# Exploratory Data Mining via Search Strategies Lab #5

Ross Jacobucci & Kevin J. Grimm

# Outline

This presentation will go over the basics of using various multivariate procedures in R. These include:

1. Exploratory Factor Analysis & PCA
2. Structural Equation Models
3. SEM Trees
4. Other Multivariate CART Models

# R Packages

EFA & SEM

```
library(psych) # efa & miscellaneous tools
library(OpenMx) # SEM
library(lavaan) # SEM
```

SEM Trees

```
# how to install
#source('http://www.brandmaier.de/semtree/getsemtree.R')
library(semtree)
```

Other Longitudinal Trees
longRPart package. Not currently maintained on CRAN. Can be accessed from:
http://cran.r-project.org/src/contrib/Archive/longRPart/
First Download the 1.0 tar.gz

```
# longRPart is not on CRAN
# have to install from source
library(longRPart)
library(REEMtree)
```

# PCA and EFA

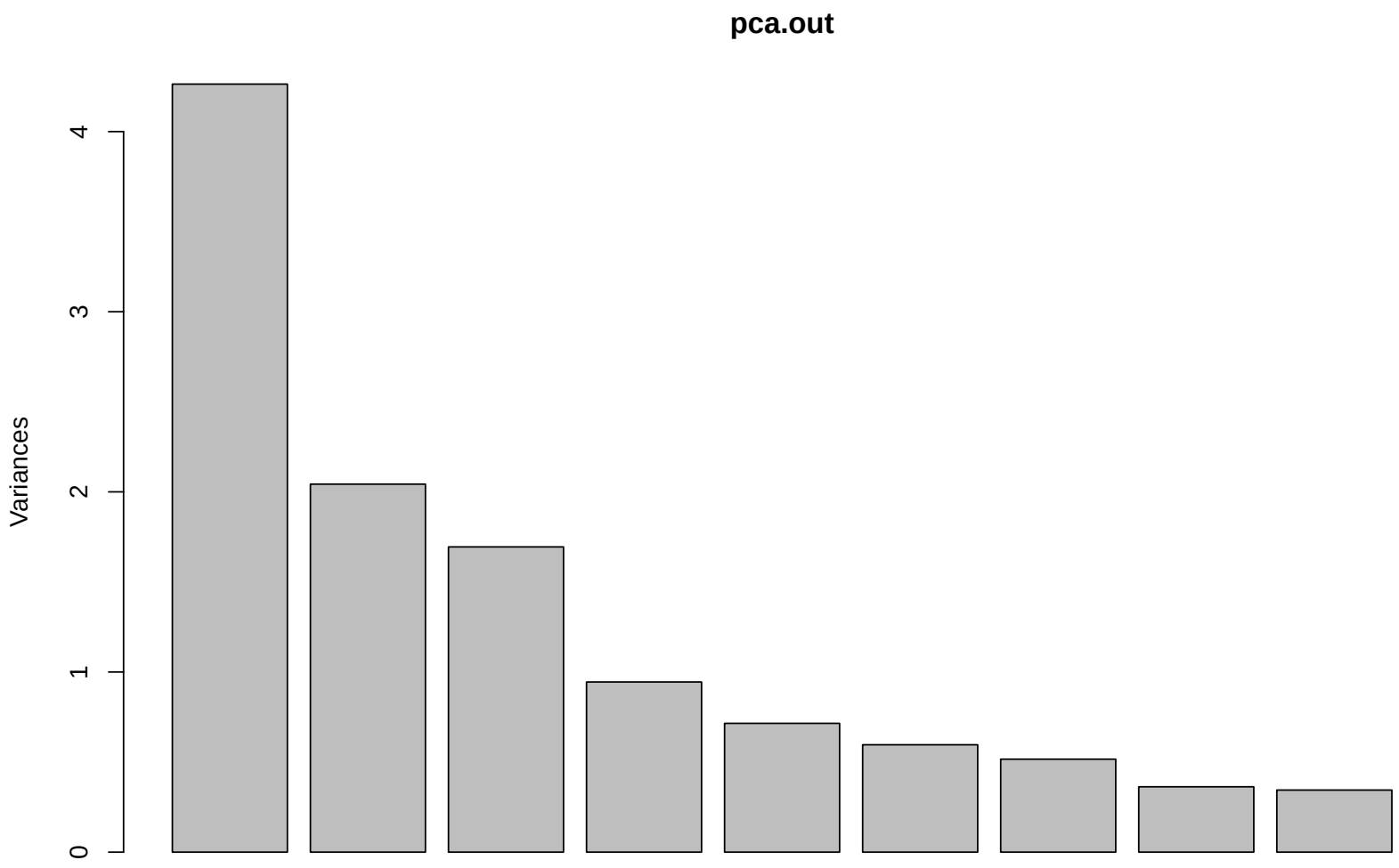
To do PCA: prcomp() – built-in
To do EFA: factanal() – builtin
fa() – from **psych**; multiple upgrades
efaUnrotate() – from **semTools**; can do FIML for missing data and WLSMV for categorical variables
GPA() – from **GPArotation** – one stop shop for factor rotations
**nFactors** package contains various 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

```
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, EFA, CFA

# PCA

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



# PCA continued

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
```

