**Overview – Single Factor CFAs in Mplus or Lavaan**

Today we will:

* Test a single factor CFA
* Calculate McDonald’s omega
* Examine discrepancies to evaluate misfit

**IN MPLUS:**

As always, start by getting your data file ready for Mplus:

* Download “Lab 9 Example Data.csv” from Canvas and save it somewhere you can find it.
* Remove the headers and the participant ID info and save it as “Lab 9 Example Data for Mplus.csv”

The first few lines of your input file will look just like those for an EFA:

TITLE: CFA of Lab 9 Example Data;

DATA: FILE IS Lab 9 Example Data for Mplus.csv;

VARIABLE: NAMES ARE item1-item6;

For a CFA, however, we don’t use the ANALYSIS command. Instead, we use MODEL. In MODEL, we can’t just tell Mplus what to do with a single keyword; instead, we have to specify what our model looks like. For a single factor CFA, it’s pretty simple:

MODEL: factor1 BY item1-item6;

Now, “factor1” is just a variable name – you can call your factor anything you like, just like you can call your items anything you like. If you are working with a large or complex data set, you should try to give your factors descriptive names (“selfesteem”, “jobsat”, etc.). The only rules are that variable names cannot contain spaces and they cannot begin with a number.

We’re also going to request some particular output from Mplus:

OUTPUT: STANDARDIZED;

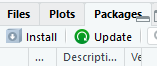
This will give us standardized (interpretable!) estimates for factor loadings, etc.

**IN R:**

As always, save your data as a .csv file (preferably *with* headers) and read it in. My preference is:

data <- read.csv(“Your File Name.csv”, header = TRUE)

For CFAs in R, you’ll need the lavaan package. In the lower right pane of your RStudio window, click on the “Packages” tab and then click “Install”:



Type ‘lavaan’ in the dialog box that pops up, check “Install Dependencies,” and let R install it for you.

Setting up your analysis in lavaan is just a little different than in Mplus. First, make sure to load lavaan:

library(lavaan)

Then, you’ll set up your model. Right now, it’s just a text string, but it needs to be in a specific format:

model.1f <- 'f1 =~ item\_1 + item\_2 + item\_3 + item\_4 + item\_5 + item\_6'

Key pieces to note

* Even though I only have one factor right now, I included “.1f” and called the factor “f1”. You can call your factor and your model whatever you like; I use this notation because it makes it easier later on when I want to compare models.
* The model needs to be inside quotation marks (because it’s text). Single or double quotes are fine – it doesn’t matter.
* The “=~” part is very important – if this is missing or inaccurate, lavaan will not read your model correctly.
* Item names are separated by spaces and “+”, not by commas as we often do elsewhere.

Once you’ve set up your model, to run it, you need the cfa() function. However, the default output from this function isn’t very detailed or helpful, so we generally do this in a two-step process:

fit.model.1f <- cfa(model.1f, data)

If you run this line, it will seem as if R doesn’t do anything in the console, but you’ll have a new (and complex!) object appear in your upper right window called “fit.model.1f.” Again, you could call this whatever you like; “fit” tends to be the conventional name among R users for the result of a model test, so you’ll tend to see it if you are looking at examples online. The argument “data” needs to be the name of *your* data set – if you called it something other than “data”, enter your name for it here.

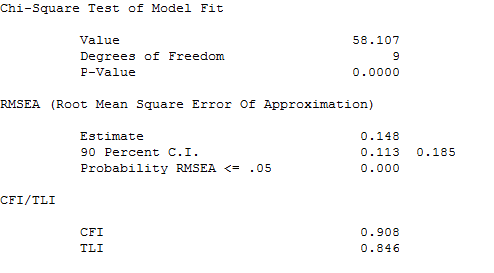
What we’ve done here is ask R to run our CFA on the model we specified, with the data we specified, and then store the results in something called “fit.model.1f”. We can then use other functions within lavaan to pull out the particular results we are interested in. To get results comparable to the Mplus solution, I recommend:

summary(fit.model.1f, fit.measures = TRUE, standardized = TRUE)

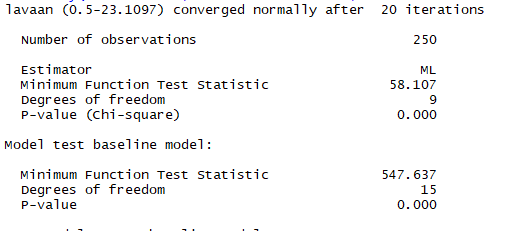
**Interpreting CFA Output**

As lavaan is designed to largely mimic Mplus, you’ll find the output from the two is quite similar. I’ll focus on the Mplus output here, and highlight differences where necessary.

