möglich), (b) mittels `grp =` Argument die identifizierende Variable für die Gruppenzugehörigkeit in Anführungszeichen (`"id"`.

```
df_cfa_long_scores <- df_cfa_long_scores |>
  de_mean(a,b, c, grp = "id")
head(df_cfa_long_scores)
```

```
# A tibble: 6 x 11
     id  time     a     b     c    a_dm   b_dm   c_dm  a_gm  b_gm  c_gm
  <int> <dbl> <dbl> <dbl> <dbl>   <dbl>  <dbl>  <dbl> <dbl> <dbl> <dbl>
1     1     1  3.58 2.02   3.06 -0.257   0.448 -0.527  3.83  1.57  3.59
2     1     2  3.62 1.68   4.23 -0.215   0.117  0.643  3.83  1.57  3.59
3     1     3  2.37 0.350  2.47 -1.47   -1.22  -1.12   3.83  1.57  3.59
4     1     4  3.60 1.44   2.85 -0.229  -0.132 -0.738  3.83  1.57  3.59
5     1     5  3.88 1.24   3.16  0.0481 -0.328 -0.427  3.83  1.57  3.59
6     1     6  4.17 1.34   4.26  0.340  -0.227  0.675  3.83  1.57  3.59
```

Als Ergebnis erhalten wir die zusätzlichen Variablen a, b, c jeweils mit "_dm" und "_gm". Was verbirgt sich dahinter? Wir haben einen Datensatz mit den unzentrierten / Rohvariablen der Skalen (ohne Suffix), den zentrierten Variablen (Suffix `_dm`) und den Mittelwerten der Personen (Suffix `_gm`), den wir zur weiteren Verwendung auch abspeichern.

Damit ist die Transformation der Daten abgeschlossen! Wir können die Datensätze - "df_cfa_long" für die Items und "df_cfa_long_scores" für die Skalen nun abspeichern.

```
save(df_cfa_long, df_cfa_long_scores, file = "../data/df_cfa_long.RData")
```

3.7 Überprüfe dein Verständnis

```
beispiel_zentrierung <- df_cfa_long_scores |>
  filter(id == 1) |>
  select(id, time, a, a_dm, a_gm)

beispiel_zentrierung
```

```
# A tibble: 10 x 5
      id  time     a    a_dm  a_gm
   <int> <dbl> <dbl>   <dbl> <dbl>
 1     1     1  3.58 -0.257   3.83
 2     1     2  3.62 -0.215   3.83
 3     1     3  2.37 -1.47    3.83
 4     1     4  3.60 -0.229   3.83
 5     1     5  3.88  0.0481  3.83
 6     1     6  4.17  0.340   3.83
 7     1     7  3.59 -0.240   3.83
 8     1     8  3.72 -0.111   3.83
 9     1     9  5.28  1.45    3.83
10     1    10  4.51  0.679   3.83
```

Warum ist der Wert, den Person 1 in "a_gm" hat, in jeder Zeile gleich, nicht aber bei "a_dm" und "a"? Wie müsste man "a" transformieren, damit man auf "a_gm" und "a_dm" kommt? Denke an die mathematischen Operationen die du in `mutate()` eingeben müsstest, wie Plus, Minus (`+`, `-`, `/`, …) und die Funktion für Mittelwert (`mean()`).

Lösung

Die Variable a ist die Rohvariable, die den gemessenen Wert auf der Skala an jedem Tag angibt. Variablen mit "_gm" und "_dm" werden von `sjmisc::de_mean()` erstellt. Variablen mit "_gm" stehen für den Mittelwert der Person über alle Tage hinweg. Variablen mit "_dm" stehen für die täglichen Abweichungen vom Mittelwert der Person. "a_dm" ergibt sich aus: $a = a - a\_gm$. Wir können dies auch in R replizieren mit:

```
beispiel_zentrierung |>
  mutate(a_gm_manuell = mean(a),
         a_dm_manuell = a - a_gm_manuell)
```

```
# A tibble: 10 x 7
      id  time     a    a_dm  a_gm a_gm_manuell a_dm_manuell
```

```
   <int> <dbl> <dbl>   <dbl> <dbl>        <dbl>        <dbl>
 1     1     1  3.58 -0.257   3.83         3.83      -0.257
 2     1     2  3.62 -0.215   3.83         3.83      -0.215
 3     1     3  2.37 -1.47    3.83         3.83      -1.47
 4     1     4  3.60 -0.229   3.83         3.83      -0.229
 5     1     5  3.88  0.0481  3.83         3.83       0.0481
 6     1     6  4.17  0.340   3.83         3.83       0.340
 7     1     7  3.59 -0.240   3.83         3.83      -0.240
 8     1     8  3.72 -0.111   3.83         3.83      -0.111
 9     1     9  5.28  1.45    3.83         3.83       1.45
10     1    10  4.51  0.679   3.83         3.83       0.679
```

3.8 Übung Datenaufbereitung

Schau dir den Datensatz `df_cfa_exercise` an und repliziere die Schritte oben:

1) Daten in Langformat transformieren und als `df_uebung_lang <-` zuweisen.
2) Skalenscores für x und y erstellen und als `df_uebung_lang_scores <-` zuweisen.
3) Skalenscores für x und y zentrieren (weiterhin in `df_uebung_lang_scores`).

