

Lösungen zu den Aufgaben

1. Aufgabe

In einer Studie untersuchte Frau Prof. Dr. Klug Ursachen von Entscheidungen im Rahmen von Einstellungen und Verhalten bei Pop-up Stores.

U.a. wurden folgende Fragen untersucht:

- Welchen (kausalen) Effekt hat die Distanz zum und Lage des Pop-up-Stores hinsichtlich der AV?
- Wie stark ist der Moderatoreffekt von Variablen wie z.B. Innovationsorientierung, Shopping-Relevanz und Soziodemografika?
- Ist ein Effekt auf Einstellung, Verhaltensintention und Verhalten zu beobachten?

Es handelt sich um ein experimentelles Design mit zwei Faktoren (Lage und Distanz) mit jeweils 3 Stufen.

Ein Teil der Daten ist (nur) für Lehrzwecke freigegeben.

Folgende Materialien stehen bereit:

- [Roh-Datensatz](#), $n = 90$, Gruppen 1-3
- [Studienkonzept](#)
- [Fragebogen](#)
- [Codebook](#)

Aufgaben

1. Entfernen Sie leere Zeilen und Spalten aus dem Datensatz. Tipp: Nutzen Sie das R-Paket `{{janitor}}`.
2. Entfernen Sie konstante Variablen. Tipp: Nutzen Sie das R-Paket `{{janitor}}`.
3. Prüfen Sie auf Duplikate, d.h. doppelte Zeilen. Tipp: Nutzen Sie das R-Paket `{{janitor}}`.
4. Entfernen Sie alle Spalten, die Zeit-Objekte enthalten.
5. Ersetzen Sie leere Zellen sowie Zellen mit Inhalt "N/A" durch NA, also durch einen fehlenden Wert. Tipp: `na_if()` aus `{{dplyr}}`.
6. Rekodieren Sie die Anker (Labels) der Ratingskala in Zahlen und zwar von -3 bis +3! Tipp: Nutzen Sie `recode()` aus `{{dplyr}}`.
7. Berechnen Sie Spalten-Mittelwerte für alle Konstrukte, die die Ratingskala verwenden. Tipp: Nutzen Sie `rowwise()` und `c_across()`.
8. Exportieren Sie die Daten als CSV- und als XLSX-Datei. Tipp: Nutzen Sie das R-Paket `{{rio}}`.
9. Berechnen Sie Cronbachs Alpha! Tipp: Nutzen Sie das R-Paket `{{psych}}`.
10. Berechnen Sie gängige deskriptive Statistiken für die Mittelwerte der Konstrukte. Tipp: Nutzen Sie das R-Paket `{{easystats}}` und daraus die Funktion `describe_distribution()`.
11. Importieren Sie diese Tabelle nach Word! Tipp: Nutzen Sie das R-Paket `{{flextable}}`.

Lösung

Ad 1.

Daten laden:

```
d_url <- "https://raw.githubusercontent.com/sebastiansauer/Lehre/main/data/popupstore/data/dla.csv"

dla <- read_csv(d_url)

## Rows: 90 Columns: 196
## — Column specification —————
## Delimiter: ","
## chr   (76): v004, v008, v009, v010, v011, v012...
## dbl   (5): v001, v003, v005, v023, v109
## lgl   (112): v024, v025, v026, v027, v028, v029...
## dtm   (3): v002, v006, v007
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

dim(dla)

## [1] 90 196
```

Die Tabelle umfasst 90 Zeilen und 196 Spalten.

Leere Zeilen/Spalten entfernen:

```
library(janitor)
d2 <-
  d1a %>%
    remove_empty()
```

Ad 2.

```
library(janitor)
d3 <-
  d2 %>%
    remove_constant()
```

Ad 3.

```
d3 %>%
  get_dupes()

## No variable names specified - using all columns.

## No duplicate combinations found of: v001, v002, v003, v005, v006, v007, v008, v009, v010, ... and 74 other variables

## # A tibble: 0 × 84
## # ... with 84 variables: v001 <dbl>, v002 <dtm>,
## #   v003 <dbl>, v005 <dbl>, v006 <dtm>,
## #   v007 <dtm>, v008 <chr>, v009 <chr>,
## #   v010 <chr>, v011 <chr>, v012 <chr>,
## #   v013 <chr>, v014 <chr>, v015 <chr>,
## #   v016 <chr>, v017 <chr>, v018 <chr>,
## #   v019 <chr>, v020 <chr>, v021 <chr>, ...
```

Keine Duplikate zu finden.

