# Analysis of Environmental Data

*Data Exploration, Screening and Adjustments*

(Written by Kevin McGarigal)

The purpose of this lab exercise is to familiarize you with options for exploring, screening and making adjustments to your data – in other words, methods for exploratory data analysis and graphics in R. These procedures will help you get to know your data set and, in particular, screen for irregularities (e.g., outliers) and transform and/or standardize data if appropriate. Data screening is an essential precursor to all statistical analyses and data transformations and standardizations are commonly applied prior to many techniques. Note, because all data sets are unique the procedures described below may not always be appropriate in all cases. You must judge the appropriateness of each procedure for your particular data set and study context. In addition, there are myriad possibilities for exploring your data, including almost limitless possibilities for statistical and graphical summaries, including extremely advanced methods for graphical display of data, that go well beyond the scope of this lab. The intent of this lab is to introduce you the simplest and most common methods for data exploration, screening and adjustment in R. Here is outline of what is included in this lab exercise:

## 1. Set up your R work session

Open R and set the current working directory to your local workspace, for example:

```
setwd('c:/work/stats/ecodata/lab/exploratory/')
```

Load the biostats library (which is not a formal library) by typing, substituting the appropriate path:

```
source('.../biostats.R')
```

The source() function is like the library() function in that it loads a set of functions into memory. The difference is that source() is used to load a personal script containing one or more functions and library() is used to load a formal R library or package that is available through the R-project.

Import the data sets by typing:

```
birds<-read.csv('bird.sub.csv',header=TRUE)
hab<-read.csv('hab.sub.missing.csv',header=TRUE)
```

In this case, the data represent standardized breeding bird counts and a variety of habitat variables for 30 landscapes in the Oregon Coast Range. See the corresponding metadata file (birdhab.meta.pdf) for a description of the variables in these data sets. Now that we've got the data into a data frame, check the structure of the data set by typing:

```
str(birds)
str(hab)
```

## 2. Summary statistics

Calculate summary statistics for logical groups of numeric variables and examine them for suspicious values that may signal a possible data entry error. In particular, note the minimum and maximum values for obvious errors. In addition, examine the summary statistics to gain a general understanding of the range of values observed and the frequency of zeros and missing values. There are two basic options here, summary by row (observation or record) and by column (variable).

Mixed data set option:

If the data set consists of unrelated variables or variables on different scales of measurement (e.g., environmental variables), as is the case for the hab dataset, then only a column summary is meaningful and the following biostats library function is available by typing:

```
col.summary(hab[,-c(3:7)])
```

Note the use of indexing to select columns 3 through 7. Alternatively, you can use the summary() function in the base library for a reduced output. For categorical variables (factors), you can use the table() function to generate a count of observations in each category, as follows:

```
table(hab[,1:2])
```

Homogeneous data set option:

If, on the other hand, the data set consists of a homogeneous set of variables measured on the same scale (e.g., species abundances), as is the case with the birds dataset, then both a row summary and column summary is meaningful.

To produce a column summary, type:

```
sum.stats(birds,var='AMCR:YRWA',margin='column')
```

To produce a row summary, type:

```
sum.stats(birds,var='AMCR:YRWA',margin='row')
```

See the help file for sum.stats for a description of the summary statistics produced.

## 3. Missing data

Review the column and row summary statistics above and note the amount and pattern of missing data. If some cases it may be suitable to simply drop all records with any missing values. For example, with a large data set, dropping a few observations may not have any effect on the results, so it might be easiest to simply drop any observation with missing data. Recall from the first lab that this can be done easily using the na.omit() function, as follows:

```
temp<-na.omit(hab)
```

However, in many cases the observations are simply too valuable to throw out because of some missing data. Some functions are smart enough to be able to handle missing data. For example, functions such as max(), mean(), var(), sd(), sum() and others have an optional na.rm argument: na.rm=TRUE drops NA values before doing the calculation. Otherwise if x contains any NAs, mean(x) and other similar functions will return an NA. For example, to calculate the mean Simpson's diversity index of stand types in the hab data set, type:

```
mean(hab$s.sidi)
```

Note, that mean() returns a value of NA because there are missing values for some of the observations. If we specify the na.rm=TRUE argument, then mean will drop the missing values before calculating the mean, as follows:

```
mean(hab$s.sidi,na.rm=TRUE)
```

Other functions have similar built-in arguments for handling missing data.

