Intro to SEM and Clustering in R

R has a number of built-in functions as well as packages available to do latent variable modelling.

To do PCA: prcomp() – built-in

To do EFA: factanal() – builtin

fa() – from **psych**; multiple upgrades

efaUnrotate() – from **semTools**; can do FIML for missing data and WLSMV for categorical variables

GPA() – from **GPArotation** – one stop shop for factor rotations

nFactors package contains various functions for determining number of factors

CFA: cfa() – from **lavaan** package

General SEM: **OpenMx** – can do cfa,sem, mixtures, differential equations… Most general package

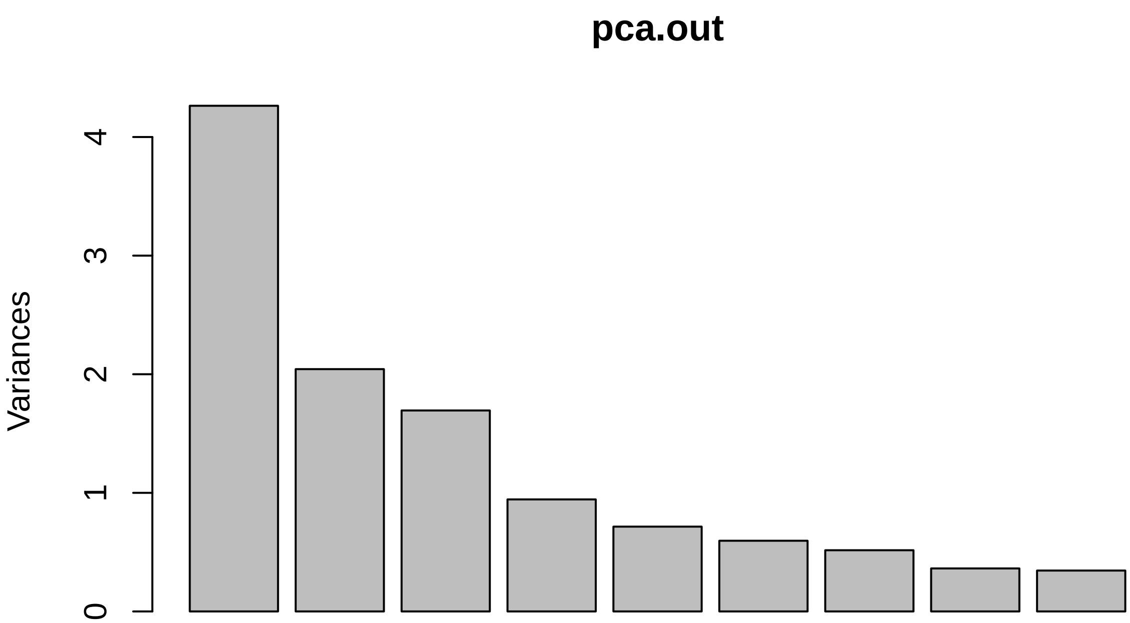
lavaan – modeled after Mplus; can do maybe 80% of the things that Mplus can

We will be using the Holzinger Swineford dataset for all of the examples. Data from lavaan package

```r
library(lavaan)
library(OpenMx)
# can't get OpenMx from CRAN
#source('http://openmx.psyc.virginia.edu/getOpenMx.R')
HS <- HolzingerSwineford1939
#summary(HS)
#str(HS)
```

PCA

```r
pca.out <- prcomp(HS[,7:15])
#quartz()
plot(pca.out)
```



Slightly ambiguous as to the number of components to retain, but we can see that the 3 components with eigenvalues above 1 (Kaiser rule). But in looking at the actual loadings, it almost looks like there is a general component, and maybe a couple specific components.

The psych package has a PCA function, principal(), which uses the same algorithm, but provides much more helpful output.

```
library(psych)
prin1 <- principal(HS[,7:15])
loadings(prin1)
```

```
##
## Loadings:
##    PC1
## x1 0.658
## x2 0.390
## x3 0.477
## x4 0.766
## x5 0.738
## x6 0.772
## x7 0.349
## x8 0.454
## x9 0.591
##
##                  PC1
## SS loadings    3.216
## Proportion Var 0.357
```

```
prin2 <- principal(HS[,7:15],2)
loadings(prin2)
```

```
##
## Loadings:
##    RC1    RC2
```

```
## x1  0.450  0.496
## x2  0.259  0.304
## x3  0.183  0.550
## x4  0.880  0.104
## x5  0.880
## x6  0.875  0.122
## x7         0.610
## x8         0.764
## x9  0.164  0.764
##
##                  RC1   RC2
## SS loadings    2.646 2.209
## Proportion Var 0.294 0.245
## Cumulative Var 0.294 0.539
```

```r
prin3 <- principal(HS[,7:15],3)
loadings(prin3)
```

```
##
## Loadings:
##    RC1    RC3    RC2
## x1  0.321  0.673  0.175
## x2         0.727 -0.102
## x3         0.779  0.155
## x4  0.889  0.124
## x5  0.903
## x6  0.869  0.178
## x7        -0.153  0.830
## x8         0.145  0.818
## x9  0.130  0.435  0.636
##
##                  RC1   RC3   RC2
## SS loadings    2.501 1.872 1.847
## Proportion Var 0.278 0.208 0.205
## Cumulative Var 0.278 0.486 0.691
```

```r
prin4 <- principal(HS[,7:15],4)
loadings(prin4)
```

```
##
## Loadings:
##    RC1    RC2    RC3    RC4
## x1  0.306         0.788
## x2  0.105         0.216  0.959
## x3                0.819  0.191
## x4  0.887         0.149
## x5  0.904
## x6  0.869         0.159
## x7         0.832        -0.148
## x8         0.833  0.127  0.108
## x9  0.123  0.603  0.471  0.114
##
##                  RC1   RC2   RC3   RC4
```

```
## SS loadings    2.490 1.783 1.629 1.017
## Proportion Var 0.277 0.198 0.181 0.113
## Cumulative Var 0.277 0.475 0.656 0.769
```