Ad 4.

```
d4 <-
  d3 %>%
    select(-c(v002, v006, v007))
```

Ad 5.

```
d4 %>%
  mutate(v001 = na_if(v001, ""),
         v001 = na_if(v001, "N/A"))

## # A tibble: 90 × 80
##   v001 v003 v005 v008 v009 v010 v011
##   <dbl> <dbl> <dbl> <chr> <chr> <chr> <chr>
## 1 794 25 1031050258 2a02:... <NA> Ja Ja
## 2 146 25 1376736840 2a02:... <NA> Ja Ja
## 3 459 4 355469450 2003:... http... Nein Ja
## 4 324 25 995386148 134.2... http... Ja Ja
## 5 257 25 689485052 2003:... http... Nein Nein
## 6 182 25 1702405999 2003:... http... Nein Nein
## 7 95 25 1699812941 93.13... http... Ja Nein
## 8 355 25 1599676009 2a02:... http... Ja Nein
## 9 570 25 809829273 2003:... http... Nein Nein
## 10 173 25 76734057 134.2... http... Ja Nein
## # ... with 80 more rows, and 73 more variables:
## #   v012 <chr>, v013 <chr>, v014 <chr>,
## #   v015 <chr>, v016 <chr>, v017 <chr>,
## #   v018 <chr>, v019 <chr>, v020 <chr>,
## #   v021 <chr>, v022 <chr>, v023 <dbl>,
## #   v033 <chr>, v034 <chr>, v035 <chr>,
## #   v036 <chr>, v037 <chr>, v038 <chr>, ...
```

Und so weiter für alle Spalten ...

Puh, geht das nicht schlauer?

Ja, geht. Hier ein kleiner Trick:

```
d5 <-
  d4 %>%
    map_df(na_if, "") %>%
    map_df(na_if, "N/A")
```

Mit `map_df()` kann man eine Funktion, hier `na_if()` auf jede Spalte der Tabelle (hier: `d5`) anwenden. Als Ergebnis dieses "Funktions-Mapping" soll wieder eine Tabelle - daher `map_df` zurückgegeben werden.

Mal ein Check: Die Anzahl der fehlenden Werte müsste sich jetzt erhöht haben im Vergleich zur letzten Version des Datensatz, `d4`:

```
sum(is.na(d4))
```

```
## [1] 1806

sum(is.na(d5))

## [1] 1893
```

Hm, gar nicht so viele mehr. Aber grundsätzlich hat es funktioniert :-)

Sie brauchen `map_df()` nicht zu verwenden. Es geht auch ohne. Mit `map_df()` ist es nur komfortabler.

Ad 6.

Die Item-Positionen, wann also die Items der Ratingskala beginnen und wann (an welcher Spaltenposition) sie enden, ist im Fragebogen ersichtlich.

```
d5 %>%
  mutate(v033_r = recode(v033,
    "lehne voll und ganz ab" = -3,
    "lehne ab" = -2,
    "lehne eher ab" = -1,
    "weder/noch" = 0,
    "stimme eher zu" = 1,
    "stimme zu" = 2,
    "stimme voll und ganz zu" = 3,
    .default = NA_real_ # Ansonsten als NA und zwar NA vom Typ "reelle Zahl"
  )) %>%
  select(v001, v033, v033_r) %>%
  head(10)
```

```
## # A tibble: 10 × 3
##   v001 v033          v033_r
##   <dbl> <chr>        <dbl>
## 1   794 stimme voll und ganz zu      3
## 2   146 stimme eher zu              1
## 3   459 <NA>                     NA
## 4   324 stimme eher zu              1
## 5   257 lehne eher ab             -1
## 6   182 stimme zu                 2
## 7    95 stimme eher zu              1
## 8   355 stimme zu                 2
## 9   570 stimme eher zu              1
## 10  173 lehne eher ab             -1
```

Das hat also funktioniert. Aber das jetzt für alle Spalte zu übernehmen, puh, viel zu langweilig. Gibt's da vielleicht einen Trick?

Ja, gibt es.

```
d6 <-
d5 %>%
  mutate(across(
    .cols = c(v033:v056, v087:v104),
    .fns = ~ recode(.,
      "lehne voll und ganz ab" = -3,
      "lehne ab" = -2,
      "lehne eher ab" = -1,
      "weder/noch" = 0,
      "stimme eher zu" = 1,
      "stimme zu" = 2,
      "stimme voll und ganz zu" = 3,
      .default = NA_real_ # Andere Werte als NA (Fehlende Werte) vom Typ "reelle Zahl" kennzeichnen
    )
  ))
```

Mit `across()` kann man eine Funktion (oder mehrere), `.fns`, über mehrere Spalten, `.cols` anwenden, hier wenden wir `recode()` auf alle Spalten der Ratingskala an.