# 2 Components

```
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
```

# 3 Components

```
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
```

# 4 Components

```
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
```

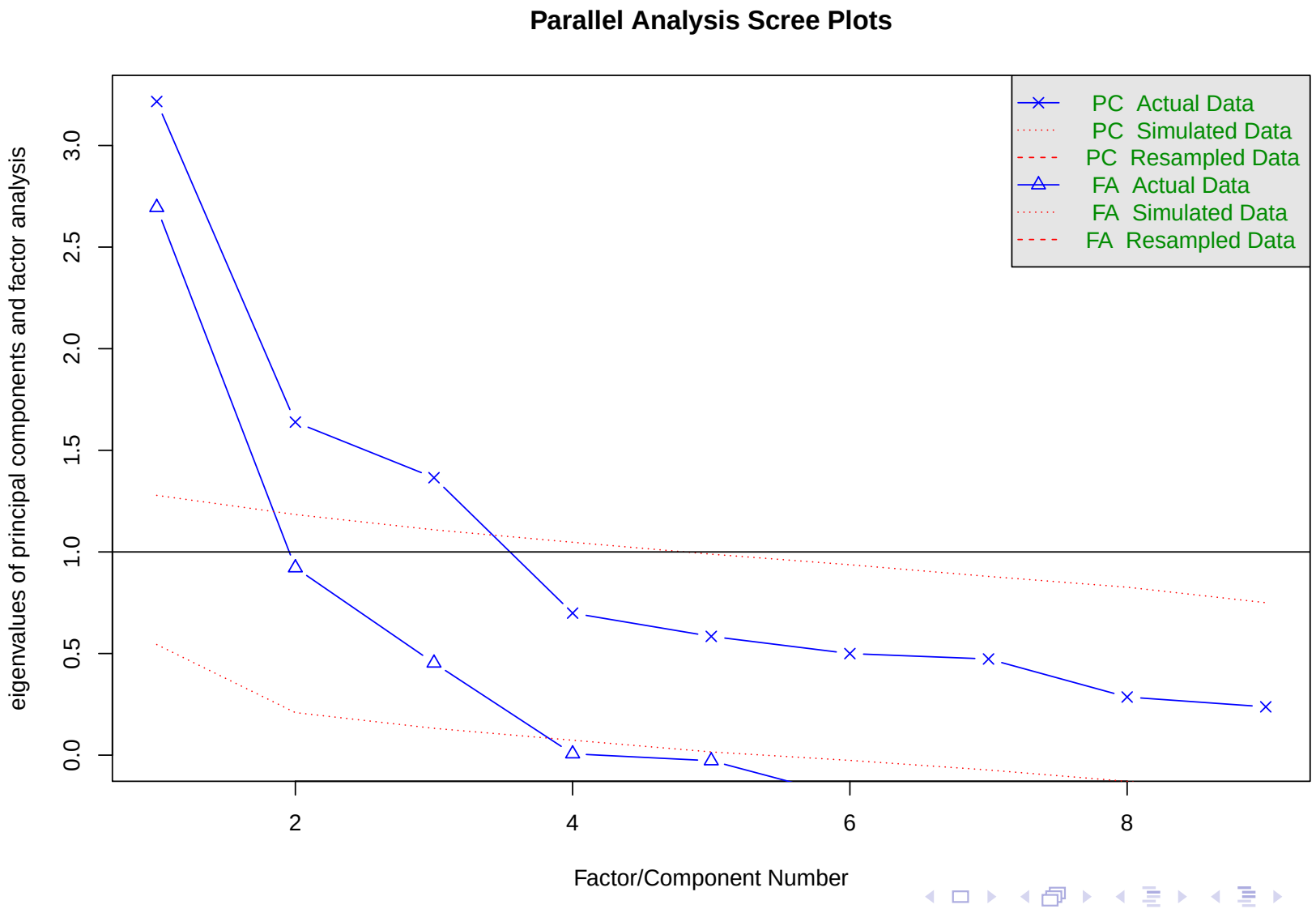
# PCA Continued

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.
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

# Parallel Analysis

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

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



# Parallel Analysis With Items

Not Run

```
library(random.polychor.pa)
data(bfi)
bfi.data<-na.exclude(as.matrix(bfi[1:200, 1:5]))
out <- random.polychor.pa(nrep=3, data.matrix=bfi.data, q.eigen=.99)
```

If your variables have 1-5 or 6 categories, then steps need to be taken to change the estimation procedure for a number of methods we will be talking about, including parallel analysis.

