Assignment 9

Adrian Bracher (Matr. Nr. 01637180)

19.05.2021

Contents

1 Preparing to analyze survey data

Note: Sadly Datacamp fails to provide a lot of necessary data and other prerequisites, therefore I cannot execute many of the code chunks.

```
library(irr)
library(psych)
library(psychometric)
library(dplyr)
library(likert)
library(car)
library(Hmisc)
library(tidyr)
library(corrplot)
library(lavaan)
```

1.1 Measuring expert agreement

In this exercise we learn how to use cor() to compute the correlation in a data frame and then also call agree() to get the agreement in percent between raters.

```
# Print beginning of sme data frame
print(head(sme))

# Correlation matrix of expert ratings
cor(sme)

# Percentage agreement of experts
agree(sme)
```

1.2 Inter-rater reliability

In the code below we learn how to compute Cohen's Kappa.

```
# Load psych package
library(psych)

# Check inter-rater reliability
cohen.kappa(sme)
```

1.3 Content validity

Here we measure content validity using Lawshe's Content Validity Ratio with CVratio().

```
# Calculate the CVR for each unique item in the data frame
cvr_by_item <- lawshe %>%
  group_by(item) %>%
  summarize(CVR = CVratio(NTOTAL = length(unique(expert)),
                      NESSENTIAL = sum(rating == 'Essential')))
```

```r
# See the results
cvr_by_item
```

1.4 Visualizing response frequencies

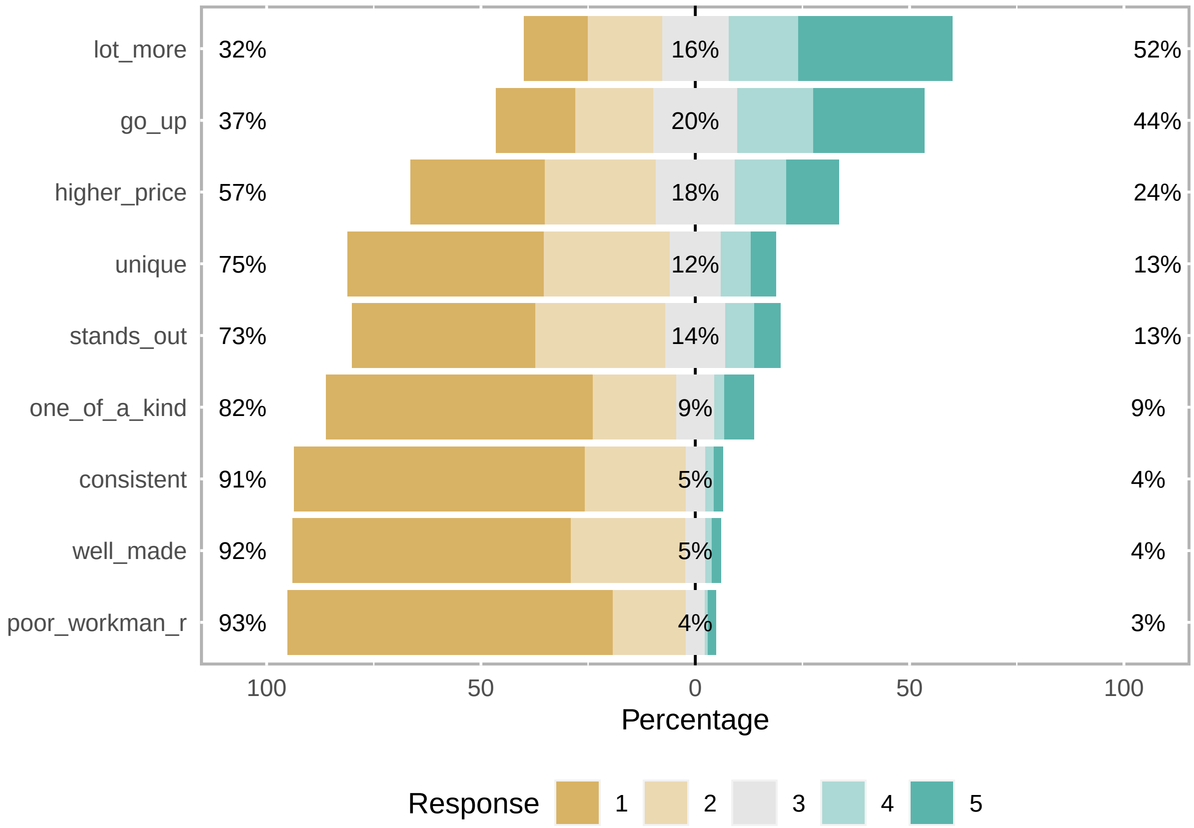
In this exercise we derive summary counts and create a visualization.

```r
brand_rep = read.csv("brandrep-cleansurvey-extraitem.csv")
# Convert items to factor
b_rep_likert <- brand_rep %>%
                mutate_if(is.integer, as.factor)

# Response frequencies - base R
summary(b_rep_likert)
```

```
##  well_made consistent poor_workman_r higher_price lot_more go_up   stands_out
##  1:363     1:379      1:424          1:175        1: 83    1:103   1:239
##  2:149     2:131      2: 95          2:145        2: 97    2:102   2:170
##  3: 27     3: 26      3: 25          3:103        3: 87    3:110   3: 78
##  4:  8     4: 11      4:  4          4: 67        4: 91    4: 99   4: 38
##  5: 12     5: 12      5: 11          5: 69        5:201    5:145   5: 34
##  unique  one_of_a_kind
##  1:256   1:348
##  2:164   2:109
##  3: 67   3: 50
##  4: 39   4: 13
##  5: 33   5: 39
```

```r
# Plot response frequencies
result <- likert(b_rep_likert)
plot(result)
```



1.5 Reverse-coding items

In the code below we "reverse-code" items.

```
# Get response frequencies from psych
response.frequencies(brand_qual)

# Print item descriptions
brand_qual_items

# Reverse code the "opposite" item
brand_qual$tired_r <- recode(brand_qual$tired,
                            "1 = 5; 2 = 4; 4 = 2; 5 = 1")

# Check recoding frequencies
brand_qual %>%
    select(tired, tired_r) %>%
    response.frequencies() %>%
    round(2)
```

1.6 Missing values

Missing values sometimes hint at underlying problems. In this exercise we try to analyze the missing values to see more.

```
# Total number of rows
nrow(missing_lots)
```

```r
# Total number of complete cases
nrow(na.omit(missing_lots))

# Number of incomplete cases by variable
colSums(is.na(missing_lots))

# Hierarchical plot -- what values are missing together?
plot(naclus(missing_lots))
```

