# ESCP 8082 R Labs

Sonja D. Winter

2024-09-05

# Table of contents

# Preface

Welcome to the R Labs page for ESCP 8082: Foundations of Educational and Psychological Measurement. Although these labs were created for use in a classroom setting, they are available to all who happen to find them and find use in them. Please don't hesitate to let me know about any bugs or errors you notice, by contacting me at sdwinter@missouri.edu

# 1 Getting Started

Before you can successfully complete the R Labs included on this page, you will need to install some software and some packages within that software. This first Lab will help you do so.

## 1.1 Installing New Software

As the title of this page suggests, all labs will be done using R (and RStudio). To use these programs, you'll need to install **both** R and RStudio. Follow the instructions below to install them.

### 1.1.1 Step 1: Install R

R is a programming language and computing environment specialized for statistical analysis and data manipulation. It's commonly used for performing statistical tests, creating data visualizations, and writing data analysis reports.

#### Installing R for Windows Computers

Go to https://cloud.r-project.org/bin/windows/base/ and click the link titled **Download R-4.3.2 for Windows** (note: the version number might be different, but the remainder of the link will be the same). This will download the R Installer into your Downloads folder, where you can double click on it and follow the prompts on the screen to finish installing R. You can accepts all default settings.

#### Installing R for Mac Computers

You will need to figure out if you have an Intel Processor or an Apple M Processor. You can do so by clicking on the Apple icon in the top-left corner of your screen and clicking on About this Mac. The window that will pop up will show you an overview of your computer, including the processor/chip used.

Once you know what processor your computer has, go to https://cloud.r-project.org/bin/macosx/, and:

- If your computer has an Intel Processor, click on the file titled **R-4.3.2-x86_64.pkg**
- If your computer has an Apple M Processor, click on the file titled **R-4.3.2-arm64.pkg**

Note: the version number might be different, but the remainder of the link will be the same. This will download the R Installer into your Downloads folder, where you can double click on it and follow the prompts on the screen to finish installing R. You can accepts all default settings.

#### Installing R for Linux Computers

If you are using a Linux-based operating system, use your system's package manager to install R. For example, here are the instructions for installing R on Ubuntu.

> **Note**
>
> R cannot be installed on Chromebooks, so you'll need to use the computers available in the classroom/computer labs.

### 1.1.2 Step 2: Install RStudio

RStudio is an integrated development environment (IDE) for reproducible scientific computing that is developed for the R programming language. An IDE is basically a nicer-looking user interface that can be customized to suit the preferences of the user. This is the actual program that we will use in class!

- Download the latest, free version of RStudio Desktop. Be sure to get the version that is appropriate for your operating system.
- Install RStudio Desktop by launching the installer after it downloads. You can accept all the defaults during installation.

> **Tip**
>
> For more detailed instructions for downloading and installing R and RStudio, you can watch this video tutorial on YouTube. To learn about (or review) R basics, you can skim this (free!) book by Navarro (2015): Learning Statistics with R. There is also the SWIRL Interactive R Tutorial that lets you learn about the basics of R while using R.

## 1.2 Install Necessary Packages

Throughout these labs, we will rely on a set of R packages, which add functionality to the base R language (like expansion sets of a game). These packages are typically available through CRAN or GitHub. You only need to install packages once (but you may need to update them!), so lets do that now.

We will start with a set of packages that we can download from CRAN, using the built-in `install.packages` function:

```
install.packages(c("rio", "ggplot2", "psych","correlation",
                   "GPArotation", "lavaan", "MBESS",
                    "devtools"))
```

Running the code above will install:

1. `rio`: makes importing lots of different data file types easy.
2. `ggplot2`: a versatile visualization package.
3. `psych`: will help us cover topics such as exploratory factor analysis and reliability.
4. `correlation`: includes fancy correlation coeficients
5. `GPArotation`: helps with exploratory factor analysis
6. `lavaan`: the main structural equation modeling package we will use to cover confirmatory factor analysis and measurement invariance.
7. `MBESS`: includes additional internal consistency measures
8. `devtools`: a package that helps us install packages that are available on GitHub.

In addition to these main packages, R might also install additional packages that are needed for these 8 packages to work (so-called dependents).

Next, you will install a package, `semTools` from GitHub. Due to a bug in the version of this package on CRAN, we need to use the unofficial, development version of the package. You may need to uncomment (remove the #) the first line of code and execute both lines for this to work. In some cases, the download of the `semTools` package is too slow and results in an error because the R session times out.

```
# options(timeout = max(300, getOption("timeout")))
devtools::install_github("simsem/semTools/semTools")
```

## 1.3 Data Used in the R Labs

Several of the R Labs require you to download data files to use for the analyses. Links to these data files are included within each lab, accompanied by an explanation and citation.

You are now ready to continue to the second R Lab, where you will learn all about correlation coefficients.

# 2 Correlations

## 2.1 Loading R Packages

If you want to use the functionality of a package, you will need to "load" the package into your environment. To do that, we use the `library` function:

```
library(rio)
library(psych)
library(ggplot2)
library(correlation)
```

## 2.2 Loading Data

You can download the data by right-clicking this link and selecting "Save Link As..." in the drop-down menu: data/tempice.csv. Make sure to save it in the folder you are using for this class.

Typically, you will import some data file into your R environment for further analysis. There are many ways of doing this. I will show you two:

1. You can use a point-and-click approach by clicking the `Import Dataset` button in the right-top window.
2. You can use a function (the one we use is from the `rio` package).

```
tempice <- import(file = "data/tempice.csv")
```

The function above will attempt to import the file `tempice.csv` from a folder called data, which is located inside your working directory.

Sometimes, running the code above doesn't work because R thinks you want to import the data from the wrong folder (which R calls the working directory). We can check what the working directory is:

```
getwd()
```

If the result of this function is not the folder containing your data file, then you can change the working directory in two ways:

1. Use a point-and-click approach by moving your cursor to the bottom-right window to navigate to the correct folder (in the Files tab).
2. Use the following R function to change the working directory:

```
# Mac OS:
setwd("~/Dropbox/Work/Teaching/Measurement/R Labs")

# Windows:
setwd("C:/Users/sonja/Dropbox/Work/Teaching/Measurement/R Labs")

# Note: the folder that you are using for this class will very
# likely be in a different location.
```

Typically, R/RStudio will set the working directory to the folder containing the R file you open. If you start RStudio by itself (instead of opening a file), then the working directory will typically be set to your home folder.

## 2.3 Basic R operations

Below are some basic operations that you can execute in R. First, you can use R as a fancy calculator:

```
1 + 1
```

```
[1] 2
```

```
5 / 3.21
```

```
[1] 1.557632
```

```
4*4
```

```
[1] 16
```

You can also save one or more values into an object (think of this as a variable) and then do math with those objects:

```
x <- 10

y <- 5

x*y
```

```
[1] 50
```

There are several ways to store multiple values in one object, but the main method is using a function you've already used before:

```
z <- c(1, 2, 3, 4)

z * x
```

```
[1] 10 20 30 40
```

The `c()` function can be used to create vectors, which contain values for a single variable (here `z`). To access specific values within an object, you can use `[]`:

```
z[1]
```

```
[1] 1
```

```
z[1:3]
```

```
[1] 1 2 3
```

```
z[c(1,2,4)]
```

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

There are also objects called data frames. These look more like your SPSS data files, or Excel files: big tables in which each row represents a case/person and each column represents a variable. The data we imported above is in a data frame. We can access several parts of the data frame using basic operations and functions:

```
# retrieve the value in the first row, first column
tempice[1,1]
```

```
[1] 67.56
```

```
# retrieve the first column
tempice[,1]
```

```
 [1] 67.56 71.52 63.42 69.36 75.30 81.78 76.92 87.18 84.12 74.58 82.68 72.96
```

```
# retrieve the second column by using its column name
tempice$x2
```

```
 [1] 215 325 185 332 406 522 412 614 544 421 445 408
```

```
# get some summary information about each column
summary(tempice)
```

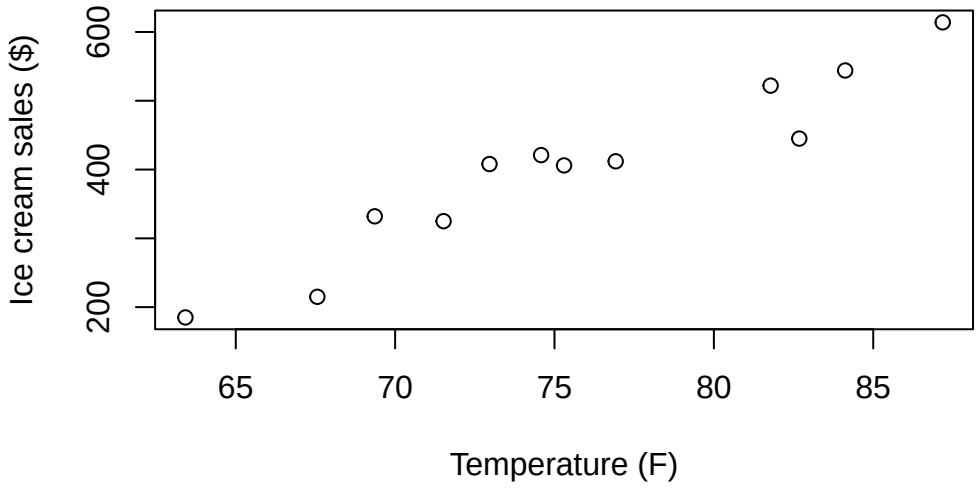
```
       x1              x2       
 Min.   :63.42   Min.   :185.0  
 1st Qu.:70.98   1st Qu.:330.2  
 Median :74.94   Median :410.0  
 Mean   :75.61   Mean   :402.4  
 3rd Qu.:82.00   3rd Qu.:464.2  
 Max.   :87.18   Max.   :614.0  
```

Throughout this course, you will learn additional operations you can use in R. This class is not meant to be a complete introduction to the R language, so your knowledge of R will be somewhat haphazard by the end of this class.

## 2.4 Visualizing Bivariate Associations

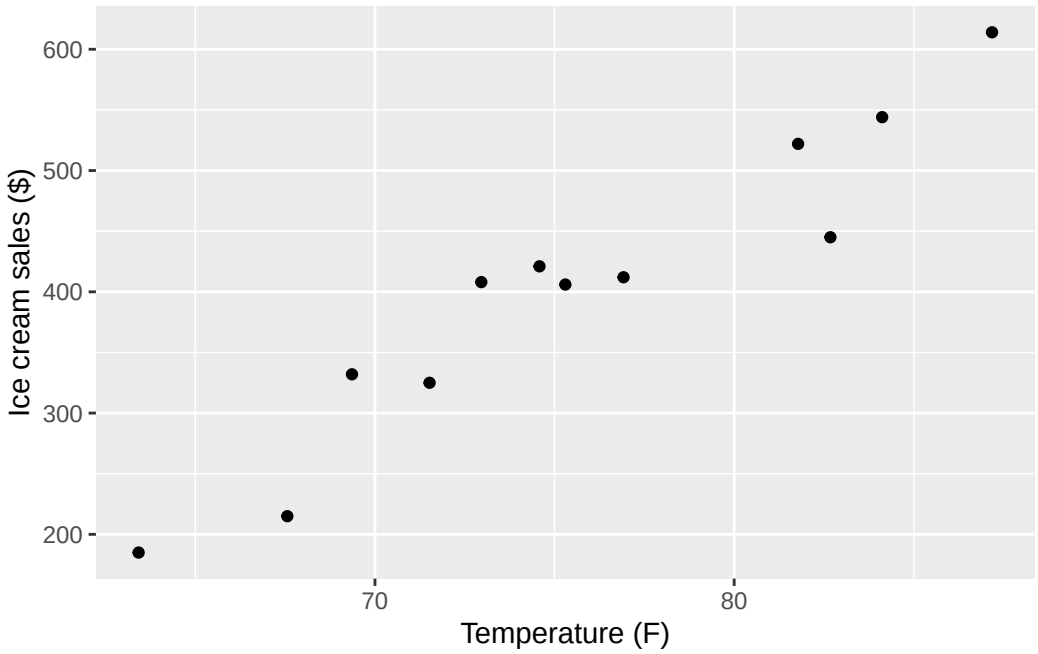
Now we can focus on the topic of this module: Correlation. We will start by producing a simple scatter plot to visualize the association between the two variables stored in `tempice`:

```
# Create scatterplot of variables x1 and x2
plot(x = tempice$x1, y = tempice$x2,
     xlab = "Temperature (F)",
     ylab = "Ice cream sales ($)")
```




We can also use the `ggplot2` package to create a similar scatter plot:

```
ggplot(tempice, aes(x = x1, y = x2)) +
  geom_point() +
  labs(x = "Temperature (F)",
       y = "Ice cream sales ($)")
```




## 2.5 Calculating Pearson's r 'by hand'

Next, we will go through the different computational steps to calculate Pearson's *r*.