# EFA

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

```
fa.out <- factanal(HS[,7:15],3);loads <- fa.out$loadings
fa.out
```

```
##
## Call:
## factanal(x = HS[, 7:15], factors = 3)
##
## Uniquenesses:
##    x1    x2    x3    x4    x5    x6    x7    x8    x9
## 0.513 0.749 0.543 0.279 0.243 0.305 0.502 0.469 0.543
##
## Loadings:
##    Factor1 Factor2 Factor3
## x1  0.277   0.623   0.151
## x2  0.105   0.489
## x3          0.663   0.130
## x4  0.827   0.165
## x5  0.861
## x6  0.801   0.212
## x7                  0.696
## x8          0.162   0.709
## x9  0.132   0.406   0.524
##
##                Factor1 Factor2 Factor3
```

# Rotation

```
# 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
```

```
# 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

# Factor Scores

Get factor scores:

```
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.

# Confirmatory Factor Analysis

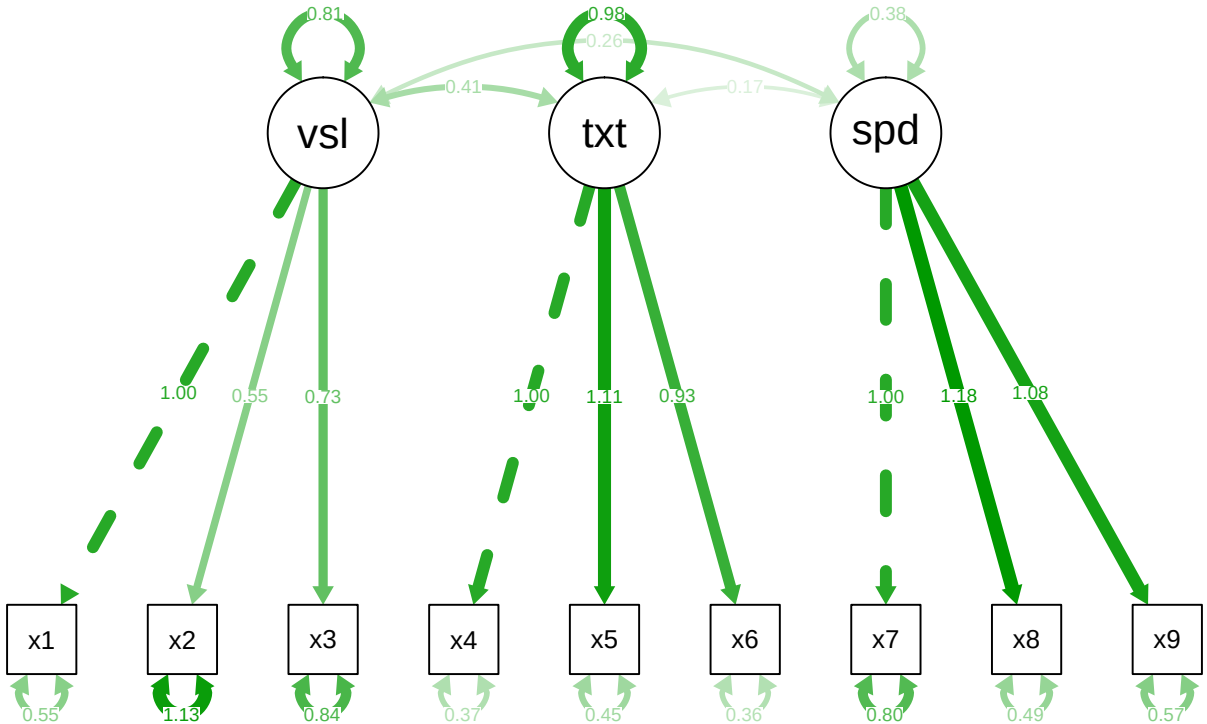
What if we have an idea as to what the structure of a model is?
Great tutorial: `http://lavaan.ugent.be/tutorial/tutorial.pdf`
This means we can specify a simple structure to the model

```
HS.model <- ' visual =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed =~ x7 + x8 + x9 '
fit <- cfa(HS.model, data=HolzingerSwineford1939)
#summary(fit, fit.measures=TRUE) too much output
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
```