In some cases it may be justifiable to replace the missing values with another value – although there

should be a good reason for doing so!. The general syntax for doing so is x[is.na(x)]=value. For example, to replace the missing values in the hab data set with 0, type:

```
temp<-hab
temp[is.na(temp)]<-0
temp$s.sidi
```

Note that we first copied the hab data set to a new object, temp, so as to preserve our original data, then we replaced the missing values with zero, and then we printed to the console the variable containing the missing data after replacement. In addition, note that this operation works equally well on vectors, matrices and dataframes.

In some cases it may be preferable to replace the missing values with either the mean or median of the variable. For example, to replace the missing value for s.sidi in the hab data set with the mean of s.sidi across all other observations, type:

```
temp<-hab
temp$s.sidi[is.na(temp$s.sidi)]<-mean(temp$s.sidi,na.rm=TRUE)
temp$s.sidi
```

Again, to preserve the original data set we first copied it to a new object. Note that the replacement with the mean always coerces the variable to a real number (i.e. class=numeric). For count variables (e.g., bird abundance data; class=integer), the median is perhaps the more logical replacement, as it maintains the integer status of all values. To accomplish a 'median' replacement, type:

```
temp<-hab
temp$s.sidi[is.na(temp$s.sidi)]<-median(temp$s.sidi,na.rm=TRUE)
temp$s.sidi
```

Note that the operation above replaces missing values for a vector, or a single column of a data frame or matrix. To repeat the same operation for multiple columns of a data frame or matrix, the above operation would have to be nested in a loop that loops through the columns one at a time. Instead of writing this function from scratch, we can use the replace.missing() function in the biostats library. To replace all missing values in the hab data set with the corresponding column medians, type:

```
temp<-replace.missing(hab,var='sub.lat:w')
temp$s.sidi
```

where 'sub.lat' and 'w' are the first and last numeric variables in the range of selected variables. Note, in this function, the 'median' replacement is the default method. To replace missing values with the mean, instead of the default median, specify the method='mean' argument, as follows:

```
temp<-replace.missing(hab,var='sub.lat:w',method='mean')
temp$s.sidi
```

If missing data values exist and are concentrated in one or more samples and/or one or more variables, you might consider dropping the offending samples and/or variables. For example, you might consider dropping variables with too many missing values, say >5 percent of observations missing, using the drop.var() function is the biostats library, as follows:

```
temp<-drop.var(hab,var='sub.lat:w',pct.missing=5)
str(temp)
```

Lastly, be aware that data imputation methods exist for handling missing data that go way beyond the simple methods presented here. If missing data is a serious issue in your data set, then it will behoove you to investigate imputation methods further.

## 4. Frequency of occurrence and abundance plots

If your data set is a community ecology set containing samples (rows) by species (columns) abundances, there are several things about the species and samples you may wish to know:

- in how many plots does each species occur?
- what is the mean abundance of each species when it occurs (not averaging zeros for plots where it is absent)?
- is the mean abundance of species correlated with the number of plots in which it occurs?
- how many species occur in each plot?
- is the total abundance of species in a plot correlated with the number of species in a plot?

Foa.plots() in the biostats library produces a series of 10 different plots that can help you answer these questions and more. To view the plots for the bird species abundance data set, type:

```
foa.plots(birds,var='AMCR:YRWA')
```

*In how many plots does each species occur?*

The first four plots portray the species frequency of occurrence among plots, but in slightly different ways – either as an empirical distribution function (EDF) of species occurrence or as a histogram of species occurrence. Because some ecologists have suggested that species occurrence and abundance distributions sometimes follow a log-normal distribution, the fourth plot depicts of histogram of log-transformed species occurrence.