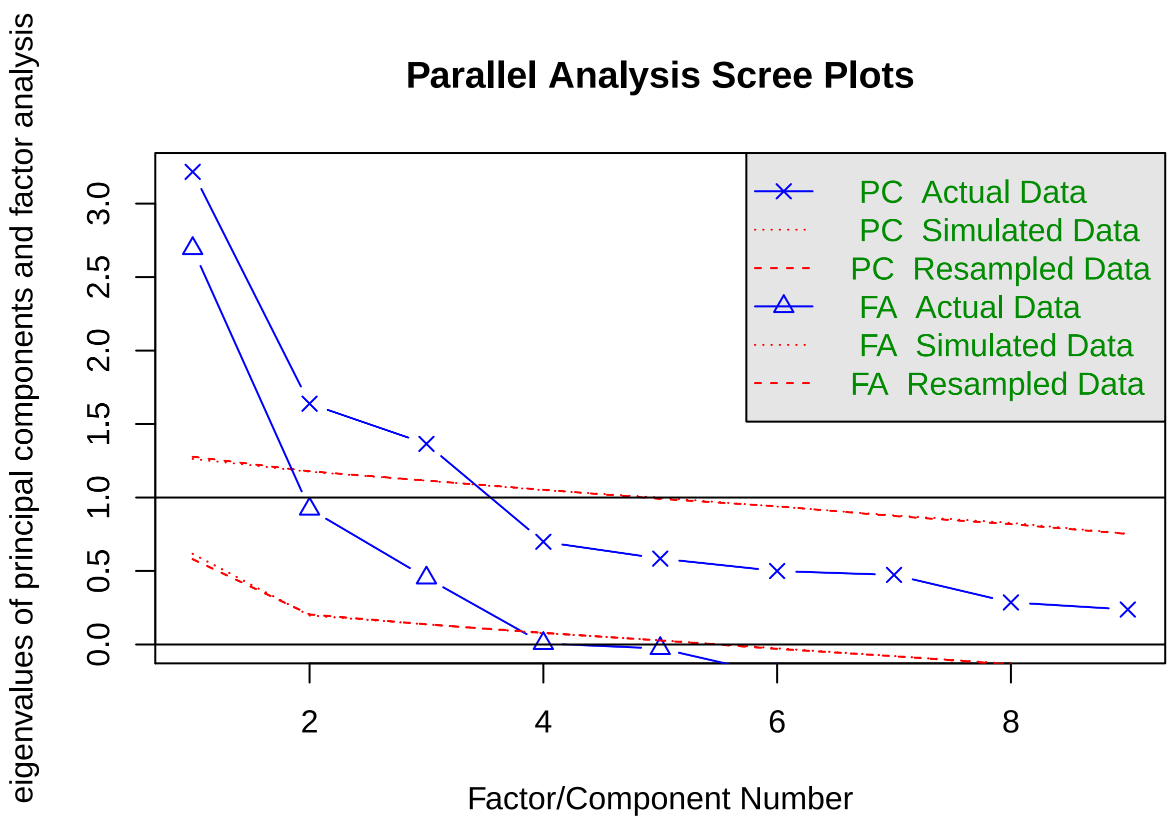
Note, PCA always extracts the same number of components as variables entered. But with principal() we have a choice of displaying a specific number of components.

In using PCA, 3 components seems to be a little bit cleaner,where we can see "clusters" in the loadings, than others in the factor structure. But still hazy. With 4 components, the last component is really only made up of 1 variable (loading > 0.9).

One of the best tools that I know of to determine the number of components(PCA) or factors(EFA) is Horn's parallel analysis from the psych package.

Although called fa.parallel() it extracts both components and factors

```
fa.parallel(HS[,7:15])
```




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

Parallel analysis compares the actual eignevalues to the eigenvalues from a simulated dataset of random noise variables. We are looking for the number of eigenvalues above what would be expected by chance. This makes it look pretty clear, both 3 components and factors

EFA

R has the built-in factanal() which gets the job done in most cases. Defaults to ML estimation and varimax(orthogonal rotation)

```r
fa.out <- factanal(HS[,7:15],3)
loads <- fa.out$loadings
# cluster.plot(fa.out)
# extract loadings and feed to rotation program.
library(GPArotation)
gpa.out <- GPFoblq(loads) # oblique rotation
# new loading matrix
round(gpa.out$loadings,2)
```

```
##    Factor1 Factor2 Factor3
## x1    0.19    0.60    0.03
## x2    0.04    0.51   -0.12
## x3   -0.07    0.69    0.02
## x4    0.84    0.02    0.01
## x5    0.89   -0.07    0.01
## x6    0.81    0.08   -0.01
## x7    0.04   -0.15    0.72
## x8   -0.03    0.10    0.70
## x9    0.03    0.37    0.46
```

```r
# new factor correlations
gpa.out$Phi
```

```
##           [,1]      [,2]      [,3]
## [1,] 1.0000000 0.3257713 0.2164403
## [2,] 0.3257713 1.0000000 0.2704747
## [3,] 0.2164403 0.2704747 1.0000000
```

Fairly clear factor structure. Not many cross-loadings.

Fancy way to plot results, from http://mindingthebrain.blogspot.com/2015/04/plotting-factor-analysis-results.html

Get factor scores:

```r
fa.out2 <- fa(HS[,7:15],scores="Bartlett")
fscor <- fa.out2$scores
head(fscor)
```

```
##              MR1
## [1,] -0.22248948
## [2,] -0.99709195
## [3,] -2.15390885
## [4,]  0.02741775
## [5,] -0.14301155
## [6,] -1.35498984
```

Not generally advisable to get factor scores as there are a number of inherent problems with them (Grice 2001), but the psych package's fa() has multiple options. see "scores=" option.

Want to do full-information maximum likelihood (FIML) with EFA? Have a decent amount of missing data, and can make the assumption it is missing at random (MAR)? The only way to do it in R is efaUnrotate() (probably can do in OpenMx, but I'm not sure how to do it easily with more than 1 factor).

efaUnrotate is a wrapper that is built around lavaan. This means you can use it for FIML, WLSMV, and whatever other options from lavaan you want.

Note: default is to estimate unrotated solution, then use GPArotation to rotate.

```
library(semTools)

efa.fiml <- efaUnrotate(HS[,7:15],3,missing="fiml") # little bit slower
#summary(efa.fiml)
rot.out <- oblqRotate(efa.fiml) # quartimin is default, same as GPFoblq()
#summary(rot.out) # very close to results from fa() and GPFoblq()
```

Check out semTools – a *ton* of great helper functions

CFA

How about we move to a confirmatory factor analysis framework. Note: this isn't really confirmatory, as we have already looked and tested out multiple factor structures.