# Plot CFA Model

```
semPlot::semPaths(fit,what="est")
```



# Various SEM (CFA) Tools

Get Fit Measures

```
fitMeasures(fit)
```

Modification Indices

```
modindices(fit)
```

Note that lavaan actually has four different functions to run models:
lavaan() - requires you to specify the full model
cfa() - only have to specify part of the model, makes assumptions for CFA models
sem() - makes assumptions typical in more complex sem models
growth() - makes specifying LGM easy, only need intercept and slope specification

# Longitudinal Analyses

# WISC Data

For this demonstration, we are going to compare the different packages and functions in analyzing longitudinal WISC data. Data is WISC4VPE.DAT.

```
wisc <- read.table("C:/Users/RJacobucci/Documents/GitHub/EDM_Labs/2015/wisc4vpe
names(wisc)<- c("V1","V2","V4","V6","P1","P2","P4", "P6", "Moeducat")
# note: V1 refers to verbal scores at grade 1, P is performance
```

# Visualization

To use many of the packages in R for longitudinal data (nlme), it is many times required to create a "long" data file, instead of the default wide.
How to do:

```
# get rid of performance variables
wisc.verb <- wisc[,c(1:4,9)]

# create subset for plotting
ntot <- nrow(wisc.verb)    # total number of observations
wisc.verb.sel <- wisc.verb[sample(ntot, 30), ]

wisc.long <- reshape(wisc.verb, varying = c("V1", "V2", "V4", "V6"), v.names =
                times = c(1, 2, 4, 6), direction = "long")

wisc.long.sel <- reshape(wisc.verb.sel, varying = c("V1", "V2", "V4", "V6"),
                         v.names = "verbal", times = c(1, 2, 4, 6),
                         direction = "long")
head(wisc.long,3)
```

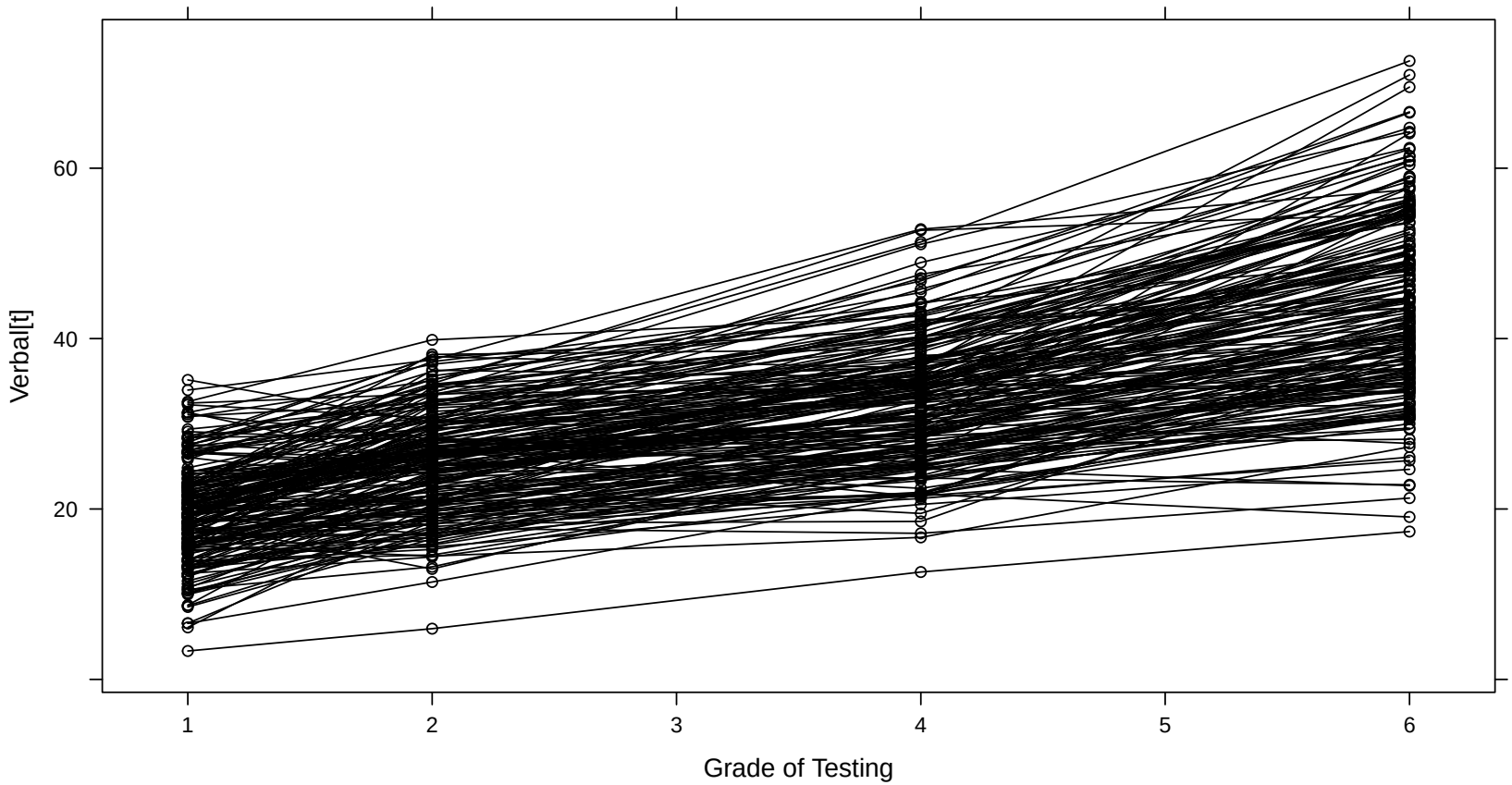
```
##     Moeducat time   verbal id
## 1.1        0    1 24.41964  1
## 2.1        0    1 12.44048  2
## 3.1        2    1 32.42560  3
```

```
names(wisc.long)[2] <- "grade"
names(wisc.long.sel)[2] <- "grade"
```

