In this module I provide a few illustrations of options within lavaan for handling various situations.

An appropriate citation for this material is

Notes: IP-056512; Support provided by the USGS Climate & Land Use R&D and Ecosystems Programs. I would like to acknowledge formal review of this material by Jesse Miller and Phil Hahn, University of Wisconsin. Many helpful informal comments have contributed to the final version of this presentation. The use of trade names is for descriptive purposes only and does not imply endorsement by the U.S. Government.
Last revised 17.02.15.
A variety of special modeling issues are automated in lavaan.

Outline:

- Missing data
- Robust estimators for non-normal endogenous variables
- Bootstrapping
- Categorical responses
- Input data as a covariance matrix
- Simulating data

Additional capabilities of lavaan are presented in modules on other specific topics (e.g., Multi-Group Modeling and Adjusting for Nested Data).

This module illustrates automated procedures in lavaan that relate to some common modeling challenges.
Important issue – dealing with missing data. Traditionally "listwise deletion" is used when packages and functions encounter an empty data cell. This discards lots of information (all non-missing information in the rows deleted). Not only is this wasteful, it is also not proper because there may be particular conditions where missing cells are more likely. lavaan has simply automated procedures that use all the data even when some cells are missing.
1. Lavaan options for working with missing data.

Two degrees of assumptions about the pattern of missingness:
(a) MCAR – missing completely at random
(b) MAR – missing at random (pattern of missingness not correlated with model predictors.)

Lavaan default is listwise deletion if you do not using "missing=".

You can invoke FIML (full-information maximum likelihood) in lavaan by declaring ‘\texttt{missing = FIML}’ in the fitting command.

```r
# specific model
mod1 <- 'cover ~ firesev
    firesev ~ age'

# Fit model
mod1.fit <- sem(mod1, data=k.dat, missing="FIML")
```

So-called "full-information maximum likelihood" is a very powerful option for performing analyses in the presence of missing data. It can be justified in two different situations, MCAR and MAR.

A useful source of information is

and

lavaan FIML methods first examine the patterns of missingness in the data. There is actually minimal reporting for the method (just the "Number of missing patterns" shown here).
Results from missing="FIML" (continued).

| Regressions: | Estimate | Std.Err | Z-value | P(>|z|) |
|-------------|----------|---------|---------|---------|
| cover ~ firesev | -0.090 | 0.018 | -4.855 | 0.000 |
| firesev ~ age | 0.057 | 0.012 | 4.568 | 0.000 |

| Intercepts: | Estimate | Std.Err | Z-value | P(>|z|) |
|-------------|----------|---------|---------|---------|
| cover       | 1.095 | 0.089 | 12.339 | 0.000 |
| firesev     | 3.061 | 0.358 | 8.557 | 0.000 |

| Variances: | Estimate | Std.Err | Z-value | P(>|z|) |
|------------|----------|---------|---------|---------|
| cover      | 0.080 | 0.012 | 6.553 | 0.000 |
| firesev    | 2.167 | 0.332 | 6.535 | 0.000 |

<table>
<thead>
<tr>
<th>R-Square:</th>
<th>Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>cover</td>
<td>0.212</td>
</tr>
<tr>
<td>firesev</td>
<td>0.194</td>
</tr>
</tbody>
</table>

Estimates are shown as usual.
Robust Estimators

Methods have been developed to provide estimates that are robust to deviations from the assumption of normal errors. Here we see what lavaan has to offer in that area.
1. Lavaan permits use of “robust” estimation.

Lavaan has two main options for robust estimation:

MLM – produces chi-squares and standard errors robust to non-normality. AKA the Satorra-Bentler correction.

MLR – similar to MLM, but uses the Yuan-Bentler method so that missing data can be accommodated.

see discussion in:

Robust means the inferences are robust to deviations from normality in the response variables. A useful and brief overview can be found at http://web.pdx.edu/~newsomj/semclass/ho_estimate.pdf
2. Robust estimation invoked with `estimator =` command.

```r
# create model
mod <- 'y2 ~ y1
    y1 ~ x1'

# estimate model
mod.fit <- sem(mod, data=dat, fixed.x=F,
               estimator="mlm")

# get results
summary(mod.fit)
```

“fixed.x=F” is required when using “mlm” option.