First, we get the model fit information just as it did with EFA:

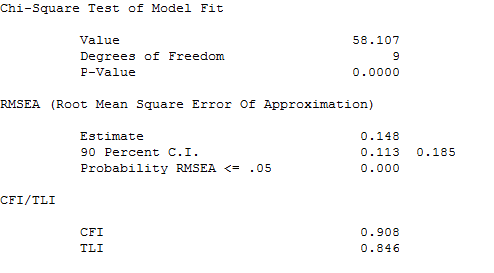


In lavaan, you’ll find this number reported as “Minimum Function Test Statistic” in the first chunk of results. Don’t confuse it with the “Minimum Function Test Statistic” under the heading “Model test baseline model” – that is the chi-square for the worst-case scenario, really unlikely null hypothesis that all of our items are uncorrelated:



You want the one circled in blue, not the one in red!

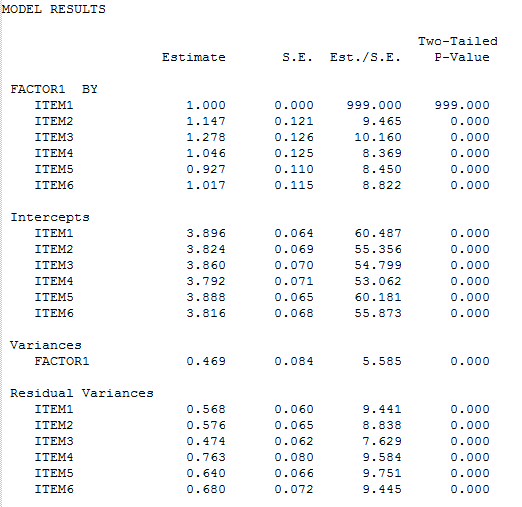
Both software packages will give you, in some order, the goodness-of-fit indices we’ve discussed in class:



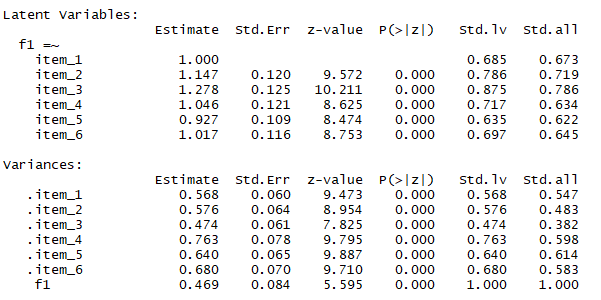
What do you think of these results? Does this model fit well? Why or why not?

Both will also give you several other fit statistics; I’m happy to explain those if you want, but you don’t need to worry about them for now.

Next, let’s look at the parameter estimates. In Mplus:



Or in lavaan:



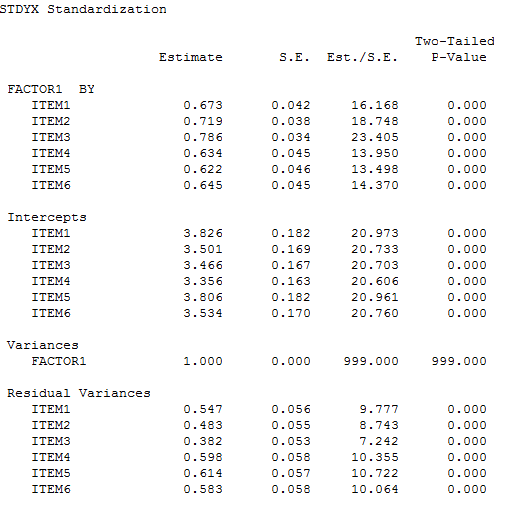
Notice that the first factor loading equals 1.000. Remember that this is an identifiability constraint – necessary for the model to work – and that the Mplus default is to fix the first loading. Can you interpret the other factor loadings?