Du kannst dazu den Code von oben wiederverwenden und auf den hier verwendeten Datensatz und seine Variablen anpassen.

```
load("../data/df_cfa_exercise.RData")
head(df_cfa_exercise)
```

```
# A tibble: 6 x 101
     id x1_t1 x1_t2 x1_t3 x1_t4 x1_t5 x1_t6 x1_t7 x1_t8 x1_t9 x1_t10 x2_t1
  <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl> <dbl>
1     1  4.93  3.48  5.21  3.78  2.82  2.87  4.40  4.00  2.83   4.53 3.57
2     2  5.02  2.47  2.26  3.38  3.41  2.93  3.72  3.10  4.07   2.92 3.66
3     3  3.66  4.52  2.63  2.10  3.30  4.67  3.85  3.43  3.42   4.35 3.37
4     4  1.93  1.84  4.25  3.23  2.46  2.82  2.60  3.22  4.20   2.27 0.707
5     5  2.27  2.91  3.18  2.94  3.71  2.72  3.51  1.55  2.73   4.95 4.31
6     6  3.18  4.05  3.54  4.06  2.37  2.07  3.39  3.49  3.66   1.89 2.91
# i 89 more variables: x2_t2 <dbl>, x2_t3 <dbl>, x2_t4 <dbl>, x2_t5 <dbl>,
#   x2_t6 <dbl>, x2_t7 <dbl>, x2_t8 <dbl>, x2_t9 <dbl>, x2_t10 <dbl>,
#   x3_t1 <dbl>, x3_t2 <dbl>, x3_t3 <dbl>, x3_t4 <dbl>, x3_t5 <dbl>,
#   x3_t6 <dbl>, x3_t7 <dbl>, x3_t8 <dbl>, x3_t9 <dbl>, x3_t10 <dbl>,
#   x4_t1 <dbl>, x4_t2 <dbl>, x4_t3 <dbl>, x4_t4 <dbl>, x4_t5 <dbl>,
#   x4_t6 <dbl>, x4_t7 <dbl>, x4_t8 <dbl>, x4_t9 <dbl>, x4_t10 <dbl>,
```

```
#   x5_t1 <dbl>, x5_t2 <dbl>, x5_t3 <dbl>, x5_t4 <dbl>, x5_t5 <dbl>, ...
```

```
# 1. Daten in Langformat transformieren - funktionen: pivot_longer(), pivot_wider()


# 2. Skalenscores für X und Y erstellen - funktionen: group_by(), summarise()

# 3. Skalenscores für X und Y zentrieren - Funktionen: de_mean()
```

Lösung

```
load("../data/df_cfa_exercise.RData")
head(df_cfa_exercise)
```

```
# A tibble: 6 x 101
     id x1_t1 x1_t2 x1_t3 x1_t4 x1_t5 x1_t6 x1_t7 x1_t8 x1_t9 x1_t10 x2_t1
  <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl> <dbl>
1     1  4.93  3.48  5.21  3.78  2.82  2.87  4.40  4.00  2.83   4.53 3.57
2     2  5.02  2.47  2.26  3.38  3.41  2.93  3.72  3.10  4.07   2.92 3.66
3     3  3.66  4.52  2.63  2.10  3.30  4.67  3.85  3.43  3.42   4.35 3.37
4     4  1.93  1.84  4.25  3.23  2.46  2.82  2.60  3.22  4.20   2.27 0.707
5     5  2.27  2.91  3.18  2.94  3.71  2.72  3.51  1.55  2.73   4.95 4.31
6     6  3.18  4.05  3.54  4.06  2.37  2.07  3.39  3.49  3.66   1.89 2.91
# i 89 more variables: x2_t2 <dbl>, x2_t3 <dbl>, x2_t4 <dbl>, x2_t5 <dbl>,
#   x2_t6 <dbl>, x2_t7 <dbl>, x2_t8 <dbl>, x2_t9 <dbl>, x2_t10 <dbl>,
#   x3_t1 <dbl>, x3_t2 <dbl>, x3_t3 <dbl>, x3_t4 <dbl>, x3_t5 <dbl>,
#   x3_t6 <dbl>, x3_t7 <dbl>, x3_t8 <dbl>, x3_t9 <dbl>, x3_t10 <dbl>,
#   x4_t1 <dbl>, x4_t2 <dbl>, x4_t3 <dbl>, x4_t4 <dbl>, x4_t5 <dbl>,
#   x4_t6 <dbl>, x4_t7 <dbl>, x4_t8 <dbl>, x4_t9 <dbl>, x4_t10 <dbl>,
#   x5_t1 <dbl>, x5_t2 <dbl>, x5_t3 <dbl>, x5_t4 <dbl>, x5_t5 <dbl>, ...
```

```r
# 1. Daten in Langformat transformieren - funktionen: pivot_longer(), pivot_wider()
df_uebung_superlang <- df_cfa_exercise |>
  pivot_longer(
    cols = -id, # All columns except id
    names_to = c("variable", "time"),
    names_sep = "_t"
  )
df_uebung_lang <- df_uebung_superlang |>
  pivot_wider(names_from = variable,
              values_from = value) |>
  mutate(time = as.numeric(time))

# 2. Skalenscores für X und Y erstellen - funktionen: group_by(), summarise()
df_uebung_lang_scores <- df_uebung_lang |> group_by(id, time) |>
  summarise(
    x = rowMeans(across(starts_with("x")), na.rm = TRUE),
    y = rowMeans(across(starts_with("y")), na.rm = TRUE),
    .groups = "drop" # group_by() wieder aufheben für den finalen Datensatz
  )

# 3. Skalenscores für X und Y zentrieren - Funktionen: de_mean()
df_uebung_lang_scores <- df_uebung_lang_scores |>
  de_mean(x, y, grp = "id")


head(df_uebung_lang_scores)
```

```
# A tibble: 6 x 8
     id  time     x     y    x_dm   y_dm  x_gm  y_gm
  <int> <dbl> <dbl> <dbl>   <dbl>  <dbl> <dbl> <dbl>
1     1     1  4.42  3.96  0.396   0.160  4.02  3.80
2     1     2  3.57  4.30 -0.451   0.499  4.02  3.80
3     1     3  4.53  3.91  0.508   0.106  4.02  3.80
4     1     4  3.45  3.49 -0.573  -0.308  4.02  3.80
5     1     5  3.90  3.56 -0.119  -0.245  4.02  3.80
6     1     6  3.92  4.20 -0.0953  0.394  4.02  3.80
```

3.9 Abspeichern der gebildeten Skalen

Zum Schluss speichern wir die Ergebnisse (sowohl die Items als auch die Skalen in Langformat) der Übung ab.

```
save(df_uebung_lang, df_uebung_lang_scores, file = "../data/df_uebung.RData")
```

4 Kapitel 3: Deskriptive Analysen und Überprüfung von Voraussetzungen

4.1 Pakete laden

```
if (!require("pacman")) install.packages("pacman")
```

```
Lade nötiges Paket: pacman
```

```
pacman::p_load(haven, psych,
               sjmisc, sjPlot, writexl, lavaan,
               tidyverse, multilevelTools, franzpak)
```

4.2 Daten laden

```
load("../data/df_cfa_long.RData")

head(df_cfa_long_scores)
```

```
# A tibble: 6 x 11
     id  time     a     b     c    a_dm   b_dm   c_dm  a_gm  b_gm  c_gm
  <int> <dbl> <dbl> <dbl> <dbl>   <dbl>  <dbl>  <dbl> <dbl> <dbl> <dbl>
1     1     1  3.58 2.02   3.06 -0.257   0.448 -0.527  3.83  1.57  3.59
2     1     2  3.62 1.68   4.23 -0.215   0.117  0.643  3.83  1.57  3.59
3     1     3  2.37 0.350  2.47 -1.47   -1.22  -1.12   3.83  1.57  3.59
4     1     4  3.60 1.44   2.85 -0.229  -0.132 -0.738  3.83  1.57  3.59
5     1     5  3.88 1.24   3.16  0.0481 -0.328 -0.427  3.83  1.57  3.59
6     1     6  4.17 1.34   4.26  0.340  -0.227  0.675  3.83  1.57  3.59
```

Schauen wir uns nochmal die Datenstruktur unserer aufbereiteten Datensätze aus dem vorhergehenden Kapitel an.