*What is the mean abundance of each species when it occurs (not averaging zeros for plots where it is absent)?*

The fifth plot is an EDF of species mean abundance. How does it compare to the EDF of species occurrence? Perhaps more interesting than the simple distribution of mean abundance is the following.

*Is the mean abundance of species correlated with the number of plots they occur in?*

The sixth plot is a scatter plot of frequency of occurrence against mean abundance. Is there any

apparent relationship between the two? Are the widespread species also generally more abundant? Are there many widespread species that occur at low abundance? Conversely, are there species much less widespread, but abundant where they occur? To see which dot is which species, answer 'y' to the question on the console and then simply click on the point and the species acronym (variable name) will appear next to the point. When your done identifying points, simply right click and chose 'stop'.

As before, it may be more meaningful to look at the log of mean abundance, as often abundance follows a log-normal distribution. The seventh plot is the same as above except that mean abundance has been log-transformed. Are the patterns the same?

*Is the total abundance of species in a plot correlated with the number of species in a plot?*

To answer this question, first it is instructive to look at the number of species per plot. The eighth plot depicts the EDF of plot richness. Are there any interesting patterns? For example, do most plots support an average number of species, while only a few plots support either very few or very many species? Or is the pattern different?

Second, what is the pattern in the distribution of plot total abundance? The ninth plot is the EDF of total plot abundance? How does it compare to the EDF of plot richness?

Finally, to answer the question on the relation between total abundance and number of species/plot, the last plot is a scatter plot of the two variables. Is there is relationship between the number of species per plot and the total abundance? Do species-rich plots generally support a greater total abundance of species as well?

## 5. Dropping variables

It is important to screen your data for insufficient variables (i.e., those that were sampled insufficiently to reliably characterize their environmental pattern). For example, in community data sets *rare species* with very few records are not likely to be accurately placed in ecological space. You must decide at what level of frequency of occurrence you want to accept the 'message' and eliminate species below this level. Conversely, in community data sets *abundant generalists* species can overwhelm the message of rarer species in some types of analysis. You must decide whether to include or exclude these "dominant" species. Lastly, variables with too little variation have no meaningful pattern (or influence on the analysis) and are therefore unnecessary and simply add meaningless dimensions to the data set.

Review the column summary statistics derived earlier and note the number and percentage of zeros and the coefficient of variation of each variable.

We might want to consider dropping variables with fewer than say 3 non-zero values by typing:

```
temp<-drop.var(birds,var='AMCR:YRWA',min.fo=3)
```

Note that we save the results to a new object named 'temp' rather than replacing the original object.

Check to see how many and which species were dropped by typing:

```
names(birds)[!names(birds) %in% names(temp)]
```

In community data sets, we might also want to consider dropping abundant generalist species, for example those species occurring on more than 95% of the plots, by typing:

```
temp<-drop.var(birds,var='AMCR:YRWA',max.po=95)
```

How many species were dropped? Depending on the subsequent analysis, dropping abundant generalist species may or may not be desirable, so this decision should be made carefully.

Consider dropping variables with too little variation, for example those with cv<5, by typing:

```
temp<-drop.var(birds,var='AMCR:YRWA',min.cv=5)
```

How many species, if any, were dropped? Note, dropping variables with too little variation is unlikely to effect subsequent analyses, because they will not exert much influence over the data cloud, but in the spirit of parsimony it may be convenient to simply drop them here.

## 6. Single variable distributions

Examine the distribution of each variable independently and, among other things, note the need for transformations to improve characteristics of the distribution (e.g., normality). Of course, there are many ways to examine distributions and we will not attempt an exhaustive review. Instead, we will focus on four common graphical approaches suitable for continuous variables. Note, categorical variables (or factors) are not suitable for these functions and will produce an error if they are included in the list of variables to plot, so you may need to subset your data first and select just the continuous variables.