1.7 Exploring item correlations

Here we visualize item correlations with corrplot.

```r
# View significance of item correlations
corr.test(brand_qual_9)

# Visualize item correlations -- corrplot
corrplot(cor(brand_qual_9), method = "circle")
```

1.8 Preparing the brand reputation survey

In this exercise we revisit recode and then use select with a "-" to drop a certain column. We then visualize the correlations in the modified data frame.

```r
# Get response frequencies
response.frequencies(brand_rep_9)

# Recode the appropriate item
b_rep_items
brand_rep_9$poor_workman_r <- recode(brand_rep_9$poor_workman,
                              "1 = 5; 2 = 4; 4 = 2; 5 = 1")

# Drop poor_workman from brand_rep_9: brand_rep_9_new
brand_rep_9_new <- select(brand_rep_9, -poor_workman)

# Visualize item correlation
corrplot(cor(brand_rep_9_new), method = "circle")
```

2 Exploratory factor analysis & survey development

2.1 From correlations to factors

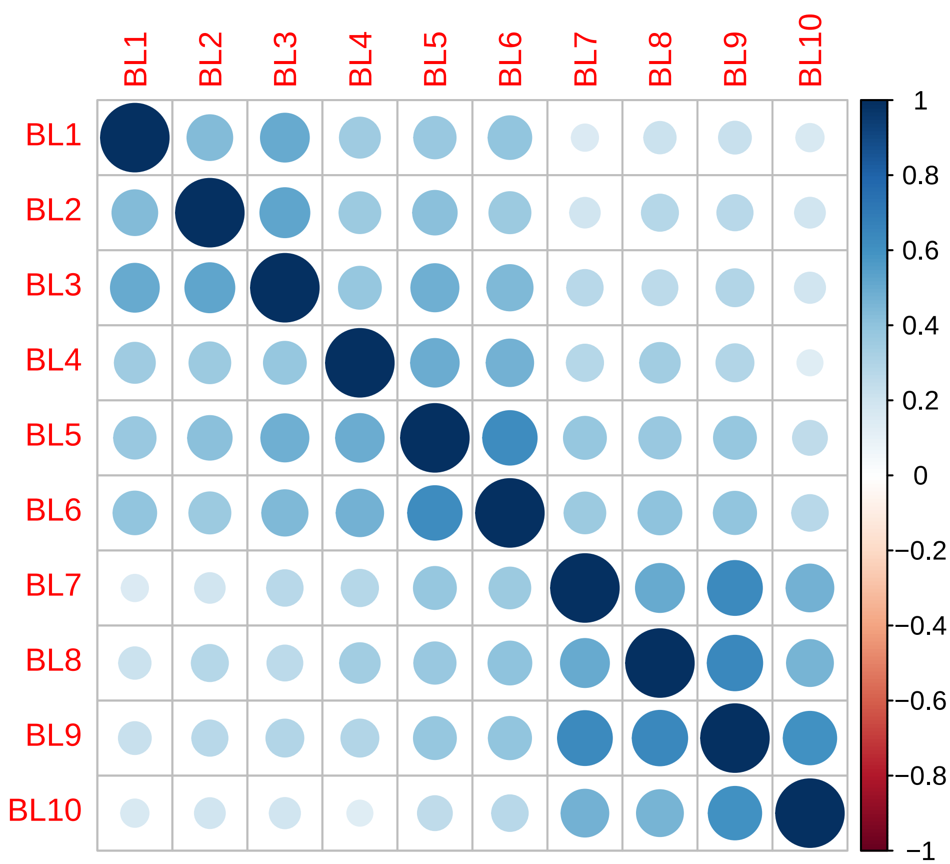
We repeat what we've learned in the previous chapter. Note that corr.test creates a correlation matrix (table). Then, we make a parallel analysis.

```r
b_loyal_10 = read.csv("brandloyalty.csv")
# Print correlation matrix
corr.test(b_loyal_10)
```

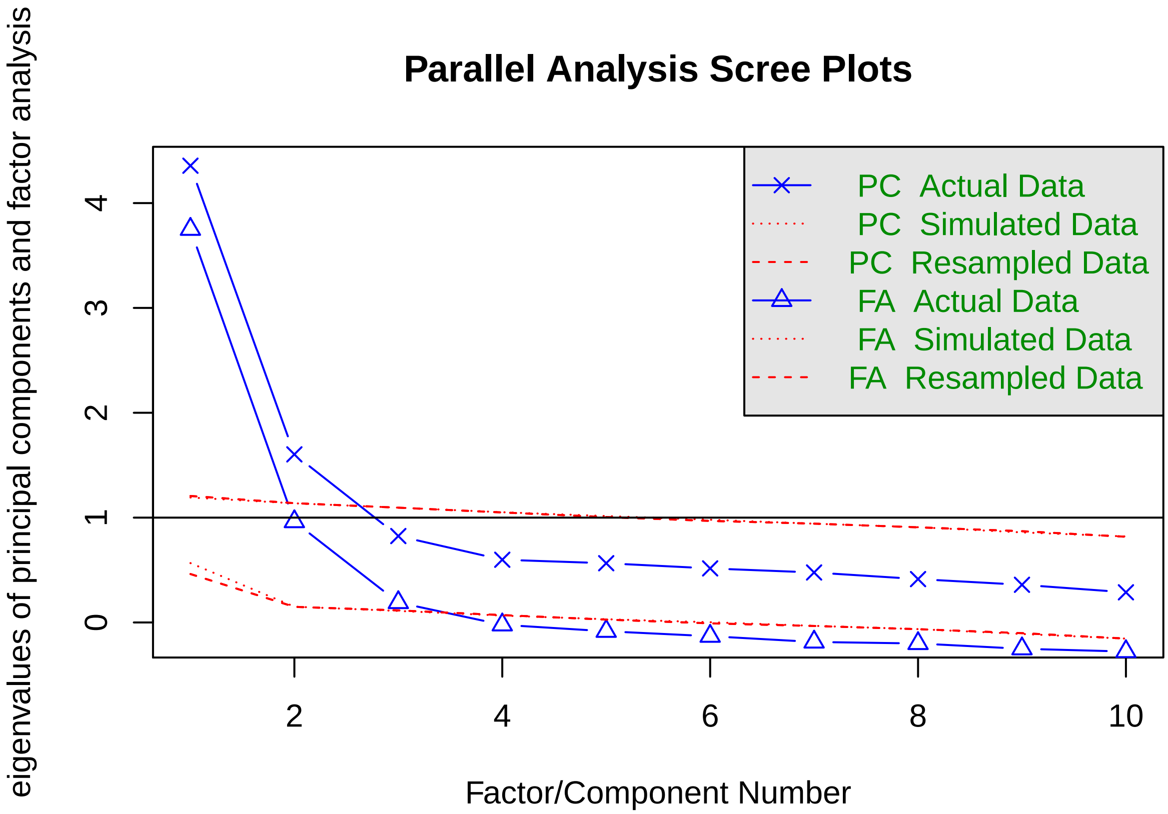
```
## Call:corr.test(x = b_loyal_10)
## Correlation matrix
##       BL1  BL2  BL3  BL4  BL5  BL6  BL7  BL8  BL9 BL10
## BL1  1.00 0.44 0.50 0.35 0.38 0.40 0.15 0.22 0.22 0.16
## BL2  0.44 1.00 0.52 0.36 0.42 0.37 0.19 0.29 0.27 0.20
```

```
## BL3  0.50 0.52 1.00 0.38 0.49 0.45 0.28 0.27 0.30 0.20
## BL4  0.35 0.36 0.38 1.00 0.50 0.47 0.29 0.34 0.30 0.14
## BL5  0.38 0.42 0.49 0.50 1.00 0.63 0.39 0.37 0.39 0.25
## BL6  0.40 0.37 0.45 0.47 0.63 1.00 0.36 0.40 0.39 0.28
## BL7  0.15 0.19 0.28 0.29 0.39 0.36 1.00 0.50 0.64 0.48
## BL8  0.22 0.29 0.27 0.34 0.37 0.40 0.50 1.00 0.65 0.46
## BL9  0.22 0.27 0.30 0.30 0.39 0.39 0.64 0.65 1.00 0.60
## BL10 0.16 0.20 0.20 0.14 0.25 0.28 0.48 0.46 0.60 1.00
## Sample Size
## [1] 639
## Probability values (Entries above the diagonal are adjusted for multiple tests.)
##      BL1 BL2 BL3 BL4 BL5 BL6 BL7 BL8 BL9 BL10
## BL1    0   0   0   0   0   0   0   0   0    0
## BL2    0   0   0   0   0   0   0   0   0    0
## BL3    0   0   0   0   0   0   0   0   0    0
## BL4    0   0   0   0   0   0   0   0   0    0
## BL5    0   0   0   0   0   0   0   0   0    0
## BL6    0   0   0   0   0   0   0   0   0    0
## BL7    0   0   0   0   0   0   0   0   0    0
## BL8    0   0   0   0   0   0   0   0   0    0
## BL9    0   0   0   0   0   0   0   0   0    0
## BL10   0   0   0   0   0   0   0   0   0    0
##
##  To see confidence intervals of the correlations, print with the short=FALSE option
```

```r
# Visualize b_loyal_10 correlation matrix
corrplot(cor(b_loyal_10))
```



```
# Parallel analysis
fa.parallel(b_loyal_10)
```