Also note the p-values – is this information helpful? Why or why not?

Mplus reports *intercepts* – these are the item means. You can calculate the means for yourself and verify this if you like. By default, lavaan does not report them – as discussed in class, they don’t affect the rest of the model, so they are often omitted when reporting results.

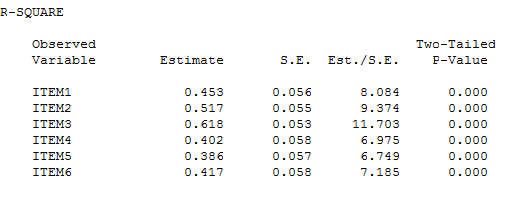
The Residual Variances (Mplus) or Variances (lavaan) are the uniquenesses – the variance in each item that is left over (or residual) after accounting for the common factor.

If you scroll down a little further in Mplus, or over to the right in R, you’ll see more sets of results. These are the standardized solutions we requested in the OUTPUT line of our input file (or the “standardized = TRUE” argument). We are primarily concerned with the STDYX solution in Mplus (labeled the “std.all” column in lavaan). The estimates from this solution are standardized with respect to *both* the variance of the items and the variance of the latent construct. These are identical to the estimates we’d get if we used the correlation matrix instead of the covariance matrix.



Now the factor loadings are on a more interpretable scale – they look like correlations. The factor variance is now equal to 1 (it has been standardized with respect to itself).

There are two other standardized solutions – STDY and STDX. These standardize the variables in different ways, neither of which is really relevant for our purposes. The last section of the output in Mplus is:



Since we have just one factor, these are just the squares of the standardized factor loadings. If we had more than one factor, these values would be adjusted to reflect the relationship between factors. They indicate the proportion of the item variance that is due to the common factor (like communalities in EFA). You can obtain these in lavaan by adding the argument “rsquare = TRUE” in your summary() command.

**Fixing the Factor Variance in Mplus**

As discussed in class, we can *either* fix one factor loading to 1.0 *or* we can fix the factor variance to 1.0. By default, Mplus does the former. If you would prefer to do the latter, you need to make certain changes to your Mplus input. First, you need to add a line to the MODEL section:

MODEL: factor1 BY item1-item6;

factor1@1;

The “@” symbol in Mplus is the symbol for fixing a parameter to a specific value. By default, if the only information you provide about the parameter you want to fix is its name, Mplus assumes you want to fix the variance (not, for example, the mean). If you wanted to fix the mean rather than the variance, you would put this information in brackets: [factor1@0]. This is counterintuitive, until you realize that we are more often interested in fixing variances than means and that the Mplus language is all about efficiency and avoiding entering any more data than is strictly necessary.

So, literally, “factor1@1” tells Mplus to fix the variance of the variable “factor1” at 1. However, Mplus is still set up to fix the first factor loading to 1 by default. In order to specify our model correctly, we have to free that parameter. This means rewriting the first line of our MODEL statement:

MODEL: factor1 BY item1\* item2-item6;

Just as “@” is the Mplus symbol for fixing a parameter, “\*” is the Mplus symbol to request that a parameter be freely estimated. Including this symbol confuses our shorthand listing of item names, though, so now we need to specify that we want Mplus to include items 2 through 6 in addition to the now-free item 1.

Save your edited input file under a different name – this will allow you to compare the output from the two side by side – and run it.

**Fixing the Factor Variance in lavaan**

In lavaan, you follow the same basic process, but the notation looks a little different:

model.1f <- 'f1 =~ NA\*item\_1 + item\_2 + item\_3 + item\_4 + item\_5 + item\_6

f1 ~~ 1\*f1'

Note that everything is all in the *same* set of quotation marks, though it spills over onto two lines.