### 2.5.1 The data

```
tempice
```

```
      x1  x2
1  67.56 215
2  71.52 325
3  63.42 185
4  69.36 332
5  75.30 406
6  81.78 522
7  76.92 412
8  87.18 614
9  84.12 544
10 74.58 421
```

```
11 82.68 445
12 72.96 408
```

### 2.5.2 Variable x1 calculations

First, we need to compute the mean, variance, and standard deviation for x1.

```
# Mean:
x1bar <- (67.56 + 71.52 + 63.42 + 69.36 + 75.30 + 81.78 +
            76.92 + 87.18 + 84.12 + 74.58 + 82.68 + 72.96) / 12

# Mean (less by hand):
x1bar_2 <- sum(tempice$x1) / nrow(tempice)

# The result is equivalent:
x1bar
```

```
[1] 75.615
```

```
x1bar_2
```

```
[1] 75.615
```

```
# Variance:
s2x1 <- ((67.56 - x1bar) ^ 2 +
           (71.52 - x1bar) ^ 2 +
           (63.42 - x1bar) ^ 2 +
           (69.36 - x1bar) ^ 2 +
           (75.30 - x1bar) ^ 2 +
           (81.78 - x1bar) ^ 2 +
           (76.92 - x1bar) ^ 2 +
           (87.18 - x1bar) ^ 2 +
           (84.12 - x1bar) ^ 2 +
           (74.58 - x1bar) ^ 2 +
           (82.68 - x1bar) ^ 2 +
           (72.96 - x1bar) ^ 2) / (12 - 1)

# Variance (less by hand):
```

```r
s2x1_2 <- sum((tempice$x1 - x1bar)^2) / (nrow(tempice) - 1)

# Standard deviation:
sx1_2 <- sqrt(s2x1)
sx1_2
```

```
[1] 7.220069
```

```r
# Getting these things by doing even less by hand:
x1bar <- mean(tempice$x1)
s2x1 <- var(tempice$x1)
sx1 <- sd(tempice$x1)
sx1
```

```
[1] 7.220069
```

### 2.5.3 Variable x2 calculations

Second, we need to compute the mean, variance, and standard deviation for x2. We will just use the built-in functions this time:

```r
# Same idea for variable x2:
x2bar <- mean(tempice$x2)
s2x2 <- var(tempice$x2)
sx2 <- sd(tempice$x2)
```

### 2.5.4 Sum of Cross-Products, Covarariance, and Correlation

Next, we have to combine these components together to finnd thee sum of cross-products:

```r
# Compute the sum of cross-products:
CP <- (tempice$x1 - x1bar) * (tempice$x2 - x2bar)
CP
```

```
 [1] 1509.64125  317.02125 2651.39625  440.45625   -1.12875  737.23125
 [7]   12.50625 2446.96125 1204.16625  -19.23375  300.85125  -14.82375
```

```r
sumCP <- sum(CP)
sumCP
```

```
[1] 9585.045
```

Finally, with help from the sample size, n, we can compute the sample covariance and (Pearson) correlation estimates:

```r
# Sample size
n <- nrow(tempice)
n
```

```
[1] 12
```

```r
# Covariance and correlation
covariance <- sumCP/(n - 1)
covariance
```

```
[1] 871.3677
```

```r
correlation <- covariance/(sx1 * sx2)

# Are ice cream sales and temperature correlated?
correlation
```

```
[1] 0.9575066
```

## 2.6 Using a function to calculate Pearson's r

Luckily, R has some built-in functions that we can use to compute Pearson's *r*:

```
cov(tempice$x1, tempice$x2)
```

```
[1] 871.3677
```

```
cor(tempice$x1, tempice$x2)
```

```
[1] 0.9575066
```

An even nicer option is to use a function that is part of the built-in `stats` package (this means you don't have to install or load it yourself), which provides a confidence interval around the estimate:

```
cor.test(tempice$x1, tempice$x2)
```

```
    Pearson's product-moment correlation

data:  tempice$x1 and tempice$x2
t = 10.499, df = 10, p-value = 1.016e-06
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.8515370 0.9883148
sample estimates:
      cor
0.9575066
```

## 2.7 Issues with Pearson's *r*

To see how misleading Pearson's *r* can be when data do not meet its assumptions, we'll look at a second data file.

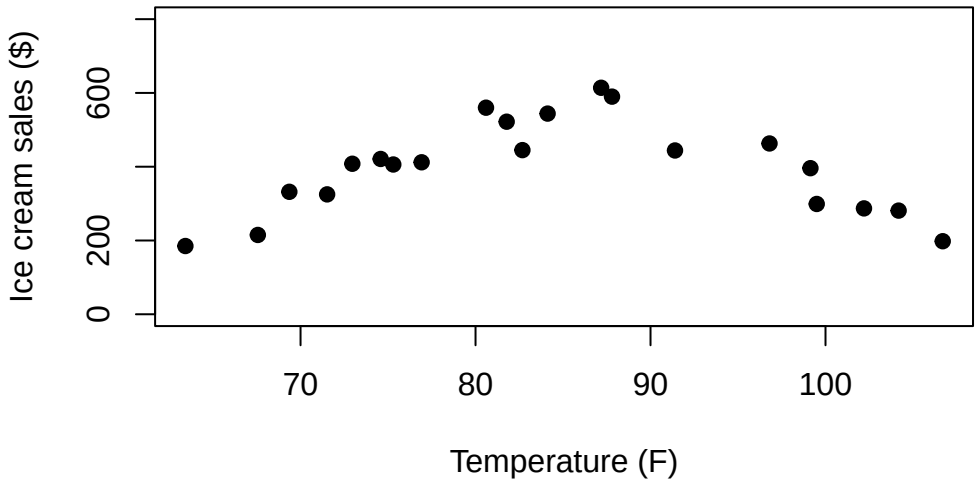
You can download this data by right-clicking this link and selecting "Save Link As..." in the drop-down menu: data/tempicecurve.csv. Again: Make sure to save it in the folder you are using for this class.

You can import the data using a version of the code below, or using the point-and-click method described above.

```
tempicecurve <- import(file = "data/tempicecurve.csv")
```

To get an idea of the problem with these data, we can visualize them in another scatter plot:

```
plot(tempicecurve$x1, tempicecurve$x2, pch=19,
     xlab = "Temperature (F)",
     ylab = "Ice cream sales ($)",
     ylim = c(0,800))
```




How will the shape of the relationship between x1 and x2 affect the Pearson's r correlation estimate?

```
cor.test(tempicecurve$x1, tempicecurve$x2)
```

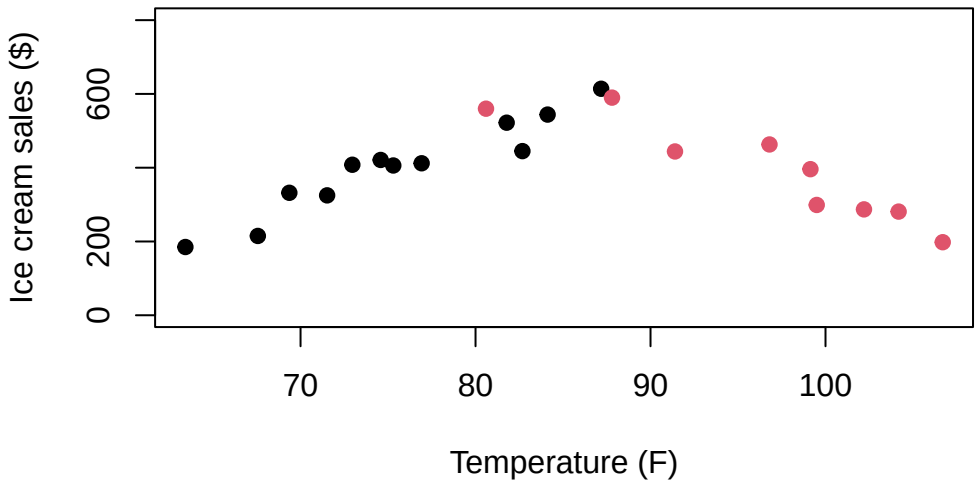
```
    Pearson's product-moment correlation

data:  tempicecurve$x1 and tempicecurve$x2
t = 0.0015808, df = 19, p-value = 0.9988
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.4313917  0.4319818
sample estimates:
          cor
0.0003626502
```

There might be an explanation for this kind of pattern. For example, it may be that there is a positive linear association up to a certain temperature after which the direction of the association changes because people don't want to leave their house to buy Ice Cream anymore.

To visualize this hypothesis, we can use the `group` variable to change the color of point below and above a vague temperature cutoff range:

```
plot(tempicecurve$x1,tempicecurve$x2,pch=19,
     xlab = "Temperature (F)",
     ylab = "Ice cream sales ($)",
     ylim = c(0,800),
     col = tempicecurve$group)
```




We can look at the correlation for each subset of data separately:

```
# select only group = 1 (cooler to hot temps)
tempicecurve1 <- subset(tempicecurve, group == 1)
cor.test(tempicecurve1$x1, tempicecurve1$x2)
```

```
    Pearson's product-moment correlation

data:  tempicecurve1$x1 and tempicecurve1$x2
t = 10.499, df = 10, p-value = 1.016e-06
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
```

```
 0.8515370 0.9883148
sample estimates:
      cor
0.9575066
```

```
# select only group = 2 (hot to hottest temps)
tempicecurve2 <- subset(tempicecurve, group == 2)
cor.test(tempicecurve2$x1, tempicecurve2$x2)
```

```
    Pearson's product-moment correlation

data:  tempicecurve2$x1 and tempicecurve2$x2
t = -6.2426, df = 7, p-value = 0.0004272
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.9834759 -0.6604377
sample estimates:
      cor
-0.920721
```

What do the separate correlation estimates tell us about the likely association between temperature and ice cream sales?

## 2.8 Alternatives to Pearson's *r*

In the above example, we were able to split the data in half to appropriately estimate two Pearson's *r* for two linear associations. But there are other alternative's to Pearson's *r* that help with other challenges.

### 2.8.1 Correlation Estimate for Data with Outliers

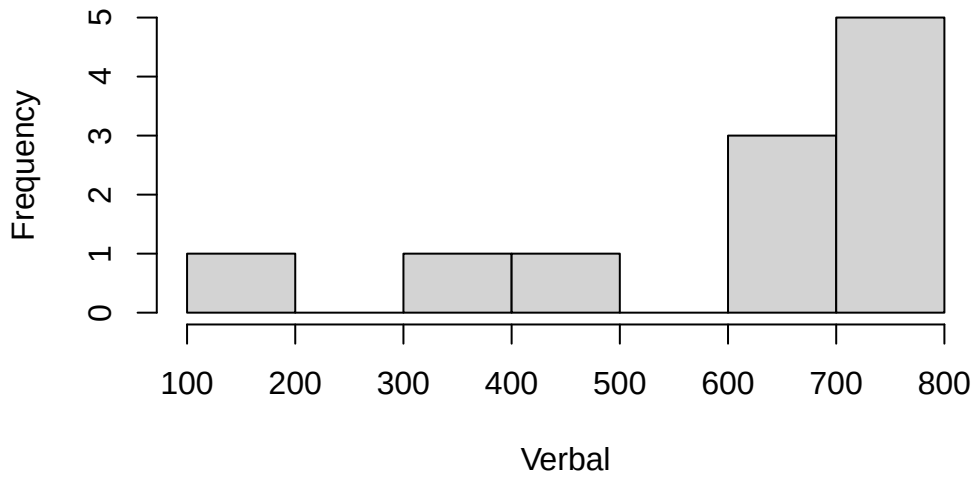
We need to import some more (fake) data:

```
SATscores_out <- rio::import("data/SATscores_outlier.csv")
```

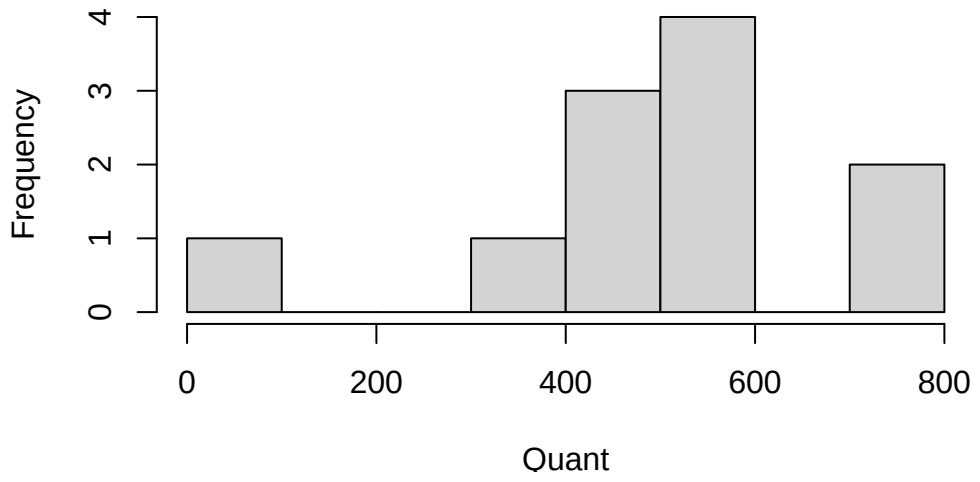
This data frame contains two variables, `verbal` and `quant`, which reflect 11 participants' verbal and quantitative SAT scores.