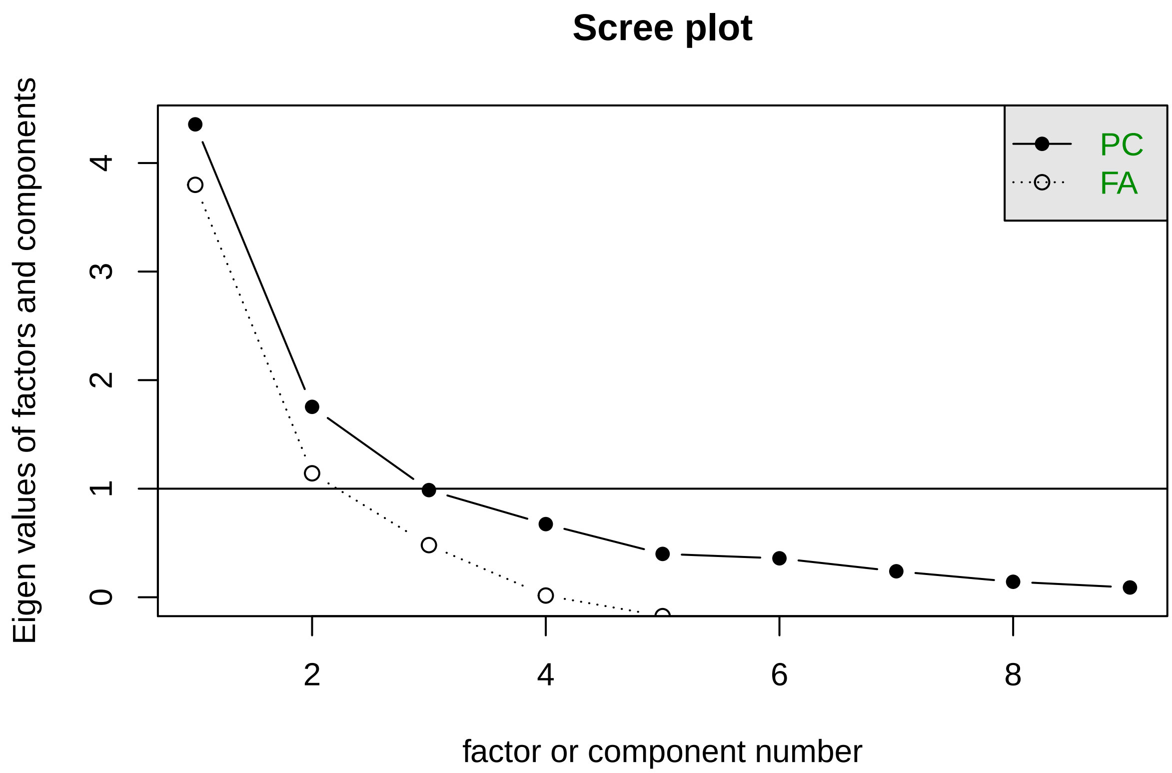



```
## Parallel analysis suggests that the number of factors =  3  and the number of components =  2
```

2.2 Building your first EFA

In this exercise we build an EFA using the psych package. We also plot a scree plot of the model.

```
brand_rep_9 = read.csv("brandrep-cleansurvey-extraitem.csv")
# Scree plot
scree(brand_rep_9)
```




```r
# Conduct three-factor EFA
brand_rep_9_EFA <- fa(brand_rep_9,  nfactors = 3)
```

```
## Loading required namespace: GPArotation
```

```r
# Print output of EFA
names(brand_rep_9_EFA)
```

```
##  [1] "residual"      "dof"           "chi"           "nh"
##  [5] "rms"           "EPVAL"         "crms"          "EBIC"
##  [9] "ESABIC"        "fit"           "fit.off"       "sd"
## [13] "factors"       "complexity"    "n.obs"         "objective"
## [17] "criteria"      "STATISTIC"     "PVAL"          "Call"
## [21] "null.model"    "null.dof"      "null.chisq"    "TLI"
## [25] "RMSEA"         "BIC"           "SABIC"         "r.scores"
## [29] "R2"            "valid"         "score.cor"     "weights"
## [33] "rotation"      "communality"   "communalities" "uniquenesses"
## [37] "values"        "e.values"      "loadings"      "model"
## [41] "fm"            "rot.mat"       "Phi"           "Structure"
## [45] "method"        "scores"        "R2.scores"     "r"
## [49] "np.obs"        "fn"            "Vaccounted"
```

