An R Companion to *Statistics for Archaeologists* by Robert Drennan

David L. Carlson
Anthropology Department
Texas A&M University
Version .6 - 04/02/12
http://people.tamu.edu/~dcarlson/quant/Drennan

Chapter 1. Batches of Numbers

This guide shows you how to use R, a free software environment for statistical computing and graphics. The guide is intended to be used in conjunction with Robert Drennan's book, *Statistics for Archaeologists*. It will be what Drennan describes as your "statpack." I suggest that you read each chapter first and then go back through the chapter with this guide while using R to produce the analyses, figures, and tables in the book. This guide does not include copies of the figures produced by using R so you will not get much out of it by just reading it. There are a number of excellent guides to R for new users, but they do not necessarily include all of the topics that archaeologists would find useful and they start with the basics and build up from there. This guide starts with the commands needed to produce the analyses in each chapter in *Statistics for Archaeologists*. At first they will seem much like magical incantations, but in each chapter we will talk about basic concepts and how they apply to the kinds of analyses we are doing. If you want to know a lot about the design of R and how it works before using it, this is not the guide for you. If you want to jump into using R and gradually learn the background, you may find this guide to be a productive way to learn.

As a platform for quantitative analysis, R has several advantages over commercial statistical packages:

1. R is completely free. You can take it with you wherever you go and you can update it whenever you want without paying any licensing fees. You obtain R by downloading it from one of over 70 websites worldwide. If you have an internet connection, you can obtain R.
2. R is available for computers running Windows, Mac OS X, and Linux.
3. R is continually being updated and extended. Because it is an open source program, people around the world continually add extensions to the basic system. Currently there are about 3000 packages that extend the basic functions in R.
4. R commands can easily be combined to perform specialized analyses and saved so that those analyses can easily be repeated with new or corrected data.
5. Using the command approach to data analysis can shift your orientation from what is convenient to what makes sense since you can easily design your own ways of summarizing, plotting, and analyzing your data.
6. Online R documentation is extensive and forums provide places for new R users to ask questions.
7. Books on various aspects of R are readily available. Springer has a growing series called "Use R" and Wiley has a series called "Statistics Using R" each with about two dozen titles covering R.

To obtain R, visit the main website: http://www.r-project.org/. You will download R from CRAN (The

Comprehensive R Archive Network) http://cran.r-project.org/. R is a command language meaning that you type commands and the R system responds to those commands. The advantage of this approach is that it is easy to record a series of commands in a file and it is easy to add new commands without overhauling the menu system each time. Typing commands does mean that R comes with a somewhat longer learning curve than a graphical program such as SPSS or Statgraphics. To ease you into the command interface we will install a graphical user interface written for R that gives you easy access to many commands in R. It will also give you an opportunity to see what the commands look like since they are printed as you select them from the menus. You can do a great deal with the Rcmdr interface and you may find that it provides everything you need. Every R command is not available from the menus and for those commands that are included in the menus, every possible option is not included. The command interface gives you access to every R command and every option. This limitation is not unique to R. The graphical menu systems of commercial programs such as SAS and SPSS are based on an underlying command language. The SPSS menus also do not include every SPSS command and some menu commands do not include every option.

For more details, see the document "Getting R" on my website which includes instructions on Installing R and Rcmdr. Once you have installed R and Rcmdr, look at the "R resources on the web." I recommend reading one of the manuals for beginners to get you started (for example "Using R for Data Analysis and Graphics" by J. H. Maindonald) and "R Commander: An Introduction" by Natasha Karp.

To get started you should first read the book chapter. Then go back through the chapter with this guide and your computer with R and Rcmdr loaded. There are no computer printouts or figures in this guide, but I will talk about them assuming that you have just run the analyses or generated the plots on your computer. My description won't make much sense if you are not working along with me. Generally you should type the commands into the Rcmdr Script Window, select them and click Submit or type them into the R Console. The Script Window lets you type multiple commands, select them and then run them all by clicking Submit. The R Console runs each command when you press the Enter key on the keyboard. If a command is not working in the Script Window, try typing it in the R Console window. It won't run, but the Console will print the whole error message.

If you are selecting and pasting commands, they should work as I have tested them while writing this guide. But somethings things happen. Most likely the problem will be conversion of special characters, principally the quotation marks and the dash. Word processors sometimes convert the simple quotation marks to open quote/close quote marks that R doesn't recognize or a dash to a hyphen. Type the command directly and it should work. Let me know about the errant command and I'll correct it in any subsequent editions.

As you are working through the chapter (or after you have finished it), take the time to look at the help page for every new command. For example, the command `?mean` brings up a help page on the mean that describes all of the options you can use with the mean. Always do this since even simple commands may have unexpected features. In the case of the `mean()` function you will discover that it can also compute trimmed means. In addition to getting help on single commands, you can browse the available functions in packages from The R Language help page (Help | Html help on the R Console or Help | Start R help system on Rcmdr). In addition to information about the packages you have installed on your computer, there are frequently asked questions, search functions and other resources.

Now that we've covered how to use the guide, let's talk about the first chapter. Drennan discusses

batches of numbers. In general, we measure objects and a set of the measurements for a particular variable or characteristic of those objects is a batch. Those objects can be postholes or stone tools. They can also be archaeological sites, bones, bronze fibulae, house pits, storage pits, a volume of excavated sediment, or just about anything else archaeologists recover or record from an archaeological site.

In R we usually represent each object as a row in a table. Each column of the table is a batch of that represents an attribute or property that we have recorded. The values can be numeric measurements (15mm, 4gm, or 5 decorative bands) or they can be non-numeric descriptions (red, cord-marked, jar-shaped, lipped, conoidal base, large). In R, this kind of table is referred to as a **`data.frame`**. A data.frame has rows labeled by **`rownames`** (e.g. Vessel 1, Vessel 2 in Table 2.1) although in many cases these are just row numbers. The columns are labeled with **`colnames`** (length, width, weight, color, temper, etc). You can refer to a specific data value in terms of its rowname and colname or in terms of its row number and column number.

Drennan introduces stem-and-leaf as a simple way of looking at the distribution of a batch of numbers. Stem-and-leaf plots we designed as a way of examining a batch of data by hand without calculators or computers, but we can make them with R. First we have to put the data on posthole diameters into R. We'll use R Commander to create the data set and the stem-and-leaf plot. Start up R on your computer and use the command `library(Rcmdr)` to load R Commander.

The R Commander window appears with a menu along the top and three windows stacked on on top of the other. The Script Window is where R Commander copies commands whenever you use the menus to perform an action. This lets you see what commands and options are being used and makes it easy for you to alter an option and re-rerun the command. You can type your own commands in this window and submit them to R for processing just by putting the cursor on the line (or selecting multiple lines) and clicking on the Submit button. You can also save the commands in this window for future reference (using the File | Save script menu option). The Output window echoes the commands and then displays any printed results that result from the commands. The last window is the Message window where error messages and status messages appear. If you produce any graphics, a separate graphics window will open.

We are going to create our first data set in R. Select the Data | New data set menu option. Call the data set “Postholes.” Now a small spreadsheet opens. Click on “var 1” at the top of the first column and type “Diam” as the variable name and click the circle next to numeric to indicate that Diam is a numeric variable. Now type the 13 numbers from Table 1.1 into the column and close the window. Notice that in the bar just underneath the menus Rcmdr lists Postholes as the current Data set. If you are working with multiple data sets (and you will be), you can click on the current data set name to access a drop down selection menu that lists all of the other data sets that are currently loaded. Next to the data set name is the Edit data set button that brings up the spreadsheet so you can add data or make corrections. Next to that is the View data set button that displays the current data set. Click on that button now to look at the Postholes data set and make sure you entered the numbers correctly. Before going any further, we need to save the data set you have created. Select File | Change working directory and select (or create) the folder you will save your R data sets in. Then select Data | Active data set | Save active data set to save Postholes. This step is important if you will be using the data set again because Rcmdr (and R) does not save data sets automatically.