In `df_cfa_long_scores` haben wir haben die Variablen mit Fokus auf die Skalenscores:

- id: Gruppierungsvariable / Personen-ID, von 1 - 100
- time: Zeitpunkt / Tag der Messung, von 1-10
- a, b, c: Rohvariablen (Skalenscores von a, b, und c. Die Items aus denen a,b,c gebildet sind brauchen wir nur bei bestimmten Abschnitten)
- a_dm - c_dm: Personen-zentrierte Variablen
- a_gm - c_gm : Personen-Mittelwerte

```
head(df_cfa_long)
```

```
# A tibble: 6 x 15
     id  time    a1    a2    a3    a4    a5    b1       b2      b3     b4    b5
  <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>    <dbl>   <dbl>  <dbl> <dbl>
1     1     1  3.34  3.38  3.22  4.40  3.54 3.37   1.70    0.584    1.58   2.85
2     1     2  2.15  5.11  5.68  2.77  2.39 2.49   0.952   2.10     0.331  2.55
3     1     3  1.61  3.00  2.50  1.90  2.81 0.992 -0.686   0.00315 -0.518  1.96
4     1     4  3.35  3.85  4.97  2.61  3.23 2.25   0.00412 0.711    2.19   2.03
5     1     5  3.32  3.69  4.70  3.48  4.21 1.95   0.676   0.922    0.285  2.36
6     1     6  4.17  5.66  3.27  4.19  3.57 1.54  -0.373   1.30     1.69   2.54
# i 3 more variables: c1 <dbl>, c2 <dbl>, c3 <dbl>
```

In `df_cfa_long` haben wir die einzelnen Item abgespeichert.

- id: Gruppierungsvariable / Personen-ID, von 1 - 100
- time: Zeitpunkt / Tag der Messung, von 1-10
- a1-a5: 5 Items aus Skala für "a"
- b1-b5: 5 Items aus Skala für "b"
- c1-c3: 3 Items aus Skala für "c"

4.3 Reliabilitätsanalyse

Die Reliabilitätsanalyse basiert auf den Items, nicht auf den Skalenwerten (`df_cfa_long`). Bei täglich erhobenen Skalen nehmen wir die `omegaSEM()` Funktion. Als erstes Argument geben wir die Items in einem Character-Vector mittels `c()`, die Items werden mit Anführungszeichen angegeben. Falls ihr die Itemnamen nicht wisst, könnt ihr sie mit `names(df_cfa_long)` nachsehen.

```r
scalea_reliab <- omegaSEM(
  items = c("a1", "a2", "a3", "a4", "a5"),
  id = "id",
  data = df_cfa_long)
scalea_reliab$Results
```

```
           label   est ci.lower ci.upper
42  omega_within 0.739    0.712    0.766
50 omega_between 0.848    0.796    0.901
```

Hier erscheint teils eine Warnung, weil nicht alle Personen (cluster) Varianz auf den Items haben. Dies können wir ignorieren. In den simulierten Daten, die wir verwenden, ist dies jedoch nicht der Fall. Dann können wir den Output ansehen. Omega_within gibt die Reliabilität für Unterschiede innerhalb der Person an, und Omega_between gibt die Reliabilität für Unterschiede zwischen Personen an. Die Reliabilitäten sollten über .70 liegen für eine gute Reliabilität auf beiden Leveln.

```r
scaleb_reliab <- omegaSEM(
  items = c("b1", "b2", "b3", "b4", "b5"),
  id = "id",
  data = df_cfa_long)

scaleb_reliab$Results
```

```
           label   est ci.lower ci.upper
42  omega_within 0.744    0.718    0.770
50 omega_between 0.888    0.850    0.927
```

```r
scalec_reliab <- omegaSEM(
  items = c("c1", "c2", "c3"),
  id = "id",
  data = df_cfa_long)

scalec_reliab$Results
```

```
           label   est ci.lower ci.upper
28  omega_within 0.628    0.585    0.670
34 omega_between 0.764    0.673    0.855
```

4.4 Korrelationstabelle

In quantitativ-empirischen psychologischen Artikeln ist (fast) immer die Korrelationstabelle die erste Tabelle des Artikels. Unser nächstes Ziel ist es, eine Korrelationstabelle anzufertigen, in der wir die a) Mittelwerte, b) Standardabweichungen, c) ICC (Anteile der Zwischen-Person Varianz) und Korrelationen Zwischen und Innerhalb von Personen integrieren.

Die Funktion `cortable_multilevel()` berechnet und stellt diese Angaben für uns zusammen. Sie nimmt die Argumente `varnames` mit den Variablennamen als Vektor `c("a", "b", "c")` und die Gruppierungsvariable, die angibt zu welcher Level-2 Einheit eine Beobachtung/Zeile gehört (`grp = "id"`).

```
cortable_integriert <- cortable_multilevel(df_cfa_long_scores,
                                           varnames = c("a", "b", "c"),
                                           grp = "id")
cortable_integriert
```

```
# A tibble: 3 x 7
  Variable M     SD    ICC   `1.`   `2.`   `3.`
  <chr>    <chr> <chr> <chr> <chr>  <chr>  <chr>
1 1.a      3.03  0.99  .49   -      .32**  .39***
2 2.b      1.95  1.07  .55   .22*** -      .29**
3 3.c      1.99  1.06  .47   .25*** .29*** -
```

Betrachten wir nun die einzelnen Elemente der Korrelationstabelle:

- Mittelwerte / M: Wir sehen, dass a einen höheren Mittelwert (M = 3.03) als b und c (M = 1.94, M = 1.99) aufweist. Für die Verteilung der Variablen sehen wir uns idealerweise auch Histogramme an.
- Standardabweichungen / SD: Die Streuung der Variablen ist ähnlich und ihre Standardabweichungen liegen zwischen 0.99 und 1.07.
- 1.-3. Wir erhalten im unteren Dreieck die Inner-Person-Korrelationen, und im oberen Dreieck die Zwischen-Person-Korrelationen. Alle Korrelationen sind signifikant positiv.