### *6.1 Empirical (cumulative) distribution functions*

First, the empirical distribution function (EDF) is a simple rank order distribution of increasing values of the variable. A variable with a perfectly uniform distribution of values within its range (minimum to maximum), so that no one value is more common than another, will have points that fall on a perfect diagonal straight line. Deviations from the diagonal indicate non-uniformity. Plot the EDF for each variable by typing:

```
edf.plots(birds,var='AMCR:YRWA')
```

Many people prefer to view the empirical distribution as a cumulative function. The empirical cumulative distribution function (ECDF) plots the rank order distribution of increasing values of the variable on the x axis, as in the EDF, but the y axis is given as the probability (0-1) of values less than or equal to the value of the variable. This has a nice intuitive interpretation, because you can easily determine what percentage of the values are less than or equal to any specified value. The ECDF is plotted just like the EDF, but using the ecdf() function instead of the edf() function, as

follows:

```
ecdf.plots(birds,var='AMCR:YRWA')
```

If the samples are grouped, consider examining the distributions separately for each group. In the case of the birds data set, the data are naturally grouped by basin. You can examine the ECDF's for each variable within each basin, by typing:

```
ecdf.plots(birds,var='AMCR:YRWA',by='basin')
```

### *6.2 Histograms*

Next, try the more conventional histogram, by typing:

```
hist.plots(birds,var='AMCR:YRWA')
```

Environmental data are often highly skewed, so evaluate each distribution for skewness. You should also pay attention to the occurrence of extreme values in the distribution (i.e., potential outliers), as we will address this issue below. In addition, for species abundance variables like the ones here, pay attention to the level of quantitative information present (i.e., whether there is a range of abundances or whether the principal signature is one of presence versus absence) since this may determine the need for a binary transformation.

If the samples are grouped, consider examining the distributions separately for each group. In the case of the birds data set, you can examine histograms for each variable by basin, by typing:

```
hist.plots(birds,var='AMCR:YRWA',by='basin')
```

### *6.3 Box-and-whisker plots*

An alternative way to examine the distribution of each variable is with a box-and-whisker plot, obtained by typing:

```
box.plots(birds,var='AMCR:YRWA')
```

Box plots can depict the skewness of the distribution quite nicely and also can be used to identify extreme observations. The central box shows the data between the 'hinges' (roughly quartiles), with the median represented by a line. 'Whiskers' go out to the extremes of the data, and very extreme points (defined as samples that are by default farther than 1.5 times the inter-quartile range from the box) are shown by themselves.

Again, if the samples are grouped, the box plots can be produced for each group separately and displayed side-by-side on the same plot. For the birds data set, try typing:

```
box.plots(birds,var='AMCR:YRWA',by='basin')
```

*6.4 Normal quantile-quantile plots*

Another useful way of examining the distribution of each variable is to compare the empirical distribution function (EDF) to the expected EDF for a normal distribution. A normal quantile-quantile (or QQ) plot does just this. Try typing:

```
qqnorm.plots(birds,var='AMCR:YRWA')
```

Note, the qqnorm plot depicts the sample quantiles on the x axis against the theoretical quantiles from a normal distribution of the same sample size on the y axis. If the data are from a perfectly normal distribution, the data will lie on a diagonal straight line. Departures from the diagonal indicate deviations from a normal distribution. Skewed distributions show up nicely as deviations from the line at the tails.