Now you are ready to make your first stem-and-leaf plot. Select Graphs | Stem-and-leaf display. In the

dialog box that opens notice that the Variable Diam is selected for you (it is the only variable). Next to Leafs digit use the slider to change the value to 0.1. Under Parts per stem select 1 and uncheck all three options at the bottom. The stem-and-leaf plot is printed in the output window. It closely resembles Figure 1.1 except that it is inverted with low numbers at the top and large numbers at the bottom.

Follow the same procedure to create the Scraper data set with a single variable, Weight. Use View data set to proof your numbers and save your new data set. Then create a stem-and-leaf plot as we did for the Postholes. The result should match Table 1.2 (except that the lightest scrapers are at the top and the heaviest are at the bottom). To get Table 1.3 leave the Leafs digit set at 1 but change the Parts per stem to 2. Select Repeated stem digits and uncheck all the options. R does not preserve the decimal value in the stem-and-leaf plot so only the one's digit is included in the plot. This does not affect the shape of the plot, but it does mean that you cannot read off the original data from this plot the way you can from Table 1.3.

To get Table 1.4 follow the instructions for Table 1.3 but change the Parts per stem to 1. To get Table 1.5 follow the instructions for Table 1.3 but use the slider to change the Leafs digit to 10 and the Parts per stem to 2. Both of these tables group the data too compactly to see the distribution.

The next section illustrates back to back stem-and-leaf plots. These are not available in R yet so we can't reproduce those examples. You could enter the post hole data from the Smith site and construct a stem-and-leaf plot of just those data. Remember we called the Black site post hole diameters Postholes so you will have to come up with a different name.

Then we look at histograms. The example involves 29 site areas from the Kiskiminetas River Valley. I'll save you a bit of typing by giving you a single command to create the data set:

```
KRV <- data.frame(Area=c(12.8, 11.5, 14, 1.3, 10.3, 9.8, 2.3, 15.3, 11.2, 3.4,
12.8, 13.9, 9, 10.6, 9.9, 13.4, 8.7, 3.8, 11.7, 1.7, 12.3, 11, 2.9, 10.7, 7.4,
8.2, 2, 2.2, 4.5))
```

Copy this command into the Script window of Rcmdr. It will probably paste as a single line. If so, just leave the cursor on the same line (it doesn't need to be at the end) and click the Submit button. Check for error messages. Now click the Data set button and select KRV from the drop down list.
To get the stem-and-leaf plot just select Graph | Stem-and-leaf display. Then specify Leafs Digit 0.1 and Parts per stem 1. Unclick all of the options. After clicking on OK, you should have your stem-and-leaf plot. Now for the histogram just select Graph | Histogram. You can leave all of the options alone and click OK. The histogram opens in a separate graphics window and is similar to Figure 1.2 but there are fewer bars. Run it again, but this time specify 15 as the Number of bins (bars). To do this you will have to put the cursor in the box next to Number of bins: and delete the <auto> before entering the number. Now the graph looks closer, but the graph window is square so the bars seem taller than in Figure 1.2. Also the bins, the groups the individual observations are assigned to are defined somewhat differently. Drennan defines the 1 to 2 group as including 1 and any value just below 2, but not including 2 whereas R defaults to defining the 1 to 2 group as including anything just above 1, but not including 1 up through 2 (including 2). R is flexible and if this bothers you just find the command:

```
Hist(KRV$Area, scale="frequency", breaks=15, col="darkgray")
```

in the Script Window and insert "`right=FALSE`":

```
Hist(KRV$Area, scale="frequency", breaks=15, right=FALSE, col="darkgray")
```

Leave the cursor on the line and click Submit. Now the histograms match. You might want to spend some time trying the different options on the commands that we have used.

R also has an extensive help system that documents all of the options of each function (command). To find out about the options of the barplot() command, use the command `help(stem.leaf)` or ? `stem.leaf`. Either command will open your web browser to a page describing the `stem.leaf()` command. The help page provides a brief description of the command followed by a list of all the options (arguments) and their default values. Read over the pages for the `stem.leaf()` and `hist()` (you were actually using the `Hist()` function but it calls `hist()` and that is where most of the options are defined) commands.

In Practice Drennan introduces a new data set, scrapers from Pine Ridge Cave and Willow Flats. The website includes a separate page that provides links to data sets introduced throughout the book. If you want to build it yourself, call the data set Scrapers and create three variables: Site, Material, and Length. Alternatively you can type the data into a spreadsheet such as Excel. If you are using Windows you can select the data and column headings and copy them to the clipboard. In Rcmdr select Data | Import Data | from text file, clipboard or url. In the dialog box enter a data set name (Scrapers); in the Location of Data file section select Clipboard; and in the Field Separator select Tabs. Then click OK and message in the Messages window should indicate that dataset Scrapers has 48 rows and 3 columns. Now save this dataset before continuing. If the clipboard does not work or you are using a Mac or a Linux computer, save the spreadsheet as a csv file ("comma separated values"). Follow the previous instructions but just enter the data set name and change the Field Separator to Commas. Click OK and Rcmdr will open a file selector window for you to locate the file you created. It is also possible to import directly from several file formats including Excel and Access by selecting the menu option Data | Import Data | from . . . and choosing the correct option.

You should be able to create a stem-and-leaf plot from the combined data using the new Scrapers data set. To produce back-to-back stem-and-leaf plots by hand, you can use R to make the work easier by running two commands:

```
Scrapers[order(Scrapers$Site, Scrapers$Length),]
Scrapers[order(Scrapers$Material, Scrapers$Length),]
```

These commands introduce some additional features of data frames in R. You can think of a data frame as a table (just as it appears when you use the View data set button. The rows and the columns are both numbered. You can refer to the 17th row in Scrapers as Scrapers[17,]. R will print the entire 17th row since we did not put anything after the comma. If we had used Scrapers[17, 3] we would get only the Length of the 17th scraper. If we want to list an entire column we have several options: Scrapers$Length, Scrapers[,3], and Scrapers[,"Length"]. Although this may seem like overkill now, you will find all of these options useful in different contexts. Not that in general you cannot just use "Length" to refer to that column in Scrapers unless you tell R to look in Scrapers to find a column with that name. For now, just type the data set each time you refer to a variable. We'll talk about alternatives later, but if you pay attention to the commands that appear in the Rcmdr Script window when yoyu

choose various commands, you will see some examples.

The commands above use the `order()` function where the row specification goes in the brackets (i.e. [row, column]) and the column specification is blank (so the entire row will be listed). The `order()` function orders the rows according to your specifications. The first command sorts by Site and then by Length. The second command sorts by Material and then by Length. These two lists should make it easier to construct the back-to-back stem-and-leaf plots.

Chapter 2. The Level or Center of a Batch

Drennan introduces statistics that describe the center of a batch including the mean, the median, and the trimmed mean. He also introduces an example of a batch that has two centers. First we'll create the data in Table 2.1 (weights of flakes recovered from two bell-shaped pits):

```
Pit1 <- data.frame(Context="Pit 1", Weight=c(9.2, 12.9, 11.4, 9.1, 28.6, 10.5,
11.7, 10.1, 7.6, 11.8, 14.2, 10.8))
Pit2 <- data.frame(Context="Pit 2", Weight=c(11.3, 9.8, 14.1, 13.5, 9.7, 12, 7.8,
10.6, 11.5, 14.3, 13.6, 9.3, 10.9))
Flakes <- data.frame(rbind(Pit1, Pit2))
```

We have created three data sets, one each for pits 1 and 2 and a combined set. The first two commands create data sets with two variables, Context (which can be Pit 1 or Pit 2) and Weight which is the weight of the flakes recovered from the pits. Run these commands and then select either Pit1 or Pit2 in Rcmdr and view it. Notice that we typed “Pit 1” only once, but R repeated that label for every weight measurement. This is called recycling and it can make it easier to create data sets. It can also cause unintended errors if you create a data set with columns of unequal length by accident since the last entry in the short column will be recycled.