4.4.1 Export von Tabellen zu Excel

Wir exportieren die Korrelationstabelle nach Excel mittels `write_xlsx()`.

```
# eval: false
write_xlsx(cortable_integriert, path = "korrelationstabelle.xlsx")
```

Die Excel-Tabelle lässt sich dann in Word kopieren und weiter verarbeitet werden, z.B. mit den richtigen Variablennamen versehen werden etc. Damit haben wir nun die Datenaufbereitung und deskriptive Datenanalyse abgeschlossen.

4.5 Übung

Wendet diese Schritte nun auf den Datensatz aus der letzten Übung an. Zunächst laden wir die Daten, die in der letzten Übung abgespeichert werden sollten.

Wir haben die beiden Datensätze `df_uebung_lang` für alle Items (x1-x5, y1-y5) im langen Datenformat und `df_uebung_lang_scores` für alle Skalenwerte (x,y) im Langformat.

```
load("../data/df_uebung.RData")
head(df_uebung_lang)
```

```
# A tibble: 6 x 12
     id  time    x1    x2    x3    x4    x5    y1    y2    y3    y4    y5
  <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1     1     1  4.93  3.57  3.12  5.76  4.70  4.33  4.48  2.85  4.68  3.48
2     1     2  3.48  3.72  2.20  4.87  3.58  3.60  4.45  5.55  4.68  3.22
3     1     3  5.21  4.03  2.42  5.11  5.88  3.55  4.76  3.77  3.61  3.85
4     1     4  3.78  2.60  2.60  4.14  4.11  3.05  3.29  3.65  4.36  3.12
5     1     5  2.82  4.92  2.18  4.98  4.60  2.59  5.68  2.27  3.40  3.84
6     1     6  2.87  4.42  3.37  4.90  4.06  3.08  4.68  5.26  5.14  2.82
```

```
head(df_uebung_lang_scores)
```

```
# A tibble: 6 x 8
     id  time     x     y    x_dm   y_dm  x_gm  y_gm
  <int> <dbl> <dbl> <dbl>   <dbl>  <dbl> <dbl> <dbl>
1     1     1  4.42  3.96  0.396   0.160  4.02  3.80
2     1     2  3.57  4.30 -0.451   0.499  4.02  3.80
3     1     3  4.53  3.91  0.508   0.106  4.02  3.80
4     1     4  3.45  3.49 -0.573  -0.308  4.02  3.80
5     1     5  3.90  3.56 -0.119  -0.245  4.02  3.80
6     1     6  3.92  4.20 -0.0953  0.394  4.02  3.80
```

```
head(df_uebung_lang_scores)
```

```
# A tibble: 6 x 8
     id  time     x     y    x_dm   y_dm  x_gm  y_gm
  <int> <dbl> <dbl> <dbl>   <dbl>  <dbl> <dbl> <dbl>
1     1     1  4.42  3.96  0.396   0.160  4.02  3.80
2     1     2  3.57  4.30 -0.451   0.499  4.02  3.80
3     1     3  4.53  3.91  0.508   0.106  4.02  3.80
4     1     4  3.45  3.49 -0.573  -0.308  4.02  3.80
5     1     5  3.90  3.56 -0.119  -0.245  4.02  3.80
6     1     6  3.92  4.20 -0.0953  0.394  4.02  3.80
```

4.5.1 Reliabilitsätsanalyse

Berechnet die Omega-Within und Omega-Between Reliabilitäten für X und Y mittels `omegaSEM()` und analysiert die Angaben. Passt dazu den Code aus dem Abschnitt 'Reliabilitätsanalyse' oben auf das Beispiel an.

- Weisen die Skalen X und Y hinreichende Omega-Werte auf, um Inner-Person Unterschiede reliabel zu messen? Begründet eure Antworten.
- Weisen die Skalen X und Y hinreichend Omega-Werte auf, um Zwischen-Person Unterschiede reliabel zu messen? Begründet eure Antworten.

Lösung

```
scalex_reliab <- omegaSEM(c("x1", "x2", "x3", "x4", "x5"), "id", df_uebung_lang)
scalex_reliab$Results
```

```
           label   est ci.lower ci.upper
42  omega_within 0.721    0.692    0.750
50 omega_between 0.877    0.834    0.919
```

```
scaley_reliab <- omegaSEM(c("y1", "y2", "y3", "y4", "y5"), "id", df_uebung_lang)
scaley_reliab$Results
```

```
           label   est ci.lower ci.upper
42  omega_within 0.362    0.297    0.428
50 omega_between 0.884    0.845    0.924
```

4.5.2 ICC und Korrelationstabelle

Berechnet den ICC für X und Y mittels `cortable_multilevel()` und analysiert die Angaben. Passt dazu den Code aus dem Abschnitt 'Korrelationstabelle' oben auf das Beispiel an.

- Weisen die Skalen X und Y hinreichend Varianz auf der Inner-Person-Ebene auf, um sie mittels Mehrebenen-Modelle zu analysieren? Begründet eure Antwort.
- Wie fällt die Korrelation zwischen X und Y auf Zwischen-Person Ebene aus? Ist es basierend auf der Prüfungen der Voraussetzungen (ICCs, Reliabilitäten) sinnvoll, diese zu interpretieren?
- Wie fällt die Korrelation zwischen X und Y auf Inner-Person Ebene aus? Ist es basierend auf der Prüfungen der Voraussetzungen (ICCs, Reliabilitäten) sinnvoll, diese zu interpretieren?

Lösung

```
mehrebenen_stats <- df_uebung_lang_scores |>
    cortable_multilevel(varnames = c("x", "y"), grp = "id")

print(mehrebenen_stats)
```

```
# A tibble: 2 x 6
  Variable M     SD    ICC   `1.`  `2.`
  <chr>    <chr> <chr> <chr> <chr> <chr>
1 1.x      3.04  0.99  .53   -     .28**
2 2.y      2.05  0.93  .69   .02   -
```

Beide Variablen haben einen ICC unter .80, unserer Faustregel, d.h. 20% oder mehr Varianz liegt auf der Inner-Person Ebene. Somit weisen sie genügend Varianz auf Inner-Person-Ebene aus und eine Mehrebenen-Analyse ist möglich. X scheint mehr Varianz auf Inner-Person zu haben (ICC: .53; Varianz auf Inner-Person-Ebene ist 1-ICC, also .47 oder 47%) als Y (31%).