Ad 7.

```
d7 <-
d6 %>%
  rowwise() %>% # Zeilenweise arbeiten im Folgenden
  mutate(
    exp_avg = mean(c_across(v033:v039), na.rm = TRUE),
    neu_avg = mean(c_across(v040:v042), na.rm = TRUE),
    att_avg = mean(c_across(v043:v047), na.rm = TRUE),
    ka_avg = mean(c_across(v048:v053), na.rm = TRUE),
    wom_avg = mean(c_across(v054:v056), na.rm = TRUE),
    innp_avg = mean(c_across(v087:v092), na.rm = TRUE),
    imp_avg = mean(c_across(v093:v096), na.rm = TRUE),
    hedo_avg = mean(c_across(v097:v100), na.rm = TRUE),
    shol_avg = mean(c_across(v101:v104), na.rm = TRUE)
  ) %>%
  relocate(ends_with("_avg"), .after = v008) # wir verschieben alle Spalten, die mit `_avg` enden nach vorne
```

`c_across()` ist wie `c()`. Allerdings funktioniert `c()` leider *nicht* für zeilenweise Operationen. Daher braucht es einen Freund, der das kann, `c_across()`.

Ad 8.

```
library(rio)
export(d7, file = "d7.csv")
export(d7, file = "d7.xlsx")
```

Ad 9.

```
library(psych)

d7 %>%
  select(v087:v092) %>%
  alpha(title = "Skala Innovationsorientierung")

##
## Reliability analysis  Skala Innovationsorientierung
## Call: alpha(x = ., title = "Skala Innovationsorientierung")
##
##   raw_alpha std.alpha G6(smc) average_r S/N ase
##   0.87      0.88      0.88      0.54  7 0.02
##   mean sd median_r
## -0.25 1.3      0.52
##
## lower alpha upper      95% confidence boundaries
## 0.83 0.87 0.91
##
## Reliability if an item is dropped:
##   raw_alpha std.alpha G6(smc) average_r S/N
## v087      0.84      0.84      0.83      0.52 5.4
## v088      0.83      0.83      0.83      0.50 5.0
## v089      0.87      0.87      0.87      0.57 6.5
## v090      0.84      0.84      0.84      0.51 5.1
## v091      0.87      0.87      0.87      0.57 6.5
## v092      0.87      0.87      0.87      0.58 6.9
##   alpha se var.r med.r
## v087 0.026 0.013 0.51
## v088 0.028 0.015 0.51
## v089 0.022 0.024 0.55
## v090 0.028 0.022 0.51
## v091 0.023 0.018 0.51
## v092 0.022 0.018 0.55
##
## Item statistics
##   n raw.r std.r r.cor r.drop mean sd
## v087 61 0.83 0.83 0.82 0.73 0.16 1.8
## v088 61 0.87 0.87 0.87 0.80 0.49 1.5
## v089 61 0.74 0.73 0.65 0.60 -0.75 1.8
## v090 61 0.86 0.86 0.83 0.78 -0.80 1.7
## v091 61 0.71 0.73 0.65 0.60 0.00 1.4
## v092 61 0.70 0.70 0.62 0.57 -0.59 1.6
##
## Non missing response frequency for each item
##   -3 -2 -1 0 1 2 3 miss
## v087 0.07 0.15 0.20 0.08 0.26 0.15 0.10 0.32
## v088 0.02 0.11 0.13 0.18 0.31 0.15 0.10 0.32
## v089 0.13 0.30 0.21 0.15 0.05 0.10 0.07 0.32
## v090 0.21 0.18 0.18 0.20 0.11 0.08 0.03 0.32
## v091 0.07 0.10 0.16 0.25 0.30 0.13 0.00 0.32
## v092 0.15 0.16 0.18 0.25 0.16 0.10 0.00 0.32
```

Ad 10.

```
library(easystats)

d7 %>%
  select(ends_with("_avg")) %>%
  describe_distribution()
```