# Trajectories

Lets take a look at what the trajectories are:
First using lattice package

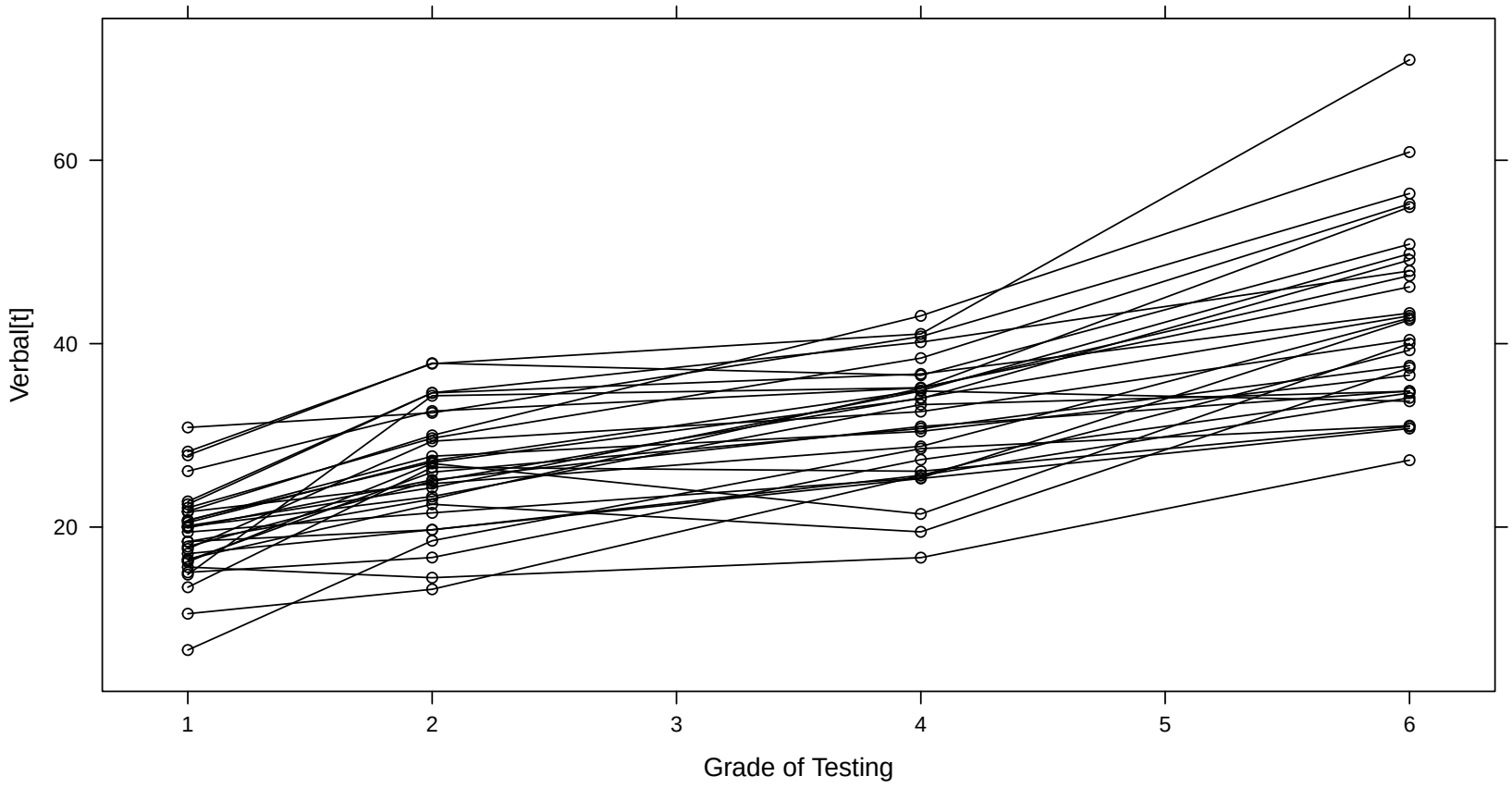
```
library(lattice)
xyplot(verbal ~ grade, groups = id, data = wisc.long, type = "o", col = "black"
       xlab = "Grade of Testing", ylab = "Verbal[t]")
```