“~~” is the notation lavaan uses to refer to a covariance – this literally means “set the covariance of f1 with f1 to equal 1.” As a variance is just the covariance of an item with itself, this works. The “NA\*” before item\_1 is equivalent to the \* in Mplus; it means “please estimate a value for this parameter.”

# Comparing Models with Fixed Loadings vs. Fixed Factor Variances

When you look at the output, notice what changes and what doesn’t:

* The fit statistics should be *exactly* the same.
* The factor loadings in the unstandardized solution are quite different.
* The factor loadings in the standardized solution, though, are the same.

Since we will be interpreting the fit statistics and the standardized parameters, it really doesn’t matter which identifiability constraint (fixed factor variance or fixed factor loading) we choose.

**Calculating Omega in Mplus**

Raykov & Marcoulides give a procedure for getting Mplus to calculate omega (Section 7.5.2). This requires slightly more elaborate input, and it can get messy in a hurry if you are dropping items, creating new factors, etc. Omega is not meaningful for a poorly fitting model, because it is directly based on the factor loadings. Therefore, I *strongly recommend that you wait to calculate omega until you have settled on your final model.*

When you are ready, though, the first changes we will make are in the MODEL statement:

MODEL: factor1 BY item1\* (L1)

item2-item6 (L2-L6);

item1-item6 (U1-U6);

factor1@1;

Note that we *need* to use the approach in which we fix the factor variance, and estimate that first factor loading freely, because we’re going to need an accurate estimate of that loading to calculate omega from. The letters in parentheses are *labels* (names) for the parameters. We’ll use these in a minute to refer to specific parameters and tell Mplus to combine them in a particular way.

“item1\* (L1)” tells Mplus to name that factor loading parameter “L1”

When you name parameters like this, the names absolutely must be the *last* thing on the line, so we separate that first line into two:

MODEL: factor1 BY item1\* (L1)

item2-item6 (L2-L6);

Note that there is **no semicolon** at the end of that first line. That tells Mplus to keep reading and treat the two lines as part of the same command. Those two lines together tell Mplus to estimate all of our factor loadings and name them L1 – L6. Then the third line tells Mplus to name the unique variances (remember, a parameter name all by itself means a variance) U1-U6. In the examples, R & M named all their parameters “P” – I have named them “L” and “U” to make it clearer just what we’re referring to.

Again, remember that our formula for omega assumes that the factor variance is equal to 1, so we want to make sure we’ve used that identifiability constraint rather than the first-factor-loading-equal-to-one constraint. Don’t forget to fix your factor variance!

Next, we add a new *section* to our input file. After the end of the MODEL section, we add:

MODEL CONSTRAINT:

NEW(COMP\_REL);

COMP\_REL = (L1+L2+L3+L4+L5+L6)\*\*2/

((L1+L2+L3+L4+L5+L6)\*\*2+U1+U2+U3+U4+U5+U6);

This tells Mplus to create a new parameter, called OMEGA, and then defines OMEGA as the square of the sum of all the loadings divided by the square of the sum of all the loadings plus the sum of all the uniquenesses. Those last 2 lines should look a lot like the formula presented in class!

Again, we’ve broken up the formula into 2 lines with **no semicolon** in between. This is because Mplus only reads ~90 characters per line, and we are pushing that limit in this formula. If you put the whole thing on one line, you run the risk of an error message. Also, this is easier to read. Note that if you drop an item, you will need to remove *all reference* to its parameters in your omega formula, and you’re now referring to that item in several places. This is why it’s a good idea to wait to calculate omega until you have your final preferred model in hand!

Finally, we’ll request one more thing in our OUTPUT section:

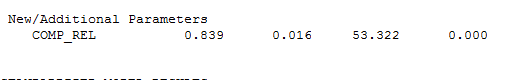
OUTPUT: STANDARDIZED; CINTERVAL;

This will give us confidence intervals around ALL of our estimated parameters, including our new parameter OMEGA.