4.5.2.1 Zwischen-Person Ebene

Die Korrelation zwischen X und Y beträgt r = .28 und ist signifikant. Die Reliabilität auf Level-2/ Zwischen-Person Ebene, durch omega_between angegeben, ist für X und Y gut (> .80). Die ICCs weisen auf ausreichend Varianz auf Zwischen-Person Ebene hin (53%-69%). Daher ist sie sinnvoll zu interpretieren. DIe Variablen sind moderat positiv miteinander assoziiert.

4.5.2.2 Inner-Person Ebene

Für Y, auch wenn es durchaus Varianz auf Inner-Person Ebene gibt (ICC = .69, damit sind 31% der Varianz auf Inner-Person Ebene), ist auf Inner-Person Ebene ist die Reliabilität sehr niedrig (Omega-within = .32). Das heisst, dass die Varianz innerhalb der Personen über die Tage auf Variable Y nicht reliabel gemessen werden. Die Inner-Person Varianz auf Y ist somit nicht sinnvoll zu interpretieren und stellt vermutlich nur "Rauschen" dar. Die Korrelation mit X ist nicht sinnvoll zu interpretieren (und fällt auch nicht signifikant aus), auch wenn X sowohl ausreichend Level-1 Varianz hat als auch reliabel gemessen ist.

4.6 Zusatz

Die folgenden Analysen sind optional, geben aber ein tieferes Verständnis des Materials.

4.6.1 Blick hinter die Kulissen: Berechnung von Omega mittels einer Mehrebenen konfirmatorischen Faktorenanalysen

Für eine genauere Auswertung können wir `omegaSEM()` mit dem Parameter `savemodel = TRUE` laufen lassen und uns mittels `summary()` die konfirmatorische Faktoranalyse (CFA) genauer ansehen.

Wie CFAs funktionieren, kann hier repetiert werden: Statistik IV - Methodenlehre

Zudem können wir uns mit `lavInspect()` die Modellparameter ansehen, um zu verstehen wie die Reliabilitätskoeffizient gebildet wird.

```
scalec_reliab <- omegaSEM(c("c1", "c2", "c3"), "id", df_cfa_long, savemodel = TRUE)
scalec_reliab$Fit |> summary(fit = TRUE, stand = TRUE)
```

```
lavaan 0.6-18 ended normally after 32 iterations

  Estimator                                         ML
  Optimization method                           NLMINB
  Number of model parameters                        15
  Number of inequality constraints                   6

  Number of observations                          1000
  Number of clusters [id]                          100
```

```
Model Test User Model:

  Test statistic                                0.000
  Degrees of freedom                                0

Model Test Baseline Model:

  Test statistic                              383.809
  Degrees of freedom                                6
  P-value                                       0.000

User Model versus Baseline Model:

  Comparative Fit Index (CFI)                   1.000
  Tucker-Lewis Index (TLI)                      1.000

Loglikelihood and Information Criteria:

  Loglikelihood user model (H0)             -4446.130
  Loglikelihood unrestricted model (H1)     -4446.130

  Akaike (AIC)                               8922.260
  Bayesian (BIC)                             8995.876
  Sample-size adjusted Bayesian (SABIC)      8948.235

Root Mean Square Error of Approximation:

  RMSEA                                         0.000
  90 Percent confidence interval - lower        0.000
  90 Percent confidence interval - upper        0.000
  P-value H_0: RMSEA <= 0.050                      NA
  P-value H_0: RMSEA >= 0.080                      NA

Standardized Root Mean Square Residual (corr metric):

  SRMR (within covariance matrix)               0.000
  SRMR (between covariance matrix)              0.000

Parameter Estimates:

  Standard errors                            Standard
  Information                                Observed
  Observed information based on               Hessian
```

```
Level 1 [within]:

Latent Variables:
                   Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
  f_within =~
    c1       (wl1)    0.596    0.043   13.816    0.000    0.596    0.595
    c2       (wl2)    0.653    0.046   14.176    0.000    0.653    0.622
    c3       (wl3)    0.587    0.043   13.605    0.000    0.587    0.580

Variances:
                   Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
    f_within          1.000                              1.000    1.000
   .c1       (wr1)    0.648    0.048   13.565    0.000    0.648    0.645
   .c2       (wr2)    0.675    0.054   12.435    0.000    0.675    0.613
   .c3       (wr3)    0.677    0.048   14.183    0.000    0.677    0.663


Level 2 [id]:

Latent Variables:
                   Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
  f_between =~
    c1       (bl1)    0.529    0.095    5.570    0.000    0.529    0.660
    c2       (bl2)    0.704    0.106    6.614    0.000    0.704    0.803
    c3       (bl3)    0.680    0.115    5.905    0.000    0.680    0.694

Intercepts:
                   Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
   .c1               2.003    0.086   23.252    0.000    2.003    2.500
   .c2               1.948    0.094   20.798    0.000    1.948    2.224
   .c3               2.021    0.103   19.615    0.000    2.021    2.063

Variances:
                   Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
    f_betwen          1.000                              1.000    1.000
   .c1       (br1)    0.362    0.081    4.446    0.000    0.362    0.564
   .c2       (br2)    0.272    0.107    2.536    0.011    0.272    0.355
   .c3       (br3)    0.497    0.120    4.132    0.000    0.497    0.518

Defined Parameters:
                   Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
```

```
    num_within        3.371    0.262   12.875    0.000    3.371    3.233
    denom_within      5.370    0.253   21.213    0.000    5.370    5.155
    omega_within      0.628    0.022   29.175    0.000    0.628    0.627
    num_between       3.659    0.764    4.787    0.000    3.659    4.657
    denom_between     4.789    0.754    6.354    0.000    4.789    6.094
    omega_between     0.764    0.046   16.440    0.000    0.764    0.764

Constraints:
                                                 |Slack|
    wr1 - 0                                       0.648
    wr2 - 0                                       0.675
    wr3 - 0                                       0.677
    br1 - 0                                       0.362
    br2 - 0                                       0.272
    br3 - 0                                       0.497
```

```
lavInspect(scalec_reliab$Fit, "list") |>
  select(lhs, op, rhs, free, level, free, label, est, se) |>
  mutate(across(where(is.numeric), round, 2)) # alternatively, parTable()
```

```
Warning: There was 1 warning in `mutate()`.
i In argument: `across(where(is.numeric), round, 2)`.
Caused by warning:
! The `...` argument of `across()` is deprecated as of dplyr 1.1.0.
Supply arguments directly to `.fns` through an anonymous function instead.

  # Previously
  across(a:b, mean, na.rm = TRUE)

  # Now
  across(a:b, \(x) mean(x, na.rm = TRUE))

            lhs op                         rhs free level         label  est
1       f_within =~                          c1    1     1           wl1 0.60
2       f_within =~                          c2    2     1           wl2 0.65
3       f_within =~                          c3    3     1           wl3 0.59
4       f_within ~~                   f_within    0     1               1.00
5             c1 ~~                         c1    4     1           wr1 0.65
6             c2 ~~                         c2    5     1           wr2 0.68
7             c3 ~~                         c3    6     1           wr3 0.68
8             c1 ~1                               0     1               0.00
9             c2 ~1                               0     1               0.00
```

```
10           c3 ~1                                0     1              0.00
11     f_within ~1                                0     1              0.00
12    f_between =~                           c1    7     2          bl1 0.53
13    f_between =~                           c2    8     2          bl2 0.70
14    f_between =~                           c3    9     2          bl3 0.68
15    f_between ~~                    f_between    0     2              1.00
16           c1 ~~                           c1   10     2          br1 0.36
17           c2 ~~                           c2   11     2          br2 0.27
18           c3 ~~                           c3   12     2          br3 0.50
19           c1 ~1                               13     2              2.00
20           c2 ~1                               14     2              1.95
21           c3 ~1                               15     2              2.02
22    f_between ~1                                0     2              0.00
23          wr1  >                             0    0     0              0.65
24          wr2  >                             0    0     0              0.68
25          wr3  >                             0    0     0              0.68
26   num_within :=               (wl1+wl2+wl3)^2    0     0   num_within 3.37
27 denom_within := (wl1+wl2+wl3)^2+(wr1+wr2+wr3)    0     0 denom_within 5.37
28 omega_within :=       num_within/denom_within    0     0 omega_within 0.63
29          br1  >                             0    0     0              0.36
30          br2  >                             0    0     0              0.27
31          br3  >                             0    0     0              0.50
32  num_between :=               (bl1+bl2+bl3)^2    0     0  num_between 3.66
33 denom_between := (bl1+bl2+bl3)^2+(br1+br2+br3)    0     0 denom_between 4.79
34 omega_between :=     num_between/denom_between    0     0 omega_between 0.76
    se
1  0.04
2  0.05
3  0.04
4  0.00
5  0.05
6  0.05
7  0.05
8  0.00
9  0.00
10 0.00
11 0.00
12 0.10
13 0.11
14 0.12
15 0.00
16 0.08
17 0.11
```

```
18 0.12
19 0.09
20 0.09
21 0.10
22 0.00
23 0.00
24 0.00
25 0.00
26 0.26
27 0.25
28 0.02
29 0.00
30 0.00
31 0.00
32 0.76
33 0.75
34 0.05
```