Below is code for visualizing the `SATscores_out` data, which reveals that there is an outlier.

```
hist(SATscores_out$verbal, xlab = "Verbal", main = "")
```




```
hist(SATscores_out$quant, xlab = "Quant", main = "")
```




We can compute the biweight and Winsorized correlation coefficients and compare those to the Pearson correlation coefficient:

```
# Pearson
cor.test(SATscores_out$verbal, SATscores_out$quant)
```

```
    Pearson's product-moment correlation

data:  SATscores_out$verbal and SATscores_out$quant
t = 3.8095, df = 9, p-value = 0.004157
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.3513338 0.9417011
sample estimates:
      cor
0.7856319
```

```r
# biweight
cor_test(SATscores_out, "verbal", "quant", method = "biweight")
```

```
Parameter1 | Parameter2 |    r |        95% CI | t(9) |     p
-------------------------------------------------------------
verbal     |      quant | 0.37 | [-0.29, 0.79] | 1.20 | 0.260

Observations: 11
```

```r
# Winsorized
cor_test(SATscores_out, "verbal", "quant", winsorize = TRUE)
```

```
Parameter1 | Parameter2 |    r |        95% CI | t(9) |     p
-------------------------------------------------------------
verbal     |      quant | 0.42 | [-0.24, 0.81] | 1.39 | 0.198

Observations: 11
```

How does the estimate of the correlation change across methods?

### 2.8.2 Correlation Estimate for Non-normal Data

Even without the outlier, the SAT scores distributions looked somewhat skewed. For this example, we will remove the outlier and focus solely on the non-normality of the two variables:

```
SATscores <- SATscores_out[1:10,]
```

Below is code to test if your variables are approximately Normally distributed. Remember, we're testing the Null hypothesis that the data are similar to a Normal distribution. If the p-value is < .05, we reject this Null hypothesis and have to conclude that the data are probably not normally distributed.

```
# Shapiro Wilk test of normality.
shapiro.test(SATscores$verbal)
```

```
    Shapiro-Wilk normality test

data:  SATscores$verbal
W = 0.82541, p-value = 0.02945
```

```
shapiro.test(SATscores$quant)
```

```
    Shapiro-Wilk normality test

data:  SATscores$quant
W = 0.82188, p-value = 0.02671
```

We can compare Spearman's $\rho$ (Rho) and Kendall's $\tau$ (Tau) to Pearson's correlation coefficient:

```
# Pearson
cor.test(SATscores$verbal, SATscores$quant)
```

```
    Pearson's product-moment correlation

data:  SATscores$verbal and SATscores$quant
t = 1.6325, df = 8, p-value = 0.1412
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
```

```
 -0.1893260  0.8591149
sample estimates:
      cor
0.4998894
```

```r
# Spearman (you can use cor.test or cor_test)
# cor.test(SATscores$verbal, SATscores$quant, method = "spearman")
cor_test(SATscores, "verbal", "quant", method = "spearman")
```

```
Parameter1 | Parameter2 |  rho |       95% CI |     S |      p
-------------------------------------------------------------
verbal     |      quant | 0.67 | [0.05, 0.92] | 54.00 | 0.033*

Observations: 10
```

```r
# Kendall (you can use cor.test or cor_test)
#cor.test(SATscores$verbal, SATscores$quant, method = "kendall")
cor_test(SATscores, "verbal", "quant", method = "kendall")
```

```
Parameter1 | Parameter2 |  tau |       95% CI |    z |      p
-----------------------------------------------------------
verbal     |      quant | 0.51 | [0.04, 0.80] | 2.06 | 0.040*

Observations: 10
```

How does the estimate of the correlation change across methods?

We can also compare the raw SAT data to the ranked SAT data to see that the correlation estimate is equivalent when using Kendall/Spearman, but that it is different when using Pearson.

We first create the rank-ordered variables:

```r
SATscores_rank <- data.frame(verbal = rank(SATscores$verbal),
                             quant = rank(SATscores$quant))
```

Next, we look at the Pearson's *r* when based on the raw or ranked data:

```r
# Camparing Pearson correlation coefficients
# (now using the cor_test function)
cor_test(SATscores, "verbal", "quant", method = "pearson")
```

```
Parameter1 | Parameter2 |    r |        95% CI | t(8) |     p
-----------------------------------------------------------------
verbal     |      quant | 0.50 | [-0.19, 0.86] | 1.63 | 0.141

Observations: 10
```

```r
cor_test(SATscores_rank, "verbal", "quant", method = "pearson")
```

```
Parameter1 | Parameter2 |    r |       95% CI | t(8) |      p
-----------------------------------------------------------------
verbal     |      quant | 0.67 | [0.07, 0.91] | 2.57 | 0.033*

Observations: 10
```

Now compare those results to what happens when we use Spearman's $rho$ (Rho):

```r
# Comparing Spearman correlation coefficients
cor_test(SATscores, "verbal", "quant", method = "spearman")
```

```
Parameter1 | Parameter2 |  rho |       95% CI |     S |      p
-----------------------------------------------------------------
verbal     |      quant | 0.67 | [0.05, 0.92] | 54.00 | 0.033*

Observations: 10
```

```r
cor_test(SATscores_rank, "verbal", "quant", method = "spearman")
```

```
Parameter1 | Parameter2 |  rho |       95% CI |     S |      p
-----------------------------------------------------------------
verbal     |      quant | 0.67 | [0.05, 0.92] | 54.00 | 0.033*

Observations: 10
```

### 2.8.3 Correlation Estimate for (Ordinal) Categorical Data

For this example, we will import some ordinal data on quality of life (QoL) and health:

```
QoLHealth <- import("data/QoLHealth.csv")
```

The variables are imported as strings, so we need to tell R what the order of the possible values is:

```
QoLHealth$health <- factor(QoLHealth$health, level = c("Poor", "Moderate","Good"
QoLHealth$QoL <- factor(QoLHealth$QoL, level = c("Low", "Medium", "High"), order
```

The cross-table below shows the categorical nature of these variables, where each only takes on 3 values that may be ordinal but are not neccesarily equally spaced:

```
table(QoLHealth)
```

```
          QoL
health     Low Medium High
  Poor      58     52    1
  Moderate  26     58    3
  Good       8     12    9
```

We can use the polychoric correlation coefficient for the ordinal QoL and Health data (here we use the `correlation` package):

```
cor_test(QoLHealth, "health", "QoL", method = "polychoric")
```

```
Parameter1 | Parameter2 |  rho |       95% CI | t(225) |         p
-------------------------------------------------------------------
health     |        QoL | 0.42 | [0.31, 0.52] |   6.94 | < .001***

Observations: 227
```

There is also an option in the `psych` package to compute the polychoric correlation coefficient, which uses the cross-table as input:

```
polychoric(table(QoLHealth))
```

```
[1] "You seem to have a table, I will return just one correlation."

$rho
[1] 0.4198846

$objective
[1] 1.790876

$tau.row
       Poor    Moderate
-0.02760955  1.13707578

$tau.col
       Low     Medium
-0.2396873  1.5781226
```

The nice thing about the psych functions is that they also return the threshold estimates that represent the point on the underlying continuous distribution (e.g., the continuum of health from poor to good) where someone is likely to change their answer from one response category to the next.

## 2.9 Summary

In this R lab, you were introduced to a host of correlation coefficients, each of which are appropriate for different variable types and distributions. Next time you want to estimate the correlation between two variables, take a momnt to consider is Pearson's *r* is the best choice or not.

# 3 Confirmatory Factor Analysis

## 3.1 Loading R Packages

Remember, you only need to install a package once. But if you want to use the functionality of a package, you will need to "load" the package into your environment. To do that for lavaan (and the psych package, which we'll also use in this lab), we use the `library()` function:

```
library(lavaan)
library(semTools)
library(psych)
```

## 3.2 Loading data into our environment

Typically, you will load your own data into your environment, like we did in the Correlations lab. However, you can also use datasets that are included with R packages. To access those datasets, you can use the `data()` function:

```
data("HolzingerSwineford1939")
```

If you look at your environment tab, you should see a new data frame called `HolzingerSwineford1939`. We can take a look at the variables this dataframe using the `describe()` function that is part of the `psych` package:

```
describe(HolzingerSwineford1939)
```

```
      vars   n   mean    sd median trimmed   mad   min    max  range  skew
id       1 301 176.55 105.94 163.00  176.78 140.85  1.00 351.00 350.00 -0.01
sex      2 301   1.51   0.50   2.00    1.52   0.00  1.00   2.00   1.00 -0.06
ageyr    3 301  13.00   1.05  13.00   12.89   1.48 11.00  16.00   5.00  0.69
agemo    4 301   5.38   3.45   5.00    5.32   4.45  0.00  11.00  11.00  0.09
```

```
school*   5 301  1.52  0.50  2.00   1.52  0.00 1.00  2.00  1.00 -0.07
grade     6 300  7.48  0.50  7.00   7.47  0.00 7.00  8.00  1.00  0.09
x1        7 301  4.94  1.17  5.00   4.96  1.24 0.67  8.50  7.83 -0.25
x2        8 301  6.09  1.18  6.00   6.02  1.11 2.25  9.25  7.00  0.47
x3        9 301  2.25  1.13  2.12   2.20  1.30 0.25  4.50  4.25  0.38
x4       10 301  3.06  1.16  3.00   3.02  0.99 0.00  6.33  6.33  0.27
x5       11 301  4.34  1.29  4.50   4.40  1.48 1.00  7.00  6.00 -0.35
x6       12 301  2.19  1.10  2.00   2.09  1.06 0.14  6.14  6.00  0.86
x7       13 301  4.19  1.09  4.09   4.16  1.10 1.30  7.43  6.13  0.25
x8       14 301  5.53  1.01  5.50   5.49  0.96 3.05 10.00  6.95  0.53
x9       15 301  5.37  1.01  5.42   5.37  0.99 2.78  9.25  6.47  0.20
         kurtosis   se
id          -1.36 6.11
sex         -2.00 0.03
ageyr        0.20 0.06
agemo       -1.22 0.20
school*     -2.00 0.03
grade       -2.00 0.03
x1           0.31 0.07
x2           0.33 0.07
x3          -0.91 0.07
x4           0.08 0.07
x5          -0.55 0.07
x6           0.82 0.06
x7          -0.31 0.06
x8           1.17 0.06
x9           0.29 0.06
```

You can also learn more about these built-in datasets by going to its help page:

```
?HolzingerSwineford1939
```

## 3.3 CFA Step 1: Model Specification

To specify a model in `lavaan`, we have to write it out in `lavaan` syntax and assign it to an object (here called HSmodel). Basic syntax for a CFA follows this template:

```
factorname =~ indicator1 + indicator2 + indicator3
```

In our model, we have three factors (visual, textual, and speed) that each load onto three items (e.g., for visual: x1, x2, and x3). You don't have to specify that factors are hypothesized to be correlated, `lavaan` does this automatically (scroll down for an example of a CFA in which we specify that the factors should not be correlated).

```
#CFA model specification
HSmodel <- "visual  =~ x1 + x2 + x3
            textual =~ x4 + x5 + x6
            speed   =~ x7 + x8 + x9"
```

## 3.4 CFA Step 2: Model Estimation

Next, we need to estimate the model, using our data. In our lecture, we went over the four phases of estimation, but in R, model estimation simplifies to using the `cfa()` function with our model syntax and data arguments:

```
fit1 <- cfa(model = HSmodel,
            data = HolzingerSwineford1939)
```