The quickest way to get the mean and median computations shown in Table 2.1 is to use the Flakes data set. Select it and then select Statistics | Summaries | Numerical Summaries. Weight is the only variable so it is already selected. Leave the various statistics selected and click on Summarize by groups. Context is the only grouping variable and it is already selected so click OK and the OK again to exit the dialog boxes. The results are displayed in the Output Window. Right now we are only interested in the mean (the first column) and the median (labeled here as the 50% quantile). The results match those in the table. If you just want the mean or median, there are functions for each. Just type the following commands into the Script Window, select them all and click Submit:

```
mean(Pit1$Weight)
median(Pit1$Weight)
mean(Pit2$Weight)
median(Pit2$Weight)
```

You can also easily get them out of the combined Flakes data set:

```
by(Flakes$Weight, Flakes$Context, mean)
by(Flakes$Weight, Flakes$Context, median)
```

After talking about the issue of outliers and how they can be eliminated, Drennan introduces the trimmed mean. R makes it relatively easy to compute the trimmed mean since there is a `trim=` option in the `mean()` function.

```
mean(Pit1$Weight, trim=.1)
mean(Pit2$Weight, trim=.1)
```

Or

```
by(Flakes$Weight, Flakes$Context, mean, trim=.1)
```

The only difference is that R does not round when calculating how many values to trim so `trim=.05` does not get rounded up from 0.6 to 1 so we use `trim=.1` to get match the results in the text (trim=.09 would give the same result).

Finally, Drennan discusses what it means when a batch has two centers or peaks. Using the following command (you could copy and paste it into the Script Window) create the BlackSmith data set and then create the stem-and-leaf plot in Table 2.2:

```
BlackSmith <- data.frame(Area=c(18.3, 18.8, 16.7, 6.1, 5.2, 21.2, 19.8, 4.2, 18.3,
3.6, 20.0, 7.5, 15.3, 26.8, 5.4, 18.7, 6.2, 7.0, 20.7, 18.9, 19.2, 6.7, 19.1,
23.4, 4.5, 16.2, 5.6, 17.5, 5.9, 6.7, 4.9, 17.9, 15.0, 13.6, 5.4, 5.8))
```

You should also be able to produce a histogram of these data that show the two centers.

To calculate the mean and medians of the two groups, we use the `by()` command again:

```
by(BlackSmith$Area, BlackSmith$Area>10, mean)
by(BlackSmith$Area, BlackSmith$Area>10, median)
```

If we were going to use these groups in further analysis we could create a new variable in the BlackSmith data set (perhaps called Size) to group the data into Large and Small sites.

The Scrapers data set is available on the website under Chapter 1. You should be able to figure out how to compute the statistics you need for the problems at the end of the chapter.

Chapter 3. The Spread or Dispersion of a Batch.

Drennan introduces statistics that describe the spread of a batch including the range, standard deviation, variance, interquartile range, and the trimmed standard deviation. We will first use the Flakes data set. Load it into Rcmdr if it is not already loaded. A quick way to get the range statistics is to use the menu option Statistics | Summaries | Table of statistics. When the dialog opens, the factor and the response variables are correct since they are the only ones in the data set. Click the Other option and type `range` in the box. Now click OK. In the Output Window you will see the minimum and maximum for Pit 1 and Pit 2. Alternatively you could type `fivenum` in the box and get the minimum, maximum, median,

and 1st and 3rd quartiles. A somewhat nicer set of descriptive statistics is provided by a function called `describe()` in the `psych` package. You will have to install this package before we can use it. Type and run the following command:

```
install.packages("psych")
library(psych)
```

R will ask you to select a mirror to download the package from. Select a mirror nearby. After the package is installed, the second command will load it into your workspace so that R can find it. You only have to install the package once, but you will have to load the package whenever you want to use it. It will then be available until you exit R. Now run the Table of statistics menu option, but this time specify `describe`. Now you will get a nice statistical summary of Pits 1 and 2. To find out what all of the columns mean run the command `?describe`.

The interquartile range or midspread is just the difference between the 3rd and 1st quartiles. You can compute the number from the quartiles you got when you ran the `fivenum()` function or you can run Table of statistics and specify `IQR` as the statistic. You should notice that the value does not agree for Pit 1 but it does agree for Pit 2. This difference highlights one of the problems with the quartile statistics (especially for small samples). There are multiple definitions of the quartiles. The R function IQR() which calls the function quantile() offers nine different ways of defining the quartiles! R uses number 7, SAS uses number 3, SPSS and Minitab use number 6 and a recent review article recommended type 8. For the details use `?quantile`.

The results of the `describe()` function give you the standard deviations for Pits 1 and 2 that you can use to follow Drennan's discussion. You can get the variance by selecting the Table of statistics again and this time typing in `var`. Because of the very large flake in Pit 1, the standard deviation and variance are much larger.

The trimmed standard deviation is not directly available in R, but we can compute it ourselves by using a function in the `psych` package called `winsor()`. Since the example is for Pit 1, lets create a new data set called Pit1 (even if you did save the one you created when we first created Flakes). With the Flakes data set selected, select Data | Active data set | Subset active data set. In the dialog box that opens, uncheck Include all variables and select Weight. In the Subset expression delete <all cases> and insert `Context=="Pit 1"`. Be sure to use two equals signs and put quotation marks around Pit 1. In the Name for new data set delete <same as active data set> and insert `Pit1`. No quotation here and no space between Pit and 1. Click OK and you should have the new data set listed under the menu bar. If you saved the Pit1 data frame you created earlier, you don't really need to do this, but the data set for the problems at the end of the chapter is similar and you may want to split it into two separate data sets. Follow this procedure twice, but make sure to re-select the combined data set after making the first subset data set since it will be selected.

To follow Drennan's example in Table 3.4 we need to reorder the data frame from largest to smallest:

```
Pit1<-Pit1[order(Pit1$Weight, decreasing=TRUE),]
```

The package `psych` contains several functions for computing Winsorized batches, as well as, Winsorized means, standard deviations, and variance. These are logically named. Type and run the

following three commands:

```
winsor(Pit1$Weight, trim=.09)
winsor.mean(Pit1$Weight, trim=.09)
winsor.sd(Pit1$Weight, trim=.09)
winsor.var(Pit1$Weight, trim=.09)
```

Note that we are using a trim of 9% rather than the 5% that Drennan uses. The authors of the `psych` package used a slightly different method of Winsorizing. It involves replacing the trimmed values with the quartiles defined by the trim. Using a 9% trim matches Drennan's results better. To get the trimmed standard deviation, let R do the calculations:

```
sqrt((12-1)*winsor.var(Pit1$Weight, trim=.09)/(10-1))
```

For the exercises at the end of the chapter, the data set Nanxiong is available on the website for you to use.