We will do this in both lavaan (very easy to use) and OpenMx (not as easy, but more flexible)

lavaan

First lavaan – which we will use for general SEM later

```
######## basic lavaan syntax #########

# latent variable definition           =~ is measured by
# regression                           ~ is regressed on
# (residual) (co)variance             ~~ is correlated (covariance) with
# intercept (mean)                   ~ 1 intercept   # same as regressed, but with 1



########## basic regression in R ######
lm.out = lm(x9 ~ 1 + x1 + x2 + x3,data=HS)   # 1 = intercept
#summary(lm.out)
coef(lm.out)
```

```
## (Intercept)          x1          x2          x3
##  3.45832040  0.25425361  0.04935937  0.16013214
```

```
##############################################
############ Confirmatory Factor Analysis ############

HS.model <- '
visual =~ x1 + x2 + x3
textual =~ x4 + x5 + x6
speed =~ x7 + x8 + x9 '

fit <- cfa(HS.model, data = HS)
coef(fit)
```

```
##       visual=~x2       visual=~x3      textual=~x5      textual=~x6
##            0.554            0.729            1.113            0.926
##        speed=~x8        speed=~x9           x1~~x1           x2~~x2
##            1.180            1.082            0.549            1.134
##           x3~~x3           x4~~x4           x5~~x5           x6~~x6
##            0.844            0.371            0.446            0.356
##           x7~~x7           x8~~x8           x9~~x9   visual~~visual
##            0.799            0.488            0.566            0.809
## textual~~textual     speed~~speed  visual~~textual    visual~~speed
##            0.979            0.384            0.408            0.262
##   textual~~speed
##            0.173
```

```
# get factor scores
fscor.lav <- predict(fit) # in flux right now -- in manual there is lavPredict()

###################### the cfa() function is a wrapper for the lavaan() function #######
##### a wraper is just a function that makes assumptions for you, so specify less code ###

# same model using lavaan()

HS.model2 <- '
visual =~ x1 + x2 + x3
textual =~ x4 + x5 + x6
speed =~ x7 + x8 + x9

# residual variances for each indicator
x1~~x1
x2~~x2
x3~~x3
x4~~x4
x5~~x5
x6~~x6
x7~~x7
x8~~x8
x9~~x9

# factor variances
visual~~visual
textual~~textual
speed~~speed

# factor covariances
visual~~textual   # think no covariance between factors? -- visual~~0*textual
visual~~speed
textual~~speed
'
fit2 <- cfa(HS.model2, data = HS)
coef(fit2)
```

```
##       visual=~x2       visual=~x3      textual=~x5      textual=~x6
##            0.554            0.729            1.113            0.926
##        speed=~x8        speed=~x9           x1~~x1           x2~~x2
##            1.180            1.082            0.549            1.134
```

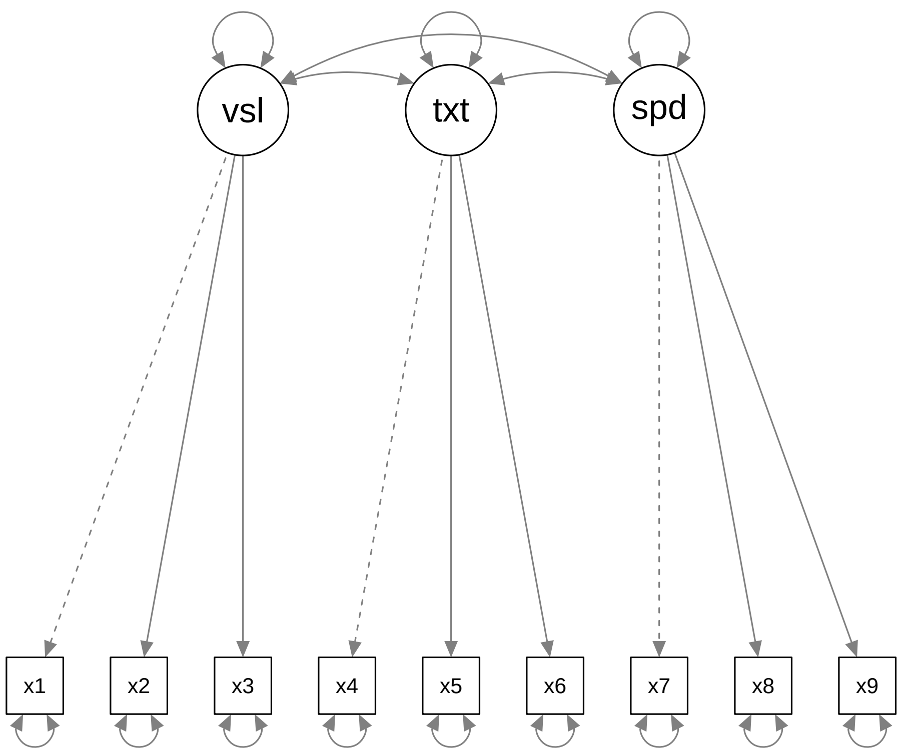
```
##          x3~~x3          x4~~x4          x5~~x5          x6~~x6
##           0.844           0.371           0.446           0.356
##          x7~~x7          x8~~x8          x9~~x9  visual~~visual
##           0.799           0.488           0.566           0.809
## textual~~textual    speed~~speed visual~~textual   visual~~speed
##           0.979           0.384           0.408           0.262
##   textual~~speed
##           0.173
```

```
#summary(fit2, fit = TRUE)
```

