

1. Guideline to developing an appropriate linear model

With any dataset, it is necessary to go through data diagnostics, transformation, and variable selection when putting together a model.

Diagnostics = checking constant variance, residual normality, outliers, leverage
Variable Selection = a priori inclusion, testing (forward, backward elimination) and criterion based methods (AIC, Adjusted R-square)

In GENERAL, the following is a good guideline: Diagnostics → Transformation → Variable Selection
→ Diagnostics

2. Getting data into R

I find it easiest to store data as “.csv” files using Microsoft Excel. This way a permanent copy of the data file can be saved, and the .csv file can be read into R at the beginning of each analysis session. Outliers can be deleted from the original .csv file if necessary, so this won't have to be done each time a data set is used. I also find it useful to save lines of code (as in the ordered factoring example below) in a .txt file, but don't save the “>” symbols if you do this. A whole set of commands can then be copied and pasted into the R console from the .txt file, which can save a considerable amount of time.

Example: Creating a dataframe from a “.csv” file. The “header=T” argument tells R to recognize column labels in the dataframe. Keep in mind you have to change the R working directory to the folder in which the “example_data.csv” is stored in order for the command below to work (⌘-D on a Mac).

```
> example.df = read.csv("example_data.csv", header=T)
```

3. Correlation and simple linear regression

Correlation quantifies how well two variables X and Y covary together. Correlation only makes sense when both X and Y are variables that you measure. If you control X and are measuring Y, then use linear regression. Correlation does not discriminate between X and Y, and linear regression does – only use regression if you can clearly define which variable is X (i.e. is independent) and which is Y (dependent).

In addition to importing data from .csv files, it is possible to create vectors of data for use with R. To demonstrate how to carry out a Pearson's correlation analysis and a linear regression, first make vectors of elevation, precipitation, temperature, and soil pH data:

```
# Input of elevation data in meters  
> m = c(1500,2020,2720,3400)  
#Input of precip data (mm) at each elevation  
> ppt = c(938.3,797.6,593.2,475.6)  
# Input of mean temperature data (C°) at each elevation  
> temp = c(17.9,14.9,10.8,7.1)  
# Input of soil C:N ratio at each elevation  
> cn = c(21.04, 21.56, 19.33, 19.27)  
# Input of 0-horizon soil pH at each elevation  
> pH = c(4.25, 4.1, 4.2, 4.12)
```

A. Calculation of Pearson's correlation coefficients for selected pairs of variables

For the "cor" function, Pearson's correlation coefficient is the default method. It is possible to add a method="spearman" or method="kendall" argument if these coefficients are desired.

```
> cor(m,ppt)
r = -0.9941432

> cor(m, temp)
r = -0.9998673

> cor(temp, ppt)
r = 0.995771

> cor(m, cn)
r = -0.8520494

> cor(m, pH)
r = -0.4933305

> cor(temp, pH)
r = 0.489425
```

B. Plot all combinations of variables to examine the shapes of the relationships

```
# First make a dataframe from the vectors previously created
> abiotic = data.frame(elevation.m = m, precip.mm = ppt, temp.C = temp, cn =
cn, pH = pH)

# Plot all possible combinations of these variables with blue symbols
> plot(abiotic, col=4)
```

C. Using linear regression to calculate a line of best fit between two variables.

In order to be theoretically valid, assume that there is a "dependent/independent" type of relationship between soil pH and temperature, with temperature as the independent variable.

```
m1 = lm(pH~temp.C, data=abiotic)

> summary(m1)

Call:
lm(formula = pH ~ temp.C, data = abiotic)

Residuals:
    1      2      3      4 
0.044610 -0.083635  0.046097 -0.007072

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  4.075586   0.121677  33.495  0.00089 ***
temp.C       0.007252   0.009136   0.794  0.51057
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Residual standard error: 0.0747 on 2 degrees of freedom
Multiple R-squared: 0.2395, Adjusted R-squared: -0.1407
F-statistic: 0.63 on 1 and 2 DF, p-value: 0.5106

Note that the R-squared value calculated by the linear regression procedure gives the same “r” – by taking the square root – as does the Pearson’s correlation analysis.

4. One-sample t-test

```
> t.test(variable.name, mu=expected mean)
```

For this test, mu=0 is the default when nothing is specified; otherwise, any number can be entered in place of “expected mean” in the code above. The dependent variable, called “variable.name” here, can be a simple vector or can be a part of a larger data frame.

5. Two-sample t-test (assuming unequal variance)

Assume a dependent variable (“dep.var”) and a categorical variable (“cat.var”) are stored in a data frame named “example.df”

```
> t.test(dep.var~cat.var, data=example.df)
```

To get the t-test in which variance is assumed to be equal, type:

```
> t.test(dep.var~cat.var, var.equal=T, data=example.df)
```

6. Checking the homogeneity of variance assumption

- A. Plotting residuals vs. fitted values is a good way to check whether the variance in the data are constant. The vertical scatter surrounding a horizontal line running through y=0 should be random:

Assuming a simple linear model “m”, and using a dataframe named “example.df”:

```
> m = lm(Y ~ X, data=example.df)  
> plot(fitted(m), resid(m))  
> abline(h=0)
```

The last two commands above will return a plot of the residual values vs. the fitted values for the model, with a horizontal line through y=0.

- B. One can also plot the absolute value of the residuals vs. the fitted values in order to increase the resolution of one's ability to detect whether there's a trend that would suggest non-constant variance. This is done with the following commands:

```
> plot(fitted(m), abs(resid(m)))
```

- C. A quick way to check non-constant variance quantitatively is this simple regression:

```
> summary(lm(abs(resid(m)) ~ fitted(m)))
```

- D. For a categorical variable, can graphically check how the levels are behaving (i.e. – is there any difference among the levels in terms of variance) like this:

```
> plot(resid(m) ~ fitted(m), pch=unclass(example.df$category))
```

The argument “pch” specifies how the “plot characters” should be drawn.
Can also use the jitter command so that categories don’t overlap:

```
> plot(jitter(fitted(m)), resid(m), xlab="Fitted", ylab="Residuals")
```

Need to look for non-constant variance across all groups, and whether there's any ordering among groups.

- E. It is also possible to use LEVENE's TEST to assess the homogeneity of variance assumption. Levene's test is quite insensitive to non-normality. Also, because most parametric statistical tests are relatively insensitive to non-constant variance, there is no need to take action unless Levene’s Test is significant at the 1% level (Faraway 2004).

```
> library(car)  
> levene.test(Y, X)
```

7. Checking the Normality assumption (residuals should be normally distributed for linear models). Note: Only long-tailed distributions cause large inaccuracies for linear models. It is possible to transform log-normal distributions, but mild non-normality can be safely ignored. Cauchy distributions present problems (See Faraway 2004, p. 59-60).

- A. Make a quantile-quantile (Q-Q) plot that compares the model’s residuals to “ideal” normal observations:

```
> qqnorm(resid(m), ylab="Residuals")  
> qqline(resid(m))
```

The latter command puts a straight line onto the graph that joins the 1st and 3rd quartiles (and is therefore not influenced by outliers).

To graphically display how the residuals are distributed, one can plot the density distribution of the residuals:

```
> plot(density(resid(m)))  
> rug(resid(m))
```

The “rug” command puts the individual points as a stripchart beneath the density plot.

- B. To produce 9 examples of normally distributed residuals:

```
> par(mfrow=c(3,3))  
> for(i in 1:9) qqnorm(rnorm(50))
```

The first command tells the basic R graphics engine to produce plots within a 3x3 matrix, rather than the default 1x1 matrix.

- C. To produce nine examples of a log-normal distribution (observations skewed/ clustered toward the left of the distribution, long right tail):

```
> par(mfrow=c(3,3))  
> for(i in 1:9) qqnorm(exp(rnorm(50)))
```

- D. To produce nine examples of a Cauchy distribution (most observations clustered in the middle of the distribution, tails typically on either side, but can be one or the other):

```
> par(mfrow=c(3,3))
> for(i in 1:9) qqnorm(rcauchy(50))
```

- E. The Shapiro-Wilk test is a formal test of normality, but should be used only in conjunction with Q-Q plots AT BEST:

```
> shapiro.test(resid(m))
```

For smaller sample sizes, Shapiro-Wilk lacks power.

8. Checking for outliers and points with undue leverage in a linear model

- A. Use of leverage plot:

```
> m = lm(Y ~ X1 + X2 + X3 + X4, data=example.df)
> gi = influence(m)
> qqnorm1(gi$coef[, 4])
```

This produces a leverage plot for the X3 term in the model, since inclusion of the intercept makes X3 the 4 term in the model.

- B. Can also use a halfnormal plot of Cook's Distance to determine points with undue influence that are likely outliers:

Again, assume dataframe=example.df

```
> library(faraway)
> halfnorm(cooks.distance(m))
```

Say two points were identified that look extreme, with Cook's Distance values > 0.2. Can find which points these are with a logical operator:

```
> example.df[cooks.distance(m) > 0.2, ]
```

- C. If outliers are identified (e.g. – cases 6 and 24 in a database), can exclude those points like this:

```
> m = lm(Y ~ ., subset=-c(6,24))
```

The “~.” notation means use all Xs initially described above.

- D. Another approach for categorical data is to plot the raw data with a boxplot and look for extreme cases:

```
> m = lm(Y ~ cat, data=example.df)
```

where “cat” is a categorical variable with a₁ levels, in the “example.df” data frame

```
> boxplot(Y ~ cat, example.df)
```

If an outlier is identified in level a₂, can look at the elements of a₂ like this:

```
> subset(example.df, example.df$cat=="a2")
```

9. Variable Selection and Model Building

- A. Adding and removing model terms, using dataframe="example.df":

```
> m = lm(Y ~ X1+X2+X3+X4, example.df)
> summary(m)
```

The "summary" command will give coefficients and p-values for all terms in the model. To remove the term "X1" from the model m, type

```
> m1 = update(m, .~. -X1) and the new model is stored as "m1"
> summary(m1)
```

- B. **Comparing models m and m1 above using an F-test** via the "anova" function: Consensus view among statisticians(?) is that this method is inferior to criterion based approaches (AIC), but is useful when certain terms must be included in the model and one wishes to respect the heirarchy principle

```
> anova(m, m1)
```

If the models are significantly different, the F-test will return a p-value <0.05

- C. **Model selection using Akaike's Information Criterion (AIC):** this method allows categorical independent variables, and respects the heirarchy principle (i.e. – if a term is in a significant interaction, but is not itself significant, it is still included).