Chapter 4. Comparing Batches

Drennan introduces a new graph type, the box-and-dot plot (also called the box-and-whisker plot)

Select all four lines and paste them into the Script Window, select them in the Script Window and then click the Submit button to make a data set combining the post hole diameters from the Brown and Smith sites:

```
BSPosts <- data.frame(Site=c(rep("Black Site", 13), rep("Smith Site", 15)),
     Diameter=c(9.7, 9.2, 12.9, 11.4, 9.1, 44.6, 10.5, 11.7, 11.1, 7.6, 11.8,
     14.2, 10.8, 20.5, 17.2, 15.3, 15.9, 18.3, 17.9, 18.6, 14.3, 19.4, 16.4,
     18.8, 15.7, 18.9, 16.8, 8.4))
```

You can now use Rcmdr to create a box-and-dot plot. Select Graphs | Boxplot. The Diameter variable is already selected. Click on the Plot by groups button. The Group variable is already selected so click OK twice and a graphics window will open with your plot. It looks like Figure 4.2 but squashed since the graphics windows is square. You can find out about the boxplot() function and its options by typing the command ?boxplot. For now, we can change the look of the graph by grabbing the lower right corner of the graphics window and resizing the window to make it a tall, narrow rectangle. This changes the proportions of the graph so that it more closely resembles the figure.

Looking at the figure it is clear that the median post hold diameter of the two sites is different. But what about the spread? Drennan discusses transforming the data by removing the median so that you can focus on the distribution. We can easily do this in Rcmdr. First use the Table of statistics menu option to get the medians for each site (Black site = 11.1, Smith Site= 17.2). Next select Data | Manage variables in active data set | Compute new variable. In the New variable name box put “MdDiam” (do not type the quotation marks). In the Expression to compute box put “`ifelse(Site=="Black Site", Diameter-11.1, Diameter-17.2)`.” Again, no quotation marks. Click OK and then View the data set. It now has a new variable. Create a box-and-dot plot with the new variable, plotting by Site. This will

match Figure 4.3. Now it is clearer that the interquartile range (midspread) is a bit less for the Black site.

The next step is to remove the spread. First use Table of statistics, select MdDiam and Other and put IQR in the box to get the midspread for each group, Now select Data | Manage variables in active data set | Compute new variable. In the New variable name box put "MdSpDiam" (do not type the quotation marks). In the Expression to compute box put "`ifelse(Site=="Black Site", MdDiam/2.1, MdDiam/2.9)`." Now create the boxplot. This one will resemble Figure 4.4.

Drennan points out the useful features of standardizing numbers by subtracting the median and dividing by the midspread. R has a function for standardizing variables called `scale()`. If we subset the BSPosts data set separately we can easily use it to scale the diameters of each group:

```
BlPosts <- BSPosts[BSPosts$Site=="Black Site", ]
BlPosts <- data.frame(BlPosts, Scaled = scale(BlPosts$Diameter, center =
median(BlPosts$Diameter), scale = IQR(BlPosts$Diameter)))
BlPosts <- data.frame(BlPosts, Z= scale(BlPosts$Diameter))

SmPosts <- BSPosts[BSPosts$Site=="Smith Site", ]
SmPosts <- data.frame(SmPosts, Scaled = scale(SmPosts$Diameter, center =
median(SmPosts$Diameter), scale = IQR(SmPosts$Diameter)))
SmPosts <- data.frame(SmPosts, Z= scale(SmPosts$Diameter))
```

You could do this with the menus in Rcmdr by first using the Subset the active data set command twice and then using the Compute new variable twice on each, but it is easier to follow the steps by looking at the commands. The first command three commands extract the Black site measurements and compute the two sets of scores while adding them to the data set. The second three do the same for the Smith site. You can save the separate data sets if you want. To put them back together again just requires one command:

```
BSPosts <- data.frame(rbind(BlPosts, SmPosts))
```

Use the Nanxiong data set from the last chapter to answer the exercises.

Chapter 5. The Shape or Distribution of a Batch

Drennan introduces the concept of the shape of a batch. Symmetry simply means that the values fall equally on either side of the median and the median falls at about the middle of the midspread. Data on storage pits (Table 5.1) are introduced and we can create a data frame for R as follows:

```
BVPits <- data.frame(Volume=c(1.23, 1.48, 1.55, 1.38, 1.10, 1.02, 1.29, 1.32,
1.35, 1.65, 1.39, 1.40, 1.12, 1.46, 1.24, 1.34, 1.21, 1.45, 1.51))
```

Create the data set and select it in Rcmdr. Then produce a stem-and-leaf plot. Look at the distribution and note its symmetry. Now create the second data set (Table 5.2):

```
BAPits <- data.frame(Volume=c(1.22, 1.64, 1.16, 1.07, 1.50, 1.84, 1.37, 1.15,
1.29, 1.32, 2.03, 1.17, 1.04, 1.43, 1.11, 1.40, 1.26))
```

Create the data set and select it in Rcmdr. Then produce a stem-and-leaf plot. Look at the distribution and note the asymmetry. Using the Buena Vista pit volumes, Drennan tries seven different transformations (Figure 5.1). You could add these one by one to the data set using the Data | Manage variables in active data set | Create new variable seven times or you can create them all with a single (long) command using the `transform()` function:

```
BVPits <- transform(BVPits, Vsqrt=sqrt(Volume), Vlog=log(Volume), VInv=-1/Volume,
VInvsq=-1/Volume^2, V2=Volume^2, V3=Volume^3, V4=Volume^4)
```

You might want to save the data set after this. We can produce all of the stem-and-leaf plots with a single command:

```
temp <- apply(BVPits, 2, stem.leaf, trim.outliers=FALSE, depths=FALSE,
style="bare")
```

The plots are pretty similar, but there are rounding differences between them (R doesn't round the numbers in the plot). You can at least see the way the transformations change the distributions. The `apply()` function is very handy, but confusing at first. It lets us apply a function to the data set row by row (1) or column by column (2). We are applying the `stem.leaf` function to each column. We could end the command there and let the `stem.leaf` function use its default settings, but we wanted to change some of them so we put the optional specifications after the name of the function. When used with `apply`, the stem.leaf function returns character strings needed to produce the stem-and-leaf plot. We are capturing that output to the variable temp so that it will not be printed in the Output Window.

Drennan also compares the various transformations using box-and-dot plots after transforming them to the the same scale. Let's plot the unstandardized values first. Rcmdr provides access to box-and-dot plots when a single variable is divided up according to categorical variable (called a factor), but it does not let us plot several columns in a data set. By typing the command, we can do this easily:

```
boxplot(BVPits)
```

Because of the scale differences it is hard to compare the plots. Let's standardize using the median and midspread:

```
BVStandard <- apply(BVPits, 2, function(x) scale(x, center=median(x),
scale=IQR(x)))
boxplot(BVStandard)
abline(h=c(-.5, .5), lty=2)
```

We plot the standardized values and add the dashed horizontal lines to mark the boundaries of the upper and lower values of the midspread. Because there are so few values, the standardized boxes do not line up exactly between the lines.

You should now be able to repeat this analysis with BAPits, the second data set we created. Drennan

only uses some of the transformations, so you can delete the ones he does not use.

At the end of the chapter Drennan introduces the normal distribution. R has functions for a number of distributions built. To take a look at the shape of a normal distribution select Distributions | Continuous distributions | normal distribution | Plot normal distribution. When the dialog box pops up, just click OK to get the standard normal distribution plot (mean = 0, standard deviation = 1).

The exercises use the Nanxiong data set from Chapter 3. You can find it on the website.