## 3.5 CFA Step 3: Interpreting Model Fit and Parameter Estimates

There are several ways of extracting the model fit and parameter estimates from our fitted lavaan model (called `fit1`). The most typical way to look at this output is by using the `summary()` function. Within this function, we can ask for some extra output (fit measures, standardized estimate, R-squares):

```
summary(fit1,
        fit.measures = TRUE,
        standardized = TRUE,
        rsquare = TRUE)
```

```
lavaan 0.6.17 ended normally after 35 iterations

  Estimator                                         ML
  Optimization method                           NLMINB
  Number of model parameters                        21
```

```
  Number of observations                           301

Model Test User Model:

  Test statistic                                85.306
  Degrees of freedom                            24
  P-value (Chi-square)                          0.000

Model Test Baseline Model:

  Test statistic                               918.852
  Degrees of freedom                            36
  P-value                                       0.000

User Model versus Baseline Model:

  Comparative Fit Index (CFI)                   0.931
  Tucker-Lewis Index (TLI)                      0.896

Loglikelihood and Information Criteria:

  Loglikelihood user model (H0)             -3737.745
  Loglikelihood unrestricted model (H1)     -3695.092

  Akaike (AIC)                               7517.490
  Bayesian (BIC)                             7595.339
  Sample-size adjusted Bayesian (SABIC)      7528.739

Root Mean Square Error of Approximation:

  RMSEA                                         0.092
  90 Percent confidence interval - lower        0.071
  90 Percent confidence interval - upper        0.114
  P-value H_0: RMSEA <= 0.050                   0.001
  P-value H_0: RMSEA >= 0.080                   0.840

Standardized Root Mean Square Residual:

  SRMR                                          0.065

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
  visual =~
    x1                1.000                               0.900    0.772
    x2                0.554    0.100    5.554    0.000    0.498    0.424
    x3                0.729    0.109    6.685    0.000    0.656    0.581
  textual =~
    x4                1.000                               0.990    0.852
    x5                1.113    0.065   17.014    0.000    1.102    0.855
    x6                0.926    0.055   16.703    0.000    0.917    0.838
  speed =~
    x7                1.000                               0.619    0.570
    x8                1.180    0.165    7.152    0.000    0.731    0.723
    x9                1.082    0.151    7.155    0.000    0.670    0.665

Covariances:
                   Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
  visual ~~
    textual           0.408    0.074    5.552    0.000    0.459    0.459
    speed             0.262    0.056    4.660    0.000    0.471    0.471
  textual ~~
    speed             0.173    0.049    3.518    0.000    0.283    0.283

Variances:
                   Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
   .x1                0.549    0.114    4.833    0.000    0.549    0.404
   .x2                1.134    0.102   11.146    0.000    1.134    0.821
   .x3                0.844    0.091    9.317    0.000    0.844    0.662
   .x4                0.371    0.048    7.779    0.000    0.371    0.275
   .x5                0.446    0.058    7.642    0.000    0.446    0.269
   .x6                0.356    0.043    8.277    0.000    0.356    0.298
   .x7                0.799    0.081    9.823    0.000    0.799    0.676
   .x8                0.488    0.074    6.573    0.000    0.488    0.477
   .x9                0.566    0.071    8.003    0.000    0.566    0.558
    visual            0.809    0.145    5.564    0.000    1.000    1.000
    textual           0.979    0.112    8.737    0.000    1.000    1.000
    speed             0.384    0.086    4.451    0.000    1.000    1.000

R-Square:
```

```
   Estimate
x1    0.596
x2    0.179
x3    0.338
x4    0.725
x5    0.731
x6    0.702
x7    0.324
x8    0.523
x9    0.442
```

### 3.5.1 Model Fit

The output above is great, but it can be a lot. To look at just the fit indices, you can also use `fitMeasures()`. This function will return *a ton* of fit indices if you do not include the `fit.measures` argument. Here, we select the main indices that we're intereted in:

```
fitMeasures(fit1,
            fit.measures = c("chisq", "df", "pvalue",
                             "cfi", "rmsea", "rmsea.ci.lower",
                             "rmsea.ci.upper", "srmr"))
```

```
        chisq            df        pvalue           cfi         rmsea
       85.306        24.000         0.000         0.931         0.092
rmsea.ci.lower rmsea.ci.upper          srmr
        0.071         0.114         0.065
```

We can include `output = "text"` to make the output look a little bit nicer (more like the summary output above):

```
fitMeasures(fit1,
            fit.measures = c("chisq", "df", "pvalue",
                             "cfi", "rmsea", "rmsea.ci.lower",
                             "rmsea.ci.upper", "srmr"),
            output = "text")
```

```
Model Test User Model:
```

```
  Test statistic                                85.306
  Degrees of freedom                            24
  P-value                                       0.000

User Model versus Baseline Model:

  Comparative Fit Index (CFI)                   0.931

Root Mean Square Error of Approximation:

  RMSEA                                         0.092
  Confidence interval - lower                   0.071
  Confidence interval - upper                   0.114

Standardized Root Mean Square Residual:

  SRMR                                          0.065
```

Based on all fit indices, the fit of this CFA is poor. First, the Chi-square statistic is significant, indicating poor fit. Sometimes, with larger sample sizes, a significant Chi-square simply means that there are a lot of small, trivial misspecifications. However, if we look at the CFI, TLI, RMSEA, and SRMR values and compare them to their suggested cutoff values (.95, .95, .06, .08), they also indicate that the model fits the data poorly.

### 3.5.2 Parameter Estimates

There is also a function we can use to extract *just* the standardized estimates, `standardizedSolution()`. Within this function, we can specify that some of the output (z-statistics and p-values) are left out. Typically, significance of parameter estimates is evaluated using the unstandardized solution (which we saw above with the `summary()` function), so we should not focus on significance of the standardized estimates. Again, we can include `output = "text"` to get output that is easier to read:

```
standardizedSolution(fit1,
                     zstat = FALSE, pvalue = FALSE,
                     output = "text")
```

```
Latent Variables:
```

```
                   est.std  Std.Err ci.lower ci.upper
  visual =~
    x1               0.772    0.055    0.664    0.880
    x2               0.424    0.060    0.307    0.540
    x3               0.581    0.055    0.473    0.689
  textual =~
    x4               0.852    0.023    0.807    0.896
    x5               0.855    0.022    0.811    0.899
    x6               0.838    0.023    0.792    0.884
  speed =~
    x7               0.570    0.053    0.465    0.674
    x8               0.723    0.051    0.624    0.822
    x9               0.665    0.051    0.565    0.765

Covariances:
                   est.std  Std.Err ci.lower ci.upper
  visual ~~
    textual          0.459    0.064    0.334    0.584
    speed            0.471    0.073    0.328    0.613
  textual ~~
    speed            0.283    0.069    0.148    0.418

Variances:
                   est.std  Std.Err ci.lower ci.upper
   .x1               0.404    0.085    0.238    0.571
   .x2               0.821    0.051    0.722    0.920
   .x3               0.662    0.064    0.537    0.788
   .x4               0.275    0.038    0.200    0.350
   .x5               0.269    0.038    0.194    0.344
   .x6               0.298    0.039    0.221    0.374
   .x7               0.676    0.061    0.557    0.794
   .x8               0.477    0.073    0.334    0.620
   .x9               0.558    0.068    0.425    0.691
    visual           1.000             1.000    1.000
    textual          1.000             1.000    1.000
    speed            1.000             1.000    1.000
```

## 3.6 CFA Step 4: Making Model Adjustments

Our model did not fit our data very well, so we may want to make some adjustments to our model. Remember, there are always ways to make a model fit better (adding

parameters), but if they are not informed and justified by theory then we end up with a model that will only fit our data (like the outfits made specifically for Taylor Swift during her Eras tour) and will not generalize to new samples (which is bad!).

We can use a function to help us identify the parameters that, when added, will result in the largest improvements in model fit (in terms of a reduction in the model Chi-square statistic), the `modindices()` function (which stands for modification indices). In the code below, we ask that only parameters with relatively large modification indices (10 or above) are returned, and we ask that the output is sorted from largest improvement to smallest improvement.

```
modindices(fit1, minimum.value = 10, sort = TRUE)
```

```
      lhs op rhs     mi    epc sepc.lv sepc.all sepc.nox
30 visual =~  x9 36.411  0.577   0.519    0.515    0.515
76     x7 ~~  x8 34.145  0.536   0.536    0.859    0.859
28 visual =~  x7 18.631 -0.422  -0.380   -0.349   -0.349
78     x8 ~~  x9 14.946 -0.423  -0.423   -0.805   -0.805
```

### 3.6.1 Model Re-Specification

We can use the information from the modification indices to re-specify our model. The largest index is for adding a factor loading that goes from the visual factor to x9 (`mi = 36.41`), which is "Speeded discrimination straight and curved capitals". From the description of x9, we can see that this test does involve visual ability as well, so I feel that we can justify this modification. This means that the item x9 now loads onto two factors. The second loading is also called a cross loading. By adding this cross loadings, we're making the interpretation of the visual and speed factors more complex.

```
#Reanalysis
HSmodel2 <- "visual  =~ x1 + x2 + x3 + x9
             textual =~ x4 + x5 + x6
             speed   =~ x7 + x8 + x9"
```

### 3.6.2 Model Estimation

Again, model estimation is straightforward:

```r
fit2 <- cfa(model = HSmodel2,
            data = HolzingerSwineford1939)
```

### 3.6.3 Model Fit and Parameter Interpretation

We can look at the model fit and parameter estimates of this new model.

```r
summary(fit2,
        fit.measures = TRUE,
        standardized = TRUE,
        rsquare = TRUE)
```

```
lavaan 0.6.17 ended normally after 34 iterations

  Estimator                                         ML
  Optimization method                           NLMINB
  Number of model parameters                        22

  Number of observations                           301

Model Test User Model:

  Test statistic                                52.382
  Degrees of freedom                                23
  P-value (Chi-square)                           0.000

Model Test Baseline Model:

  Test statistic                               918.852
  Degrees of freedom                                36
  P-value                                        0.000

User Model versus Baseline Model:

  Comparative Fit Index (CFI)                    0.967
  Tucker-Lewis Index (TLI)                       0.948

Loglikelihood and Information Criteria:
```

```
  Loglikelihood user model (H0)              -3721.283
  Loglikelihood unrestricted model (H1)      -3695.092

  Akaike (AIC)                                7486.566
  Bayesian (BIC)                              7568.123
  Sample-size adjusted Bayesian (SABIC)       7498.351

Root Mean Square Error of Approximation:

  RMSEA                                          0.065
  90 Percent confidence interval - lower         0.042
  90 Percent confidence interval - upper         0.089
  P-value H_0: RMSEA <= 0.050                    0.133
  P-value H_0: RMSEA >= 0.080                    0.158

Standardized Root Mean Square Residual:

  SRMR                                           0.045

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
  visual =~
    x1                1.000                               0.885    0.759
    x2                0.578    0.098    5.918    0.000    0.511    0.435
    x3                0.754    0.103    7.291    0.000    0.667    0.590
    x9                0.437    0.081    5.367    0.000    0.387    0.384
  textual =~
    x4                1.000                               0.989    0.851
    x5                1.115    0.066   17.016    0.000    1.103    0.856
    x6                0.926    0.056   16.685    0.000    0.916    0.838
  speed =~
    x7                1.000                               0.666    0.612
    x8                1.207    0.185    6.540    0.000    0.804    0.795
    x9                0.675    0.112    6.037    0.000    0.450    0.447

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

```
  visual ~~
    textual           0.396    0.072    5.506    0.000    0.453    0.453
    speed             0.177    0.055    3.239    0.001    0.301    0.301
  textual ~~
    speed             0.136    0.051    2.675    0.007    0.206    0.206

Variances:
                   Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
   .x1                0.576    0.100    5.731    0.000    0.576    0.424
   .x2                1.120    0.100   11.153    0.000    1.120    0.811
   .x3                0.830    0.087    9.515    0.000    0.830    0.651
   .x9                0.558    0.060    9.336    0.000    0.558    0.550
   .x4                0.373    0.048    7.800    0.000    0.373    0.276
   .x5                0.444    0.058    7.602    0.000    0.444    0.267
   .x6                0.357    0.043    8.285    0.000    0.357    0.298
   .x7                0.740    0.086    8.595    0.000    0.740    0.625
   .x8                0.375    0.094    3.973    0.000    0.375    0.367
    visual            0.783    0.134    5.842    0.000    1.000    1.000
    textual           0.978    0.112    8.728    0.000    1.000    1.000
    speed             0.444    0.097    4.567    0.000    1.000    1.000

R-Square:
                   Estimate
    x1                0.576
    x2                0.189
    x3                0.349
    x9                0.450
    x4                0.724
    x5                0.733
    x6                0.702
    x7                0.375
    x8                0.633
```

But how do we know if this model is better (in terms of fitting our data) than the original CFA?

## 3.7 Comparing Multiple Models to Each Other

To compare the fit of the two models, we can use the `compareFit()` function:

```r
comp_fit1_fit2 <- compareFit(fit1, fit2)
summary(comp_fit1_fit2)
```

```
################### Nested Model Comparison #########################

Chi-Squared Difference Test

     Df    AIC    BIC  Chisq Chisq diff   RMSEA Df diff Pr(>Chisq)
fit2 23 7486.6 7568.1 52.382
fit1 24 7517.5 7595.3 85.305     32.923 0.32567       1  9.586e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

######################## Model Fit Indices ###########################
       chisq df pvalue rmsea   cfi   tli  srmr       aic       bic
fit2 52.382† 23   .000 .065† .967† .948† .045† 7486.566† 7568.123†
fit1 85.306  24   .000 .092  .931  .896  .065  7517.490  7595.339