Save this file (again, with a new name) and run it. In the output, note that:

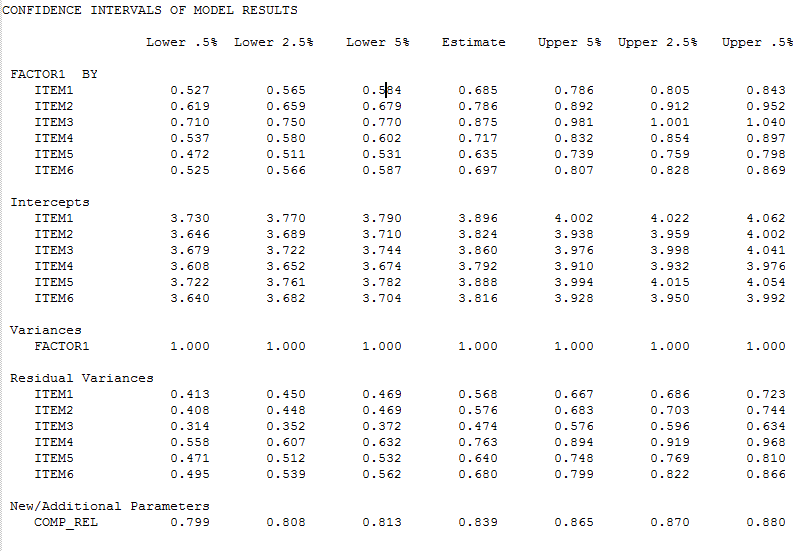
* The fit statistics are exactly the same.
* The unstandardized parameter estimates are exactly the same… except…

At the end of the unstandardized estimates, there is a new line:



This is our estimate of omega!

If you scroll to the end of the standardized solutions (which haven’t changed), you will see something else new, titled “Confidence Intervals of Model Results.” This section provides bootstrap confidence intervals for every estimated parameter (much more useful than significance tests!), including omega:

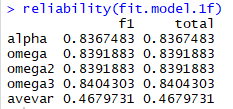


We can now report that the reliability of this test is .84 (90% CI: .81 - .87). This is that *point and interval estimation* that R & M make such a big deal about – we have a specific point estimate (.84) and also a confidence interval around that estimate.

**Calculating Omega in lavaan**

In R, this is vastly easier: install the package semTools and use the reliability() function:

reliability(fit.model.1f)

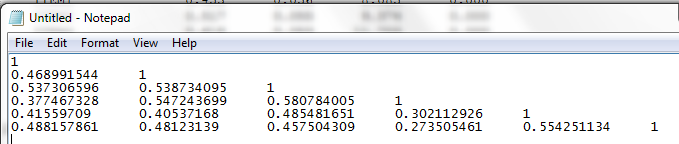


It’s almost like cheating compared to the Mplus version – but we’d have to work a lot harder to get the confidence intervals. If you’d like to know more about the variants of omega this function produces, see here: <https://rdrr.io/cran/semTools/man/reliability.html>

**Running the Analysis from the Correlation Matrix in Mplus**

Our single-factor model doesn’t fit too badly, but it’s hard to tell based on a single set of fit statistics whether it might be better. To evaluate fit in more detail, we need to look at the discrepancy matrix. Doing this in Mplus is a little complicated, because Mplus analyzes the covariance matrix by default and then produces a matrix of *covariance* discrepancies, which are not interpretable. To get interpretable discrepancies, we *have* to analyze the *correlation* matrix.

The only way to do this in Mplus is to give it the correlation matrix rather than the raw data. So, go back to your Excel file and use the Analysis Toolpak to find the correlation matrix for these 6 items (see Lab 2 handout for reminders on how to do this). Copy your correlation matrix (**just** the correlations, no headers, etc!) into a Notepad file, which should look like this:



Save that Notepad file. Now, return to Mplus, and make some changes to your input. Start by changing the data file to your new correlation matrix (!) and then give Mplus some information about that file:

DATA: FILE IS Lab 9 Example Correlation for Mplus.txt;

TYPE IS CORRELATION;

NOBSERVATIONS IS 250;

The “TYPE IS” keyword is fairly self-explanatory. ☺ NOBSERVATIONS is your sample size – Mplus needs this information and it is not contained in the correlation file. Other than that, you can leave the rest of your input file as is (including the omega calculation). Save this input (under a new name), run it, and compare to your last syntax.