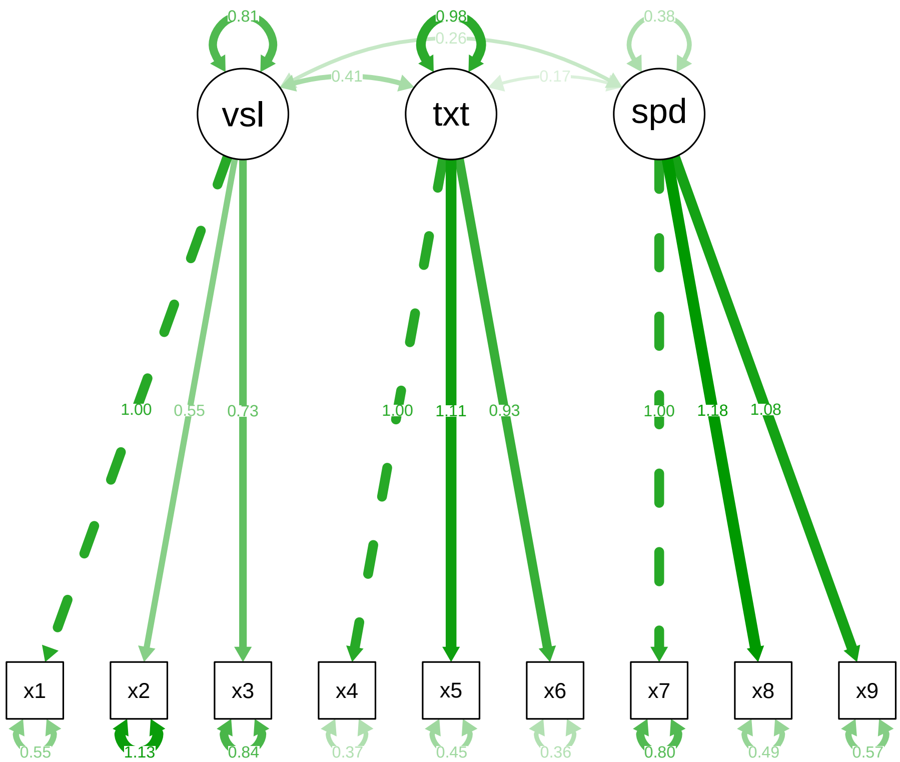
We can see that we get identical parameter estimates across the two equivalent specifications.

So what the heck did we create? I find it much more helpful to visual what the model looks like. The "semPlot" package is great – makes it super easy to visualize SEM models.

```
library(semPlot)
semPaths(fit)
```




```
# dashed lines refer to fixed parameters
# also
semPaths(fit,what="est")
```



Output Breakdown

Using the summary() function, we get a lot of output, but we will go piece by piece for what it means

```
#summary(fit, fit.measures = TRUE)
#get standardized estimates
summary(fit, fit = TRUE,std=T)
```

```
## lavaan (0.5-17) converged normally after  35 iterations
##
##   Number of observations                           301
##
##   Estimator                                         ML
##   Minimum Function Test Statistic               85.306
##   Degrees of freedom                                24
##   P-value (Chi-square)                           0.000
##
## Model test baseline model:
##
##   Minimum Function 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
##
##   Number of free parameters                        21
##   Akaike (AIC)                               7517.490
##   Bayesian (BIC)                             7595.339
##   Sample-size adjusted Bayesian (BIC)        7528.739
##
## Root Mean Square Error of Approximation:
##
##   RMSEA                                         0.092
##   90 Percent Confidence Interval          0.071  0.114
##   P-value RMSEA <= 0.05                         0.001
##
## Standardized Root Mean Square Residual:
##
##   SRMR                                          0.065
##
## Parameter estimates:
##
##   Information                                Expected
##   Standard Errors                            Standard
##
##                    Estimate  Std.err  Z-value  P(>|z|)   Std.lv  Std.nox
## Latent variables:
##   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:
##   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:
##     x1                0.549    0.114                      0.549    0.404
##     x2                1.134    0.102                      1.134    0.821
##     x3                0.844    0.091                      0.844    0.662
##     x4                0.371    0.048                      0.371    0.275
##     x5                0.446    0.058                      0.446    0.269
##     x6                0.356    0.043                      0.356    0.298
```

```
##     x7                  0.799    0.081                            0.799    0.676
##     x8                  0.488    0.074                            0.488    0.477
##     x9                  0.566    0.071                            0.566    0.558
##     visual              0.809    0.145                            1.000    1.000
##     textual             0.979    0.112                            1.000    1.000
##     speed               0.384    0.086                            1.000    1.000
```

Fit

```r
round(fitMeasures(fit)[c("chisq","df","pvalue","rmsea","tli","cfi")],3)
```

```
##  chisq     df pvalue  rmsea    tli    cfi
## 85.306 24.000  0.000  0.092  0.896  0.931
```

```r
# to get all
# use fitMeasures(fot)
```

The χ^2 is significant, which means there is a significant amount of misfit (opposite of most p-values,not good). With larger sample sizes, it is almost impossible to get a non-significant χ^2. Generally, we are looking for CFA & TLI > 0.95 and RMSEA > 0.06. We don't quite get this.

So lets first look at the loadings. We can see the values in the second plot from semPlot

```r
pars <- parameterEstimates(fit)
pars[pars$op=="=~",]
```

```
##       lhs op rhs   est    se      z pvalue ci.lower ci.upper
## 1  visual =~  x1 1.000 0.000     NA     NA    1.000    1.000
## 2  visual =~  x2 0.554 0.100  5.554      0    0.358    0.749
## 3  visual =~  x3 0.729 0.109  6.685      0    0.516    0.943
## 4 textual =~  x4 1.000 0.000     NA     NA    1.000    1.000
## 5 textual =~  x5 1.113 0.065 17.014      0    0.985    1.241
## 6 textual =~  x6 0.926 0.055 16.703      0    0.817    1.035
## 7   speed =~  x7 1.000 0.000     NA     NA    1.000    1.000
## 8   speed =~  x8 1.180 0.165  7.152      0    0.857    1.503
## 9   speed =~  x9 1.082 0.151  7.155      0    0.785    1.378
```

Overall, the loadings are pretty high, there aren't any weak loadings.

Note, in CFA you have to scale the latent variable by either fixing one factor loading (usually to 1), or fix the factor variance. The default in most programs is to fix first factor loading.

In lavaan, if we wanted to change this:

```r
HS.model <- '
visual =~ NA*x1 + x2 + x3 # have to override default fix of first factor loading
textual =~ NA*x4 + x5 + x6
speed =~ NA*x7 + x8 + x9
visual~~1*visual
textual~~1*textual
speed~~1*speed'
```

This results in the exact same level of fit, but the scale of the parameter estimates shift to being standardized.