```
> m = lm(Y ~ X1*X2*X3*X4, example.df)
> m1 = step(m, trace=F)
```

The "trace=F" argument blocks the large amount of intermediate output we would otherwise see.

- D. **Model selection using "drop1" command** – uses AIC and F-tests to test each term relative to the full model.

```
> drop1(m, test="F", scope=~X2+X3+X4)
```

Where test="F" requests an F-test of the new "drop1" model vs. the original, and the "scope" argument specifies which terms should be considered for dropping.

Useful to remember that for ANOVA, the order of entry of terms into the model is important, and "drop1" provides a way to test all ways in which terms are entered.

- E. **Model selection using Mallow's Cp statistic and adjusted R-square:** only useful when independent variables are quantitative rather than categorical (Faraway 2006, p. 21)

```
> library(leaps)
> b = regsubsets(Y ~ X1 + X2 + X3 + X4, force.in=1, data=example.df)
> rs = summary(b)
> rs
> rs$adj          ## This term lists the adjusted R-square values as the
```

stepwise procedure adds each term listed in the
regsubsets" command

"regsubsets" generates all possible model permutations, starting with the strongest single "X" predictor and testing inclusion of all other variables.

The "force.in" command says always have term "X1" in this case.
Can also use this as a short-cut when specifying the full model:

```
> b = regsubsets(Y ~ ., data=exampe.df)
```

1. Looking at Mallows' Cp statistic graphically, using the above example:

```
> plot(2:4, rs$cp, xlab="No. of Parameters", ylab="Cp statistic")  
> abline(0,1)
```

Cp is a measure of the average mean square error of prediction for a given model. For a good model, Cp should fall on or below the (0,1) line.

2. Looking at adjusted R-square graphically, using the above example:

```
> plot(2:4, rs$adjr2, xlab="No. of Parameters", ylab="Adjusted R-square")
```

The "rs\$adjr2" term extracts the adjusted r-square values for each term in the model and plots them on the y-axis. This way it is possible to quickly see which number of predictors gives the best fit, but is not over-fit.

To obtain output describing the model that optimizes the adjusted R-square:

```
> rs$which[which.max(rs$adjr),]
```

10. Graphically checking multiple regression model structure

A. PARTIAL REGRESSION: Obtaining the effect of an independent variable on a dependent when there is more than one independent, i.e. $Y = X1 + X2$

If we're interested in how X1 affects Y, first calculate the variance in Y that is left over after we have modeled the effect of X2 on Y,

```
> resY = residuals(lm(Y ~ X2))
```

Then, remove the effect of X2 on X1,

```
> resX1 = resid(lm(X1 ~ X2))
```

Then plot the residuals of X1 vs. the residuals of Y,

```
> plot(resX1, resY)  
> coef(resY ~ resX1)
```

The last command gives the intercept and slope for the partial regression of X1 vs. Y. Could also use "summary" command to get the coefficients and see if the relationship is significant.

- B. PARTIAL RESIDUAL PLOT: Useful for determining the effect of X1 on Y while factoring out the effect of X2 on Y. “b” and “c” are equivalent ways of obtaining partial residual plots, but I am not sure if “a” gives the same analysis or not.

- a. Again, assume $Y = X1 + X2$, in `dataframe=example.df`

```
> m = lm(Y ~ X1 + X2, data=example.df)
> termplot(m, partial=T, term=1)
```

The “term=1” refers to variable X1, and “term=2” would refer to X2 and so on.

- b. Can also use command “prplot” from library(faraway):

```
> library(faraway)
> prplot(m, 1)
```

where “m” is the model specified, and 1 is the first independent variable in the model. Can use “summary(m)” to determine what number corresponds to what variable in the model.

- c. This is the manual way of obtaining a plot like that we obtain in “b”. Doing it manually is sometimes advantageous when one wants more control over plotting characteristics or if one wants to save the “new” adjusted dependent variable as a new vector.

Assume $Y = X1 + X2$, in `dataframe=example`; “m” is defined as above. We are interested in factoring out the effect of X2 on Y, and plotting the remaining variance in Y against X1.

```
> adj.Y = resid(m) + coef(m)[‘X1’]*example.df$X1
> plot(adj.Y~X1, data=example.df)
> abline(0, coef(m)[‘X1’])
> adjY.X1 = cbind(example.df$X1, adj.Y)
> write.csv(adjY.X1, file=”partial_residual.csv”)
```

The first line creates a vector containing the variation in Y that is left over after variation due to the X2 term has been factored out. The second and third lines plot `adj.Y` versus X1, and produce a line of best fit. The fourth and fifth arguments create a data table with the X1 values in one column and the `adj.Y` values in the second column, and then this data table is written to a .csv file that can be stored for later use.

11. Analysis of Variance (ANOVA) and Analysis of Covariance (ANCOVA)

If the independent categorical variable is numerically coded, need to make sure that R knows the numbers used for coding represent levels of a factor, and that the codes are not treated in a quantitative manner, which would lead to some sort of regression rather than the desired ANOVA.

First, read in the data file, creating a working data frame in R out of a .csv file saved from Excel, then change the numerical codes of the categorical variable to levels of a factor.

Example of “factoring” a categorical variable with 6 levels, assuming dependent variable “Y”:

```
> example.df = read.csv("example_data.csv", header=T)
> example.df$cat = factor(example.df$cat, levels=1:6)
> levels(example.df$cat) = c("Cat1", "Cat2", "Cat3", "Cat4", "Cat5", "Cat6")
```

```

> summary(example.df$cat)
> attach(example.df)
> m = lm(Y ~ cat)
> summary(m)
> anova(m)

```

The “summary” command will give coefficients for the terms in the model, as well as R-square statistics, etc. The “anova” command gives F-statistics and p-values. In this case, it was not necessary to specify “data=example.df” when we wrote out the model in line 6 because the data frame was “attached” in line 5. “Attaching” a data frame makes all of its variables accessible by name without having to specify which data frame they come from. This can be useful when only a small number of data frames are defined, and multiple data frames do not have variables with the same name.

- A. **Note about ordered factors:** can use the command "as.ordered" to tell R that the levels of a factor correspond to increasing or decreasing amounts of that factor. Consider the example in which plants are fertilized with "Ambient" "Low" and "High" amounts of N, resulting in the model:

Growth = N.treatment + error, in dataframe “N.fert”

```

> N.fert$N.treatment = as.ordered(N.fert$N.treatment)
> m = lm(Growth ~ N. treatment, data=N.fert)
> summary(m)

```

The final “summary” command gives a test of linear and quadratic contrasts for the ordered variable.

- B. When a dataset has missing values, the missing data must be coded as NA (must use capitals) within Microsoft Excel.
- C. The command to use when performing ANOVA when data include NA values is:

```

> anova(lm(Y ~ category, na.action=na.exclude))

```

It is also possible to use na.action=na.omit, though I haven’t figured out exactly what the difference is. I know that the number of residuals is different when you use these two different approaches though...

- D. For ANOVA models, it is still necessary to plot the residual and fitted values, and to make a Q-Q plot of the residuals. Transformation of the response is worth considering, but transformation of the predictor makes no sense.

- a. Example: model $Y = X + e$, where X is categorical and e is the error, in dataframe="example.df"

```

> m = lm(Y ~ X, example.df)
> qqnorm(resid(m))
> qqline(resid(m), lty=2)      ## “lty=2” gives a dashed line

> plot(jitter(fitted(m), resid(m), xlab="Fitted", ylab="Residuals"))

```

The “jitter” command above makes the fitted values, which would otherwise all be horizontally on top of each other since they are in the same category, appear staggered. This makes visually assessing non-homogeneity of variance a little easier.

b. Can also use Levene's Test to assess homogeneity of variance

E. Syntax for ANCOVA models (or ANOVAs with more than one variable)

a. Example: $Y = \text{cont} + \text{cat1} + \text{cat2}$, in data frame = "example.df"

```
> m = lm(Y ~ cont*cat1*cat2, example.df)
```

The "cont*cat1*cat2" term specifies a model with all main effects and all interaction terms. If not all interaction terms are desired, can specify which ones are needed:

```
> m = lm(Y ~ cont + cat1 + cat2 + cat1:cat2)
```

When making these models, recall that order of entry is important to determining whether a term is "significant." Using the "drop1" command, detailed above, can test whether each predictor is significant compared to the full model.

b. To graphically see the effect of more than one variable, use "interaction.plot" syntax:

```
> interaction.plot(x.factor, trace.factor, y)
```

where the dataframe has already been "attached".

F. **Within treatment comparisons** (i.e. – contrasting levels within a factor)

- a. A priori comparisons: It is possible to specify a priori contrasts with lm objects by using the "contrasts" argument. Crawley (see References) details how to do this, but I have not done this yet myself...
- b. Post-hoc comparisons: Use the TukeyHSD command to make all possible pair-wise comparisons of levels within a factor

Example:

```
> m = aov(Y ~ cont + cat1*cat2, example.df)
> TukeyHSD(m, cat1)
```

The "TukeyHSD" command allows one to specify which variable is used to produce the pair-wise comparisons. The above code specifies comparisons will only be made for levels within the "cat1" variable.

Note that the "aov" syntax is needed rather than the "lm" syntax, in order for "TukeyHSD" to extract the pair-wise comparisons.

G. **Obtaining and plotting predicted values from a linear model object**

Example: have a linear model object called "test.lm", based on X and Y data from a dataframe called "example.df". We want to see how the predicted values for Y from this model compare to the observed values for Y

```
> test.lm = lm(Y~X, data=example.df)
> predict(test.lm)
```

To plot the observed values versus the predicted values:

```
> plot(example.df$Y~predict(test.lm))
```

To save the predicted values for use in another program:

```
> write.csv(predict(test.lm), file="predicted_example.csv")
```

12. **Block Design Experiments** – note: the preferred design is to have the number of blocks EQUAL to the number of treatments. Also, models should always include the “block” term entered into the model first, as one is typically looking for the explanatory power of treatments after variability due to “block” has already been removed.

- A. Example: 8 different oat varieties grown in 5 different blocks, each variety is grown once per block. Have variables "yield" "variety" "block" in dataframe = “oats”

Graphically, we can use an interaction plot:

```
> attach(oats)
> interaction.plot(variety, block, yield)
> interaction.plot(block, variety, yield)
```

- B. Linear mixed effects models are also commonly used when an experiment has a blocked design, with “block” typically incorporated into the model as a random effect.

13. **Robust Regression**: useful when the residual error is not normally distributed (i.e. – when a Q-Q plot indicates long-tail errors due to a Cauchy distribution). Robust regression downweights the effects of any datum with a large residual error.

Found in

```
> library(MASS)
> m = rlm(Y ~ X, data=example.df)
```

14. **Linear mixed effects models**: These models are used when there are both fixed and random effects that need to be incorporated into a model. Fixed effects usually correspond to experimental treatments for which one has data for the entire population of samples corresponding to that treatment. Random effects, on the other hand, are assigned in the case where we have measurements on a group of samples, and those samples are taken from some larger sample pool, and are presumed to be representative. As such, linear mixed effects models treat the error for fixed effects differently than the error for random effects.

- A. Estimation of random effects for lme models in the NLME package is accomplished through use of both EM (Expectation-Maximization) algorithms and Newton-Raphson algorithms. EM iterations bring estimates of the parameters into the region of the optimum very quickly, but convergence to the optimum is slow when near the optimum.

Newton-Raphson iterations are computationally intensive and can be unstable when far from the optimum. However, close to the optimum they converge quickly.

The LME function implements a hybrid approach, using 25 EM iterations to quickly get near the optimum, then switching to Newton-Raphson iterations to quickly converge to the optimum. If convergence problems occur, the “control” argument in LME can be used to change the way the model arrives at the optimum.

- B. A general method for comparing nested models fit by maximum likelihood is the likelihood ratio test. This test can be used for models fit by REML (restricted maximum likelihood), but only if the fixed terms in the two models are invariant, and both models have been fit by REML. Otherwise, the argument: `method="ML"` must be employed (ML = maximum likelihood).

Example of a likelihood ratio test used to compare two models:

```
> anova(model1, model2)
```

The output will contain a p-value, and this should be used in conjunction with the AIC scores to judge which model is preferred. Lower AIC scores are better.

Generally, likelihood ratio tests should be used to evaluate the significance of terms on the random effects portion of two nested models, and should not be used to determine the significance of the fixed effects.

A simple way to more reliably test for the significance of fixed effects in an LME model is to use conditional F-tests, as implemented with the simple "anova" function.

Example:

```
> anova(model1)
```

will give the most reliable test of the fixed effects included in model1.

- C. Example: An experiment in which plots were fertilized with three different levels of N for 5 years ($N_1 < N_2 < N_3$), and plot above-ground biomass was recorded each year. Data are in hypothetical dataframe = "Nfert"

```
> library(nlme)
> m = lme(biomass ~ N*year, data="Nfert", random=~year|plot,
correlation=corAR1(), method="ML", na.action=na.omit,
control=lmeControl(maxIter=200, msMaxIter=200, niterEM=100))
```

The following explain what each term in the model specifies:

- "biomass ~ N*year" states the response variable as a function of the fixed effects, in this case "N" fertilization level, the time term "year", and their interaction.
 - "random=~year|plot" states that the slope of the biomass response through time, as well as the intercept are allowed to vary randomly with plot.
 - "correlation=corAR1()" specifies an autoregressive correlation structure for the error associated with biomass measurements from the plots through time. That is, measurements at a given time point from a given plot will be more similar to temporally proximate measurements than temporally distant ones. Other correlation structures exist (described in Pinheiro and Bates 2000, p.234), and it is worth testing which correlation structure best fits the data.
 - The "control" and "method" arguments are described above in A and B, respectively.
- D. It is also possible to specify a nested error structure using the "random" term, but keep in mind that there must be replication available for the nested factor. For example, an experiment

employing some treatment replicated across blocks CANNOT use “random=~1|block/treatment” UNLESS the treatment is replicated at each block.

- E. Repeated measures analyses in R takes an identical form to the example above, and demonstrates how lme may be used for the analysis of repeated measures data.

- F. Identifying outliers with a lme model object

The process is slightly different than with standard lm model objects, since the “influence” function does not work on lme model objects. Given “mod.lme”, we can use the plot function to identify outliers:

```
## Plotting a lme object produces a fitted vs. residuals plot using lattice.
```

```
> plot(mod.lme, id=0.05, idLabels=row.names(example.df))
```

```
## The “id” argument indicates that data points with absolute standardized residuals that lie outside of the 95% confidence interval of the standard normal distribution are identified as outliers
```

```
## The “idLabels” argument allows you to label those points identified as outliers. Here, I have simply used the row names of the example dataframe.
```

15. Other considerations for repeated measures analysis:

- A. The function “make.rm” : takes data where there is a column for a grouping variable and a column for sample I.D., and each time point in a repeated measures experiment is a separate column (i.e. column headers = “SampleID”, “Treatment”, “time1”, “time2”, etc.), and turns it into the form used by R for repeated measures. The new form of the data frame will have time as an ordered categorical variable.