* Once again, the fit statistics don’t change.
* You’ll see that now there are no “Intercepts” (item means) estimated in the correlation-based analysis. Mplus doesn’t have the information it would need to calculate them from the raw data; however, they aren’t relevant to the model fit or the other parameter estimates, so those don’t change.
* You’ll see small differences in the unstandardized factor loadings. This is due to the fact that Mplus has to work backwards to get these – it has to “unstandardize” them from the standardized loadings, and since it doesn’t have the actual item variances & covariances, this is a tricky proposition. If you need to report or use the unstandardized loadings, **always** run your analysis on the raw data or the covariance matrix.
* However, our estimate of omega comes out the same.
* The standardized (STDYX) solutions are also the same. This reflects the fact that the standardization process works the same way whether we standardize before analyzing the data (i.e., use the correlation matrix) or after (standardizing the unstandardized loadings).

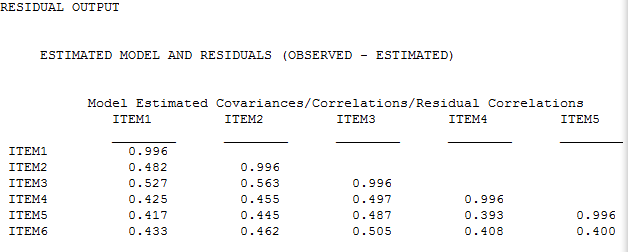
**Obtaining the Discrepancy Matrix in Mplus**

The point of running the analysis using the correlation matrix is so we can get the **standardized discrepancy matrix** or residual matrix. Again, you can get this matrix from a covariance-based or raw data analysis, but it will contain covariance discrepancies rather than correlation discrepancies and it will be very difficult if not impossible to interpret!

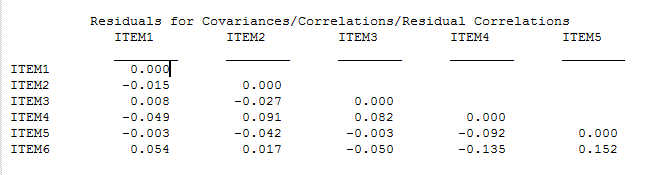
To get this matrix, all you need to do is add another keyword (RESIDUAL) in your OUTPUT line:

OUTPUT: STANDARDIZED RESIDUAL;

Now, in your output, you will have several more pieces of information. First, there will be a table called:



These are the predicted item correlations based on your model. The next table is the one we really care about:



These are the discrepancies (or residuals) between the observed and predicted correlations. Remember that our target is for these to be < .10 in absolute value. In this example, most of the discrepancies meet that criterion except for item 6, which has large-ish discrepancies with item 4 (negative – these 2 items are correlated less than we would expect) and item 5 (positive – these items are correlated more than we would expect). Since the pattern isn’t particularly consistent, we might decide to consider Item 6 as a problematic item and drop it (FYI, when we do this our fit improves substantially). Alternately, since our fit is already OK and these are not much above .10, if the content of item 6 seems really important to our scale and we want to keep it, we could choose to disregard those large discrepancies for now.

**Obtaining the Discrepancy Matrix in lavaan**

Again, in lavaan this is much easier:

resid(fit.model.1f, type = “cor”)

With 6 items, this matrix isn’t too hard to read, but a larger matrix can be hard to interpret. You can write the discrepancy matrix directly to Excel if you use:

write.csv(resid(fit.model.1f, type = “cor”)$cor, “New File Name.csv”)

We’ve used write.csv()before; all we’re doing here is selecting just the piece we want (“cor”) from the output of the resid() function using the $ operator, and then telling R where to send that information. This will come in handy for your project, where you will have a much larger discrepancy matrix, and it’s nice to be able to use Excel functions such as conditional formatting to easily spot large values.

**Lab Exercise:**

Your turn! Download the “Lab 9 Data.csv” file from Canvas. You may use R or Mplus, your choice; I recommend choosing something different from the person next to you so that you can compare!