2.3 EFA: How many factors?

Here we look into different number of factors and then print the loadings for a two/four factor EFA.

```r
brand_rep_9_EFA_3 = brand_rep_9_EFA
# Summarize results of three-factor EFA
```

```
summary(brand_rep_9_EFA_3)

# Build and print loadings for a two-factor EFA
brand_rep_9_EFA_2 = fa(brand_rep_9, nfactors = 2)
brand_rep_9_EFA_2$loadings


# Build and print loadings for a four-factor EFA
brand_rep_9_EFA_4 = fa(brand_rep_9, nfactors = 4)
brand_rep_9_EFA_4$loadings
```

2.4 Refining the brand reputation survey

We extract further information like eigenvalues and factor score correlations.

```
# Three factor EFA - brand_rep_9
brand_rep_9_EFA_3 <- fa(brand_rep_9, nfactors = 3)

# Eigenvalues
brand_rep_9_EFA_3$e.values
```

```
## [1] 4.35629549 1.75381015 0.98701607 0.67377072 0.39901205 0.35865598 0.23915591
## [8] 0.14238807 0.08989556
```

```
# Factor score correlations
brand_rep_9_EFA_3$score.cor
```

```
##           [,1]      [,2]      [,3]
## [1,] 1.0000000 0.3687866 0.4733475
## [2,] 0.3687866 1.0000000 0.4977679
## [3,] 0.4733475 0.4977679 1.0000000
```

```
# Factor loadings
brand_rep_9_EFA_3$loadings
```

```
##
## Loadings:
##                MR2    MR1    MR3
## well_made       0.896
## consistent      0.947
## poor_workman_r  0.739
## higher_price    0.127  0.632  0.149
## lot_more               0.850
## go_up                  0.896
## stands_out                    1.008
## unique                        0.896
## one_of_a_kind   0.309  0.295  0.115
##
##                  MR2   MR1   MR3
## SS loadings    2.361 2.012 1.858
## Proportion Var 0.262 0.224 0.206
## Cumulative Var 0.262 0.486 0.692
```

2.5 Comparing EFA model fits

In this exercise we repeat the previously learned usage of fa, select and their variables.

```r
# Create brand_rep_8 data frame
brand_rep_8 <- select(brand_rep_9, -one_of_a_kind)

# Create three-factor EFA
brand_rep_8_EFA_3 <- fa(brand_rep_8, nfactors=3)

# Factor loadings
brand_rep_8_EFA_3$loadings
```

```
##
## Loadings:
##                 MR2    MR3    MR1
## well_made       0.887
## consistent      0.958
## poor_workman_r  0.735
## higher_price    0.120  0.596  0.170
## lot_more               0.845
## go_up                  0.918
## stands_out                    0.990
## unique                        0.916
##
##                   MR2   MR3   MR1
## SS loadings    2.261 1.915 1.850
## Proportion Var 0.283 0.239 0.231
## Cumulative Var 0.283 0.522 0.753
```

```r
# Factor correlations -- 9 versus 8 item model
brand_rep_9_EFA_3$score.cor
```

```
##           [,1]      [,2]      [,3]
## [1,] 1.0000000 0.3687866 0.4733475
## [2,] 0.3687866 1.0000000 0.4977679
## [3,] 0.4733475 0.4977679 1.0000000
```

```r
brand_rep_8_EFA_3$score.cor
```

```
##           [,1]      [,2]      [,3]
## [1,] 1.0000000 0.2826536 0.4247302
## [2,] 0.2826536 1.0000000 0.4977679
## [3,] 0.4247302 0.4977679 1.0000000
```

2.6 EFA model iteration

Again, in this exercise we use previously learned functions to compute loadings and a scree plot.

```r
# Three factor EFA loadings
brand_rep_8_EFA_3$loadings
```

```
##
## Loadings:
##                 MR2    MR3    MR1
## well_made       0.887
## consistent      0.958
## poor_workman_r  0.735
## higher_price    0.120  0.596  0.170
## lot_more               0.845
```

```
## go_up                 0.918
## stands_out                   0.990
## unique                       0.916
##
##                  MR2   MR3   MR1
## SS loadings    2.261 1.915 1.850
## Proportion Var 0.283 0.239 0.231
## Cumulative Var 0.283 0.522 0.753
```

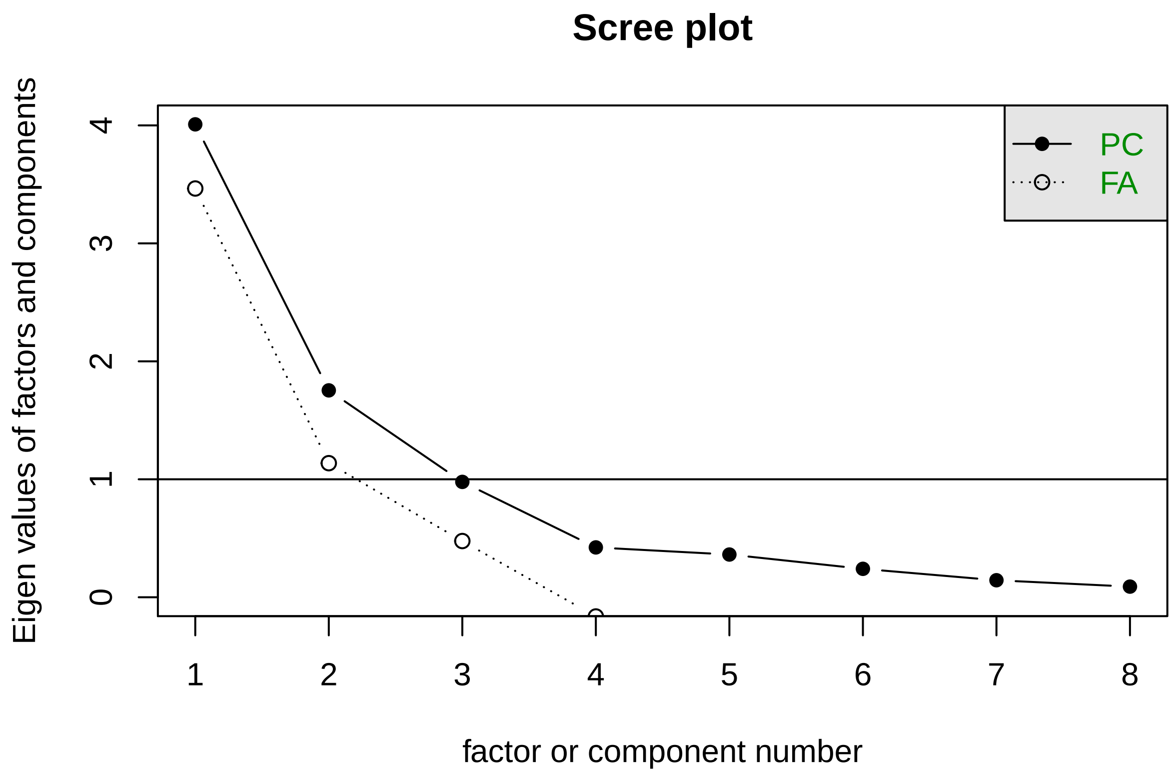
```r
# Two factor EFA & loadings
brand_rep_8_EFA_2 <- fa(brand_rep_8, nfactors = 2)
brand_rep_8_EFA_2$loadings
```

```
##
## Loadings:
##                MR1    MR2
## well_made              0.878
## consistent             0.937
## poor_workman_r         0.725
## higher_price    0.724
## lot_more        0.782 -0.124
## go_up           0.859
## stands_out      0.595  0.293
## unique          0.546  0.315
##
##                  MR1   MR2
## SS loadings    2.529 2.392
## Proportion Var 0.316 0.299
## Cumulative Var 0.316 0.615
```

```r
# Four factor EFA & loadings
brand_rep_8_EFA_4 <- fa(brand_rep_8, nfactors = 4)
brand_rep_8_EFA_4$loadings
```

```
##
## Loadings:
##                MR2    MR3    MR1    MR4
## well_made       0.831
## consistent      1.011                -0.126
## poor_workman_r  0.699                 0.253
## higher_price    0.129  0.601  0.168
## lot_more               0.850
## go_up                  0.913
## stands_out                    0.992
## unique                        0.915
##
##                  MR2   MR3   MR1   MR4
## SS loadings    2.220 1.920 1.854 0.097
## Proportion Var 0.278 0.240 0.232 0.012
## Cumulative Var 0.278 0.518 0.749 0.761
```

```r
# Scree plot of brand_rep_8
scree(brand_rep_8)
```