Chapter 6. Categories

Drennan introduces the concept of categorical variables with an example of 140 pottery sherds from three sites. You will find the data set on the website called Sherds.RData. If you load the data set you can use R commands to follow the examples. Once you have the data loaded into Rcmdr we can try out the menu commands for producing tables. For Tables 6.2 and 6.3 we are only looking at the number of pottery sherds in each site (6.2) or the number of each style (6.3). In Rcmdr select Statistics | Summaries | Frequency distributions. Select Site and click OK. Rcmdr displays the number of sherds in each site and then the percentage. The total is not displayed though and the sites are in alphabetical order instead of the ordering that Drennan uses. R stores categorical variables as factors. In the absence of other instructions, R orders factors alphabetically. You can tell that Site is a factor with the commands `str()` (to look at the underlying structure of the variable and `levels()` to get a list of the categories and their sequence:

```
str(Sherds$Site)
levels(Sherds$Site)
```

Rcmdr will let you change the ordering of a factor. While the site ordering is not that important here, there are times when you will want the factor levels to be in a particular order so we'll change the order of the Sites to see how it is done. Select Data | Manage variables in active data set | Reorder factor levels. In the dialog box, select Site and click OK. Click Yes to overwrite the existing variable. In the next dialog box make Cypress Swamp 3 and Oak Grove 1. Then click OK and regenerate the table.

Now generate Table 6.3 showing the number of sherds and their percentages by Style. To produce Table 6.4 we need to use a different menu option. Select Statistics | Contingency tables | Two way table. In the dialog box select Style as the row variable and Site as the column variable. Check Column percentages and uncheck Chi square test of independence. The cross-tabulation of Style and Site appears in the Output Window along with a separate table of column percentages.

We have more control over the tables if we use R commands. For example we can add margins (row and/or column totals) and compute percentages by row, column, or across the entire table. The two principal commands are `table()`, `xtabs()`, `addmargins()`, and `prop.table()`. To create Table 6.2 try the following commands:

```
addmargins(table(Sherds$Site))
addmargins(prop.table(table(Sherds$Site)))*100
```

We could save the result of the `table()` function and save some typing:

```
SiteTable <- table(Sherds$Site)
addmargins(SiteTable)
round(addmargins(prop.table(SiteTable))*100, 2)
```

This time we rounded the results to two decimal places.

For the cross-tabulation of the two variables we can use either `table()` or `xtabs()`.

```
Xtable <- table(Sherds$Style, Sherds$Site)
Xtable
Xtable <- xtabs(~Style+Site, data=Sherds)
Xtable
```

These commands produce almost identical tables. In many circumstances you can use either version. With `table()` you just list the row variable followed by the column variable. If you list one variable, it will produce a one-way table. The `xtabs()` function uses a `formula` to specify the table. Basically a formula consists of a dependent variable, then a tilde (~), then independent variables joined by plus signs (+). Formulas are a flexible way of specifying a statistical model and you will use them more in the coming chapters. Here the dependent variable is blank because it is not a variable, it is frequency of sherds in cell. There is another advantage of `xtabs()`. If your categorical data is stored in a compact form, `xtabs()` can produce a table directly, but `table()` cannot. The following commands convert `Xtable` (the table we just produced to a data.frame.

```
Ftable <- data.frame(Xtable)
Ftable
xtabs(Freq~Style+Site, data=Ftable)
```

The data set `Ftable` uses six lines to provide all of the information presented in the 140 lines of data in Table 6.1 so it is a very compact way of representing categorical data and you will often find categorical data presented this way. The function `xtabs()` can convert these data into cross-tabulation tables for further analysis. Also, xtabs() has a data= option that lets us specify the just the variable name so it saves some typing.

Toward the end of the chapter Drennan introduces bar graphs as a way of summarizing categorical data. To make some in Rcmdr select the Sherds data set. First just make some single variable bar graphs. Select Graphs | Bar graph and then choose Site. This gives you a bar graph of the number of sherds at each site. Then follow the same procedure and choose Style to get a bar graph of the number of incised to unincised sherds. Unfortunately this is as far as Rcmdr can take us, but the barplot() command has many variations as long as we type commands directly:

```
Xtable <- xtabs(~Style+Site, data=Sherds)  # Build Xtable
barplot(Xtable, ylab="Frequency")          # Stacked bar graph
barplot(Xtable, ylab="Frequency", beside=TRUE) # Side-by-side bar graph
barplot(Xtable, ylab="Frequency", beside=TRUE, legend.text=TRUE) # with legend
barplot(t(Xtable), ylab="Frequency")       # Stacked bar graph, flip rows/columns
barplot(t(Xtable), ylab="Frequency", beside=TRUE) # Side-by-side bar graph
```

```
barplot(t(Xtable), ylab="Frequency", beside=TRUE, legend.text=TRUE) # with legend
barplot(t(Xtable), beside=TRUE, legend.text=TRUE, args.legend=list(x="topleft"))
# The next two use percentages to match Figure 6.1
barplot(t(prop.table(Xtable, 2)*100), ylim=c(0,75), ylab="Percent", beside=TRUE,
legend.text=TRUE)
barplot(prop.table(Xtable, 2)*100, ylim=c(0,75), ylab="Percent", beside=TRUE,
legend.text=TRUE)
# Two more illustrating color
barplot(t(prop.table(Xtable, 2)*100), ylim=c(0,75), ylab="Percent", beside=TRUE,
legend.text=TRUE, col=c("red", "green", "blue"))
barplot(prop.table(Xtable, 2)*100, ylim=c(0,75), ylab="Percent", beside=TRUE,
legend.text=TRUE, col=c("red", "green", "blue"))
```

The exercises focus on survey data from Al Amadiyah. You will find the data set on the website.

Chapter 7. Samples and Populations.

Most of the chapter deals with practical methods of sampling. Drennan discusses how to use a table of random numbers to draw various kinds of samples. You can also use R to draw samples quickly and easily (and without closing your eyes). The main function for drawing samples in R is `sample()`.

For example, you are drawing a random sample of grid squares to survey as the first stage of a field project. If survey blocks are 1 x 1 km and the survey area is 15 x 45 km, there are 675 grid squares. If your plan is to survey 100 squares, a simple way to draw the sample is to assign numbers 1 through 675 to the grid squares and then use `sample()` to choose 100 squares:

```
Grids <- sample(1:675,100)
Grids
sort(Grids)
```

The first command draws 100 numbers from the range 1 to 675 without replacement (so no grid will be selected more than once). The second command lists the grid numbers in the order they were selected and the third command lists them from smallest to largest.
The only drawback is that you have to assign a number to each grid and then figure out which grids were selected. If you have a list of the coordinates of the survey grids, you can select them randomly and print out the coordinates of the chosen units. Assume the southeast corner of the survey area is North 100, East 100. We have North grid corners from 100 to 114 and East corners from 100 to 144. To create a list of survey grids we need two commands:

```
SurveyUnits <- expand.grid(N=100:115, E=100:144)
ProjectGrids <- paste("N", Survey$N, "/E", Survey$E, sep="")
```

SurveyUnits is data.frame with two columns and 675 rows. It contains the North and East coordinates for all of the units in the survey area. The second command creates ProjectGrids, a single column of grid square labels (e.g. N100/E100). In any real project there will probably units that have been previously surveyed or cannot be surveyed because of extensive development or lack of landowner permission. If there are many of these, you could export the data.frame to a spreadsheet to remove the

rows. If there are only a few, they can be eliminated before drawing the sample:

```
Exclude <- c("N101/E104", "N104/E129", "N106/E121", "N111/E129", "N110/E101",
"N100/E135")
ProjectGrids <- ProjectGrids[!ProjectGrids %in% Exclude]
SampleGrids <- sample(ProjectGrids, 100)
```

SampleGrids contains a list of the units that should be sampled. A similar procedure lets you draw a random sample of artifacts by their catalog number for further study (e.g. compositional analysis, usewear, thin sections). If for some reason you need to sample with replacement, just add the option `replace=TRUE` to the `sample()` function.