# Subset Trajectories

This is hard to see, better to use subset

```
xyplot(verbal ~ grade, groups = id, data = wisc.long.sel, type = "o",
       col = "black", xlab = "Grade of Testing", ylab = "Verbal[t]")
```




```
# on average, scores went up over time
```

Thats a little better.

# GGplot2

But, can we simultaneously view the trajectories over time while seeing the influence that Mother's education may have?

```
library(ggplot2)
qplot(grade,verbal,group=id,data=wisc.long,alpha=I(1/2),colour=factor(Moeducat)
      geom = c("line","point"),xlab = "Grade of Testing", ylab = "Verbal[t]")
```


It seems pretty clear that mothers with higher levels of education have children that are consistently higher in verbal performance across time.
So now that we have an idea what will we find if we look at the relationship between mother's education and trajectory, lets test it with statistical models

# SEM Trees

# SEM Trees LGCM

**Note: SEM Trees can be used with any type of SEM that you can specify in lavaan or OpenMx**

Our first example is going to be using a latent growth curve model (lgcm) as our outcome, and attempting to find subgroups based on mother's education and the performance scores

Previous demonstrations using SEM Trees have used OpenMx. In this case, we will use lavaan.

```
linearGCM <- '
    inter =~ 1*V1 + 1*V2 + 1*V4 + 1*V6
    slope =~ 1*V1 + 2*V2 + 4*V4 + 6*V6
    inter ~~ vari*inter; inter ~ meani*1;
    slope ~~ vars*slope; slope ~ means*1;
    inter ~~ cov*slope;
    V1 ~~ residual*V1; V1 ~ 0*1;
    V2 ~~ residual*V2; V2 ~ 0*1;
    V4 ~~ residual*V4; V4 ~ 0*1;
    V6 ~~ residual*V6; V6 ~ 0*1;'
run <- lavaan(linearGCM,wisc) # could also have used growth()
#summary(run)
coef(run)
```

```
##     vari    meani     vars    means      cov residual residual residual
##   15.196   15.151    1.529    4.673    1.565   12.828   12.828   12.828
## residual
##   12.828
```

# 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 wit
# mediation parameter definition    :=
```

# Mediation Example

Example taken from: http://lavaan.ugent.be/tutorial/mediation.html

```
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)
```

# Mediation Continued

```
summary(fit.med)
```

```
## lavaan (0.5-20) 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
##
## Regressions:
##                    Estimate  Std.Err  Z-value  P(>|z|)
##   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:
```

# OpenMx

Same LGM as specified in lavaan

```
resVars <- mxPath( from=c("V1", "V2", "V4", "V6"), arrows=2,
                   free=TRUE, values = c(1,1,1,1),
                   labels=c("residual","residual","residual","residual") )
latVars<- mxPath( from=c("intercept","slope"), arrows=2, connect="unique.pairs",
                   free=TRUE, values=c(1,.4,1), labels=c("vari1","cov1","vars1")
intLoads<- mxPath( from="intercept", to=c("V1", "V2", "V4", "V6"), arrows=1,
                free=FALSE, values=c(1,1,1,1) )
sloLoads<- mxPath( from="slope", to=c("V1", "V2", "V4", "V6"), arrows=1,
                  free=FALSE, values=c(1,2,4,6) )
manMeans<- mxPath( from="one", to=c("V1", "V2", "V4", "V6"), arrows=1,
                 free=FALSE, values=c(0,0,0,0) )
latMeans<- mxPath( from="one", to=c("intercept","slope"), arrows=1,
                   free=TRUE,  values=c(0,1), labels=c("meani1","means1") )
dataRaw<- mxData( observed=wisc[,c(1:4,9)], type="raw" )
lgm.mod<- mxModel("LGM", type="RAM",
                   manifestVars=c("V1", "V2", "V4", "V6"),
                   latentVars=c("intercept","slope"),dataRaw,
                   resVars, latVars, intLoads, sloLoads, manMeans, latMeans)
mod.run <- mxRun(lgm.mod);coef(mod.run)
```