As with the other plots, you can examine normal QQ plots for each variable by basin and subbasin in the birds data set, by typing:

```
qqnorm.plots(birds,var='AMCR:YRWA',by='basin')
```

*6.5 Four-in-one plots*

The four plots described above can be depicted together in a single 4-part plot for each variable using the uv.plots() function in the biostats library. However, this function does not allow for groups. Thus, if you seek plots by group then use the separate functions above. The four-in-one plot is generated by typing:

```
uv.plots(birds,var='AMCR:YRWA')
```

## 7. Relationships between pairs of variables

Before considering a formal statistical analysis involving multiple variables, it is always useful to examine the nature of the relationships between pairs of variables, including both dependent and interdependent relationships. At the risk of being accused of data-dredging, inspecting these relationships can be very useful in determining the form of the statistical relationship between the independent and dependent variables and evaluating the underlying assumptions (e.g., linearity, multicolinearity) of the model to be employed.

*7.1 Correlations*

It can be useful to examine the linear correlation between pairs of variables. A simple way to do this is the cor() function, as follows:

```
cor(birds[,-c(1:3)])
```

Note, the brackets are used here to exclude the first three columns (which contain plot ID information) from the resulting correlation matrix. By default the cor() function produces Pearson

correlation coefficients, but the method= argument can be used to ask for kendall or spearman rank correlation coefficients, as follows:

```
cor(birds[,-c(1:3)],method='spearman')
```

*7.2 Scatterplots*

In cases involving dependent and independent variables, it may be most useful to examine scatterplots between pairs of dependent and independent variables, which can provide an indication of the strength and nature of the dependent relationship and thereby guide the selection of the appropriate statistical analysis to follow.

The plot() function can be used to produce a scatterplot of two numeric variables, for example:

```
plot(hab$ls,birds$BRCR)
```

We can add a robust locally weighted regression (lowess) line to the plot using the lowess() function, as follows:

```
lines(lowess(hab$ls,birds$BRCR))
```

Note, since lines() is a low-level plotting function it will automatically plot on top of the previous scatterplot created using the high-level plot() function.

Scatterplots between multiple x's and y's can be quickly generated using the scatter.plots() function in the biostats library, as follows:

```
birdhab<-merge(hab,birds,by=c('basin','sub'))
scatter.plots(birdhab,y='AMCR:YRWA',x='ls')
```

Note, here we merged the birds and hab data first. Then we used the scatter.plot() function to create scatterplots between each combination of bird and hab variables. If more than one y-variable and/or x-variable is specified, a separate scatterplot will be produced for each unique combination of x and y. Note, by default a Lowess regression line is plotted as well, but this can be turned off by setting the lowess= argument to FALSE.

*7.3 Scatterplot matrix*

It can also be quite useful to produce bivariate scatterplots for all pairs of variables (i.e., no distinction between independent and dependent variables). A simple way to do this is with a scatterplot matrix using the pairs() function, as follows:

```
pairs(hab[,9:16])
```

The scatterplot matrix can be difficult to read if there are too many variables, so here we restricted the variables to columns 9 through 16. For a more elegant scatterplot matrix that combines the

pairwise scatterplots with Lowess regression lines with the pairwise correlation coefficient, try the following:

```
pairs(hab[,9:16],lower.panel=panel.smooth,upper.panel=panel.cor, method='spearman')
```

In the resulting scatterplot matrix, a scatterplot for each unique pair of x and y variables is ploted in the lower triangle of the matrix, and the corresponding correlation coefficient is given in the upper triangle of the matrix - and the size of the coefficient is used to scale the size the characters so that large correlations standout.

### *7.4 Coplots*

In cases involving the relationship between a dependent and independent variable, the relationship may be obscured by the effects of other variables. In such cases it may be useful to examine a scatterplot of x and y, but conditioned on a third variable, say z. The coplot() function does just this, as follows:

```
coplot(BRCR~ls|sub.elev,data=birdhab,panel=panel.smooth)
```

The coplot() function specifies the x and y, implemented here as y~x (in formula notation), with the conditioning variable after the conditioning operator | (here read is 'given elev'). The panels in the resulting plot are ordered from lower left to upper right, associated with the values of the conditioning variable in the upper panel from left to right. The coplot in this case highlights a consistent pattern of increasing brown creeper abundance as the percentage of the landscape comprised of large sawtimber forest increases across a gradient in subbasin elevation.