Chapter 8. Different Samples from the Same Population

This chapter explores the concept of a sampling distribution by taking samples from a group of 17 post hole measurements. The following commands will construct the vector of measurements for you and compute the mean and standard deviation:

```
Postholes <- c(10.4, 10.7, 11.1, 11.5, 11.6, 11.7, 12.2, 12.6, 12.9, 13.2, 13.7,
14.0, 14.3, 15.0, 16.4, 18.4, 20.3)
length(Postholes)
mean(Postholes)
sd(Postholes)
plot(density(Postholes))
```

Drennan wants to use this group of measurements as our population and we will try to estimate the population mean from samples taken from this population. We first select 17 samples of 1 as a way of estimating the mean of the population. We can use R to make a figure that may help to illustrate the process:

```
plot(density(Postholes), las=1, xlim=c(7, 24), xaxp=c(7, 24, 17), ylim=c(0, .18),
yaxp=c(0, .18, 9), lwd=2)
abline(h=c(0:18/100), v=c(7:24), lty="dotted", col="gray")
rug(Postholes)
abline(v=mean(Postholes), lwd=2, col="red")
arrows(mean(Postholes)-3, .02, mean(Postholes)+3, .02, length=.1, angle=90,
code=3, lwd=2, col="blue")
```

These commands produce a density plot using a procedure called kernel density estimation. You can see that the measurements are not completely symmetrical. The red line identifies the mean post hole diameter and the small tick marks just above the x-axis represent the value of each of the 17 post hole diameters. Since the sample size is one, these tick marks also represent the possible samples. The blue line indicates the range of ± 3.0 cm, the maximum error that Drennan has selected for the example.

Drennan looks at all the possible samples of 1 post hole measurement, then 2 measurements, and then 3

measurements and shows how well these samples estimate the mean. We can use R to take a slightly different approach that more resembles bootstrapping. We assume the population consists of the same 17 measurements. Then we can draw random samples from this population in various sizes with replacement. We can then look at the distribution of the means of those different sized samples. The R commands are straightforward and we will be repeating blocks of commands:

```
plot(density(Postholes), ylim=c(0, .45), xlab="Mean Diameter", main="Sampling
Distributions")
abline(v=mean(Postholes))

samples <- sapply(1:500, function(x) mean(sample(Postholes, 2, replace=TRUE)))
mean(samples)
sd(samples)
lines(density(samples), col="red")

samples <- sapply(1:500, function(x) mean(sample(Postholes, 3, replace=TRUE)))
mean(samples)
sd(samples)
lines(density(samples), col="blue")

samples <- sapply(1:500, function(x) mean(sample(Postholes, 4, replace=TRUE)))
mean(samples)
sd(samples)
lines(density(samples), col="green")

samples <- sapply(1:500, function(x) mean(sample(Postholes, 10, replace=TRUE)))
mean(samples)
sd(samples)
lines(density(samples), col="violet")
legend("topright", c("Sample Size =  1", "Sample Size =  2", "Sample Size =  3",
"Sample Size =  4", "Sample Size = 10"), lty=1, col=c("black", "red", "blue",
“green", "violet"))
```

We start off plotting the kernel density of the original post hole measurements just as we did before except this time we have increased the range of the y axis. The vertical line marks the mean. Then we have four blocks of commands. The first command in each block draws a random sample with replacement for sample sizes of 2, 3, 4, and then 10 and computes the mean of that sample. That process is repeated 500 times. The next two commands print out the mean and standard deviation of those 500 means. The last command uses the kernel density function to plot the sampling distribution for that sample size. For example, the red line shows the mean posthole diameters for samples of two post holes where we drew 500 samples of two postholes. The last line adds a legend to the plot.

Looking at the plot you can see that as the sample size increases from 1 (the black line) to 10 (the violet line) the distributions become narrower and taller. In other words the various estimates of the mean diameter are closer to one another. Looking at the Output Window you can see that the means stay about the same, but the standard deviation decreases as the sample size increases. By a sample size of 10, the means are already much more symmetrical than the original data distribution.

Using the results you can compare the standard deviations of the sampling distributions we just computed to the expected values based on the equation on page 105.

```
sd(Postholes)/sqrt(c(2, 3, 4, 10))
```

The values will not be exact, but they should be close.

Chapter 9. Confidence and Population Means

The first half of the chapter explores the concept of confidence limits. You already know that Rcmdr can plot normal distributions, but you could use it to plot some of the distributions in Figures 9.1 – 9.5. Shading the distributions as in Figures 9.6 and 9.7 is possible, but not really relevant to understanding the concepts presented.

The section on Student's t provides us with more opportunity to use Rcmdr. The Student's t distribution is one of the distributions included on the Distributions tab. Select Distributions | Continuous distributions | t distribution | Plot t distribution. Put a number in the degrees of freedom box and click OK. Try several different values. At about 60 degrees of freedom, the t distribution is very similar to the normal distribution (which is on Table 9.1 on the bottom lined marked ∞). Unlike the table you can get the t distribution for any number, for example 99. We can use Rcmdr to follow along with Drennan's presentation at the bottom of page 118. We have an error range of two standard errors and we want to know what the probability is that the population mean falls within that range. Select Distributions | Continuous distributions | t distribution | t probabilities. In the dialog box put the values 2 and 99 degrees of freedom and click upper tail. Click OK. You get a small number (.02411985). This is the proportion of the t distribution curve that includes values greater than 2. It is the upper tail or the right hand side of the distribution. Only .02411985 percent of the time will your population mean fall in that area (which is outside the confidence interval). But we also have a lower limit at -2. We could plug in -2, 99, and lower tail, but you will get exactly the same answer because the t distribution is symmetrical. You could just double .02411985 and subtract that from 1 to get the probability that your population mean lies within two standard errors. A pretty compact R command will to it all for you:

```
diff(pt(c(-2, 2), df=99))
diff(pt(c(-3, 3), df=99))
diff(pt(c(-1, 1), df=99))
```

The first command returns .9517603. So the probability is about 95.2% for a confidence interval of plus or minus two standard errors. The second command returns .9965845 so the probability is over 99.7% for a confidence interval of plus or minus three standard errors. The third command returns . 6802515. So the probability is about 68.0% for a confidence interval of plus or minus one standard error.

To estimate the number of standard errors for a particular confidence level select Distributions | Continuous distributions | t distribution | t quantiles. If we want a 90% confidence interval we want each tail to contain 5% so let's find the quantile for 95% measured from the lower (left) side of the distribution. We get 1.66 so we need plus or minus 1.66 standard errors to construct a 90% confidence interval. The command is just:

```
qt(.95, df=99)
qt((.9+1)/2, df=99)
```

The first command is shorter, but the second does the math for you so you just insert .9 for 90% confidence. Both give the same result.

The finite population adjustment reduces the standard error when the size of the population is finite and you know what the size of the population is. For example a standard error for some estimate for the 50 states in the U.S. or for the 195 countries in the world. We'll use the finite population corrector to illustrate the way that R allows you to write functions to handle computations that you may do repeatedly:

```
FPC <- function(std, n, N) {std/sqrt(n)*sqrt(1-n/N)}
FPC
FPC(3.21, 25, 53)
```

The function name is FPC (finite population corrector). When you define the function, there is no output unless there is an error. If you just type the function name, R prints the definition of the function. The function takes three values (arguments), the standard deviation, the sample size, and the population size. If you put the values in this order, you do not need to identify them. If you identify them, you can put them in any order(e.g. Using the sample of 25 rim sherds: `FPC(N=53, n=25, std=3.21)`).