```
##  residual     vari1      cov1     vars1    meani1    means1
## 12.827561 15.195543  1.564602  1.528757 15.150807  4.673396
```

# Run semtree()

Now use "run" with semtree()

**Note: semtree() works better with OpenMx at this time**

Just used defaults

```
mytree <- semtree(mod.run,mydata=wisc[,c(1:4,9)]) # only moeducat as covariate
plot(mytree)
```

# Plot Trajectories

```
# create expected trajectories from parameters
expected.growth <- matrix(
    rep(t(parameters(mytree))[, "meani"], each=4) +
    rep(t(parameters(mytree))[, "means"], each=4)*c(1,2,4,6), nrow=2, byrow=T)
# plot expected trajectories for each leaf
plot(c(1,6), c(10,50), xlab="Grade", ylab="Verbal Score", type="n",main="SEM Tr
lines(c(1,2,4,6), expected.growth[1,], col="red", type="b", lw=3)
lines(c(1,2,4,6), expected.growth[2,], col="blue", type="b", lw=3)
legend("bottomright", c("Mom Ed. = 0", "Mom Ed. = 1 or 2"),col=c("red","blue"),
```

# SEM Trees Results Continued

We should get same results as in the left node of the tree by just subsetting the dataset based on Moeducat = 0
ould get same results as in the left node of the tree by just subsetting the dataset based on Moeducat = 0

```
wisc.sub <- wisc[wisc$Moeducat == 0,]
run.sub <- lavaan(linearGCM,wisc.sub)
coef(run.sub)
```

```
##      vari     meani      vars     means       cov residual residual residual
##    12.016    12.473     1.394     4.085    -0.936    10.950    10.950    10.950
## residual
##    10.950
```

Yup, everything checks out. This should make it clear that SEM Trees is really just subsetting the dataset into subgroups based on values of the covariates entered.

# SEM Trees Options

# Invariance of parameters

```
model <- ' i =~ 1*t1 + 1*t2 + 1*t3 + 1*t4
           s =~ 0*t1 + l1*t2 + l2*t3 + 3*t4 '
fit <- growth(model, data=Demo.growth)
tree.inv = semtree(fit,Demo.growth,invariance=c("l1","l2"))
```

This allows parameters to be freely estimated, but forces them to be the same in each group tested

# Additional Options

```
#?semtree.control
control <- semtree.control(method="fair",min.N=20,bonferroni=TRUE)
tree2 = semtree(fit,Demo.growth,control=control)
```

I prefer to set the method=“fair” as the default “naive” exhibits a preference for covariates with a large number of response options.

Additionally, I usually set the minimum number per node to be something greater than 20. Remember, each node is an actual SEM model.

Finally, if there are a large number of covariates, setting bonferroni=TRUE corrects for the number of comparisons. Usually doesn't make a difference

# Mixed Effects Trees

# longRPart

Instead of running the models in a SEM framework, longRPart uses mixed-effects models. This works just as well, as many LGCM can be re-specified as mixed-effects models.
For this we will use the nlme package to run a linear mixed effects model

# nlme

```r
#Linear growth
mix1 <- lme(fixed = verbal ~ grade, random = ~ grade | id,
            data = wisc.long, method="ML" )
summary(mix1) # get same estimates as in LGM, notice SD not VAR
```

```
## Linear mixed-effects model fit by maximum likelihood
##  Data: wisc.long
##        AIC      BIC    logLik
##   5050.817 5079.043 -2519.408
##
## Random effects:
##  Formula: ~grade | id
##  Structure: General positive-definite, Log-Cholesky parametrization
##             StdDev   Corr
## (Intercept) 3.898140 (Intr)
## grade       1.236429 0.325
## Residual    3.581559
##
## Fixed effects: verbal ~ grade
##                 Value Std.Error  DF  t-value p-value
## (Intercept) 15.150809 0.3681966 611 41.14870       0
## grade        4.673395 0.1085627 611 43.04789       0
##  Correlation:
##       (Intr)
## grade -0.155
##
```

# longRPart