### *7.5 Redundancy plots*

In some applications it may be useful to know how much redundancy exists in the data and whether it is more than you might expect by chance alone given the dimensionality and empirical characteristics of the data set. Redundancy is a measure of coordination or covariation among variables (e.g., species). Importantly, all multivariate techniques and especially ordination procedures depend on redundancy among variables. For example, in the context of ordination, if all variables were independently distributed, then a canonical projection to fewer dimensions would not be effective. In this case we may wish to know whether there is sufficient redundancy among the variables (i.e., correlations) to expect dimensionality reduction to be effective? Conversely, many procedures expect variables to be independent, so that too much redundancy can be troublesome. In either case, it can be very useful to assess the degree of redundancy among your variables.

One way to do this is to plot the rank order distribution of pairwise correlations against the null distribution obtained by randomly permuting the data, which can be done with the redun.plot() function by typing:

```
redun.plot(birds[,-c(1:3)])
```

If the data set contains lots of redundancy, that is, more than we would expect by chance alone, then

the actual pairwise correlations (both positive and negative ones) should be mostly larger than those from the random data. In the redundancy plot, the degree to which the 'actual' line falls outside the 'random' envelope is a measure of how much 'real' redundancy exists.

The previous redundancy plot provided a 'global' assessment of redundancy for the entire data set. We can also assess each variable's redundancy separately and compare it to what we might expect for a randomized data set, by typing:

```
redun.plot(birds[,-c(1:3)],var='AMCR:YRWA')
```

## 8. Outliers

Check your data set for potential outliers and eliminate them if, and only if, it is justified. As a general rule, observations should not be dropped automatically just because they have extreme values. It is one thing to identify extreme observations that have high leverage on the results (i.e., have a strong affect on the parameter estimates), but it is another thing altogether to delete these observations from the data set just because they have high leverage. What constitutes a true "outlier" depends on the question being asked and the analysis being conducted. Moreover, just because a value may be extreme (i.e., far from the mean) it does not mean that it will have high leverage on the parameter estimates. Make sure you understand why this statement is true.

There is no general rule for deciding whether extreme observations should be considered "outliers" and deleted from the data set before proceeding with the analysis. Nevertheless, it is important to have an understanding of the number and pattern of extreme observations in order to gauge the robustness of the results. A good practice is to repeat the analyses with and without the suspect points and determine if the results are sensitive or robust to their inclusion. If the results are sensitive to the inclusion of these extreme (high-leverage) points, you should probably carefully consider whether those points represent a meaningful environmental condition, and act on them accordingly.

There are several ways to identify extreme values, including both univariate and multivariate methods. It is always a good idea to begin with a univariate inspection. The uv.outliers() function in the biostats library computes the standardized values for each variable (i.e., z-scores) and returns a data frame with a list of samples and the variables that are greater than a specified number of standard deviations from the mean (default = 3). Try typing:

```
uv.outliers(hab,id=c('basin','sub'),var='sub.lat:w')
```

Note the number and distribution of extreme values. Are there observations with extreme values on several variables, or are the extreme values dispersed among samples and variables?

In the context of a multivariate data set, just because an observation is extreme on a single variable, doesn't mean it is going to be a multivariate outlier. More importantly, an observation may not be a univariate outlier and yet still be an outlier when two or more variables are considered jointly. Thus, it is perhaps more instructive to determine if each observation is extreme in multivariate space. To do this, use the multivar.outliers() function in the biostats library. Specifically, select an appropriate

distance measure – ideally the distance measure to be used in subsequent analyses – and use this function to compute the average distance of each sample to all other samples. For example, for the species variables in the bird data set, try typing:

```
mv.outliers(birds,var='AMCR:YRWA',method='bray')
```

Note, this function utilizes functions from the vegan library, so you must have previously installed the vegan library to use this function. If you have not already installed the vegan library, do so now. This will produce a list of samples with an average Bray-Curtis distance >3 (default) standard deviations from the mean of average distances, in addition to paired histograms depicting the distribution of average distances and the distribution of standard deviates.