2.7 Measuring coefficient (Cronbach's) alpha

Here we use the function alpha() to calculate Cronbach's coefficient.

```
brand_rep_9 = read.csv("brandrep-cleansurvey-extraitem.csv")
# Standardized coefficient alpha
psych::alpha(brand_rep_9)$total$std.alpha
```

```
## [1] 0.8648896
```

```
# 3-factor EFA
brand_rep_9_EFA_3 <- fa(brand_rep_9, nfactors = 3)
brand_rep_9_EFA_3$loadings
```

```
##
## Loadings:
##                 MR2    MR1    MR3
## well_made       0.896
## consistent      0.947
## poor_workman_r  0.739
## higher_price    0.127  0.632  0.149
## lot_more               0.850
## go_up                  0.896
## stands_out                    1.008
## unique                        0.896
## one_of_a_kind   0.309  0.295  0.115
##
```

```
##                    MR2   MR1   MR3
## SS loadings    2.361 2.012 1.858
## Proportion Var 0.262 0.224 0.206
## Cumulative Var 0.262 0.486 0.692
```

```
# Standardized coefficient alpha - refined scale
psych::alpha(brand_rep_8)$total$std.alpha
```

```
## [1] 0.8557356
```

2.8 Coefficient alpha by dimension

We use the previously learned functions to check the standardized alpha for each of the three dimensions "Product Quality", "Willingness to Pay" and "Product Differentiation".

```
# Get names of survey items
names(brand_rep_8)
```

```
## [1] "well_made"      "consistent"     "poor_workman_r" "higher_price"
## [5] "lot_more"       "go_up"          "stands_out"     "unique"
```

```
# Create new data frames for each of three dimensions
p_quality <- brand_rep_8[1:3]
p_willingness <- brand_rep_8[4:6]
p_difference <- brand_rep_8[7:8]

# Check the standardized alpha for each dimension
psych::alpha(p_quality)$total$std.alpha
```

```
## [1] 0.8918025
```

```
psych::alpha(p_willingness)$total$std.alpha
```

```
## [1] 0.8517566
```

```
psych::alpha(p_difference)$total$std.alpha
```

```
## [1] 0.951472
```

```
psych::alpha(brand_rep_8)$total$std.alpha
```

```
## [1] 0.8557356
```

2.9 Split-half reliability

In this exercise we use the function splitHalf() to compute the split-half reliability of brand_rep_8.

```
# Get split-half reliability
splitHalf(brand_rep_8)
```

```
## Split half reliabilities
## Call: splitHalf(r = brand_rep_8)
##
## Maximum split half reliability (lambda 4) =  0.93
## Guttman lambda 6                          =  0.92
## Average split half reliability            =  0.86
## Guttman lambda 3 (alpha)                  =  0.86
## Guttman lambda 2                          =  0.87
## Minimum split half reliability  (beta)    =  0.66
## Average interitem r =  0.43  with median =  0.4
```

```r
# Get averages of even and odd row scores
even_items <- colMeans(brand_rep_8[,c(FALSE,TRUE)])
odd_items <- colMeans(brand_rep_8[,c(TRUE,FALSE)])

# Correlate scores from even and odd items
cor(even_items, odd_items)
```

```
## [1] 0.7441724
```

```r
# Get Cronbach's alpha
psych::alpha(brand_rep_8)
```

```
##
## Reliability analysis
## Call: psych::alpha(x = brand_rep_8)
##
##   raw_alpha std.alpha G6(smc) average_r S/N    ase mean   sd median_r
##       0.85      0.86    0.92      0.43 5.9 0.0096  2.2 0.81      0.4
##
##  lower alpha upper     95% confidence boundaries
## 0.83 0.85 0.87
##
##  Reliability if an item is dropped:
##                raw_alpha std.alpha G6(smc) average_r S/N alpha se var.r med.r
## well_made           0.83      0.83    0.89      0.42 5.0   0.0102 0.042  0.39
## consistent          0.84      0.84    0.89      0.42 5.1   0.0100 0.039  0.41
## poor_workman_r      0.85      0.85    0.92      0.45 5.7   0.0098 0.041  0.41
## higher_price        0.82      0.83    0.91      0.42 5.0   0.0120 0.052  0.39
## lot_more            0.84      0.85    0.91      0.45 5.7   0.0105 0.041  0.41
## go_up               0.82      0.84    0.90      0.43 5.3   0.0115 0.044  0.39
## stands_out          0.81      0.83    0.88      0.41 4.8   0.0116 0.045  0.34
## unique              0.82      0.83    0.88      0.41 4.9   0.0113 0.046  0.38
##
##  Item statistics
##                  n raw.r std.r r.cor r.drop mean   sd
## well_made      559  0.64  0.73  0.72   0.56  1.5 0.83
## consistent     559  0.62  0.71  0.70   0.52  1.5 0.85
## poor_workman_r 559  0.52  0.62  0.56   0.43  1.4 0.77
## higher_price   559  0.78  0.73  0.68   0.68  2.5 1.36
## lot_more       559  0.71  0.62  0.57   0.56  3.4 1.48
## go_up          559  0.76  0.69  0.65   0.64  3.1 1.45
## stands_out     559  0.78  0.77  0.78   0.69  2.0 1.18
## unique         559  0.76  0.76  0.76   0.66  2.0 1.18
##
## Non missing response frequency for each item
##                   1    2    3    4    5 miss
## well_made      0.65 0.27 0.05 0.01 0.02    0
## consistent     0.68 0.23 0.05 0.02 0.02    0
## poor_workman_r 0.76 0.17 0.04 0.01 0.02    0
## higher_price   0.31 0.26 0.18 0.12 0.12    0
## lot_more       0.15 0.17 0.16 0.16 0.36    0
## go_up          0.18 0.18 0.20 0.18 0.26    0
## stands_out     0.43 0.30 0.14 0.07 0.06    0
## unique         0.46 0.29 0.12 0.07 0.06    0
```