```
make.rm<-function(constant,repeated,data,contrasts) {  
  if(!missing(constant) && is.vector(constant)) {  
    if(!missing(repeated) && is.vector(repeated)) {  
      if(!missing(data)) {  
        dd<-dim(data)  
        replen<-length(repeated)  
        if(missing(contrasts))  
          contrasts<-  
            ordered(sapply(paste("T",1:length(repeated),sep=""),rep,dd[1]))  
        else  
          contrasts<-matrix(sapply(contrasts,rep,dd[1]),ncol=dim(contrasts)[2])  
        if(length(constant) == 1) cons.col<-rep(data[,constant],replen)  
        else cons.col<-lapply(data[,constant],rep,replen)  
        new.df<-data.frame(cons.col,  
          repdat=as.vector(data.matrix(data[,repeated])),  
          contrasts)  
        return(new.df)  
      }  
    }  
  }  
  cat("Usage: make.rm(constant, repeated, data [, contrasts])\n")  
  cat("\tWhere 'constant' is a vector of indices of non-repeated data and\n")
```

```

    cat("\t'repeated' is a vector of indices of the repeated measures data.\n")
  }

```

- B. Given a data frame named “example.df” with “sampleID” and “Treatment” columns, as well as three time points, “t1”, “t2”, and “t3”, use the following syntax to generate a dataframe called “example.RM” that will be suitable for repeated measures analysis in R using a linear mixed effects model:

```

> example.RM = make.rm(constant=c("sampleID","Treatment"),
repeated=c("t1","t2","t3"), data=example.df)

```

- C. To save make.rm output in comma delimited text file:

```

> write.csv(x, file="example.csv")

```

The response variable will be re-named “repdat”, and the time points will appear in a column called “contrasts”.

- D. Using MS Excel to read “example.csv”, one can then rename the “contrasts” column to “time”, and change the ordered time points (T1, T2, T3, etc.) to numerical values, which is necessary only if data were collected with uneven spacing in time. If the repeated measures were collected at evenly spaced intervals, then leave the “time” column as an ordered factor.
- E. Read the modified “example.csv” file back into R and proceed with a lme model as described above.

16. Non-linear regression models: Non-linear regression models are useful when there is *a priori* knowledge of the theoretical non-linear relationship between two variables. For example, Michaelis-Menten enzyme kinetics, exponential decay of litter mass through time during decomposition, radioactive decay of ¹⁴C through time, or the logarithmic growth of populations reaching equilibrium in a density dependent manner are all non-linear functions. For all non-linear models, it is necessary to know the equation linking the independent variables to the dependent variable, and it is also necessary to have starting estimates for the parameters of the equation. Some non-linear equations are “self-starting” within R; that is, the R function can generate initial estimates of the parameters in order to carry out the non-linear regression analysis. For other non-linear equations, it is necessary for the data analyst to provide the initial estimates of the parameters. This can often be accomplished by graphing the data, and estimating the parameters graphically while solving the non-linear equation of interest.

- A. The following example shows how a non-linear model can be fit to data from dataframe “example.df” that are believed to be related via the simple exponential decay function: $M_t = M_0 \times e^{(-kt)} + c$, where for example, M_t is the amount of litter mass remaining at time t , M_0 is the initial litter mass, k is the decomposition constant, t is time, and c is a constant. Here, I have used the non-linear least squares (nls) function in R to analyze the data.

- B. Make a plot of the data in order to estimate initial values for M_0 , k , and c :

```

> plot(Mt~t, example.df)

```

- C. Based on the plot of the data, values for M_0 and c are graphically estimated, and k is estimated by solving:

$$k = -\ln[(M_t - c)/M_0]/t \text{ for chosen values of } M_t \text{ and } t \text{ on the plot above.}$$

For this example, assume the following:

```
M0 = 0.4
k = 20
c = 0.05
```

D. Next, construct the non-linear regression model.

```
> mod1 = nls(Mt~M0*exp(-k*t)+c, start=list(M0=0.4, k=20, c=0.05),
data=example.df)
```

E. Check the model for normality of residuals and homogeneity of variance of the residuals.

```
> qqnorm(resid(mod1))
> qqline(resid(mod1))
> plot(fitted(mod1), resid(mod1))
```

F. Examine the summary of model output.

```
> summary(mod1)
```

Formula: $Mt \sim M0 * \exp(-k * t) + c$

Parameters:

	Estimate	Std. Error	t value	Pr(> t)	
M0	0.33227	0.06980	4.760	1.49e-05	***
k	53.16972	21.42001	2.482	0.016197	*
c	0.12351	0.03152	3.918	0.000253	***

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

Residual standard error: 0.1261 on 54 degrees of freedom

Number of iterations to convergence: 4

Achieved convergence tolerance: 4.02e-07

In this example, all the parameters of the non-linear model are significant at $p < 0.05$

G. To obtain a plot of the data, as well as the non-linear curve estimated by the model, first plot the data and then add a curve informed by model output.

```
> plot(Mt~t, example.df)
> t.indep = seq(0,0.25, len=100)
```

#The above code defines the range and number of t's within that range for which predicted values of Mt are desired.