| ## Variable | Mean | SD | IQR | Range | Skewness | Kurtosis | n | n_Missing |
|-------------|-------|------|------|---------------|----------|-----------|----|-----------|
| ## exp_avg | 0.90 | 1.12 | 1.57 | [-1.86, 3.00] | -0.45 | -0.16 | 76 | 14 |
| ## neu_avg | 1.22 | 1.25 | 1.33 | [-2.67, 3.00] | -0.96 | 0.79 | 70 | 20 |
| ## att_avg | 1.04 | 1.13 | 1.20 | [-2.60, 3.00] | -1.16 | 1.93 | 68 | 22 |
| ## ka_avg | 0.91 | 1.20 | 1.21 | [-2.17, 3.00] | -1.07 | 0.54 | 66 | 24 |
| ## wom_avg | 0.31 | 1.16 | 1.25 | [-2.33, 3.00] | 0.35 | 0.23 | 64 | 26 |
| ## innp_avg | -0.25 | 1.28 | 1.50 | [-2.83, 2.67] | 0.07 | -0.28 | 61 | 29 |
| ## imp_avg | -0.28 | 1.18 | 1.50 | [-3.00, 2.50] | -0.32 | -7.40e-03 | 61 | 29 |
| ## hed_avg | 0.34 | 1.40 | 1.50 | [-3.00, 3.00] | -0.69 | 0.40 | 60 | 30 |
| ## shol_avg | -0.80 | 1.51 | 2.75 | [-3.00, 2.25] | 0.28 | -0.85 | 60 | 30 |

Ad 11.

Es gibt mehrere Wege, das Ziel zu erreichen. Einer sieht so aus.

```
library(flextable)

flex1 <-
  d7 %>%
    select(ends_with("_avg")) %>%
    describe_distribution() %>%
    flextable()

flex1
```

| Variable | Mean | SD | IQR | Min | Max | Skewness | Kurtosis | n | n_Missing |
|----------|------------|----------|----------|-----------|----------|-------------|--------------|----|-----------|
| exp_avg | 0.8966165 | 1.116262 | 1.571429 | -1.857143 | 3.000000 | -0.44852912 | -0.155202928 | 76 | 14 |
| neu_avg | 1.2238095 | 1.248150 | 1.333333 | -2.666667 | 3.000000 | -0.95903822 | 0.790211603 | 70 | 20 |
| att_avg | 1.0441176 | 1.133925 | 1.200000 | -2.600000 | 3.000000 | -1.16348010 | 1.927188688 | 68 | 22 |
| ka_avg | 0.9090909 | 1.202981 | 1.208333 | -2.166667 | 3.000000 | -1.06540680 | 0.539760618 | 66 | 24 |
| wom_avg | 0.3072917 | 1.161257 | 1.250000 | -2.333333 | 3.000000 | 0.35118856 | 0.232676307 | 64 | 26 |
| innp_avg | -0.2486339 | 1.276795 | 1.500000 | -2.833333 | 2.666667 | 0.06946976 | -0.280800172 | 61 | 29 |
| imp_avg | -0.2827869 | 1.177458 | 1.500000 | -3.000000 | 2.500000 | -0.32165605 | -0.007398344 | 61 | 29 |
| hedo_avg | 0.3375000 | 1.401290 | 1.500000 | -3.000000 | 3.000000 | -0.68659230 | 0.397527258 | 60 | 30 |
| sho1_avg | -0.8041667 | 1.506407 | 2.750000 | -3.000000 | 2.250000 | 0.27889917 | -0.847516886 | 60 | 30 |

Vielleicht noch die Anzahl der Dezimalstellen beschneiden:

```
flex1 <-
  d7 %>%
    select(ends_with("_avg")) %>%
    describe_distribution() %>%
    adorn_rounding(digits = 2) %>%
    flextable()

flex1
```

| Variable | Mean | SD | IQR | Min | Max | Skewness | Kurtosis | n | n_Missing |
|----------|-------|------|------|-------|------|----------|----------|----|-----------|
| exp_avg | 0.90 | 1.12 | 1.57 | -1.86 | 3.00 | -0.45 | -0.16 | 76 | 14 |
| neu_avg | 1.22 | 1.25 | 1.33 | -2.67 | 3.00 | -0.96 | 0.79 | 70 | 20 |
| att_avg | 1.04 | 1.13 | 1.20 | -2.60 | 3.00 | -1.16 | 1.93 | 68 | 22 |
| ka_avg | 0.91 | 1.20 | 1.21 | -2.17 | 3.00 | -1.07 | 0.54 | 66 | 24 |
| wom_avg | 0.31 | 1.16 | 1.25 | -2.33 | 3.00 | 0.35 | 0.23 | 64 | 26 |
| innp_avg | -0.25 | 1.28 | 1.50 | -2.83 | 2.67 | 0.07 | -0.28 | 61 | 29 |
| imp_avg | -0.28 | 1.18 | 1.50 | -3.00 | 2.50 | -0.32 | -0.01 | 61 | 29 |
| hedo_avg | 0.34 | 1.40 | 1.50 | -3.00 | 3.00 | -0.69 | 0.40 | 60 | 30 |
| sho1_avg | -0.80 | 1.51 | 2.75 | -3.00 | 2.25 | 0.28 | -0.85 | 60 | 30 |

Und so speichert man als Word-Datei:

```
save_as_docx(flex1, path = "flex1.docx")
```