Another way of assessing multivariate outliers is to compute the Mahalanobis distance between each sample and the group of all other samples and to compare this against the expected distribution of Mahalanobis distances for a multivariate normal distribution, using a very conservative probability, e.g., $p<0.001$ based on a chi-square distribution with degrees of freedom equal to the number of variables. This can be done with the method='mahalanobis' argument in mv.outliers(), as follows:

```
mv.outliers(birds,var='AMCR:YRWA',method='mahalanobis')
```

Unfortunately, in this case the function fails because there are too few observations. It can also fail for a variety of other reasons, the most common being too many zeros in the data.

## 9. Data transformations

Once you have thoroughly examined your data, you may deem it necessary or useful to transform your data. A data "transformation" involves applying a mathematical function separately to each data value (i.e., a single cell in the data frame) and is distinct from a "standardization" as discussed below. Often times environmental data are highly skewed and/or range over several orders of magnitude, and as such can benefit from a transformation, such as the log or square-root transformation, that compress large values. For community data sets involving species abundances, it is sometimes useful or more meaningful to transform the data to binary (presence/absence) data. Use the data.trans() function in the biostats library to transform variables using the log, power or arcsine square-root transformations.

### *9.1 Log transformation*

To log-transform the species abundances in the bird data set, type:

```
data.trans(birds,var='AMCR:YRWA',method='log')
```

Examine the paired histograms comparing the raw (untransformed) and transformed distributions to see the effect of the transformation.

### *9.2 Power transformation*

To power-transform the species abundances in the bird data set, try a square-root transformation, by typing:

```
data.trans(birds,var='AMCR:YRWA',method='power',exp=.5)
```

Note, the square-root transformation is simply a special case of the power transformation when the exponent is equal to 0.5.

To transform the species abundances into presence/absence (i.e., binary transformation), use the power method with an exponent equal to zero, by typing:

```
data.trans(birds,var='AMCR:YRWA',method='power',exp=0)
```

### *9.3 Logit and Arcsine square-root transformation*

For proportional data (i.e., ranges 0-1), the logit and arcsine square-root transformations are often recommended by statisticians. To accomplish these for the stand-level Simpson's diversity index, type:

```
data.trans(hab,var='s.sidi',method='logit')
data.trans(hab,var='s.sidi',method='asin')
```

Note, for any of these transformations, it is common to apply the same transformation to an entire set of related variables so that they are all on the same scale. For example, if some of the species variables benefit from a log transformation, it would generally be preferably to log-transform all species variables, not just the ones in need. In any case, once you have decided to use a particular transformation, you will need to save the transformed data into a new object by assigning the results, or you can use the 'outfile' argument in the data.trans() function to automatically save the transformed data set to a new permanent data set.

## 10. Data standardizations

In many environmental data sets, especially community data sets involving species abundances, it is often quite useful to standardize (or relativize) the data before conducting subsequent analyses. A data "standardization" (sometimes referred to as "relativization") involves adjusting a data value relative to a specified standard derived from the corresponding row (sample) or column (variable) of the data frame. Keep in mind that standardizations can fundamentally alter the patterns in the data and can make the difference "between illusion and insight, fog and clarity" (McCune and Grace, 2002).

If you are working with community data involving species abundances, such as in the plant species cover data set and the bird abundance data set, check the coefficient of variation (cv) in row and column totals for the species variables (see sum.stats() described previously). Specifically, examine the cv's for the column/row *sums* reported in the table.summary. For the column summary, this is the cv in species' total abundances (i.e., the cv in column totals). The corresponding value in the row summary is the cv in plot totals (i.e. ,the cv in row totals). If these values are small, say <50, it is

unlikely that standardization will accomplish much. However, if these values are large, say >100, then it is likely that standardization will have a large effect on the results.