```
> Mt.predict = predict(mod1, list(t=t.indep))
> lines(t.indep, Mt.predict, col=4)
```

17. Principal Components Analysis: Principal components analysis (PCA) is useful when multiple measurements have been made on the same sample or plot, and one wishes to distinguish between those samples/plots on the basis of the measurements that were made. For example, suppose 8 soil chemical characteristics were measured on each of 50 plots. PCA would allow grouping of those plots in reference to

all 8 of their soil chemical characteristics. Another example: suppose activities of 10 soil enzymes were measured in each of 30 soil samples; PCA could determine which soils responded similarly (or dissimilarly) in terms of all 10 enzymes. Finally, consider 100 plots for which species relative abundance was determined on each plot; PCA could group similar plots together in terms of their species composition.

- A. The following example shows how a PCA is performed on 6 different C fractions generated from *A. rossii* fresh leaves and leaf litter. There are 3 replicates per fraction, and on each of the 18 samples, the following measurements were made: %N, % sugars, %anthocyanins, %coumarins, %phenolic acids, %flavonoid glycosides, %ellagitannins, %gallotannins.

```
> frxn = read.csv("Aco_Frxn_PCA.csv", header=T, row.names="sampleID")
> str(frxn)

'data.frame': 18 obs. of  8 variables:
 $ N.perc      : num  0.089 0.073 0.074 0.265 0.263 0.239 0.092 0.082 0.084
0.112 ...
 $ sugars      : num   0.0  0.0  0.0 41.3 43.1 ...
 $ anthocyanin: num   0  0  0  0  0  0  0  0  0 ...
 $ coumarin    : num   0  0  0  2.29 2.3 2.32 0  0  0 ...
 $ phen.acid   : num   1.12 1.27 1.42 0.58 0.57 ...
 $ flav.gluc   : num   2.73 3.08 3.04 5.01 5.07 ...
 $ e.tannin    : num  101.45 103.46 103.74  1.66  1.48 ...
 $ g.tannin    : num   6.3 6.82 6.62 0.52 0.5 0.52 0  0  0 1.35 ...

> frxn.pca = prcomp(frxn, scale=T)
```

Problems may be encountered during a principal component analysis if the variables differ significantly in the relative magnitude of their respective values. The reason for this is that linear regression favors those variables that show the greatest variance, which generally will be those with the larger absolute values.

The "scale" argument sets the variance for each variable to 1 to avoid this problem. The data are automatically "centered" by prcomp, meaning the mean is subtracted from each data point. Whether scaling is used depends strongly on the nature of the question for which PCA is employed.

If the data contain "NAs", the variables used for the principle components analysis must be specified as a formula, so that the "na.action" argument can be used. For example:

```
> frxn.pca = prcomp(~., data=frxn, center=T, scale=T, na.action=na.omit)
```

The "~." notation means all variables in the dataframe are included in the formula.

```
> summary(frxn.pca)
```

Importance of components:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8
Standard deviation	2.042	1.413	0.983	0.912	0.16191	0.06471	0.05914	0.03320
Proportion of Variance	0.521	0.250	0.121	0.104	0.00328	0.00052	0.00044	0.00014
Cumulative Proportion	0.521	0.771	0.892	0.996	0.99890	0.99942	0.99986	1.00000

The above output summarizes the amount of variance explained by each of the PC axes, as well as the cumulative variance explained.

Note: the function “princomp” is favored by some over the function “prcomp”. “princomp” allows one to use either a correlation structure or a covariance structure, and “prcomp” automatically uses the correlation structure. When a correlation matrix is used, output from the two functions appears to be identical, though the variances are handled in different ways (see Brian Ripley comments in R help pages for more details).

- B. Plotting the results and the sample “scores” from a principal components analysis:

```
> plot(frxn.pca)    ## Plots the variances associated with each PCA axis.  
  
> plot(frxn.pca$x)  
> abline(h=0,col=8)  
> abline(v=0,col=8)
```

For each sample, plots the scores associated with PC1 and PC2, by default. The “abline” commands put horizontal and vertical gray lines through the origin of the plot. For the color palette argument: 1=black; 2=red; 3=green; 4=blue; 5=light blue; 6=purple; 7=yellow; 8=gray.

Samples are represented by simple circles in the above scores plot. Identify samples by creating a label for them, and using the “identify” function:

```
> sampleid = c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18)  
> identify(frxn.pca$x, labels=sampleid)
```

Clicking on the graph over each point, then either right clicking or returning to the console and hitting enter identifies each point according to the “sampleid” vector just defined.

- C. Plotting the “loadings”, which indicate how variation in the chemistry measurements is aligned with variation in the principal components axes:

```
> plot(frxn1.pca$rotation)  
> chem = c("N","sug","an","cou","pa","fg","et","gt")
```

These labels are ordered to correspond with the column order in the “frxn” data frame [see str(frxn) output above]

```
> identify(frxn.pca$rotation, labels=chem)
```

- D. Creating a “biplot” – overlays loading plot and scores plot in one figure.

```
> biplot(frxn.pca)  
> abline(h=0, col=8)  
> abline(v=0, col=8)
```

One drawback to the “biplot” function is that it is apparently impossible to control the plot character that is used for the individual samples using the “pch” argument. Samples are plotted as numbers or row names corresponding to their row position, which can look ungainly. I have used Adobe Illustrator to work around this drawback.

- E. To create plots and biplots using PC axes other than the default choices (e.g. – a graph of PC1 and PC3):

```
> plot(c.pca$x[,1],c.pca$x[,3], xlab="PC1",ylab="PC3")
```

```
## The numbers in brackets indicate which PC axes will be plotted
```

```
> biplot(c.pca,choices=c(1,3))
```

```
## The "choices=c(1,3)" argument indicates PC1 and PC3 will be plotted.
```

18. **Factor analysis:** Factor analysis is another ordination technique for multi-variate data, similar to PCA, but more flexible in that factors are not necessarily orthogonal to each other. Ordination axes derived from factor analysis may therefore align better with measured variables than PCA axes. That is, the factor analysis axes may be easier to interpret in terms of the traits that were measured.

- A. The following example shows how a factor analysis is performed. For this example, I took four plant species, and measured 9 chemical traits on each species (n=5 per species). Using the scores from a factor analysis, I was able to describe the chemistry of each species in terms of three significant factor axes.