################## Differences in Fit Indices ########################
           df rmsea    cfi    tli srmr    aic    bic
fit1 - fit2  1 0.027 -0.036 -0.052 0.02 30.923 27.216
```

The output of this function includes a comparison of fit based on the model Chi-square: the Chi-square difference test. If this test is significant, then it means that the model with fewer estimated parameters (here `fit1`) fits the data significantly *worse* than the model with more estimated parameters (here `fit2`), so we should select the model with more parameters (`fit2`). If this test is not significant, then it means that the additional parameters of `fit2` did not improve fit enough to result in a *better* model fit than that of `fit1`, which means that we should stick with the simpler model (`fit1`). What do the resuts above tell us?

The output also includes comparisons of the relative fit indices (AIC and BIC). Remember: lower values indicate better fit. Finally, the output also includes comparisons of other fit indices (e.g., CFI, RMSEA). These are sometimes also used to compare the fit of several models. However, there are no clear, universal guidelines on how different these indices need to be before they indicate an improvement/worsening in model fit.

## 3.8 How to specify different types of CFAs

Here is an example of how to specify a hierarchical CFA, where the three factors are indicator of a higher-order `ability` factor:

```
HSmodel3 <- "visual =~ x1 + x2 + x3
            textual =~ x4 + x5 + x6
            speed   =~ x7 + x8 + x9

            ability =~ visual + textual + speed"
```

Here is an example of a CFA in which the factors are specified to be uncorrelated with each other:

```
HSmodel4 <- "visual =~ x1 + x2 + x3
            textual =~ x4 + x5 + x6
            speed   =~ x7 + x8 + x9

            visual ~~ 0*textual
            visual ~~ 0*speed
            textual ~~ 0*speed"
```

Factor correlations can be fixed to 0 (i.e., removed from the CFA) using the following template:

`factorname1 ~~ 0*factorname2`

Here is an example of a CFA in which the factors are specified to be correlated *but* the correlations are constrained to be equal (i.e., their parameter estimate is going to be the exact same value):

```
HSmodel5 <- "visual =~ x1 + x2 + x3
            textual =~ x4 + x5 + x6
            speed   =~ x7 + x8 + x9

            visual ~~ a*textual
            visual ~~ a*speed
            textual ~~ a* speed"
```

To constrain parameters to be equal, we can give them the same label (this is different from the factorname that we use to specify/name latent factors). The general format for these equality constraints is:

```
factorname1 ~~ label*factorname2
```

Or for constraining factor loadings to be equivalent:

```
factorname1 =~ label*x1 + label*x2 + label*x3
```

Your label can be any text string (e.g., a, b, eq), but remember to use different labels for different equality constraints. So, if you want to constrain your loadings *and* your factor correlations, use the label `a` for the loadings and `b` for the correlations.

## 3.9 Summary

In this R lab, you learned how to specify, estimate, evaluate and interpret CFAs. In addition, you learned how to re-specify a CFA and compare the fit across several models to select the best-fitting model. Finally, you were introduced to some examples of alternative CFA configurations, such as the higher-order CFA.

# 4 Exploratory Factor Analysis

## 4.1 Loading R Packages

Remember, you only need to install a package once. But if you want to use the functionality of a package, you will need to "load" the package into your environment. To do that for lavaan (and the psych package, which we'll also use in this lab), we use the `library()` function:

```
library(lavaan)
library(psych)
library(GPArotation)
```

## 4.2 Loading data into our environment

We're using the same dataset as we used in the CFA R Lab, so we can use the `data()` function like we did before:

```
data("HolzingerSwineford1939")
```

Remember that for the CFA analysis, we did not have to remove any variables from the data frame, because `lavaan` extracted the relevant variables automatically. With EFA, using the `psych` package, we have to do that extraction ourselves. We can do that as follows:

```
# print the variable names of the full data frame
# and locate the relevant variables
colnames(HolzingerSwineford1939)
```

```
 [1] "id"     "sex"    "ageyr"  "agemo"  "school" "grade"  "x1"     "x2"
 [9] "x3"     "x4"     "x5"     "x6"     "x7"     "x8"     "x9"
```

```
# use the [,] operator to select only the relevant
# columns/variables (here in column 7 to 15)
HSdata <- HolzingerSwineford1939[,7:15]
```

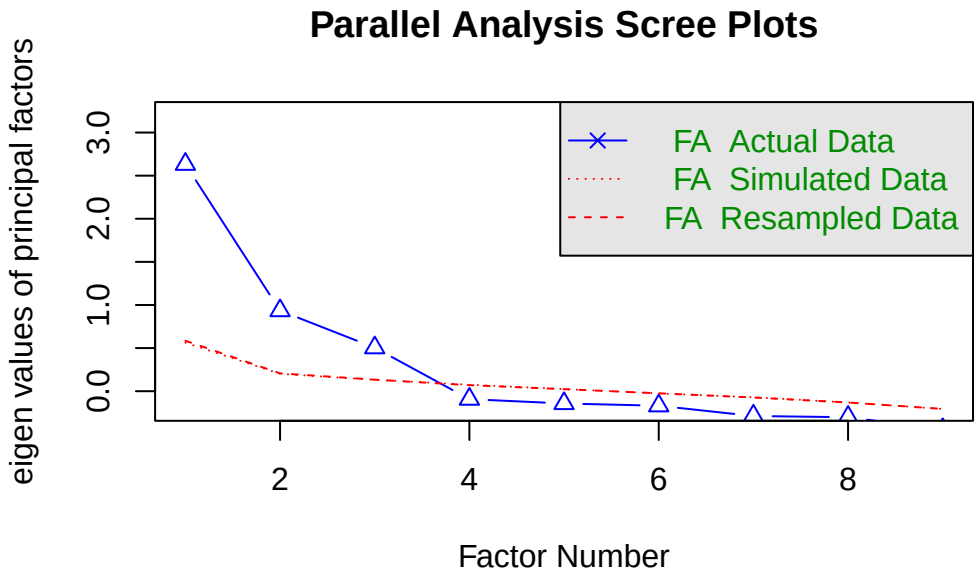
There are many other ways of selecting variables from a larger data frame, and if you have a different method that you like better, feel free to use it!

## 4.3 EFA Step 1: How many factors should I extract?

In the first step of the EFA, we will use parallel analysis to see what the algorithm identifies as the optimal number of factors to extract from the data. This algorithm generates random correlation matrices, and when doing so, it may return an error message because something went wrong with those random matrices. If this happens, you can simply re-run the `fa.parallel()` function and the error should disappear.

The code below will return a parallel analysis for the Holzinger Swineford data using the factor analysis method and based on 50 random correlation matrices. You can increase that number to 100 or 1000 if you want to be more certain of the results, but note that that will take longer to run.

```
fa.parallel(HSdata, fa = "fa", n.iter = 50)
```




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

Based on the plot, how many factors should we extract?

## 4.4 EFA Step 2: Factor Extraction and Rotation

In the first cycle of the EFA process, we will follow the parallel analysis results and estimate a 3-factor EFA. To estimate the parameters and rotate those results to be more interpretable, we just need to use one function:

```
efa_3f <- fa(HSdata, nfactors = 3,
             fm = "minres",
             rotate = "oblimin")
```

Technically, you don't even need to include `fm = "minres", rotate = "oblimin"`, but I wanted to show you what arguments you need to use if you want to change the default estimation method (here "minres") or if you want to change the default rotation method (here "oblimin").

## 4.5 EFA Step 3: Interpreting the EFA estimates

### 4.5.1 Commmunalities

```
round(efa_3f$communalities,
      digits = 3)
```

```
   x1    x2    x3    x4    x5    x6    x7    x8    x9
0.477 0.255 0.453 0.728 0.754 0.691 0.519 0.520 0.460
```

Most communalities are between .4 and .6 (one even above .6), indicating that the factors are able to account for a good chunk of the variability in the item responses. One exception is x2, which has a communality of .255. Overall, these values look acceptable.

### 4.5.2 Factor Loadings

It can be helpful to hide low factor loadings from your output to see if the factor extraction and rotation has resulted in a simple structure. We can do that by including `cutoff = .3` in the `print()` function:

```
print(efa_3f$loadings,
      cutoff = .3)
```

```
Loadings:
   MR1    MR3    MR2
x1          0.592
x2          0.509
x3          0.686
x4  0.846
x5  0.886
x6  0.805
x7                 0.737
x8                 0.686
x9          0.382  0.456

                 MR1   MR3   MR2
SS loadings    2.197 1.275 1.239
Proportion Var 0.244 0.142 0.138
Cumulative Var 0.244 0.386 0.523
```

The factor loadings appear to follow a pretty clear, simple structure. The exception is x9, which has a factor loading > .3 on two factors.

This output also includes information about the variance in the items that is explained by each factor. `SS loadings` refers to the sum of the squared loadings (i.e., the factor's Eigenvalue). The columns (even in the loadings table) are sorted from the highest Eigenvalue to the lowest. That's why the order here is MR1, MR3, and then MR2 (and MR refers to the estimation method, minres). The second row shows the variance that is accounted for by each factor, and the bottom row shows the cumulative variance accounted for by all factors. Here, the three factors explain 52.3% of the variance in the items.

Note: if you want to see all the factor loading estimates, you need to set the cuttof at the lowest possible value for factorloadings (-1):

```
print(efa_3f$loadings,
      cutoff = -1)
```

```
Loadings:
   MR1    MR3    MR2
```

```
x1  0.196  0.592  0.031
x2  0.043  0.509 -0.122
x3 -0.062  0.686  0.019
x4  0.846  0.016  0.008
x5  0.886 -0.065  0.007
x6  0.805  0.080 -0.013
x7  0.044 -0.152  0.737
x8 -0.034  0.125  0.686
x9  0.032  0.382  0.456

                 MR1   MR3   MR2
SS loadings    2.197 1.275 1.239
Proportion Var 0.244 0.142 0.138
Cumulative Var 0.244 0.386 0.523
```

### 4.5.3 Factor Correlations

Finally, we can look at the correlations between the factors:

```
round(efa_3f$Phi,
      digits = 3)
```

```
      MR1   MR3   MR2
MR1 1.000 0.323 0.213
MR3 0.323 1.000 0.261
MR2 0.213 0.261 1.000
```

Extremely large correlations between factors may be an indication of overextraction; the two factors could be combined into one factor. In this case, the correlations between the factors are small to moderate, indicating that they are tapping into distinct but correlated subconstructs.

## 4.6 EFA Step 4: Comparing to other factor solutions

To understand if the three-factor model makes the most sense, it is typical to also estimate an EFA with one factor less and one factor more to see if those analyses result in more clearly interpretable results. Let's start by estimating a two-factor EFA.

### 4.6.1 Two-Factor EFA

```
efa_2f <- fa(HSdata, nfactors = 2,
             fm = "minres",
             rotate = "oblimin")
```

### 4.6.2 Interpreting the results of the Two-Factor EFA

```
round(efa_2f$communalities,
      digits = 3)
```

```
   x1    x2    x3    x4    x5    x6    x7    x8    x9
0.341 0.100 0.223 0.728 0.708 0.705 0.179 0.381 0.545
```

Many of the communalities are low, indicating that this factor solution does not do a good job of accounting for variability in the items.

```
print(efa_2f$loadings,
      cutoff = .3)
```

```
Loadings:
   MR1    MR2
x1          0.430
x2
x3          0.449
x4  0.851
x5  0.854
x6  0.828
x7          0.434
x8          0.640
x9          0.736

                 MR1   MR2
SS loadings    2.244 1.588
Proportion Var 0.249 0.176
Cumulative Var 0.249 0.426
```

Although there are no cross-loadings, one item (x2) doesn't have a loading > .3 on either of the factors! These two factors cannot capture the variance in x2 that is common with the other items.

```
round(efa_2f$Phi,
      digits = 3)
```

```
     MR1  MR2
MR1 1.00 0.34
MR2 0.34 1.00
```

The factor correlation does not indicate any issues.

### 4.6.3 Four-Factor EFA

```
efa_4f <- fa(HSdata, nfactors = 4,
             fm = "minres",
             rotate = "oblimin")
```

#### Interpreting the results of the Four-Factor EFA

```
round(efa_4f$communalities,
      digits = 3)
```

```
   x1    x2    x3    x4    x5    x6    x7    x8    x9
0.454 0.230 0.554 0.740 0.787 0.687 0.995 0.424 0.568
```

Compared to the three-factor EFA, the communalities have not changed a lot, except for the communality of x7, which is now a whopping .995. Such a high communality indicates that there is a factor (or combination of factors) that can account for almost all variability in x7. Although this may sound good, it may stand in the way of our goal of dimension reduction (as we'll see next).

```
print(efa_4f$loadings,
      cutoff = .3)
```

```
Loadings:
   MR1    MR2    MR3    MR4
x1                 0.453
x2                 0.397
x3                 0.735
x4  0.850
x5  0.887
x6  0.804
x7         0.986
x8                        0.499
x9                        0.674

                 MR1   MR2   MR3   MR4
SS loadings    2.216 1.089 0.954 0.787
Proportion Var 0.246 0.121 0.106 0.087
Cumulative Var 0.246 0.367 0.473 0.561
```

When asked to estimate four factors, the EFA algorithm resulted in a factor that only represents one item (x7). This one-to-one association gets in the way of our goal of dimension reduction, and is an indication that this factor solution is not appropriate.