How about covariances:

```
pars <- parameterEstimates(fit)
# residual variances
pars[pars$op=="~~",][1:9,]
```

```
##   lhs op rhs   est    se      z pvalue ci.lower ci.upper
## 1  x1 ~~  x1 0.549 0.114  4.833      0    0.326    0.772
## 2  x2 ~~  x2 1.134 0.102 11.146      0    0.934    1.333
## 3  x3 ~~  x3 0.844 0.091  9.317      0    0.667    1.022
## 4  x4 ~~  x4 0.371 0.048  7.779      0    0.278    0.465
## 5  x5 ~~  x5 0.446 0.058  7.642      0    0.332    0.561
## 6  x6 ~~  x6 0.356 0.043  8.277      0    0.272    0.441
## 7  x7 ~~  x7 0.799 0.081  9.823      0    0.640    0.959
## 8  x8 ~~  x8 0.488 0.074  6.573      0    0.342    0.633
## 9  x9 ~~  x9 0.566 0.071  8.003      0    0.427    0.705
```

```
# covariance of residuals
#residuals(fit)
# factor covariances
pars[pars$op=="~~",][1:9,]
```

```
##   lhs op rhs   est    se      z pvalue ci.lower ci.upper
## 1  x1 ~~  x1 0.549 0.114  4.833      0    0.326    0.772
## 2  x2 ~~  x2 1.134 0.102 11.146      0    0.934    1.333
## 3  x3 ~~  x3 0.844 0.091  9.317      0    0.667    1.022
## 4  x4 ~~  x4 0.371 0.048  7.779      0    0.278    0.465
## 5  x5 ~~  x5 0.446 0.058  7.642      0    0.332    0.561
## 6  x6 ~~  x6 0.356 0.043  8.277      0    0.272    0.441
## 7  x7 ~~  x7 0.799 0.081  9.823      0    0.640    0.959
## 8  x8 ~~  x8 0.488 0.074  6.573      0    0.342    0.633
## 9  x9 ~~  x9 0.566 0.071  8.003      0    0.427    0.705
```

Note: "~~" between observed variables refers to residual variance "~~" between latent variables are factor variances and covariances

Also note that in lavaan, it is default to allow covariances between latent factors. You can change this in the syntax (factor1~~1*factor1), or set orthogonal=T in cfa().

Although not recommended in some instances, we can use Modification Indices to improve our model fit. Note that modification indices refer to the improvement in the chi-square fit statistic with a change in 1 degree of freedom.

```
mod = modificationIndices(fit)
mod[mod$mi > 10 & is.na(mod$mi) ==F,]
```

```
##      lhs op rhs     mi    epc sepc.lv sepc.all sepc.nox
## 1 visual =~  x7 18.631 -0.422  -0.380   -0.349   -0.349
## 2 visual =~  x9 36.411  0.577   0.519    0.515    0.515
## 3     x7 ~~  x8 34.145  0.536   0.536    0.488    0.488
## 4     x8 ~~  x9 14.946 -0.423  -0.423   -0.415   -0.415
```

From this, it looks like both x7 and x8 might also be a part of the “visual” factor.

```
HS.model2 <- '
visual =~ x1 + x2 + x3 + x7 + x8
textual =~ x4 + x5 + x6
speed =~ x7 + x8 + x9 '

fit3 <- cfa(HS.model2, data = HS)
#summary(fit3, fit.measures = TRUE)
fitMeasures(fit3)[c("rmsea","tli","cfi")]
```

```
##      rmsea        tli        cfi
## 0.06031355 0.95535107 0.97271454
```

Now we meet the “recommended” guidelines for fit criteria cutoff.

Anymore big MI’s?

```
mod2 = modificationIndices(fit2)
mod2[mod2$mi > 10 & is.na(mod2$mi) ==F,]
```

```
##      lhs op rhs     mi    epc sepc.lv sepc.all sepc.nox
## 1 visual =~  x7 18.631 -0.422  -0.380   -0.349   -0.349
## 2 visual =~  x9 36.411  0.577   0.519    0.515    0.515
## 3     x7 ~~  x8 34.145  0.536   0.536    0.488    0.488
## 4     x8 ~~  x9 14.946 -0.423  -0.423   -0.415   -0.415
```

One potential change, but in examining the epc (expected parameter change), the value isn’t too large. This means that if we added x1 to the textual factor, the expected loading would be 0.306.

CFA in OpenMx

How about the same model in OpenMx:

```
require(OpenMx)
dataRaw <- mxData( observed=HS, type="raw" )
manifests <- c("x1","x2","x3","x4","x5","x6","x7","x8","x9")
latents <- c("visual","textual","speed")
# residual variances
resVars <- mxPath( from=manifests, arrows=2,free=TRUE, values=1)
# latent variance
latVar <- mxPath(from=latents, arrows=2,free=T, values=1)
# latent covariances
cov1 <- mxPath(from="visual",to="textual",arrows=2,value=1)
cov2 <- mxPath(from="visual",to="speed",arrows=2,value=1)
cov3 <- mxPath(from="textual",to="speed",arrows=2,value=1)
# factor loadings
facLoads1 <- mxPath( from="visual", to=c("x1","x2","x3"),
                     arrows=1,free=c(F,T,T), values=c(1,1,1))
facLoads2 <- mxPath( from="textual", to=c("x4","x5","x6"),
                     arrows=1,free=c(F,T,T), values=c(1,1,1))
facLoads3 <- mxPath( from="speed", to=c("x7","x8","x9"),
```

```r
                     arrows=1,free=c(F,T,T), values=c(1,1,1))
# means
means <- mxPath( from="one", to=c(manifests,latents), arrows=1,
                 free=c(T,T,T,T,T,T,T,T,T,F,F,F), values=c(1,1,1,1,1,1,1,1,1,0,0,0))

threeFactorModel <- mxModel("Three Factor Model", type="RAM",manifestVars=manifests,
                            latentVars=latents,dataRaw, resVars, latVar, facLoads1,
                            facLoads2,facLoads3, means,cov1,cov2,cov3)
threeRun <- mxRun(threeFactorModel)
#threeRun <- mxTryHard(threeFactorModel) # don't have to be as close with starting values
#summary(threeRun)

# fit indices
factorSat <- mxRefModels(threeRun,run=T)
#summary(threeRun,refModels=factorSat)
summary(threeRun,refModels=factorSat)$CFI
```

```
## [1] 0.9305597
```

```r
summary(threeRun,refModels=factorSat)$TLI
```

```
## [1] 0.8958395
```

```r
summary(threeRun,refModels=factorSat)$RMSEA
```

```
## [1] 0.09212148
```

Good example of lavaan to OpenMx and how to write code compactly in OpenMx: http://industrialcodeworkshop.blogspot.com/2012/10/from-lavaan-to-openmx.html

Compare estimates to lavaan (exact same). Note, in OpenMx when you enter raw data, you are forced to specify the mean structure. This is not a requirement in lavaan, but possible with "meanstructure=T"