Wie der Output zeigt, ergibt sich die Omega-Reliabilität aus dem Anteil der durch den Faktor (f_within für den Faktor auf Level-1 bzw. f_between für den Faktor auf Level-2) erklärten Varianz der Items (Summe aller Item-Ladungen, quadriert für Varianz) geteilt durch die Gesamtvarianz der Items (durch Faktor erklärte Varianz der Items + Residualvarianz, d.h. übrigbleibende Varianz der Items). Diese Formel wird pro Varianzebene (Level-1, also tägliche Schwankungen innerhalb der Person) und Level-2 (Unterschiede zwischen Personen) getrennt berechnet.

4.6.2 Berechnung der einzelnen Kenngrössen der Korrelationstabelle

4.6.2.1 ICC und Within-Person Variance

Uns interessiert wie gross der Anteil der Varianz ist, der jeweils auf die zwei Ebenen der Daten entfallen (Inner-Person, Zwischen-Person-Ebene). Dies kann uns der ICC angeben. Mittels der Funktion `statsBy()` bekommen wir einige Analysen zu unseren Mehrebenen-Daten geliefert. Die Funktion benötigt zwei Argumente: den Datensatz und und die Gruppierungsvariable. Wir wählen entsprechend in `select()` die Variablen, die uns interessieren. Dies sind die Gruppierungsvariable "id" und die Rohvarianten der Variablen aus, da nur diese die Informationen über **beide** Ebenen enthalten (personen-zentrierte Variablen beinhalten nur Varianz auf Inner-Person-Ebene, Personen-Mittelwerte nur Varianz auf Zwischen-Person-Ebene). Die zerlegten Variablen mit den Kürzeln _dm und _gm brauchen wir erst später.

```r
mehrebenen_stats <- df_cfa_long_scores |>
  select(id, a, b, c) |>
    statsBy(group = "id")
```

Wir bekommen hier manchmal Warnungen, wenn wir auch reine Level-2 Variablen eingeschlossen haben. Dies können wir jedoch ignorieren. Mit `print()` bekommen wir eine Übersicht über die Ergebnisse der Resultate der `statsBy()` Funktion.

```r
print(mehrebenen_stats)
```

```
Statistics within and between groups
Call: statsBy(data = select(df_cfa_long_scores, id, a, b, c), group = "id")
Intraclass Correlation 1 (Percentage of variance due to groups)
  id    a    b    c
1.00 0.49 0.55 0.47
Intraclass Correlation 2 (Reliability of group differences)
  id    a    b    c
1.00 0.91 0.92 0.90
eta^2 between groups
a.bg b.bg c.bg
0.54 0.59 0.52

To see the correlations between and within groups, use the short=FALSE option in your print s
Many results are not shown directly. To see specific objects select from the following list:
 mean sd n F ICC1 ICC2 ci1 ci2 raw rbg ci.bg pbg rwg nw ci.wg pwg etabg etawg nwg nG Call
```

Uns interessiert nur die Intraclass Correlation 1. Intraclass Correlation (2) und Eta-Quadrat interessieren uns nicht.

Den ICC können wir uns auch direkt angeben lassen, indem wir aus dem Listenobjekt `mehrebenen_stats` mit dem Dollarzeichen `$` die Untervariable `ICC1` anwählen.

```r
icc <- mehrebenen_stats$ICC1 |>
  round(2) # runden
icc
```

```
  id    a    b    c
1.00 0.49 0.55 0.47
```

Alle Skalen im Beispiel haben ICCs in einem angemessenen Bereich (<.80). Dies heisst, dass genug tägliche Varianz vorhanden ist, um Mehrebenen-Analysen durchzuführen.