Likewise we can make a function for computing sample size:

```
Sample <- function(std, t, ER) {(std*t/ER)^2}
Sample(.9, 1.96, .5)
qt((.95+1)/2, df=12)
Sample(.9, 2.18, .5)
qt((.95+1)/2, df=15)
```

We define the sample and use it to estimate the sample size. But there is a small problem. We estimated 7 as if the degrees of freedom was infinity, but the suggested sample is much smaller. Using the `qt()` function, we see that a better estimate of t for 12 degrees of freedom (a sample size of 13) would be 2.18. Plugging that into the function, we get 15.4 as our sample size. For a df of 15 (sample size 16) the t value is 2.13, not much different from the 2.18 we used so we are probably better with a sample size of 15 (only two more than the original estimate).

The chapter ends with a discussion of assumptions and robust measures. As you will remember, R has features for computing trimmed means and standard deviations. We can run through his example using the data in Table 9.3:

```
PtWgt <- c(96, 37, 28, 34, 52, 18, 21, 39, 156, 43, 44, 19, 30, 108, 55, 24, 28,
47, 39, 31)
mean(PtWgt)
mean(PtWgt, trim=.15)
library(psych)
winsor.var(PtWgt, trim=.17)
```

The mean and trimmed mean match Drennan's results. We load the psych package and compute the Winsorized variance but since R Winsorizes differently than Drennan, we have to do some trial and error adjustment of the trim factor to get close to his value.

The data sets for the Exercises are all relatively compact:

```
CsLFlakes <- data.frame(Length=c(4.7, 4.1, 8.0, 3.2, 9.7, 7.5, 5.7, 5.0, 6.8, 6.2,
9.3, 6.9, 7.3, 4.5, 3.9, 5.4, 3.5, 6.0, 8.3, 5.5, 4.3, 4.8, 5.6, 6.1, 5.9, 7.8,
8.1, 4.3, 4.7, 3.0, 6.1, 5.1, 6.5, 8.8, 7.4, 8.5, 6.3, 7.0, 5.3, 2.6))

BuTHearths <- data.frame(Diameter=c(0.91, 0.51, 0.76, 1.64, 0.85, 0.88, 0.72,
0.77, 0.69, 0.75, 0.80, 0.90, 0.58, 0.60, 0.70, 2.47, 0.84, 1.00, 1.03, 0.66,
0.76, 0.96, 0.74, 0.64, 0.62, 0.86, 0.84, 0.82, 0.93, 0.95, 0.56, 0.78, 0.89,
0.98, 1.08, 0.83, 2.13, 0.66, 0.62, 1.93, 0.68, 0.80, 0.74, 0.93))

HuancaBlades <- data.frame(Zinc=c(53, 37, 60, 49, 66, 55, 41, 33, 82, 59, 48, 22,
74, 57)
```

Chapter 10. Medians and Resampling

Drennan provides a stem-and-leaf plot of the data that he uses to illustrate bootstrapping, but he does not present the raw data. The site areas are clearly measured to at least a tenth of a hectare (since there are some sites listed as having 0 area), but we can only get the data to the nearest hectare from Figure 10.1. You will find data set EClassic on the website. Load it to run the following commands:

```
MedianArea<-sapply(1:10000, function(x) median(sample(EClassic$Area,
replace=TRUE)))
plot(density(MedianArea))
quantile(MedianArea, p=c(.01, .025, .05, .95, .975, .99))
```

Three commands run the bootstrap, plot, and print the results. The first command does the bootstrap. We draw a sample of 113 from the areas of the Early Classic sites with replacement and compute the median. We do this 10,000 times and save the results in `MedianArea`. Then we plot the kernel density of the results. Finally we generate the numbers for three confidence limits: 98% (the first and last values), 95% (the second and fifth), and 90% (the third and fourth). The results will not match Drennan's exactly because we are using an approximation of the original data set.

To bootstrap the mean just change `median` in the first command to `mean` (you don't have to change the variable name to match) and run the three commands again. You could also bootstrap the trimmed mean, but you will have to insert the `trim=` option carefully to make sure it is inside the mean parenthesis but outside the sample parentheses.

It should not be difficult to modify the three commands to work for the Late Bronze Age sites from Nanxiong for the exercise.

Chapter 11. Categories and Population Proportions.

Here we are concerned with putting confidence intervals around proportions computed from samples. Drennan presents the basic formulas for computing the standard deviation and the standard error of a proportion . You can use R to compute the values. Proportions are actually discrete data. We "measure" 13 points out of 100. Discrete simply means that we could not have measured 13.5 or 14.5 points out of 100. Discrete data can often be modeled with a binomial distribution. This is the same distribution that governs the tossing of a coin (the probability is .5 for a head and .5 for a tail) or rolling a die (the probability is 1/6 that you will roll a 1). The procedure Drennan describes takes advantage of the fact that large samples and middling percentages (not very small or very large) are well described by a normal distribution.

The binomial distribution is available in Rcmdr. To get a plot of a normal distribution select Distributions | Discrete distributions | binomial distribution | Plot binomial distribution. Use 100 trials and a probability of .13 to plot the distribution for the obsidian projectile points. Instead of the continuous curve that you get with a normal or t distribution, here you get a series of vertical lines showing the probability of any outcome. Select Distributions | Discrete distributions | binomial distribution | Binomial quantiles and enter .025, .975 as the probabilities, 100 as the trials, and .13 as the probability. Rcmdr gives you 7 and 20 as the range (7 to 20%) a bit narrower than the interval that the normal approximation gives us. Part of the difference is that Rcmdr is expressing the confidence interval in discrete values, the percentages provided by the normal approximation are more useful for comparing with samples of different sizes. The result does suggest that the lower bound is too small so that 7% – 20% would be a better interval. If we look at the probability of getting 6 successes and subtract that from the probability of getting 20, it indicates that the confidence interval is actually 98% rather than 95% (since we are limited to integers in defining the range).

The finite correction and the estimation of sample size follows the format of the previous chapter. As before you can create a function to perform the calculations.

The exercises do not introduce new data sets, but if you made functions to compute confidence intervals or sample sizes, you will be able to put them to use.

Chapter 12. Comparing Two Sample Means

The chapter looks at ways to compare the means of two groups to see if they are different from one another in some fundamental way. The data are presented in Figure 12.1. The following commands create separate data sets for Formative and Classic sites and a combined data set for both:

```
Formative <- data.frame(Area=c(29.9, 29.6, 28.7, 27.7, 27.5, 26.5, 26.5, 26.3,
25.6, 25.6, 25.4, 24.9, 24.8, 24.5, 24.5, 24.3, 24.3, 23.4, 23.3, 23.3, 22.9,
22.3, 22.1, 21.9, 21.8, 21.5, 20.7, 20.2, 19.4, 18.3, 17.0, 16.1))
Classic <- data.frame(Area=c(34.1, 34.2, 33.1, 32.8, 32.4, 32.3, 31.7, 31.5,
31.2, 30.4, 30.3, 30.0, 29.9, 29.6, 29.6, 29.3, 28.9, 28.3, 28.3, 28.0, 27.9,
27.3, 27.1, 26.5, 26.3, 26.3, 26.3, 25.9, 25.8, 25.8, 25.8, 25.2, 24.5, 24.4,
24.0, 24.0, 23.7, 23.4, 23.3, 22.8, 22.7, 22.6, 22.3, 21.7, 21.2, 21.7, 20.4,
```

```
19.7, 19.4, 18.9, 17.5, 14.7))
HouseFloors <- data.frame(rbind(cbind(Period="Formative", Area=Formative),
 cbind(Period="Classic", Area=Classic)))
```

You can plot stem-and-leaf graphs for each group and then use the combined data set to produce box-and-dot plots. R does not have a function for bullet graphs, but it would not be hard to implement. Load the `psych` package and use `describe.by` to produce descriptive statistics for the two groups:

```
library(psych)
describe.by(HouseFloors$Area, HouseFloors$Period)
```

To simplify computing different kinds of confidence intervals, we can create a simple function and use that with the `by` function to generate confidence intervals for both periods:

```
confidence <- function(x, conf){
     return(c(mean(x)+qt((1-conf)/2, df=length(x)-1)*sd(x)/sqrt(length(x)),
mean(x)-qt((1-conf)/2, df=length(x)-1)*sd(x)/sqrt(length(x))))
}
by(HouseFloors$Area, HouseFloors$Period, confidence, .8)
```

Create the function and then use it to compute the 80%, 95%, and 99% confidence intervals for the floor areas for both periods.