2.10 Measuring loyalty

In this exercise we learn by repetition to compute EFAs.

```r
# 3 factor EFA
b_loyal_10_EFA_3 <- fa(b_loyal_10, nfactors = 3)

# Factor loadings, eigenvalues and factor score correlations
b_loyal_10_EFA_3$loadings
```

```
##
## Loadings:
##      MR2    MR1    MR3
## BL1                 0.643
## BL2                 0.682
## BL3                 0.700
## BL4          0.545  0.124
## BL5          0.772
## BL6          0.712
## BL7   0.643  0.207 -0.114
## BL8   0.619  0.165
## BL9   0.903
## BL10  0.718 -0.134
##
##                  MR2   MR1   MR3
## SS loadings    2.134 1.495 1.406
## Proportion Var 0.213 0.149 0.141
## Cumulative Var 0.213 0.363 0.503
```

```r
b_loyal_10_EFA_3$e.values
```

```
##  [1] 4.3564537 1.6031839 0.8242739 0.5982539 0.5649528 0.5155507 0.4767388
##  [8] 0.4136564 0.3594158 0.2875202
```

```r
b_loyal_10_EFA_3$score.cor
```

```
##           [,1]      [,2]      [,3]
## [1,] 1.0000000 0.4795981 0.3440793
## [2,] 0.4795981 1.0000000 0.5924107
## [3,] 0.3440793 0.5924107 1.0000000
```

```r
# 2 factor EFA
b_loyal_10_EFA_2 <- fa(b_loyal_10, nfactors = 2)

# Factor loadings, eigenvalues and factor score correlations
b_loyal_10_EFA_2$loadings
```

```
##
## Loadings:
##      MR1    MR2
## BL1   0.673 -0.111
## BL2   0.654
## BL3   0.746
## BL4   0.581
## BL5   0.661  0.129
## BL6   0.616  0.161
## BL7          0.701
```

```
## BL8   0.116  0.657
## BL9          0.901
## BL10         0.688
##
##                  MR1   MR2
## SS loadings    2.610 2.270
## Proportion Var 0.261 0.227
## Cumulative Var 0.261 0.488
```

```
b_loyal_10_EFA_2$e.values
```

```
##  [1] 4.3564537 1.6031839 0.8242739 0.5982539 0.5649528 0.5155507 0.4767388
##  [8] 0.4136564 0.3594158 0.2875202
```

```
b_loyal_10_EFA_2$score.cor
```

```
##           [,1]      [,2]
## [1,] 1.0000000 0.4623472
## [2,] 0.4623472 1.0000000
```

3 Confirmatory factor analysis & construct validation

3.1 Factor loadings in EFA & CFA

We learn how we can print CFA loadings with inspect() and also plot CFA diagrams via semPaths().

```
# Factor loadings -- EFA
brand_rep_EFA$loadings

# Factor loadings -- CFA
inspect(brand_rep_CFA, what = "std")$lambda

# Plot diagram -- EFA
fa.diagram(brand_rep_EFA)

# Plot diagram -- CFA
semPaths(brand_rep_CFA)
```

3.2 Building a CFA in lavaan

Here we use the package lavaan to build a CFA and summarise the results. CFA allows the researcher to test the hypothesis that a relationship between manifest and latent variables exist. Unlike EFA, we define the relationships between variables and we have a specific theory to test.

```
b_loyal_10 = read.csv("brandloyalty.csv")
# Rename items based on proposed dimensions
colnames(b_loyal_10) <- c("ID1", "ID2", "ID3",
                          "PV1", "PV2", "PV3",
                          "BT1", "BT2", "BT3", "BT4")

# Define the model
b_loyal_cfa_model <- 'ID =~ ID1 + ID2 + ID3
                    PV =~ PV1 + PV2 + PV3
                    BT =~ BT1 + BT2 + BT3 + BT4'

# Fit the model to the data
```

```r
b_loyal_cfa <- cfa(model = b_loyal_cfa_model, data = b_loyal_10)

# Check the summary statistics -- include fit measures and standardized estimates
summary(b_loyal_cfa, fit.measures = TRUE,
        standardized = TRUE)
```

```
## lavaan 0.6-8 ended normally after 33 iterations
##
##   Estimator                                         ML
##   Optimization method                           NLMINB
##   Number of model parameters                        23
##
##   Number of observations                           639
##
## Model Test User Model:
##
##   Test statistic                                63.953
##   Degrees of freedom                                32
##   P-value (Chi-square)                           0.001
##
## Model Test Baseline Model:
##
##   Test statistic                              2485.786
##   Degrees of freedom                                45
##   P-value                                        0.000
##
## User Model versus Baseline Model:
##
##   Comparative Fit Index (CFI)                    0.987
##   Tucker-Lewis Index (TLI)                       0.982
##
## Loglikelihood and Information Criteria:
##
##   Loglikelihood user model (H0)              -7214.586
##   Loglikelihood unrestricted model (H1)      -7182.610
##
##   Akaike (AIC)                               14475.173
##   Bayesian (BIC)                             14577.751
##   Sample-size adjusted Bayesian (BIC)        14504.727
##
## Root Mean Square Error of Approximation:
##
##   RMSEA                                          0.040
##   90 Percent confidence interval - lower         0.025
##   90 Percent confidence interval - upper         0.054
##   P-value RMSEA <= 0.05                          0.885
##
## Standardized Root Mean Square Residual:
##
##   SRMR                                           0.030
##
## Parameter Estimates:
##
##   Standard errors                             Standard
```

```
##   Information                                 Expected
##   Information saturated (h1) model          Structured
##
## Latent Variables:
##                    Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
##   ID =~
##     ID1               1.000                               0.450    0.646
##     ID2               1.186    0.090   13.235    0.000    0.534    0.675
##     ID3               1.445    0.102   14.209    0.000    0.651    0.778
##   PV =~
##     PV1               1.000                               0.555    0.626
##     PV2               1.311    0.087   15.012    0.000    0.728    0.800
##     PV3               1.340    0.091   14.765    0.000    0.744    0.772
##   BT =~
##     BT1               1.000                               0.723    0.717
##     BT2               1.106    0.065   17.064    0.000    0.799    0.729
##     BT3               1.174    0.060   19.529    0.000    0.848    0.888
##     BT4               0.886    0.057   15.507    0.000    0.641    0.660
##
## Covariances:
##                    Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
##   ID ~~
##     PV                0.192    0.020    9.511    0.000    0.768    0.768
##     BT                0.142    0.019    7.423    0.000    0.436    0.436
##   PV ~~
##     BT                0.234    0.026    8.966    0.000    0.583    0.583
##
## Variances:
##                    Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
##    .ID1               0.282    0.019   14.506    0.000    0.282    0.582
##    .ID2               0.341    0.024   13.918    0.000    0.341    0.545
##    .ID3               0.277    0.026   10.637    0.000    0.277    0.395
##    .PV1               0.479    0.031   15.569    0.000    0.479    0.608
##    .PV2               0.299    0.026   11.301    0.000    0.299    0.361
##    .PV3               0.374    0.030   12.369    0.000    0.374    0.403
##    .BT1               0.492    0.033   14.907    0.000    0.492    0.485
##    .BT2               0.563    0.038   14.670    0.000    0.563    0.468
##    .BT3               0.192    0.024    8.013    0.000    0.192    0.211
##    .BT4               0.530    0.034   15.777    0.000    0.530    0.564
##     ID                0.203    0.025    8.166    0.000    1.000    1.000
##     PV                0.308    0.038    8.097    0.000    1.000    1.000
##     BT                0.522    0.053    9.879    0.000    1.000    1.000
```