Declaring the estimator when fitting a lavaan model is simple.
3. Results

<table>
<thead>
<tr>
<th>Estimator</th>
<th>ML</th>
<th>Robust</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chi-square</td>
<td>4.213</td>
<td>4.082</td>
</tr>
<tr>
<td>Degrees of freedom</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P-value</td>
<td>0.040</td>
<td>0.043</td>
</tr>
<tr>
<td>Scaling correction factor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>for the Yuan-Bentler correction</td>
<td></td>
<td>1.032</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Standard Errors</th>
<th>Robust.mlm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression:</td>
<td>Estimate</td>
</tr>
<tr>
<td>y2 ~ y1</td>
<td>-0.154</td>
</tr>
<tr>
<td>y1 ~ x1</td>
<td>1.185</td>
</tr>
</tbody>
</table>

The results output shows the adjusted values.
A commonly used approach to estimating probabilities is resampling and one particularly popular form of resampling is bootstrapping (sampling with replacement).
1. Lavaan has resampling methods for non-normal data.

```r
# fit model and request bootstrapped results
mod.fit <- sem(mod, dat=dat,
               test="boot", se="boot", bootstrap=200)
```

Bootstrapping is likewise a simple operation in lavaan.
2. Results

<table>
<thead>
<tr>
<th>Estimator</th>
<th>ML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chi-square</td>
<td>4.213</td>
</tr>
<tr>
<td>Degrees of freedom</td>
<td>1</td>
</tr>
<tr>
<td>P-value</td>
<td>0.040</td>
</tr>
<tr>
<td>P-value (Bollen-Stine)</td>
<td>0.053</td>
</tr>
</tbody>
</table>

We can expect bootstrapped results to give both different standard errors and p-values because it is not just an adjustment of the standard errors.
Another common issue is when one has response variables that are ordered categorical. In this situation, errors are generally non-normal.
Fitting a straight line through such a set of points represents the points quite poorly and leads to illogical extrapolations, like intercepts > 1 or < 0. It also violates assumptions about normality of residuals. What we need is a way to interpret binary outcomes that makes sense. Often this is accomplished by assuming that behind the binary outcomes lies a continuous probability of observing a 1 or 0 response, as shown on the next slide.
Two of the most common ways of representing the probability of observing a 1 or 0 outcome are the probit and logit models.

For the probit model, we link our predictor to our responses using a cumulative normal probability function. With the logit model, we link our predictor to our responses using an inverse log transformation of the ratio of probabilities of outcomes.
Two requirements
(1) Declare categorical variable to be “ordered” object in R.
(2) Declare variables that are ordered categorical in the “sem” statement.
Regression weight of -3.885 specifies the effect of one unit change in flood-level on the probability of observing mass_class = 1.

Error variance = 1.0 because it is set to that value to identify the model.

He also gives more technical background at http://www.personality-project.org/r/tutorials/summerschool.14/rosseel_sem_cat.pdf
It is possible to input the covariance matrix as the data for analysis in lavaan.
The "getCov" function is used to tell lavaan a covariance matrix is being used as input. Here only the lower matrix values are inputted, which implies the matrix is symmetrical.

You have to tell lavaan the sample size for estimation to take place.
A full set of results can be obtained from the covariance matrix alone.
Using the lavaan "simulateData" function

In this module I illustrate a particular lavaan option.
An appropriate citation for this material is

Notes: IP-000000; Support provided by the USGS Climate & Land Use R&D and Ecosystems Programs. I would like to acknowledge formal review of this material by XXXXX and XXXXX. The use of trade, firm, or product names is for descriptive purposes only and does not imply endorsement by the U.S. Government. Last revised 16.02.15. Questions about this material can be sent to sem@usgs.gov.
To simulate data in lavaan, you have to provide the values for the population parameters (in red). You also have to set a seed (if you want results each time to be the same).

A "simulateData" function is evoked, along with a statement about the number of samples requested. All the examples I have found use the capital letter "L" at the end of the number of samples, but I have not found an explanation.

This produces a new data frame "sim.dat1".

Syntax for the simulateData function in lavaan can be found at www.inside-r.org/packages/cran/lavaan/docs/simulateData

```r
library(lavaan)
# Here is a simple example to get us started:
sim.mod <- 'y2 ~ 0.5*y1
       y1 ~ 0.7*x1
'

# generate data
set.seed(1)
sim.dat1 <- simulateData(sim.mod, sample.nobs=10000L)
```

USGS
Continued:

```r
# fit model
test.mod <- 'y2 ~ y1
   y1 ~ x1
'
fit <- sem(test.mod, data=sim.dat1)
summary(fit)
```

We can test to see what values we recover from our simulated data.
At this large sample size, we recover the estimated parameters.