1. Fit a single factor model to these items. **Report** and interpret the fit statistics.
   1. Chi Square = 57.005
   2. CFI = 0.911
   3. TLI = .851
   4. RMSEA = 0.146 90%CI = (0.111, 0.183)

The overall fit statistics are not ideal. IN reality, the only indicator of pretty good fit is our CFI. However, our chi-square, TLI, and RMSEA fit statsistcs sway our items are not fitting very well to the overall construct. Next, we will look into our factor loadings and other important item component analyses. These will likely give us insight as to why our fit statistics are not fitting as well as expected.

1. **Report** the standardized factor loadings. Do these give you any insight into the items?
   1. **Standardized loadings:**

* Item\_1 = 0.683
* Item\_2 = 0.775
* Item\_3 = 0.754
* Item\_4 = 0.247
* Item\_5 = 0.728
* Item\_6 = 0.717

From what I can tell, Item 4 seems like it is likely to be a problem as the standardized loading is below 0.3.

1. Calculate and **report** omega.
   1. **Omega: 0.822**
2. Obtain, **report**, and interpret the discrepancy matrix.

Residuals for Covariances/Correlations/Residual Correlations

ITEM1 ITEM2 ITEM3 ITEM4 ITEM5

\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_

ITEM1 0.000

ITEM2 0.011 0.000

ITEM3 -0.010 -0.046 0.000

ITEM4 0.123 -0.048 0.104 0.000

ITEM5 -0.077 0.001 0.098 -0.078 0.000

ITEM6 0.056 0.052 -0.053 -0.086 -0.037

Looking at the discrepancy matrix for these items, I see a few concerns. The discrepancy between predicted and observed values for item 1 & item 4 is 0.123, between 3 & 4 is 0.104. Lastly, although not above our a-priori indicated threshold for our discrepancy matrix, item 5 & 3 are at least a little bit concerning with a discrepancy of 0.098. Personally, I find item 4 has been a culprit since the start, and this discrepancy matrix is helping to conclude that we may want to remove item 4 entirely.

It seems as though

1. Based on the results of 1-4 above, would you make any changes to this scale? If so, make them and report the results of your revised analyses for 1-4 above. If not, provide a **very** persuasive argument for why no changes are necessary.

Based on these results, I would remove item 4 from the scale. It has a very low standardized factor loading (0.247) and sems to show the most discrepancy between the other items, which makes a reason for concern. I will re-run the analyses with item 4 removed now.

**New model:**

**Fit statistics:**

* 1. Chi Square = 31.081
  2. CFI = 0.948
  3. TLI = 0.897
  4. RMSEA = 0.144 90%CI = (0.098, 0.195)

**Standardized loadings:**

* Item\_1 = 0.675
* Item\_2 = 0.781
* Item\_3 = 0.745
* Item\_5 = 0.732
* Item\_6 = 0.724

**Omega: 0.853**

Residuals for Covariances/Correlations/Residual Correlations

ITEM1 ITEM2 ITEM3 ITEM5 ITEM6

\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_

ITEM1 0.000

ITEM2 0.013 0.000

ITEM3 0.002 -0.043 0.000

ITEM5 -0.074 -0.006 0.102 0.000

ITEM6 0.057 0.042 -0.051 -0.045 0.000

Looking at the results with the removal of item 4, we see much better statistics overall. All of our fit statistics seem to be in much better range of what we would expect. Although our TLI is not exactly at the point that is “acceptable”, we have enough evidence from our other fit statistics to say that the overall fit is pretty good. Next, our factor loadings and omega improved dramatically. Item 4 was originally so different and low and was not helping our interpretation of the construct much at all. Additionally, the omega appears to be in much better range. Lastly, The only factor of concern given our discrepancy matrix is item 5 – which has a discrepancy of 0.102 with item 3. However, given that this is close t the threshold, and the model seems to be doing well overall, I believe it should stay in our scale. Additionally, there is no evidence from the other results that item 5 or item 4 are necessarily bad items. Therefore, I think we should stick with our second model and remove item 4 from the analysis.

Turn in (a) a Word doc with your answers to the above and (b) relevant output.