```
round(efa_4f$Phi, digits = 3)
```

```
      MR1   MR2   MR3   MR4
MR1 1.000 0.122 0.250 0.286
MR2 0.122 1.000 0.046 0.417
MR3 0.250 0.046 1.000 0.436
MR4 0.286 0.417 0.436 1.000
```

Factor correlations indicate that there is no extremely strong correlation ($r = .417$) between the factor that represents x7 and the factor that represents x8 and x9 (these three items are hypothesized to measure one subconstruct: speed). This indicates that, although these three items share some common variance, they also tap into distinct sub-subconstructs that may need to be explored further.

## 4.7 Some Final Conclusions

For this sample, a three-factor solution appeared to best balance dimension reduction and representing the associations among the observed variables. However, the results

did indicate that there may be an issue with x2 (low communality) and x9 (cross loading). In addition, the four-factor EFA seemed to indicate that the three items measuring speed are not as related as we'd hoped they'd be. A second sample could indicate whether these findings were due to sampling variability or whether they reflect true issues that need to be resolved.

## 4.8 Summary

In this R lab, you learned how to specify, estimate, evaluate and interpret EFAs. You also learned how to evaluate different sources of information about the appropriateness of the EFA solutions.

# 5 Reliability

## 5.1 Installing and Loading the R packages

In this lab we will be using a new package: `MBESS` (Methods for the Behavioral, Educational, and Social Sciences), which includes a function that calculates coefficient omega and (more importantly) its confidence interval. Although other packages can also estimate coefficient omega, they often do not provide a confidence interval.

```
install.packages("MBESS")
```

> Tip
>
> If you experience issues installing this package on macOS, you likely need to install a few additional tools. Go to this page to download and install those tools: Compile Tools for macOS.

In this lab, we will also use the `psych` package, which we have already installed in earlier labs. So, we can simply get those packages from the R package library:

```
library(psych)
library(MBESS)
```

## 5.2 Loading data into our environment

For this lab, we will use a dataset that is included in the `psych` package, so we can use the `data()` function like we did before:

```
data("attitude")
```

These data come from a survey of clerical employees of a large financial organization. Each variable represents a rating (on the percentage scale) of how well the company performs on that item's topic (e.g., complaints).

```
describe(attitude)
```

```
           vars  n  mean    sd median trimmed   mad min max range  skew
rating        1 30 64.63 12.17   65.5   65.21 10.38  40  85    45 -0.36
complaints    2 30 66.60 13.31   65.0   67.08 14.83  37  90    53 -0.22
privileges    3 30 53.13 12.24   51.5   52.75 10.38  30  83    53  0.38
learning      4 30 56.37 11.74   56.5   56.58 14.83  34  75    41 -0.05
raises        5 30 64.63 10.40   63.5   64.50 11.12  43  88    45  0.20
critical      6 30 74.77  9.89   77.5   75.83  7.41  49  92    43 -0.87
advance       7 30 42.93 10.29   41.0   41.83  8.90  25  72    47  0.85
           kurtosis   se
rating        -0.77 2.22
complaints    -0.68 2.43
privileges    -0.41 2.23
learning      -1.22 2.14
raises        -0.60 1.90
critical       0.17 1.81
advance        0.47 1.88
```

## 5.3 Are the items tau-equivalent?

When determining which internal consistency coefficient may be most appropriate for our measurement instrument, we can look at whether the items are tau equivalent (equivalent factor loadings for all items). A simple way to do so is the run a unidimensional EFA and inspect the factor loadings:

```
fa(attitude)$loadings
```

```
Loadings:
           MR1
rating     0.758
complaints 0.834
privileges 0.603
```

```
learning   0.789
raises     0.841
critical   0.284
advance    0.491

                 MR1
SS loadings    3.285
Proportion Var 0.469
```

Based on the loadings, do you think the items are tau equivalent? What does this mean for our choice of internal consistency coefficient?

## 5.4 Coefficient Omega

We will use the MBESS package to compute coefficient omega:

```
ci.reliability(attitude, type = "omega",
               conf.level = 0.95,
               interval.type = "mlr")
```

```
$est
[1] 0.8563268

$se
[1] 0.04619404

$ci.lower
[1] 0.7657882

$ci.upper
[1] 0.9468655

$conf.level
[1] 0.95

$type
[1] "omega"

$interval.type
[1] "robust maximum likelihood (wald ci)"
```

How internally consistent are the scores on this measurement instrument with this sample? Here is how you'd report the reliability: Internal consistency of the Attitudes survey was good, $\omega$ = .86 (SE = .05), 95% CI = [.77, .95].

We can use the estimated internal consistency to get an estimate of the overall standard error of measurement (sem):

```
# compute the SD of the attitude sum scores
sd_x <- sd(rowSums(attitude))

# compute sem: sd_x * sqrt(1 - reliability)
sem <- sd_x * sqrt(1 - 0.8563268)
sem
```

```
[1] 21.88914
```

So, the average size of the error scores is 21.89.

## 5.5 Cronbach's Alpha

We will use the psych package to compute Cronbach's alpha.

```
attitude_alpha <- alpha(attitude)
```

```
Number of categories should be increased  in order to count frequencies.
```

This function returns a bunch of output that we can look at, starting with some basic summary statistics:

```
summary(attitude_alpha)
```

```
Reliability analysis
 raw_alpha std.alpha G6(smc) average_r S/N   ase mean  sd median_r
      0.84      0.84    0.88      0.43 5.2 0.042   60 8.2     0.45
```

We can also look at how Cronbach's alpha would change if specific items were removed from the instrument. This can help us identify items that are measured with more measurement error:

```
attitude_alpha$alpha.drop
```

```
           raw_alpha std.alpha   G6(smc) average_r      S/N   alpha se
rating     0.8097602 0.8081915 0.8317701 0.4125442 4.213534 0.05244967
complaints 0.7969175 0.7956468 0.8201749 0.3935404 3.893487 0.05653049
privileges 0.8278478 0.8230879 0.8661986 0.4367533 4.652525 0.04757251
learning   0.8030310 0.7983665 0.8367262 0.3975597 3.959493 0.05429802
raises     0.7953866 0.7847196 0.8261984 0.3779228 3.645105 0.05558872
critical   0.8638723 0.8634716 0.8900481 0.5131641 6.324481 0.03839722
advance    0.8404649 0.8346265 0.8563876 0.4568621 5.046918 0.04287642
                 var.r     med.r
rating     0.03484463 0.4454779
complaints 0.03454755 0.4261169
privileges 0.05381799 0.5316198
learning   0.04457495 0.3768830
raises     0.04790794 0.3432934
critical   0.03015342 0.5582882
advance    0.04811380 0.4933310
```

And finally, we can find the 95% CI:

```
attitude_alpha$feldt
```

```
  95% confidence boundaries (Feldt)
lower alpha upper
 0.74  0.84  0.92
```

Here is how you'd report the internal consistency using Cronbach's alpha: Internal consistency of the Attitudes survey was good, $\alpha$ = .84 (SE = .04), 95% CI = [.74, .92].

Note that you can also use the MBESS package to compute Cronbach's alpha:

```
ci.reliability(attitude, type = "alpha",
               conf.level = 0.95,
               interval.type = "feldt")
```

```
$est
[1] 0.8431428

$se
[1] NA

$ci.lower
[1] 0.7393757

$ci.upper
[1] 0.9157731

$conf.level
[1] 0.95

$type
[1] "alpha"

$interval.type
[1] "feldt"
```

## 5.6 Split-Half Reliability

We can also use the psych package to estimate the split-half reliability. This function returns (among some other things) the minimum, maximum, and average split-half reliability. Ideally, we want those numbers to be close together and close to 1. If the minimum and maximum are far apart, it indicates that only some specific splits can be considered essentially parallel.

```
splitHalf(attitude)
```

```
Split half reliabilities
Call: splitHalf(r = attitude)

Maximum split half reliability (lambda 4) =  0.89
Guttman lambda 6                          =  0.88
Average split half reliability            =  0.82
Guttman lambda 3 (alpha)                  =  0.84
Guttman lambda 2                          =  0.85
Minimum split half reliability  (beta)    =  0.68
```

```
Average interitem r =  0.43  with median =  0.45
```

## 5.7 Disattenuation of correlations

As discussed in class, when two measures are not perfectly reliable, then the correlation between them will be biased, or attenuated (i.e., lower than it should be). In this part of the lab, we'll see this phenomenon in action.

First, we'll split the attitude survey in two parts, so we can look at the correlation between the two parts. In Assignment 6, you will do something similar, but for two different surveys. However, you will still need to split the Assignment 6 data frame into two parts, so the code below is still relevant.

```
# Split attitude data into two parts
# for demonstration
part1 <- attitude[,c(1:4)]
part2 <- attitude[,c(5:7)]
```

To compute a correlation between the two tests, we need to compute each participant's sumscore across the items. We can use a function called `rowSums()` to do this:

```
# Compute summed scores of
# each part using rowSums()
sumscore1 <- rowSums(part1)
sumscore2 <- rowSums(part2)
```

Now, we can compute the observed correlation of the summed scores:

```
# Compute correlation between summed scores
obscor <- cor(sumscore1,sumscore2)
obscor
```

```
[1] 0.5438004
```

But we already know from our earlier assessment above that the full attitude test is not perfectly reliable. Now we also need to see if these two parts are perfectly reliable or not. To decide between using Cronbach's alpha and coefficient omega, we need to assess the (lack of) tau equivalence of the two parts:

```
#examine tau equivalence
fa(part1)$loadings
```

```
Loadings:
           MR1
rating     0.855
complaints 0.920
privileges 0.590
learning   0.713

                 MR1
SS loadings    2.434
Proportion Var 0.608
```

```
fa(part2)$loadings
```

```
Loadings:
         MR1
raises   0.874
critical 0.431
advance  0.657

                 MR1
SS loadings    1.381
Proportion Var 0.460
```

These loadings do not look tau equivalent, so we will use coefficient omega to quantify the internal consistency. This time, we're using the `$` operator to extract just the omega estimate (`est`) from the `ci.reliability()` output:

```
#record omega reliability estimates of both parts
omega1 <- ci.reliability(part1)$est
omega2 <- ci.reliability(part2)$est

omega1
```

```
[1] 0.8615586
```

```
omega2
```

```
[1] 0.7097988
```

The omegas above show that the two tests are not perfectly reliable, so it's important to disattenuate the observed correlation:

```
# Disattenuated correlation between tests
discor <- obscor / sqrt(omega1 * omega2)
discor
```

```
[1] 0.6953916
```

> **Note**
>
> A nice feature of CFA (or structural equation modeling more generally) is that correlations between factors are disattenuated for (lack of) reliability, because the factors only represent the `true score` part of the item's variability, while the `error` variance is separated into the residual or error variance of the indicators.

## 5.8 Summary

In this R lab, you learned how to determine whether a set of items are tau-equivalent and how to compute coefficient omega and Cronbach's alpha to evaluate internal consistency reliability. You also learned how to get the split-half reliability. Finally, you used the dissatenuation formula to dissatenuate a correlation for measurement error.

# 6 Measurement Invariance

## 6.1 Loading the R packages

In this lab, we will use the lavaan, semTools, and ggplot2 packages, which we have already installed in earlier labs. So, we can simply get those packages from the R package library:

```
library(lavaan)
library(semTools)
library(ggplot2)
```

## 6.2 Loading data into our environment

For this lab, we will use a dataset that contains 3811 item responses to 10 items about financial well-being. You can download the data by right-clicking this link and selecting "Save Link As..." in the drop-down menu: data/finance.csv. Make sure to save it in the folder you are using for this class.

```
finance <- read.csv("data/finance.csv")
```

We can look at the variables in the dataset using `describe()` from the psych package. In addition to the 10 items, the dataset also includes a variable denoting whether a participant worked in the public or private sector.

```
psych::describe(finance, skew = FALSE)
```

```
      vars    n mean   sd median min max range   se
item1    1 3811 2.89 1.23      3   1   5     4 0.02
item2    2 3811 3.09 1.10      3   1   5     4 0.02
item3    3 3811 2.69 1.19      3   1   5     4 0.02
item4    4 3811 3.15 1.04      3   1   5     4 0.02
```

```
item5       5 3811 2.87 1.24      3   1   5     4 0.02
item6       6 3811 3.25 1.13      3   1   5     4 0.02
item7       7 3811 2.48 1.19      2   1   5     4 0.02
item8       8 3811 3.27 1.25      3   1   5     4 0.02
item9       9 3811 2.22 1.14      2   1   5     4 0.02
item10     10 3811 2.83 1.13      3   1   5     4 0.02
sector*    11 3811 1.55 0.50      2   1   2     1 0.01
```

These items make up the Financial Well-Being scale, which was developed by the Consumer Financial Protection Bureau (CFPB):

1. I could handle a major unexpected expense (P)
2. I am securing my financial future (P)
3. Because of my money situation, I will never have the things I want in life (N)
4. I can enjoy life because of the way I'm managing my money (P)
5. I am just getting by financially (N)
6. I am concerned that the money I have or will save won't last (N)
7. Giving a gift would put a strain on my finances for the month (N)
8. I have money left over at the end of the month (P)
9. I am behind with my finances (N)
10. My finances control my life (N)

The items measure positive (P) and negative (N) financial well-being.

## 6.3 The Theoretical Measurement Model

Next, we will set up the theoretical measurement model representing the hypothesized internal structure of this measure. In this case, the construct of financial well-being is represented by two correlated sub-constructs: financial stability and financial instability.