```
lcart.mod1 <- longRPart(verbal ~ grade,~ Moeducat, ~ 1 | id,wisc.long)
```

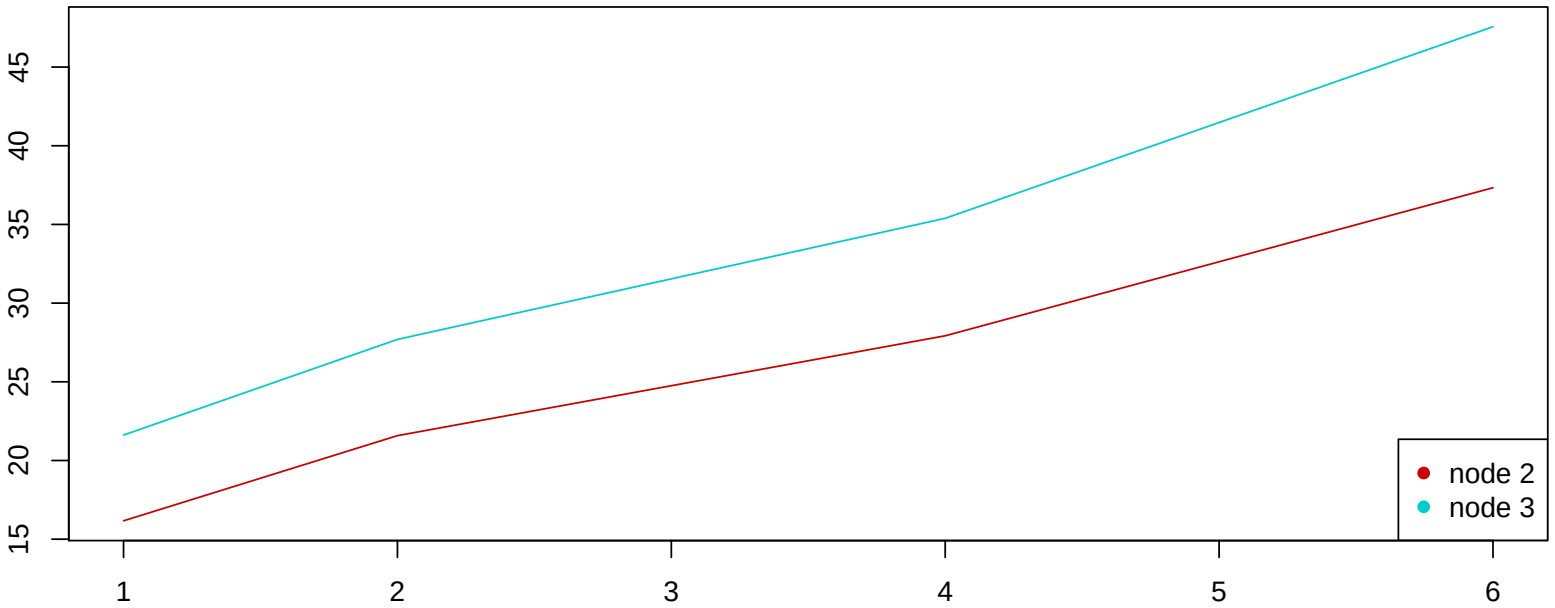
```
summary(lcart.mod1)
```

```
## Call:
## rpart(formula = paste(groupingName, c(rPartFormula)), data = data,
##     method = list(eval = evaluation, split = split, init = initialize),
##     parms = data, control = control)
##   n= 816
##
##           CP nsplit rel error
## 1 0.01981397      0  1.000000
## 2 0.01000000      1  0.980186
##
## Variable importance
## Moeducat
##      100
##
## Node number 1: 816 observations,    complexity param=0.01981397
## deviance (-2logLik) 5220.46 slope 4.7
##   left son=2 (304 obs) right son=3 (512 obs)
##   Primary splits:
##       Moeducat < 0.5 to the left,  improve=2661.947, (0 missing)
##
## Node number 2: 304 observations
```

# Plot Tree

```
lrpPlot(lcart.mod1)
```

```
## [[1]]
## [1] "1"
##
## [[1]]
## [1] "1"
```




```
lrpTreePlot(lcart.mod1,use.n=F)
```

# Additional Multivariate Trees

### REEMtree

Tree partitioning for longitudinal data where random effects exist. This doesn't really accomplish what we did previously with longRPart or SEM Trees. Interested, see the examples in following links.

`http://pages.stern.nyu.edu/~jsimonof/REEMtree/`

`http://www.r-bloggers.com/a-brief-tour-of-the-trees-and-forests/`

### mvpart

Like longRPart, also archived:

`http://cran.r-project.org/src/contrib/Archive/mvpart/`

If we treat the longitudinal data just as a multivariate outcome, we can accomplish a very similar process.

### mvtboost

Multivariate boosting

See for example code: `https://github.com/patr1ckm/mvtboost`