Mediation

Mediation in lavaan – lavaan just added a new operator (":=") Example taken from: http://lavaan.ugent.be/tutorial/mediation.html

```r
set.seed(1234)
X <- rnorm(100)
M <- 0.5*X + rnorm(100)
Y <- 0.7*M + rnorm(100)
Data <- data.frame(X = X, Y = Y, M = M)
model <- ' # direct effect
             Y ~ c*X
           # mediator
             M ~ a*X
             Y ~ b*M
           # indirect effect (a*b)
             ab := a*b
```

```
          # total effect
            total := c + (a*b)
        '
fit.med <- sem(model, data = Data)
summary(fit.med)
```

```
## lavaan (0.5-17) converged normally after  12 iterations
##
##   Number of observations                           100
##
##   Estimator                                         ML
##   Minimum Function Test Statistic                0.000
##   Degrees of freedom                                 0
##   Minimum Function Value               0.0000000000000
##
## Parameter estimates:
##
##   Information                                 Expected
##   Standard Errors                             Standard
##
##                      Estimate  Std.err  Z-value  P(>|z|)
## Regressions:
##   Y ~
##     X          (c)      0.036    0.104    0.348    0.728
##   M ~
##     X          (a)      0.474    0.103    4.613    0.000
##   Y ~
##     M          (b)      0.788    0.092    8.539    0.000
##
## Variances:
##     Y                   0.898    0.127
##     M                   1.054    0.149
##
## Defined parameters:
##     ab                  0.374    0.092    4.059    0.000
##     total               0.410    0.125    3.287    0.001
```

Remove mediator

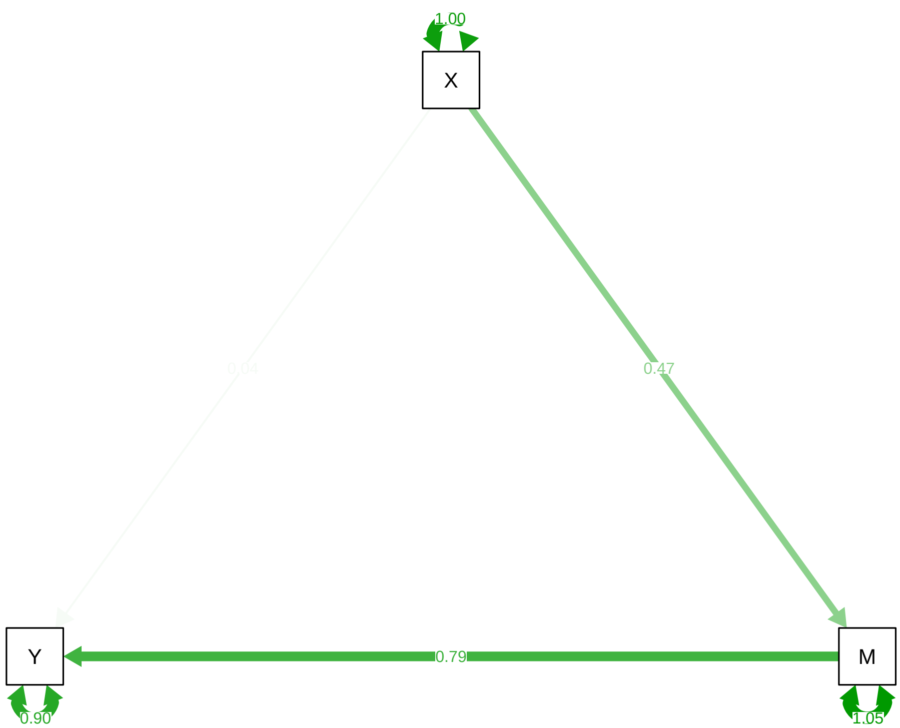
```
med2 <- '
Y ~ X
'
fit.med2 <-  sem(med2,data=Data)
coef(fit.med2)
```

```
##   Y~X  Y~~Y
## 0.410 1.553
```

```
semPaths(fit.med2,what="est")
```

What does this look like?

```r
semPaths(fit.med,what="est")
```



General SEM

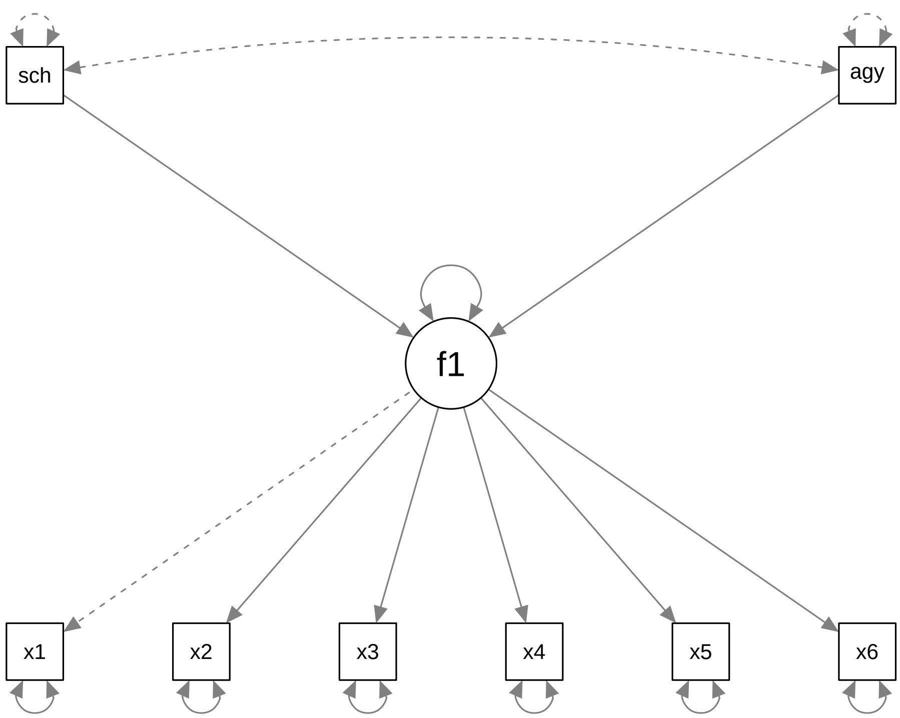
How about fitting more general SEM? Easy to do with sem() from lavaan. Note, this is again a wrapper for the lavaan().

```
sem.mod <- '
f1 =~ x1 + x2 + x3 + x4 + x5 + x6
f1 ~ school + ageyr
'
sem.out <- sem(sem.mod,HS)
#summary(sem.out,fit=T)
coef(sem.out)
```

```
##    f1=~x2    f1=~x3    f1=~x4    f1=~x5    f1=~x6 f1~school  f1~ageyr
##     0.517     0.446     2.057     2.287     1.914    -0.226    -0.078
##    x1~~x1    x2~~x2    x3~~x3    x4~~x4    x5~~x5    x6~~x6    f1~~f1
##     1.128     1.320     1.229     0.375     0.454     0.352     0.206
```