3.3 A not-so-good CFA

In the code below we create two_dimensions ODD and EVEN according to specification and then fit a cfa model to them.

```r
# Two dimensions: odd- versus even-numbered items
c_sat_bad_model <- 'ODD =~ CS1 + CS3 + CS5 + CS7 + CS9
                EVEN =~ CS2 + CS4 + CS6 + CS8 + CS10'

# Fit the model to the data
c_sat_bad_CFA <- cfa(model = c_sat_bad_model, data = c_sat)
```

```r
# Summary measures
summary(c_sat_bad_CFA, fit.measures = TRUE, standardized = TRUE)
```

3.4 Adjusting for non-normality

Multivariate normal distribution is an assumption under CFA. In this exercise we learn how to check that.

```r
# Mardia's test for multivarite normality
mardia(c_sat_50)

# Fit model to the data using robust standard errors
c_sat_cfa_mlr <- cfa(model = c_sat_model,
                     data = c_sat_50,
                     estimator = "MLR")

# Summary including standardized estimates and fit measures
summary(c_sat_cfa_mlr, fit.measures = TRUE, standardized = TRUE)
```

3.5 Comparing models using absolute fit measures

In this exercise we compare models using lavaan and determine which model fits better.

```r
# Fit the models to the data
c_sat_cfa_a <- cfa(model = c_sat_model_a, data = c_sat)
c_sat_cfa_b <- cfa(model = c_sat_model_b, data = c_sat)

# Print the model definitions
cat(c_sat_model_a)
cat(c_sat_model_b)

# Calculate the desired model fit statistics
fitMeasures(c_sat_cfa_a, fit.measures = c("cfi", "tli"))
fitMeasures(c_sat_cfa_b, fit.measures = c("cfi", "tli"))
```

3.6 Comparing CFA models using ANOVA

In this exercise we use analysis of variance (ANOVA) to compare nested models.

```r
# View current c_sat model
cat(c_sat_model_a)

# Add EU1 to the CSU factor
c_sat_model_b <- 'CSU =~ CSU1 + CSU2 + CSU3 + CSU4 + EU1
                EU =~ EU1 + EU2 + EU3
                PS =~ PS1 + PS2 + PS3'

# Fit Models A and B to the data
c_sat_cfa_a <- cfa(model = c_sat_model_a, data = c_sat)
c_sat_cfa_b <- cfa(model = c_sat_model_b, data = c_sat)

# Compare the nested models
anova(c_sat_cfa_a, c_sat_cfa_b)
```

3.7 Group CFA

In the code below we learn how to add groups to a cfa model.

```
# Fit the model to the data
c_sat_cfa <- cfa(model = c_sat_model, data = c_sat_group, group = "COUNTRY")

# Summarize results -- include fit measures and standardized estimates
summary(c_sat_cfa, fit.measures = TRUE, standardized = TRUE)

# Get average estimate for both groups
standardized_solution <- standardizedSolution(c_sat_cfa)
standardized_solution %>%
    filter(op == "=~") %>%
    group_by(group) %>%
    summarize(mean(est.std))
```

3.8 Construct validity & model fit

In the code below we use different reliability measures.

```
# Fit three-factor CFA
c_sat_cfa_3 <- cfa(model = c_sat_cfa_model_3, data = c_sat)

# Inspect key fit measures - three-factor CFA
fitMeasures(c_sat_cfa_3, fit.measures = c("cfi","tli","rmsea"))

# Fit two-factor CFA
c_sat_cfa_2 <- cfa(model = c_sat_cfa_model_2, data = c_sat)

# Inspect key fit measures - two-factor CFA
fitMeasures(c_sat_cfa_2, fit.measures = c("cfi","tli","rmsea"))

# Compare measures of construct validity for three- versus two-factor models
reliability(c_sat_cfa_3)
reliability(c_sat_cfa_2)
```

3.9 Construct validity & reliability

In this exercise, we find out how we can measure validity and realiability.

```
# Print CFA model
cat(brand_rep_CFA_model)

# semTools reliability measures
reliability(brand_rep_CFA)

# psych coefficient alpha measure
alpha(brand_rep_9)$total$std.alpha
```

3.10 Deeper into AVE & CR

In this exercise we compare dplyr methods to semTools methods and compute reliability scores.

```
# Store F1 estimates as object loadings
loadings <- standardizedSolution(c_sat_cfa) %>%
```

```r
 filter(op == "=~", lhs == "F1") %>% select(est.std)

# Composite reliability -- the squared sum of all loadings divided by that same figure plus the sum of
com_rel <- sum(loadings) ^ 2 / ((sum(loadings)^ 2)  + sum(1 - loadings ^ 2))
com_rel

# Average variance extracted -- sum of all factor squares divided by the number of items
avg_var <- sum(loadings ^ 2) / nrow(loadings)
avg_var

# Compare versus semTools
reliability(c_sat_cfa)
```