If you are working with environmental data involving variables measured in different units or on different scales, consider column standardization (by norm or standard deviates) if the environmental variables are to be used in a distance-based analysis that does not automatically standardize the variables (e.g., multi-response permutation procedures). Note, column standardization is not necessary for analyses that use the variables one at a time (e.g., ordination overlays) or for analyses with built-in standardization (e.g., principal components analysis of a correlation matrix).

There are many possible row and column standardizations and they will not be reviewed here. Refer to the biostats help file for data.stand() for details on the available standardizations. Before applying any standardization, be sure to understand what the standardization does and whether the units for all the included variables are the same. Note, in same cases, standardization is built into the subsequent analyses and therefore unnecessary – but to know this requires that you already understand the mechanics of the techniques you intend to use (which we haven't gotten to yet). At this point, you might simply explore what various standardizations do to the data so that you are ready and able to standardize your data as needed when you decide on a particular statistical procedure.

As an example, to conduct a 'row normalization' (i.e., to rescale each row so that the sums of squares are equal), type the following:

```
data.stand(birds,var='AMCR:YRWA',method='normalize',margin='row')
```

Examine the paired histograms that are produced by default to see the effect of the standardization on the distribution of each variable. In addition, after each standardization you may want to recalculate and examine the row and column summary statistics as before. Remember, if you intend to save the standardized results for use in a later analysis, you must save the results to a permanent data set. You can either assign the results to an object and then write that object to a file using the write.table() or write.csv() functions, or you can simply save the results directly with the data.stand() function by using the outfile='' argument.

Regardless of your decision to standardize or not, you should state your decision and justify in briefly on environmental grounds.

## 11. Dissimilarity matrices

In some applications it may be meaningful to convert the data set into an appropriate dissimilarity or distance matrix. A distance matrix is a square matrix (same number of rows and columns) where the elements represent the dissimilarity between the corresponding observations. A common dissimilarity matrix is a Euclidean distance matrix in which the elements represent the straight-line distance between every pair of samples based on their x-y coordinates. Given a data frame containing 10 observations (rows) and two columns representing x and y coordinates, the corresponding distance Euclidean distance matrix would be a 10-by-10 matrix representing the

distances between each pair of points. In this case, the diagonals would be 0 reflecting the fact that each point has zero distance to itself.

Of course, distances don't have to be Euclidean, nor do they have to represent distance in geographic space. Distances can represent any dissimilarity between samples, e.g. how many species sample plots share in common, the genetic relatedness of individuals, the least cost path between sample locations, etc., and there are numerous dissimilarity metrics created for such purposes. Importantly, there are a number statistical methods based entirely on distance matrices. For example, the entire family of methods known as Cluster Analysis works on distance matrices and there is a distance-based procedure for conducting analysis of variance (which is a nonparametric alternative to the least squares based classical approach). Consequently, it will be useful to learn how to convert a data frame or matrix into a distance matrix.

There are several functions available for creating dissimilarity matrices: dist[base], vegdist[vegan], daisy[cluster], dsvdis[labdsv], and data.dist[biostats]. They differ largely in the distance metrics provided.

To convert the birds data set into a distance matrix based on the Euclidean differences in species abundances among sample plots, type:

```
data.dist(birds[,-c(1:3)],method='euclidean')
```

Note, because the first three columns in the birds data set contain plot id information, including variables classified as "factors" (i.e., containing non-numeric characters) and not species abundances, we need to select the relevant columns. In the example above we used the index notation to deselect columns 1 thru 3; however, we could have also used the var= argument to select the variables by name.

For community data such as the species counts in the birds data set we would probably want to consider one of the proportional city-block dissimilarity measures instead of Euclidean distance, for example the Bray distance measure, as follows:

```
data.dist(birds[,-c(1:3)],method='bray')
```

OK, that's enough of an introduction to data exploration, screening and adjustments. Can you think of other diagnostics?