4.6.2.2 Mittelwerte

Mittelwerte wurden bereits - pro Person - durch die `statsBy()` Funktion gebildet. Den allgemeinen Mittelwert bekommen wir mit der Funktion `summarise()`. Diese erlaubt uns, zusammenfassende Werte zu bilden. Da wir dies gleich für mehrere Variablen machen, benutzen wir zudem `across()`, um die Summary gleich für mehrere Variablen zu bilden. Die Funktion benötigt als Argumente (a) die Namen der Variablen mit `c()` als einen Vektor zusammengefasst, (b) die Funktionen, wie sie gebildet werden (hier: `~mean(.x, na.rm = TRUE)` für das arithmetische Mittel unter Ausschluss aller nicht vorhandenen Werte) und (c) optional die Namen der ausgegebenen Variablen mittels ".names". Wir verwenden "m_{.col}". Abschliessend runden wir die Werte.

Ersetzt im folgenden Code in der Klammer von `c()` die Variablennamen mit denen, die euch interessieren, hier sowohl die täglichen als auch Baselinevariablen.

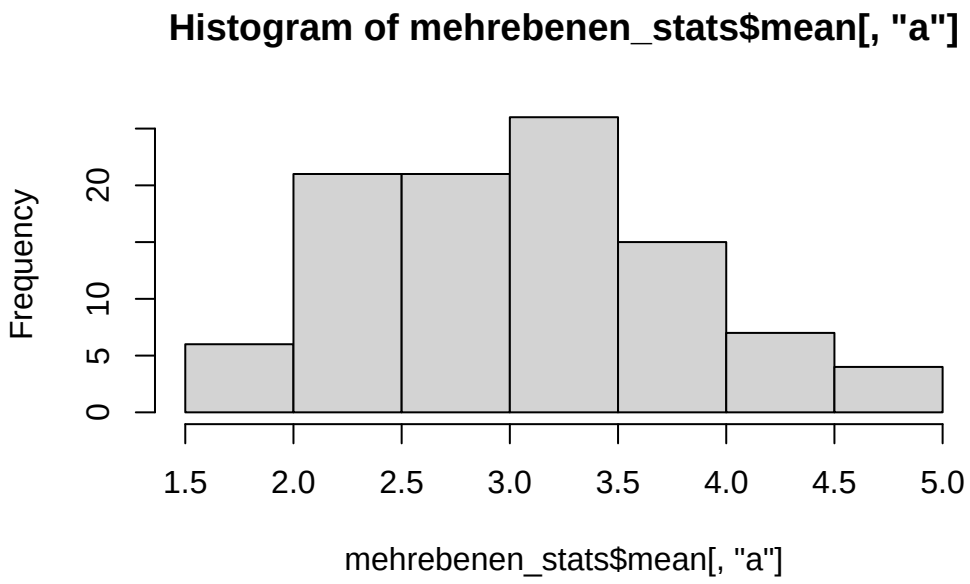
```
mittelwerte <- mehrebenen_stats$mean |>
  as_tibble() |>
  summarise(across(c(a,b,c), ~mean(.x, na.rm = TRUE)))

mittelwerte
```

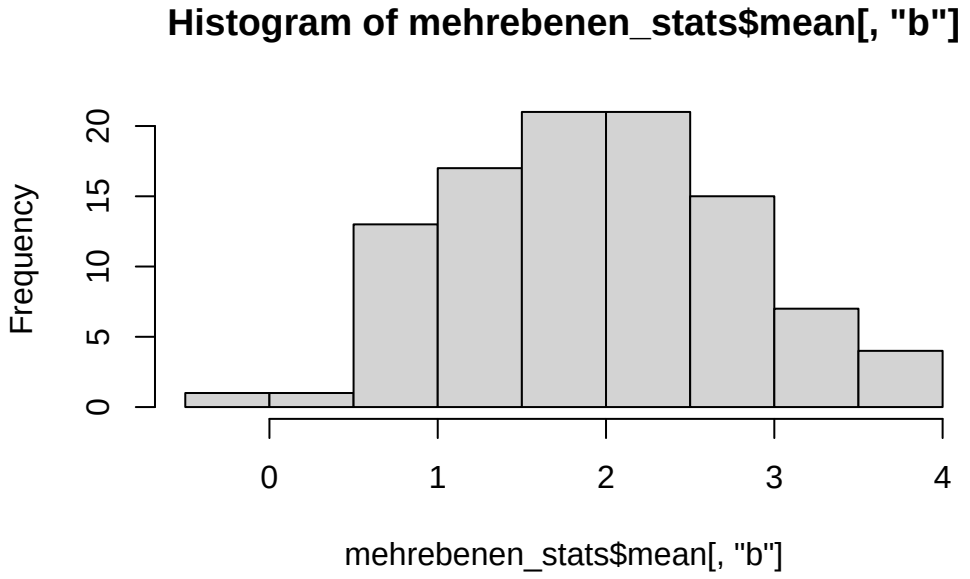
```
# A tibble: 1 x 3
      a     b     c
  <dbl> <dbl> <dbl>
1  3.03  1.95  1.99
```

Wir sehen, dass a einen höheren Mittelwert (M = 3.03) als b und c (M = 1.94, M = 1.99) aufweist. Für die Verteilung der Variablen sehen wir uns idealerweise auch Histogramme an.