3.11 CFA of the brand reputation survey

In this exercise we build a lavaan model and then print the summary and also construct validity with the reliability() function.

```r
# Print brand_rep_factors
brand_rep_factors

# Build model for lavaan
brand_rep_8_cfa_model <- "QUAL =~ consistent + well_made + poor_workman_r
                PRICE =~ go_up + lot_more + higher_price
                UNIQUE =~ stands_out + unique"
# Summarize results with fit measures and standardized estimates
summary(brand_rep_8_CFA, standardized = T, fit.measures = T)

# Construct validity
reliability(brand_rep_8_CFA)
```

4 Criterion validity & replication

4.1 Preparing a scaled data frame

In this exercise we learn how to use scale() for scaling and then describe() to print descriptive statistics.

```r
# Check if brand_rep and brand_rep_spend have the same number of rows
same_rows <- nrow(brand_rep) == nrow(brand_rep_spend)
same_rows

# Append spend column to brand_rep
brand_rep <- cbind(brand_rep, brand_rep_spend)

# Scale the data
b_rep_scale <- scale(brand_rep)

# Compare descriptive statistics of raw and scaled data frames using psych
describe(brand_rep)
describe(b_rep_scale)
```

4.2 Plotting and analyzing a concurrent validity model

We create a standardized model and establish concurrent validity. Then we print the standardized covariances with standardizedSolution() and plot the result with semPaths().

```
# Correlate F1, F2 and F3 to spend_f, the 'latentized' spend
brand_rep_model <- 'F1 =~ well_made + consistent + poor_workman_r
                    F2 =~ higher_price + lot_more + go_up
                    F3 =~ stands_out + unique
                    spend_f =~ spend
                    spend_f ~~ F1 + F2 + F3'

# Fit the model to the data -- sem()
brand_rep_cv <- sem(data = brand_rep_scaled, model = brand_rep_model)

# Print the standardized covariances b/w spend_f and other factors
standardizedSolution(brand_rep_cv) %>% filter(rhs == "spend_f")

# Plot the model with standardized estimate labels
semPaths(brand_rep_cv, whatLabels = "est.std", edge.label.cex = .8)
```

4.3 Concurrent validity & Likert-style items

In this exercise we learn by repetition.

```
# Bind & scale the variables
c_sat_rec_scale <- cbind(c_sat, c_sat_recommend) %>% scale()

# Define the model - Rec_f covaries with F1, F2, F3
c_sat_rec_model <- 'F1 =~ CS1 + CS2 + CS3 + CS4
F2 =~ CS5 + CS6 + CS7
F3 =~ CS8 + CS9 + CS10
Rec_f =~ Rec_1
Rec_f ~~ F1 + F2 + F3'

# Fit the model to the data
c_sat_rec_sem <- sem(model = c_sat_rec_model, data = c_sat_rec)

# Look up standardized covariances
standardizedSolution(c_sat_rec_sem) %>% filter(rhs == "Rec_f")
```

4.4 Statistical significance & r-square

In this exercise we practice fitting models, summarizing results and plotting.

```
# Define the model
b_q_model <- 'HIP =~ trendy + latest + tired_r
            VALUE =~ happy_pay + reason_price + good_deal
            PERFORM =~ strong_perform + leader + serious
            spend ~ HIP + VALUE + PERFORM'

# Fit the model to the data
b_q_pv <- sem(data = b_q_scale, model = b_q_model)

# Check fit, r-square, standardized estimates
summary(b_q_pv, standardized = T, fit.measures = T, rsquare = T)
```

```r
# Plot the model -- rotate from left to right
semPaths(b_q_pv, rotation = 2, whatLabels = "est.std", edge.label.cex = .8)
```

4.5 Prediction & causation

In the following we exercise the use of standardizedSolution(), inspect() and semPaths().

```r
# Plot the new model
semPaths(brand_rep_sem, rotation = 2)

# Get the coefficient information
standardizedSolution(brand_rep_sem) %>% filter(op == "~")

# Get the r-squared
r_squared <- inspect(brand_rep_sem, 'r2')["F2"]
r_squared
```

4.6 Exploring factor scores

Factor scores represent individual respondents' standing on a latent factor. We learn how we can combine predict() with as.data.frame() to compute such a factor score.

```r
# Compute factor scores in lavaan -- store as data frame
brand_rep_scores <- as.data.frame(predict(brand_rep_cfa))

# Descriptive statistics of our factor scores
describe(brand_rep_scores)

# Plot histograms for each variable
multi.hist(brand_rep_scores)
# Are they normally distributed? Check using map()
map(brand_rep_scores, shapiro.test)
```

4.7 Factor scores & regression

We learn how to combine lm with summary and round as well as inspect to compare results.

```r
# Linear regression of standardized spending and factor scores
bq_fs_reg <- lm(spend ~ F1 + F2 + F3, data = bq_fs_spend)

# Summarize results, round estimates
rounded_summary <- round(summary(bq_fs_reg)$coef, 3)
rounded_summary

# Summarize the results of CFA model
summary(brand_qual_pv)

# Compare the r-squared of each
inspect_rsq <- inspect(brand_qual_pv, "r2")["spend"]
inspect_rsq
summary(bq_fs_reg)$r.squared
```

4.8 Test-retest reliability

describeBy() returns descriptive statistics like describe(), but grouped. testRetest() is used to measure the test-retest reliability of the survey.

```
# Descriptive statistics grouped by 'time'
describeBy(brand_rep_t1_t2, "time")

# Test retest: time == 1 versus time == 2 by id = "id"
brand_rep_test_retest <- testRetest(t1 = filter(brand_rep_t1_t2, time == 1),
                                    t2 = filter(brand_rep_t1_t2, time == 2),
                                    id = "id")

# Access correlation of scales scores between t1 and t2
brand_rep_test_retest$r12
```

4.9 CFA, EFA & replication

In this exercise we practice previously acquired knowledge.

```
# Split data into odd and even halves
brand_rep_efa_data <- brand_rep[c(TRUE,FALSE),]
brand_rep_cfa_data <- brand_rep[c(FALSE,TRUE),]

# Get factor loadings of brand_rep_efa_data EFA
efa <- fa(brand_rep_efa_data, nfactors = 3)
efa$loadings

# Confirm the data that the model was fit to
inspect(brand_rep_cfa, what = "call")

# Check fit measures
fitmeasures(brand_rep_cfa)[c("cfi","tli","rmsea")]
```