Notice the inclusion of the regression of school and ageyr on the factor.

```
semPaths(sem.out)
```



Here we get almost identical options for fit and output as we do in CFA. Using sem() it is possible to create a wide variety of models, models that encompass both CFA, path analysis, mediation etc…

Multiple Group Models & Invariance

This topic will be covered in more depth later, but as an example here is the code.

```r
HS.group <- '
visual =~ x1 + x2 + x3
textual =~ x4 + x5 + x6
speed =~ x7 + x8 + x9 '

fit.config <- cfa(HS.group, data = HS,group="school")
fit.metric <- cfa(HS.group, data = HS,group="school",group.equal="loadings")
fit.strong <- cfa(HS.group, data = HS,group="school",
                  group.equal=c("loadings","intercepts"))
fit.strict <- cfa(HS.group, data = HS,group="school",
                  group.equal=c("loadings","intercepts","residuals"))
anova(fit.config,fit.metric,fit.strong,fit.strict)
```

```
## Chi Square Difference Test
##
##            Df    AIC    BIC  Chisq Chisq diff Df diff Pr(>Chisq)
## fit.config 48 7484.4 7706.8 115.85
## fit.metric 54 7480.6 7680.8 124.04      8.192       6    0.22436
## fit.strong 60 7508.6 7686.6 164.10     40.059       6  4.435e-07 ***
## fit.strict 69 7508.1 7652.6 181.51     17.409       9    0.04269 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note: have categorical indicators? Have to do slightly different specifications. See: http://www.myweb.ttu.edu/spornpra/catInvariance.html

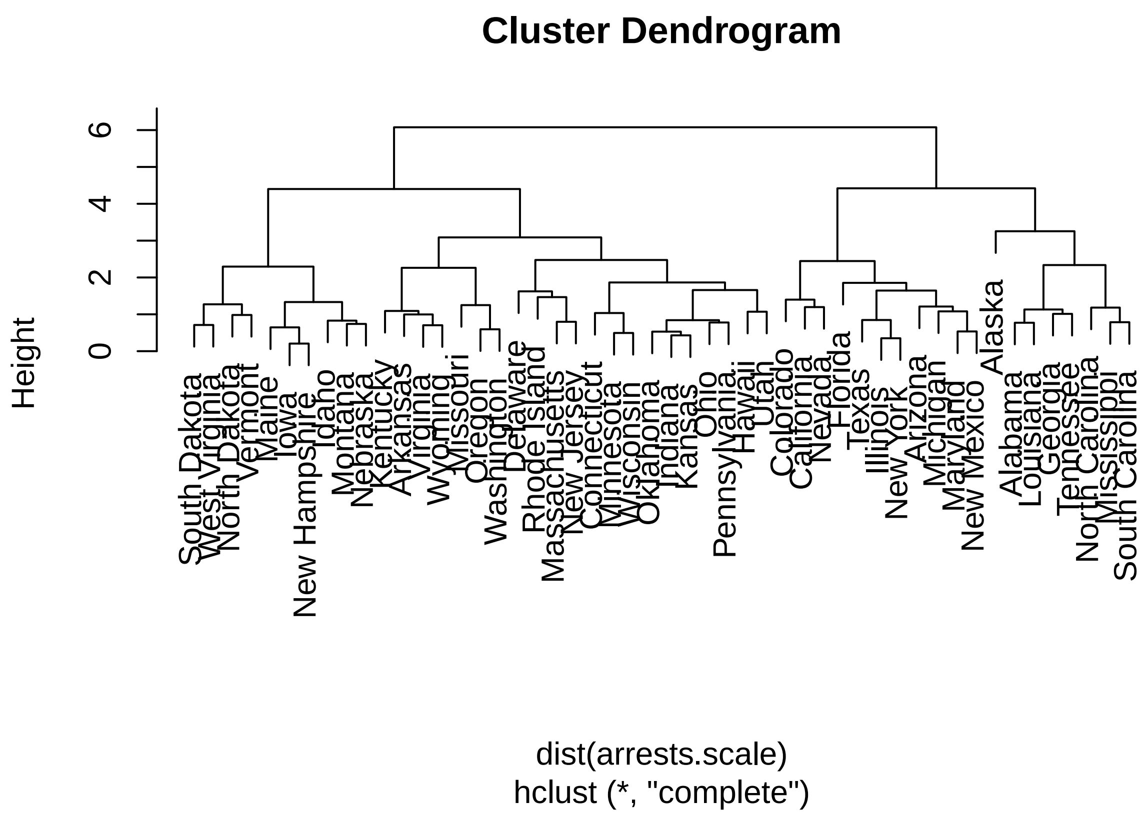
Clustering

This is a rather brief section. For additional resources, see: http://www.statmethods.net/advstats/cluster.html http://www.r-tutor.com/gpu-computing/clustering/hierarchical-cluster-analysis Ch.10 in Introduction to Statistical Learning

For this, we will be using hclust() from stats package (built-in). The dataset is on the number of arrests per 100,000 residents.

Similar to PCA, it is important to scale the variables beforehand

```r
library(ISLR)
arrests.scale = scale(USArrests)
hc.s.complete = hclust(dist(arrests.scale), method="complete")
plot(hc.s.complete)
```



This dendrogram is a little big and convoluted, and it is hard to glean any information from.

Similar to decision trees where we can prevent overfitting by pruning, in clustering we can cut off levels of the dendrogram past a certain level.

with this, we can either cut the tree at a prespecified number of groups:

```
cutree(hc.s.complete, k=3)
```

```
##        Alabama         Alaska        Arizona       Arkansas     California 
##              1              1              2              3              2 
##       Colorado    Connecticut       Delaware        Florida        Georgia 
##              2              3              3              2              1 
##         Hawaii          Idaho       Illinois        Indiana           Iowa 
##              3              3              2              3              3 
##         Kansas       Kentucky      Louisiana          Maine       Maryland 
##              3              3              1              3              2 
##  Massachusetts       Michigan      Minnesota    Mississippi       Missouri 
##              3              2              3              1              3 
##        Montana       Nebraska         Nevada  New Hampshire     New Jersey 
##              3              3              2              3              3 
##     New Mexico       New York North Carolina   North Dakota           Ohio 
##              2              2              1              3              3 
##       Oklahoma         Oregon   Pennsylvania   Rhode Island South Carolina 
##              3              3              3              3              1 
##   South Dakota      Tennessee          Texas           Utah        Vermont 
##              3              1              2              3              3 
##       Virginia     Washington  West Virginia      Wisconsin        Wyoming 
##              3              3              3              3              3
```

```r
table(cutree(hc.s.complete, k = 3))
```

```
##
##  1  2  3
##  8 11 31
```

In this, we can see that cluster 1 looks mostly like the southern states. Cluster 3 looks someone like the Northwest. Not sure about cluster 2.

Or a pre-specified height:

```r
cutree(hc.s.complete, h=3)
```

```
##        Alabama         Alaska        Arizona       Arkansas     California
##              1              2              3              4              3
##       Colorado    Connecticut       Delaware        Florida        Georgia
##              3              5              5              3              1
##         Hawaii          Idaho       Illinois        Indiana           Iowa
##              5              6              3              5              6
##         Kansas       Kentucky      Louisiana          Maine       Maryland
##              5              4              1              6              3
##  Massachusetts       Michigan      Minnesota    Mississippi       Missouri
##              5              3              5              1              4
##        Montana       Nebraska         Nevada  New Hampshire     New Jersey
##              6              6              3              6              5
##     New Mexico       New York North Carolina   North Dakota           Ohio
##              3              3              1              6              5
##       Oklahoma         Oregon   Pennsylvania   Rhode Island South Carolina
##              5              4              5              5              1
##   South Dakota      Tennessee          Texas           Utah        Vermont
##              6              1              3              5              6
##       Virginia     Washington  West Virginia      Wisconsin        Wyoming
##              4              4              6              5              4
```

```r
table(cutree(hc.s.complete, h = 3))
```

```
##
##  1  2  3  4  5  6
##  7  1 11  7 14 10
```

This changes our answer drastically. This is where domain knowlegdge comes in to play by using this information to cut the dendrogram at the place that gives the most amount of information.

IRT

Item Response Theory in lavaan. This uses weighted least squares estimation with mean and variance adjustment (WLSMV). This is what Mplus defaults to when manifest variables are dichotomous or ordinal.

http://lavaan.ugent.be/tutorial/cat.html

Two ways to specify. Change class() of variables in dataset to

Example:

```r
library(psych)
data(bfi)
sapply(bfi,class)
```

```
##        A1        A2        A3        A4        A5        C1        C2
## "integer" "integer" "integer" "integer" "integer" "integer" "integer"
##        C3        C4        C5        E1        E2        E3        E4
## "integer" "integer" "integer" "integer" "integer" "integer" "integer"
##        E5        N1        N2        N3        N4        N5        O1
## "integer" "integer" "integer" "integer" "integer" "integer" "integer"
##        O2        O3        O4        O5    gender education       age
## "integer" "integer" "integer" "integer" "integer" "integer" "integer"
```

```r
# get same things as
# str(bfi)

agree <-'
f1 =~ A1 + A2 + A3 + A4 + A5
'
irt.out <- cfa(agree,data=bfi,ordered=c("A1","A2","A3","A4","A5"))
#summary(irt.out)
coef(irt.out)
```

```
## f1=~A2 f1=~A3 f1=~A4 f1=~A5  A1|t1  A1|t2  A1|t3  A1|t4  A1|t5  A2|t1
## -1.648 -1.860 -1.182 -1.534 -0.441  0.321  0.739  1.232  1.893 -2.112
##  A2|t2  A2|t3  A2|t4  A2|t5  A3|t1  A3|t2  A3|t3  A3|t4  A3|t5  A4|t1
## -1.526 -1.184 -0.475  0.485 -1.840 -1.309 -0.952 -0.323  0.610 -1.668
##  A4|t2  A4|t3  A4|t4  A4|t5  A5|t1  A5|t2  A5|t3  A5|t4  A5|t5 f1~~f1
## -1.143 -0.867 -0.366  0.236 -2.018 -1.345 -0.911 -0.245  0.685  0.190
```

```r
fitMeasures(irt.out)[c("rmsea","tli","cfi")]
```

```
##      rmsea        tli        cfi
## 0.07142267 0.98142573 0.99071286
```

By changing to categorical, lavaan automatically changes estimator from ML to WLSMV. One of the large benefits to using WLSMV as opposed to marginal maximum likelihood in traditional IRT is that you get fit indices (rmsea, cfi, etc…)

Instead of specifying ordered= , we could have changed the class in the dataframe, and lavaan would have recognized this automatically.

Equivalent:

```r
bfi[,c("A1","A2","A3","A4","A5")] <- lapply(bfi[,c("A1","A2","A3","A4","A5")],ordered)

agree <-'
f1 =~ A1 + A2 + A3 + A4 + A5
'
irt.out <- cfa(agree,data=bfi)
#summary(irt.out)
```

Now " | " is introduced for thresholds.

In OpenMx, to do a form of IRT you have to follow a different specification. See: http://openmx.psyc.virginia.edu/docs/OpenMx/2.0.0-3756/ItemFactorAnalysis.html

and

http://faculty.virginia.edu/humandynamicslab/pubs/PritikinHunterBoker-IFA-2014.pdf

For traditional IRT models in R: ltm package – for unidimensional models mirt package – for multidimensional models

Simple example using the ltm package: Using grm() (graded response model) as the data are ordinal (# of cats > 2 and ordered)

```r
library(ltm)
ltm.out <- grm(bfi[,1:5])
#plot(ltm.out)
coef(ltm.out)
```

```
##    Extrmt1 Extrmt2 Extrmt3 Extrmt4 Extrmt5 Dscrmn
## A1  -0.905   0.744   1.654   2.774   4.459  0.862
## A2   3.030   2.139   1.645   0.660  -0.650 -1.839
## A3   2.276   1.604   1.170   0.404  -0.730 -2.527
## A4   3.352   2.232   1.670   0.709  -0.414 -1.047
## A5   3.005   1.955   1.319   0.369  -0.948 -1.701
```