```
# Two-factor CFA model
financemodel <- "positive =~ item1 + item2 + item4 + item8
                 negative =~ item3 + item5 + item6 + item7 + item9 + item10"
```

## 6.4 Step 1: Configural Invariance

To test measurement invariance (MI), we will use the `lavaan` package. First, we will assess configural invariance, which we can do using the code below. The main difference from

previous CFAs is that we now use the `group = "sector"` argument to denote which variable in our data denotes what group a participant is a member of. In addition, we will tell lavaan to scale the latent factors so they have a mean of 0 and a standard deviation of 1, using the `std.lv = TRUE` argument (note that there are other methods for scaling the latent variable, but going into those technical details is beyond the scope of this class):

```
# Configural model
config <- cfa(model = financemodel, data = finance,
              std.lv = TRUE,
              group = "sector")
```

To evaluate whether configural invariance holds, we can look at the summary output. In this case, I'm mostly interested in looking at the fit of the model to the data, so we will use `estimates = F` to omit the parameter estimates from the summary output.

```
summary(config, fit.measures = T, estimates = F)
```

```
lavaan 0.6.17 ended normally after 27 iterations

  Estimator                                         ML
  Optimization method                           NLMINB
  Number of model parameters                        62

  Number of observations per group:
    public                                        2110
    private                                       1701

Model Test User Model:

  Test statistic                               875.170
  Degrees of freedom                                68
  P-value (Chi-square)                           0.000
  Test statistic for each group:
    public                                     465.024
    private                                    410.146

Model Test Baseline Model:

  Test statistic                             20856.369
  Degrees of freedom                                90
  P-value                                        0.000
```

```
User Model versus Baseline Model:

  Comparative Fit Index (CFI)                    0.961
  Tucker-Lewis Index (TLI)                       0.949

Loglikelihood and Information Criteria:

  Loglikelihood user model (H0)              -49649.240
  Loglikelihood unrestricted model (H1)      -49211.655

  Akaike (AIC)                                99422.480
  Bayesian (BIC)                              99809.711
  Sample-size adjusted Bayesian (SABIC)       99612.704

Root Mean Square Error of Approximation:

  RMSEA                                          0.079
  90 Percent confidence interval - lower         0.074
  90 Percent confidence interval - upper         0.084
  P-value H_0: RMSEA <= 0.050                    0.000
  P-value H_0: RMSEA >= 0.080                    0.359

Standardized Root Mean Square Residual:

  SRMR                                           0.031
```

Overall, the fit of the model to the data, allowing all parameters to be freely estimated across groups, is decent: CFI > .95, TLI = .95, RMSEA = .079, 95% CI [.074, .084] SRMR < .08. Not ideal in terms of RMSEA, but otherwise okay. More importantly, this output shows us what part of the overall Model Chi-square is stemming from each of the two groups (under `Test statistic for each group`). If those two numbers are relatively equal, then the model fits about equally well to each group's data. If you notice that the Chi-square contribution is much larger for one group than the other, it is an indication that you may not be able to conclude that there is configural invariance. In this case, both groups contribute about equally to the total Chi-square, indicating similar model-data fit across groups.

### 6.4.1 If configural invariance is not supported

If configural invariance is not supported, then you can start your investigation by looking at the estimates across each group to see if there are noticeable issues (e.g., a lot of low

factor loadings in one group). You can follow this investigation up by running separate EFAs for each group, to examine what kind of factor structure emerges for each group. The code below shows you how to start this investigation for this example dataset, but I do not include the output to reduce the length of this (already lengthy) lab:

```
# Examine parameter estimates
summary(config)

# Split data is two
public <- subset(finance, sector == "public")
private <- subset(finance, sector == "private")

# Run parallel analysis for each group
# (can be followed up by full EFA examination)
library(psych)
fa.parallel(public[,1:10], fa = "fa")
fa.parallel(private[,1:10], fa = "fa")
```

## 6.5 Step 2: Metric Invariance

Next, we will estimate the metric invariance model. To do so, we again use the `cfa()` function and specify our grouping variable and latent factor scale. In addition, we will add `group.equal = "loadings"`:

```
# Metric model
metric <- cfa(model = financemodel, data = finance,
              group = "sector",
              std.lv = TRUE,
              group.equal = "loadings")
```

To see if the metric model fits the data significantly worse than the configural model, we will use the `compareFit()` function from the `semTools` package:

```
summary(compareFit(config, metric))
```

```
################### Nested Model Comparison #########################

Chi-Squared Difference Test
```

```
       Df   AIC   BIC  Chisq Chisq diff    RMSEA Df diff Pr(>Chisq)
config 68 99422 99810 875.17
metric 76 99417 99754 885.35     10.179 0.011957       8     0.2527

####################### Model Fit Indices ###########################
         chisq df pvalue rmsea   cfi   tli  srmr       aic       bic
config 875.170† 68   .000 .079  .961† .949  .031† 99422.480  99809.711
metric 885.349  76   .000 .075† .961  .954† .033  99416.660† 99753.925†

################## Differences in Fit Indices ######################
                df  rmsea cfi   tli  srmr    aic     bic
metric - config  8 -0.004   0 0.005 0.002 -5.821 -55.786
```

The null hypothesis that we're testing is that the fit of the metric and configural model is equivalent (i.e., adding equality constraints to the loadings does not make the model fit worse). Thus, if the *p*-value associated with the Chi-square difference test is > .05, we can retain that null hypothesis and conclude that metric invariance holds for these data. If the *p*-value associated with the Chi-square difference test is < .05, then we need to reject the null hypothesis and conclude that the metric invariance model fit the data significantly worse, and so metric invariance does not hold.

Note: If any loadings were found to be non-invariant (i.e., there is partial metric invariance), then the intercepts of those items also need to be estimated freely across groups in the next step. In other words, you start your investigation with a model that is already partially invariant at the scalar level (see below how to run partial invariance models).

## 6.6 Step 3: Scalar Invariance

Next, we will evaluate if scalar invariance holds for our data. To do so, we simply add the intercepts to the `group.equal = c("loadings","intercepts")` argument:

```r
# Scalar model
scalar <- cfa(model = financemodel, data = finance,
              group = "sector",
              std.lv = TRUE,
              group.equal = c("loadings","intercepts"))
```

Now, we compare the scalar model's fit to the fit of the metric model, to see if adding the equality constraints on the intercepts results in significantly worse fit:

```r
summary(compareFit(metric, scalar))
```

```
################## Nested Model Comparison #########################

Chi-Squared Difference Test

       Df   AIC   BIC  Chisq Chisq diff    RMSEA Df diff Pr(>Chisq)
metric 76 99417 99754 885.35
scalar 84 99453 99740 937.72     52.371 0.053951       8  1.427e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

####################### Model Fit Indices ##########################
         chisq df pvalue rmsea   cfi   tli  srmr       aic       bic
metric 885.349† 76   .000 .075  .961† .954  .033† 99416.660† 99753.925
scalar 937.720  84   .000 .073† .959  .956† .034  99453.030  99740.330†

################## Differences in Fit Indices ######################
                df  rmsea    cfi   tli  srmr    aic     bic
scalar - metric  8 -0.002 -0.002 0.002 0.001 36.371 -13.595
```

From the Chi-square difference test above, we can see that the scalar model fit the data significantly worse than the metric model. This means that, at least for some items, the expected response for those with an average (i.e., 0) score on the latent factor differs across groups.

### 6.6.1 Step 3B: Partial Scalar Invariance

To easily get an overview of what equality constraints should be released to result in the largest improvement in model fit, we can use the `lavTestScore` function from the `lavaan` package:

```r
# Adjust the model for partial invariance testing
lavTestScore(scalar)
```

```
$test

total score test:
```

```
   test     X2 df p.value
1 score 62.622 20       0

$uni

univariate score tests:

     lhs op   rhs     X2 df p.value
1   .p1. == .p36.  0.125  1   0.724
2   .p2. == .p37.  0.594  1   0.441
3   .p3. == .p38.  0.323  1   0.570
4   .p4. == .p39.  0.327  1   0.567
5   .p5. == .p40.  1.733  1   0.188
6   .p6. == .p41.  1.909  1   0.167
7   .p7. == .p42.  1.219  1   0.270
8   .p8. == .p43.  2.986  1   0.084
9   .p9. == .p44.  2.129  1   0.144
10 .p10. == .p45.  0.011  1   0.916
11 .p24. == .p59. 15.296  1   0.000
12 .p25. == .p60.  0.215  1   0.643
13 .p26. == .p61. 16.963  1   0.000
14 .p27. == .p62.  0.257  1   0.612
15 .p28. == .p63.  9.145  1   0.002
16 .p29. == .p64.  0.445  1   0.505
17 .p30. == .p65.  0.101  1   0.751
18 .p31. == .p66. 21.472  1   0.000
19 .p32. == .p67.  1.433  1   0.231
20 .p33. == .p68.  4.348  1   0.037
```

To figure out what equality constraints map onto which item's intercept, we can look at the parameter table of the scalar model. In the code below, I filter the output to only include intercept parameters (`op = "~1"`), only show parameters from group 2 (`group == 2`) and then to only include certain columns and rows of that output. This was mostly done to keep this PDF from becoming too large. You don't need to do any of this filtering yourself.

```
# To view the entire parameter table, simply use this code
# (remove the hashtag in front of the next line):
# parTable(scalar)

# To filter the output (this literal code will only work for
```

```
# this example):
subset(parTable(scalar), op == "~1" & group == 2)[,c(1:4, 11:15)]
```

```
   id      lhs op rhs label plabel start    est    se
59 59    item1 ~1     .p24.  .p59. 2.733  2.994 0.025
60 60    item2 ~1     .p25.  .p60. 2.978  3.173 0.022
61 61    item4 ~1     .p26.  .p61. 3.093  3.232 0.021
62 62    item8 ~1     .p27.  .p62. 3.162  3.370 0.025
63 63    item3 ~1     .p28.  .p63. 2.708  2.638 0.024
64 64    item5 ~1     .p29.  .p64. 2.931  2.820 0.024
65 65    item6 ~1     .p30.  .p65. 3.301  3.203 0.022
66 66    item7 ~1     .p31.  .p66. 2.600  2.432 0.024
67 67    item9 ~1     .p32.  .p67. 2.256  2.174 0.022
68 68   item10 ~1     .p33.  .p68. 2.855  2.785 0.022
69 69 positive ~1            .p69. 0.000 -0.217 0.035
70 70 negative ~1            .p70. 0.000  0.121 0.035
```

In the table above, we can see the the intercept of Item 7 is associated with the largest potential improvement in model fit. So, we estimate a partial scalar invariance model. To release the equality constraint for the intercept of Item 7, we add the `group.partial = c("item7 ~ 1")` argument. Once the model is estimated, we will compare its fit to the metric model, to see if we need to release additional intercept parameters:

```
scalar2 <- cfa(model = financemodel, data = finance,
               group = "sector",
               std.lv = TRUE,
               group.equal = c("loadings","intercepts"),
               group.partial = c("item7 ~ 1"))
# for a loading, group.partial would look like: "negative =~ item7"

summary(compareFit(metric, scalar2))
```

```
################### Nested Model Comparison #########################

Chi-Squared Difference Test

        Df   AIC   BIC  Chisq Chisq diff    RMSEA Df diff Pr(>Chisq)
metric  76 99417 99754 885.35
scalar2 83 99434 99727 916.21    30.863 0.042297       7   6.59e-05 ***
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

####################### Model Fit Indices ###########################
         chisq df pvalue rmsea   cfi   tli  srmr       aic       bic
metric  885.349† 76   .000 .075  .961† .954  .033† 99416.660† 99753.925
scalar2 916.212  83   .000 .073† .960  .956† .034  99433.523  99727.068†

################## Differences in Fit Indices #######################
                 df  rmsea    cfi   tli  srmr    aic     bic
scalar2 - metric  7 -0.002 -0.001 0.003 0.001 16.863 -26.857
```

The fit of the first partial scalar model is still worse than the metric model. So, we need to release additional equality constraints. After examining the modification indices again, we release the equality constraint of Item 4's intercept, estimate the model again, and compare this second partial model to the metric model:

```
# Adjust the model for partial invariance testing
lavTestScore(scalar2)
```

```
$test

total score test:

   test     X2 df p.value
1 score 41.208 19   0.002

$uni

univariate score tests:

     lhs op   rhs     X2 df p.value
1   .p1. == .p36.  0.125  1   0.723
2   .p2. == .p37.  0.594  1   0.441
3   .p3. == .p38.  0.323  1   0.570
4   .p4. == .p39.  0.327  1   0.567
5   .p5. == .p40.  2.131  1   0.144
6   .p6. == .p41.  1.823  1   0.177
7   .p7. == .p42.  1.128  1   0.288
8   .p8. == .p43.  1.924  1   0.165
9   .p9. == .p44.  1.885  1   0.170
10 .p10. == .p45.  0.045  1   0.832
```

```
11 .p24. == .p59. 15.297  1   0.000
12 .p25. == .p60.  0.215  1   0.643
13 .p26. == .p61. 16.964  1   0.000
14 .p27. == .p62.  0.257  1   0.612
15 .p28. == .p63.  2.901  1   0.089
16 .p29. == .p64.  2.900  1   0.089
17 .p30. == .p65.  1.991  1   0.158
18 .p32. == .p67.  0.000  1   0.989
19 .p33. == .p68.  0.840  1   0.359
```

```r
scalar3 <- cfa(model = financemodel, data = finance,
               group = "sector",
               std.lv = TRUE,
               group.equal = c("loadings","intercepts"),
               group.partial = c("item7 ~ 1", "item4 ~ 1"))

summary(compareFit(metric, scalar3))
```

```
################### Nested Model Comparison #########################

Chi-Squared Difference Test

        Df   AIC   BIC  Chisq Chisq diff   RMSEA Df diff Pr(>Chisq)
metric  76 99417 99754 885.35
scalar3 82 99418 99718 899.16     13.812 0.02614       6     0.0318 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

####################### Model Fit Indices ###########################
         chisq df pvalue rmsea   cfi   tli  srmr       aic       bic
metric  885.349† 76   .000 .075  .961† .954  .033† 99416.660† 99753.925
scalar3 899.161  82   .000 .072† .961  .957† .033  99418.472  99718.263†

################## Differences in Fit Indices #######################
                df  rmsea cfi   tli srmr   aic     bic
scalar3 - metric  6 -0.002   0 0.003    0 1.812 -35.661
```

The fit of this partial model is still worse than the metric model. So, we need to release additional equality constraints. After examining the modification indices again, we release the equality constraint of Item 1's intercept, estimate the model again, and compare this second partial model to the metric model:

```
# Adjust the model for partial invariance testing
lavTestScore(scalar3)
```

```
$test

total score test:

   test     X2 df p.value
1 score 24.243 18   0.147

$uni

univariate score tests:

     lhs op   rhs    X2 df p.value
1   .p1. == .p36. 0.276  1   0.599
2   .p2. == .p37. 0.946  1   0.331
3   .p3. == .p38. 1.352  1   0.245
4   .p4. == .p39. 0.127  1   0.722
5   .p5. == .p40. 2.132  1   0.144
6   .p6. == .p41. 1.826  1   0.177
7   .p7. == .p42. 1.128  1   0.288
8   .p8. == .p43. 1.919  1   0.166
9   .p9. == .p44. 1.883  1   0.170
10 .p10. == .p45. 0.046  1   0.830
11 .p24. == .p59. 6.551  1   0.010
12 .p25. == .p60. 0.744  1   0.388
13 .p27. == .p62. 3.488  1   0.062
14 .p28. == .p63. 2.901  1   0.089
15 .p29. == .p64. 2.900  1   0.089
16 .p30. == .p65. 1.991  1   0.158
17 .p32. == .p67. 0.000  1   0.989
18 .p33. == .p68. 0.840  1   0.359
```

```
scalar4 <- cfa(model = financemodel, data = finance,
               group = "sector",
               std.lv = TRUE,
               group.equal = c("loadings","intercepts"),
               group.partial = c("item7 ~ 1", "item4 ~ 1",
                                 "item1 ~ 1"))
```

```
summary(compareFit(metric, scalar4))
```

```
################## Nested Model Comparison #########################

Chi-Squared Difference Test

        Df   AIC   BIC  Chisq Chisq diff    RMSEA Df diff Pr(>Chisq)
metric  76 99417 99754 885.35
scalar4 81 99414 99720 892.61     7.2647 0.015418       5     0.2017

####################### Model Fit Indices ##########################
          chisq df pvalue rmsea   cfi   tli  srmr       aic       bic
metric  885.349† 76   .000 .075  .961† .954  .033† 99416.660  99753.925
scalar4 892.614  81   .000 .073† .961  .957† .033  99413.925† 99719.961†

################## Differences in Fit Indices ######################
                 df  rmsea cfi   tli srmr    aic     bic
scalar4 - metric  5 -0.002   0 0.003    0 -2.735 -33.964
```

This time, the fit of the partial scalar model is not worse than that of the metric model ($p$ > .05), so we can conclude that partial scalar invariance holds for these data.

## 6.7 Step 4 Strict Invariance (Optional)

Finally, we can examine strict invariance by constraining the residuals to be equal across groups. To do so, we simply add the residuals to the `group.equal = c("loadings","intercepts","residuals)` argument. Note that we keep the partial intercepts from the previous step and need to add partial residuals for those items.

```
#Strict model
strict <- cfa(model = financemodel, data = finance,
              group = "sector",
              std.lv = TRUE,
              group.equal = c("loadings","intercepts","residuals"),
              group.partial = c("item7 ~ 1", "item4 ~ 1",
                                "item1 ~ 1",
                                "item7 ~~ item7",
                                "item4 ~~ item4",
```

```
                                        "item1 ~~ item1"))
```

Similar to previous steps, we can compare the fit of this model to the previous (partial scalar) model:

```
summary(compareFit(scalar4, strict))
```

```
################### Nested Model Comparison #########################

Chi-Squared Difference Test

        Df   AIC   BIC  Chisq Chisq diff    RMSEA Df diff Pr(>Chisq)
scalar4 81 99414 99720 892.61
strict  88 99423 99686 915.96     23.351 0.035012       7    0.00148 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

####################### Model Fit Indices ###########################
          chisq df pvalue rmsea   cfi   tli  srmr       aic       bic
scalar4 892.614† 81   .000 .073  .961† .957  .033† 99413.925† 99719.961
strict  915.965  88   .000 .070† .960  .959† .033  99423.276  99685.593†

################## Differences in Fit Indices #######################
                df  rmsea    cfi   tli srmr   aic     bic
strict - scalar4  7 -0.002 -0.001 0.003    0 9.351 -34.368
```

## 6.8 Step 5: Interpreting the Mean Difference between Public and Private Sector Groups (Optional)

As we've been able to establish partial scalar invariance, we can compare the latent factor means across groups. To do so, we use the final partial scalar model:

```
subset(parameterEstimates(scalar4),
       (op == "~1" & (lhs == "positive" | lhs == "negative")))
```

```
       lhs op rhs block group label   est    se     z pvalue ci.lower
34 positive ~1        1     1       0.000 0.000    NA     NA    0.000
```

```
35 negative ~1        1     1       0.000 0.000     NA     NA    0.000
69 positive ~1        2     2      -0.216 0.037 -5.823  0.000   -0.289
70 negative ~1        2     2       0.086 0.035  2.425  0.015    0.016
   ci.upper
34    0.000
35    0.000
69   -0.143
70    0.155
```

To help with interpretation, the means of the latent factors in the first group (Public) are fixed to 0 and their variances are fixed to one. This constraint means that the freely estimated latent means in the second group (Private) can be interpreted as "relative to" the first group. In the output, we can see that the positive financial well-being mean is negative (and significant) whereas the negative financial well-being mean is positive (and significant). However, the factors' variances in this group are not equal to 1 (they are slightly smaller) and so these means are not measured on the same scale as the means of the Public sector group. There are several ways to make the means comparable.

### 6.8.1 Step 5: Method 1 for making mean differences comparable

The first is to evaluate whether the latent factor variances can be constrained to equivalence without resulting in worse model fit. This would place both groups' factors on the standardized scale (variance or *sd* = 1) and mean differences can be interpreted in terms of standard deviation units. To test this model, we can use the following code, adding `"lv.variances"` to the `group.equal` argument and then testing whether the resulting model fit significantly worse than the partial scalar model. Note that we do not have to meet the strict invariance level to test this latent variance level of invariance.

```
lvvar <- cfa(model = financemodel, data = finance,
               group = "sector",
               std.lv = TRUE,
               group.equal = c("loadings","intercepts",
                               "lv.variances"),
               group.partial = c("item7 ~ 1", "item4 ~ 1",
                                 "item1 ~ 1"))

summary(compareFit(scalar4, lvvar))
```

```
################### Nested Model Comparison #########################
```

```
Chi-Squared Difference Test

        Df   AIC   BIC  Chisq Chisq diff RMSEA Df diff Pr(>Chisq)
scalar4 81 99414 99720 892.61
lvvar   83 99410 99704 893.05    0.43956     0       2     0.8027

####################### Model Fit Indices ############################
          chisq df pvalue rmsea   cfi   tli  srmr       aic       bic
scalar4 892.614† 81   .000 .073  .961  .957  .033† 99413.925  99719.961
lvvar   893.053  83   .000 .072† .961† .958† .034  99410.364† 99703.910†

################## Differences in Fit Indices ######################
                df  rmsea cfi   tli  srmr   aic     bic
lvvar - scalar4  2 -0.001   0 0.001 0.001 -3.56 -16.052
```

Constraining the latent factor variances to equivalence across groups does not result in significantly worse model fit ($p$ = .803), so we can use the estimates from the `lvvar` output to interpret mean differences between the Public and Private sector groups.

```
subset(parameterEstimates(lvvar),
       (op == "~1" & (lhs == "positive" | lhs == "negative")))
```

```
       lhs op rhs block group label    est    se      z pvalue ci.lower
34 positive ~1        1     1       0.000 0.000     NA     NA    0.000
35 negative ~1        1     1       0.000 0.000     NA     NA    0.000
69 positive ~1        2     2      -0.217 0.037 -5.829  0.000   -0.290
70 negative ~1        2     2       0.087 0.036  2.422  0.015    0.016
   ci.upper
34    0.000
35    0.000
69   -0.144
70    0.157
```

Based on the results above, we can conclude that positive financial well-being is .22 standard deviations ($SE$ = .04) lower for those working in the Private sector compared to those working in the Public section ($p$ < .001). In addition, negative financial well-being is .09 standard deviations ($SE$ = .04) higher for those working in the Private sector compared to those working in the Public section ($p$ = .015).

### 6.8.2 Step 5: Method 2 for making mean differences comparable

If the previous method would have shown that factor variances are not comparable, then we could have used the formula for standardized mean differences (Cohen's *D*) to compute mean differences.

$$Cohen'sD = \frac{(m_1 - m_2)}{\sqrt{\sigma^2}}$$

To do so, we need the latent factor means and variances from the partial scalar model. We already know that the means and variances in the first group are 0 and 1 respectively, so we only need to know the means and variances for group 2 (i.e., Private sector):

```
subset(parameterEstimates(scalar4),
       (group == 2 &
          (op == "~1" | op == "~~") & (lhs == "positive" | lhs == "negative")))
```

```
      lhs op     rhs block group label   est    se      z pvalue ci.lower
56 positive ~~ positive    2    2      0.973 0.051  19.042  0.000   0.873
57 negative ~~ negative    2    2      0.967 0.051  18.882  0.000   0.867
58 positive ~~ negative    2    2     -0.818 0.042 -19.583  0.000  -0.899
69 positive ~1             2    2     -0.216 0.037  -5.823  0.000  -0.289
70 negative ~1             2    2      0.086 0.035   2.425  0.015   0.016
   ci.upper
56    1.074
57    1.068
58   -0.736
69   -0.143
70    0.155
```

When variances are not equal across groups, we can compute separate standardized mean differences using each group's variance in the denominator, which will give us a range of plausible mean difference effect sizes (not to be confused with a confidence interval!). Here, we first compute the Cohen's *D* using the first group's variance (= 1):

```
# positive
((-.216) - 0) / sqrt(1)
```

```
[1] -0.216
```

```r
# negative
((.086) - 0) / sqrt(1)
```

```
[1] 0.086
```

Next, we compute the Cohen's *D* using the second group's variance estimates:

```r
# positive
((-.216) - 0) / sqrt(.973)
```

```
[1] -0.2189764
```

```r
# negative
((.086) - 0) / sqrt(.967)
```

```
[1] 0.08745511
```

So, depending on the standardizer (which variance) used, the Cohen's *D* for positive financial well-being is approximately -.217 to -.219, and the Cohen's *D* for negative financial well-being is approximately .086 to .087. Note that the ranges here are really narrow because the variances are so similar, the range would be larger if the difference in the variances was larger.

## 6.9 Summary

In this R lab, you were introduced to the steps involved in measurement invariance testing, an important quantitative method that can help us collect evidence regarding the fairness of our measurement scale. You also learned how to compare means of latent variables, using several different approaches. This is the final R lab of this class!