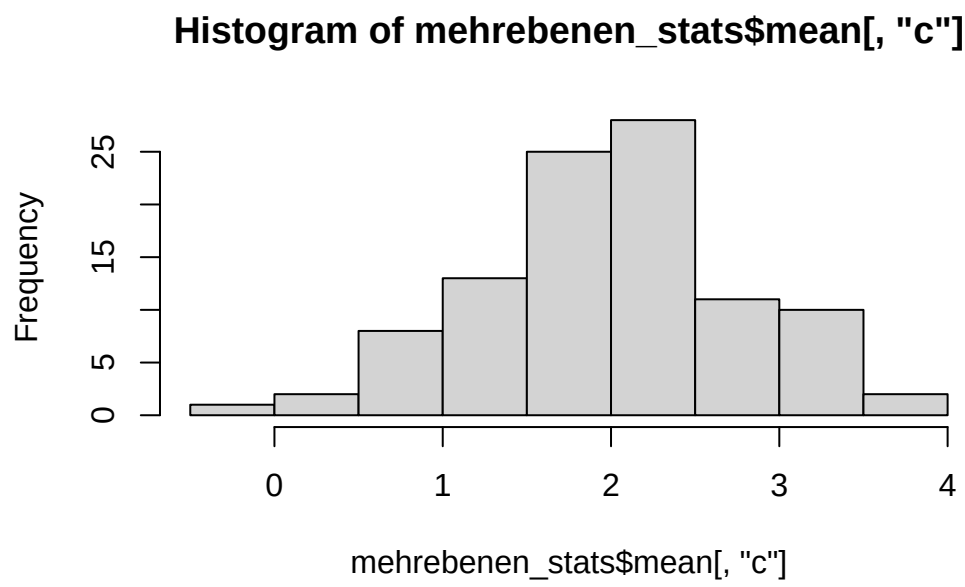
```
hist(mehrebenen_stats$mean[,"a"])
```




```
hist(mehrebenen_stats$mean[,"b"])
```



```
hist(mehrebenen_stats$mean[,"c"])
```



Alle Variablen scheinen vom Histogram her hinreichend normalverteilt.

4.6.2.3 Standardabweichungen

Ganz ähnlich wie mit Mittelwerten verfahren wir für die Standardabweichung, nur dass wir hier als Funktion `~sd(.x, na.rm = TRUE)` verwenden. Ersetzt auch hier im folgenden Code in der Klammer von `c()` die Variablennamen mit denen, die euch interessieren, hier sowohl die täglichen als auch Baselinevariablen. Die Baselinevariable "w" zeigt hier keine SD mit dieser Berechnung und muss separat berechnet werden.

```
standardabweichung <- mehrebenen_stats$sd |>
  as_tibble() |>
  summarise(across(c(a, b, c), ~mean(.x, na.rm = TRUE)))
standardabweichung
```

```
# A tibble: 1 x 3
      a     b     c
  <dbl> <dbl> <dbl>
1 0.688 0.696 0.755
```

4.6.2.4 Korrelationen

Wir wollen eine Korrelationstabelle, in der wir auf einen Blick sowohl die Zwischen-Person-Korrelationen als auch die Inner-Person-Korrelationen sehen. Die `statsBy()` Funktion, die wir bereits aufgerufen haben, gibt uns beides separat aus. und

```
mehrebenen_stats$rbg |> round(2) # Zwischen Person Kor.
```

```
     a.bg b.bg c.bg
a.bg 1.00 0.32 0.39
b.bg 0.32 1.00 0.29
c.bg 0.39 0.29 1.00
```

```
mehrebenen_stats$rwg |> round(2) # Inner Person Kor.
```

```
     a.wg b.wg c.wg
a.wg 1.00 0.22 0.25
b.wg 0.22 1.00 0.29
c.wg 0.25 0.29 1.00
```

Wir erhalten im unteren Dreieck die Inner-Person-Korrelationen, und im oberen Dreieck die Zwischen-Person-Korrelationen.

5 Kapitel 4: Hypothesentests - Teil 1

In diesem Kapitel verwenden wir verschiedene Regressionsmodelle die zur Überprüfung von Hypothesen eingesetzt werden.

- Random Intercept Modell / Null-Model
- Random Intercept, fixed slope Modell
- Random intercept, random slope Modell
- Erweiterung um Level-2 Prädiktoren

5.1 Vorbereitung

Install packages

```
if (!require("pacman")) install.packages("pacman")
```

```
Lade nötiges Paket: pacman
```

```
pacman::p_load(lmerTest, haven, brms, psych,
               sjmisc, sjPlot, sjlabelled, writexl, broom.mixed, qgraph,
               tidyverse, multilevelTools, parameters)
```

5.2 Daten einlesen

```
load("../data/df_example1.RData")
load("../data/df_example1c.RData")
```

Für diese Einheit verwenden wir den folgenden Datensatz (data.frame/tibble):

- df_example1: Alle Skalenscores im Long Format, mit personen-zentrierten Variablenvarianten ("_dm") und Personen-Mittelwerten der täglich gemessenenen Variablen ("_gm"). Struktur des Datensatzes kann man sich ansehen mit `head()` oder `print()`.

```r
head(df_example1)
```

```
# A tibble: 6 x 10
  id        y     m     x    y_dm   m_dm    x_dm  y_gm  m_gm  x_gm
  <fct> <dbl> <dbl> <dbl>   <dbl>  <dbl>   <dbl> <dbl> <dbl> <dbl>
1 1      4.00  4.77 2.37  -0.302   0.895  0.685   4.31  3.87  1.68
2 1      4.93  2.51 0.749  0.620  -1.36  -0.932   4.31  3.87  1.68
3 1      4.60  3.10 1.40   0.293  -0.777 -0.286   4.31  3.87  1.68
4 1      4.29  4.61 1.86  -0.0194  0.736  0.179   4.31  3.87  1.68
5 1      4.18  4.55 2.29  -0.122   0.674  0.607   4.31  3.87  1.68
6 1      3.63  3.59 1.70  -0.674  -0.285  0.0156  4.31  3.87  1.68
```

Im Folgenden betrachten wir ein Modell in dem y durch x vorhergesagt wird.

5.3 Random Intercept Modell / Null-Model

Die Funktion lmer() benötigt zwei Argumente, (a) die Formel und (b) den Datensatz. Zum Aufbau und Details der Formeln s. Folien.

Level 1: $y_{ij} = \beta_{0j} + e_{ij}$

Level 2 (random intercept): $\beta_{0j} = \gamma_{00} + u_{0j}$

```r
nullmodel <- lmer(y ~ (1 | id), data = df_example1)
```

Zur Ansicht der Ergebnisse haben wir zwei Optionen: Den `summary()` Befehl - die Standardansicht, wie von den Paketautoren implementiert, den `tidy()` Befehl aus dem broom-Package, und den `model_parameters()` Befehl aus dem parameters Package. tidy() und model_parameters() Funktionsoutputs könnten nach Excel/Word exportiert werden mittels der `write_xlsx()` Funktion, oder indem das Notebook als .docx "gerendert" (ausgegeben) wird.

```r
summary(nullmodel)
```

```
Linear mixed model fit by REML. t-tests use Satterthwaite's method [
lmerModLmerTest]
Formula: y ~ (1 | id)
   Data: df_example1

REML criterion at convergence: 2742.9
```