```
> chem = read.csv("Chemsimp_for_FA.csv", header=T, row.names="sampleID")
```

```
> library(psy)
> scree.plot(chem, type="R")
```

```
## The scree plot helps visually determine how many factors will be required to account for the
variability in the original 9 chemical traits measured (3 factors in this case).
```

```
> chem.fact = factanal(chem, 3, rotation="varimax", scores="regression")
```

```
## Within the parentheses: chem = dataframe on which to perform analysis; 3 = number of
factors to fit; rotation="varimax", indicates factors will be orthogonal to each other;
scores="regression", indicates factor scores will be obtained for each row in the original
dataframe, in this case each replicate of the species sampled.
```

```
## rotation="promax" allows the factor axes to be non-orthogonal to each other. Note that
factor axes will still appear orthogonal when plots are made. Factor axes may associate more
strongly with measured traits when rotation="promax" is used. But they may not.
```

- B. To obtain factor scores for each replicate of each species:

```
> chem.score = chem.fact$scores
```

- C. To obtain loadings for the 9 traits that were measured:

```
> chem.load = chem.fact$loadings
```

- D. Constructing a bi-plot, containing scores (for replicates of each species), and loadings (of chemical traits onto factors):

Plots first two factors only:

```
> par(mfrow=c(1,2))
> plot(chem.score, cex=0.8)
> abline(h=0,col=8)
> abline(v=0,col=8)
> text(chem.load, labels=names(chem), cex=0.8)
```

Plots first and third factors:

```
> plot(chem.score[,1],chem.score[,3],xlab="Factor1",ylab="Factor3",cex=0.8)
> abline(h=0,col=8)
> abline(v=0,col=8)
> text(chem.load[,1],chem.load[,3],cex=0.8,labels=names(chem))
```

19. **Curve fitting and plotting polynomial functions.** This procedure is an expansion of a simple linear regression that allows fitting of polynomials up to degree “n”, where n is the number of datapoints in the dataset.

A. Assume an example dataframe named “example.df” with Y and X as dependent and independent variables, respectively. Want to test significance of the model $Y = aX^2 + bX + C + \epsilon$

```
> library(MASS)
> poly.mod = lm(Y ~ poly(X, 2), data=example.df)
> grid = seq(Xmin, Xmax, len=50)
```

for above line, Xmin is the minimum X value and Xmax is the maximum X value

```
> plot(Y ~ X, example.df)
> lines(grid, predict(poly.mod, data.frame(X=grid)), col=4)
```

col = 4 produces a blue line, but other colors can be chosen as desired.

20. **Controlling plot symbols with categorical or continuous variables.** It is often nice when making a plot of two continuous variables to distinguish between individual plotted points on the basis of some third categorical or continuous variable.

A. Coloring XY plot symbols according to a third categorical variable

a. For example, suppose we have a dataframe called example.df and we plot Y as a function of X, and wish to distinguish the points by a third categorical variable Z. To accomplish this, first create an additional column in your dataframe to assign a specific color to all the levels of Z – e.g. a column called “z.color”. It is simplest to use standard numerical color codes that R recognizes (i.e. 2 for red, 3 for green, 4 for blue, etc.), but it is also possible to specify colors using hexadecimal notation, e.g. #FF6600 for orange.

b. Next, to plot Y versus X, color coded by Z, use the following code:

```
> plot(Y~X, example.df, col=example.df$z.color)
```

B. Scaling the size of XY plot symbols according to a third continuous variable

a. Suppose a dataframe called “example.df” with X and Y continuous variables that we wish to plot, and a third continuous variable “z.size” that we wish to use to control the size of the symbols. Note that it is helpful if the units of “z.size” are the same as for X and Y.

b. Plot Y as a function of X, with the symbol size specified by “z.size”:

```
> symbols(X, Y, circles=z.size, inches=FALSE)
```

- c. Note: The “symbols” function employs many other modifier arguments not specified here that can be used to control plot appearance.
- d. Note: The “circles” argument specifies the size of the radius of the circle symbols that are plotted. As such “z.size” should specify the size of the circle radius.
- e. Note: The “inches=FALSE” statement ensures that the units of “z.size” are not scaled to a 1-inch maximum size.

References used:

Crawley, Michael J. (2005) Statistics: An Introduction using R

Daalgard, Peter (2002) Introductory Statistics with R

Faraway, Julian (2004) Linear Models with R

Faraway, Julian (2006) Extending the Linear Model with R

Pinheiro, José C. and Bates, Douglas M. (2000) Mixed Effects Models in S and S-PLUS

http://www.graphpad.com/articles/interpret/corl_n_linear_reg/correlation.htm