Now let's use Rcmdr to compute the difference of means test comparing the Classic and Formative house floor areas. Select (load if necessary) the HouseFloors data set. There are two ways to perform a t test. One assumes that the variances of the two samples are equal. This is the one that Drennan describes. The other assumes they are not equal. R does both and the default is to use the second one (variances not equal). We can test the variances first. Select Statistics | Variances | Two Variances F-Test. Period is already selected as the Group and Area as the Response so just click OK unless you want to change the confidence level. Look at the output. F is 0.5625 with a p-value of 0.089 which is not less than .05 (the 95% confidence level). We accept that the group variances are not different from one another (we would consider them to be different if p was less than .05). That means that we can use the pooled variance t test (the one Drennan describes) instead of the separate variance t test.

Select Statistics | Means | Independent samples t test. There is only one group variable and one response variable so you don't have to select anything. Under “Assume equal variances?” click Yes (No is the default) and click OK. Look at this output. The t value is -2.6742 with a p value of .009 which is considerably less than .05 so we conclude that the floor areas are different in the different periods. The 95% confidence interval indicates that the difference probably falls between -4.3 and -0.63 (Mean Formative – Mean Classic). The last line shows the mean floor areas for Formative and Classic sites.

Drennan describes the one sample t-test with an example of sex ratios. We can run that test as well using Rcmdr. First we create the data set coding females as 1 and males as 0:

```
Burials <-data.frame(Sex=c(rep(1, 21), rep(0, 25)))
```

Then select the Burials data set. Select Statistics | Means | Single sample t test. You don't have to

change anything so click OK. The one sample t value is -0.5855 (the same as Drennan's .589 since negative or positive is irrelevant to the test (%Males – %Females vs. %Females - %Males). The p value is .5611 nowhere near being less than .05 so we accept the null hypothesis that the sex ratio is 0.5. The 95% confidence interval is 30.7% to 60.6% females. Other opportunities to use the single sample t-test come from pre-test, post-test experiments. For example you could produce a sample of stone flakes and record edge damage from the edges, then trample the flakes and record the edge damage again. Subtracting before from after gives you a measurement to use in a one sample t-test with a null hypothesis of no difference between the before and after observations (a difference of 0).

Under robust methods Drennan uses a notched box-and-dot plot. To make this in Rcmdr first produce the box-and-dot plot. Now look at the command (the last one in the Script Window) that begins with `boxplot(` at the end of the line, but inside the right parenthesis insert “, notch=TRUE” (but don't type the quotation marks). Then click Submit while the cursor is still on the line with the command. As Drennan points out, the notches overlap suggesting that the difference in floor area is not great.

We can actually test for a difference in medians by selecting Statistics | Nonparametric tests | Two sample Wilcoxon test. Click OK and look at the results. The p value is .01, well below .05 so we would still reject the null hypothesis of no difference between the medians for the two groups.

For the exercises Drennan introduces a new data set with results of a compositional analysis of obsidian artifacts from Ollantaytambo. The data set is available on the website as Zirconium.RData.

Chapter 13. Comparing Means of More Than Two Samples

Load the ArchaicPts data set (you can find it on the website) and get the Numerical summaries of the Wgt variable using Archaic as the grouping variable. The table has most of the statistics in Table 13.2 except for the variance and the standard error, but you could compute these from the other statistics in the table. Alternatively load the `psych` package and use the `describe.by` function.

Now produce a box-and-dot plot for the three groups. Modify the Rcmdr command to make a notched box-and-dot plot to see how different the medians are. If we type commands, we can produce several other kinds of informative graphs comparing point weight by time period. The best package for producing multiple plots of the kind we need here, is the `lattice` package. The package is already installed, so just load it with `library(lattice)`. The format of lattice commands is a bit different from the graphic commands we have already been using. There is a helpful online manual (. . .) that you should download and read if you want to learn how to modify the graphs we will produce here. First we create boxplots for the three groups. You have already created similar plots, but lattice uses a slightly different style. Also we use the formula method for specifying the graph (which we will use more in coming chapters). Here Wgt is the dependent variable (on the vertical or y-axis) and Archaic period is the independent variable (on the horizontal or x-axis):

```
bwplot(Wgt~Archaic, data=ArchaicPts, lwd=2)
```

The dotplot simply plots points representing the measurements along a line. Things get a bit crowded if we have too much data:

```
dotplot(Wgt~Archaic, data=ArchaicPts, lwd=2)
```

We can plot the box plot differently if we view Wgt as independent and Archaic as a grouping variable so that we separate the single plot into three different plots:

```
bwplot(~Wgt | Archaic, data=ArchaicPts, layout=c(1,3), lwd=2)
```

While the first approach seems more straightforward for the box plot, it would not work for a histogram or a density plot where we have to use this approach:

```
histogram(~Wgt | Archaic, data=ArchaicPts, layout=c(1,3))
```

and for the density plot:

```
densityplot(~Wgt | Archaic, data=ArchaicPts, layout=c(1,3), lwd=2)
```

Computing the analysis of variance is straightforward in Rcmdr. Select Statistics | Means | One way ANOVA. The variables are already selected since the are only two in the data set. Also check Pairwise comparison of means box. Rcmdr will run the `aov()` function to compute the analysis and then print the results in the Output Window and draw a plot. Ignore the plot for now. Scroll up in the Output Window to the line that says

```
AnovaModel.1 <- aov(Wgt ~ Archaic, data=ArchaicPts)
```

The summary is similar to Table 13.3 except that things are labeled differently. “Between Groups” in Table 13.3 is “Archaic” in R and Within Groups” is “Residuals” in R. This labeling emphasizes that the Between Groups variation is explained by Archaic time period and the Within variation is the leftover unexplained variation. You can see the F value and its probability which is very small and is followed by three asterisks indicating a very significant difference between the groups. The output also summarizes the groups by printing the mean and standard deviation of each. That is the end of the ANOVA test. The line that begins `.Pairs` begins a test of the differences between the three groups. The ANOVA tests a null hypothesis of no difference between the means of the groups. That is the mean weight of the Early, Middle, and Late projectile points are all equal to one another. We now know that we can reject that hypothesis. At least one mean is different from the other two. The next test is a multiple comparison analysis to see which means are different. Look at the summary of this test. There are three lines that test different possibilities: Early and Late are the same (have a difference of 0), Middle and Early are the same, and Middle and Late are the same. The probabilities of these hypotheses are presented in the last column and they are all smaller than .05 so we can conclude that all of the groups are different. The multiple comparison test is similar to performing t-tests between all pairs of means, but the test is adjusted to take into account the fact that we are doing three different tests. We want the significance level of the group of tests to be less than .05 so we have to adjust the probabilities of the individual tests. The next table in the Output Window displays the confidence limits of the differences between each pair of means. Those are the numbers that are displayed in the plot window. Look at that plot. If the difference between two means is significant, the horizontal line will not touch the vertical zero line. It does not matter if it is left or right of the zero line, only that is does not cross or touch the line. You can see that none of the lines cross the zero line so we conclude that all

of the groups are different from one another. Also the order of the periods is that Middle Archaic points are heaviest, then Early Archaic points, and Late Archaic points are lightest.

If we are concerned about outliers in the data, we can use a more robust non-parametric test. Select Statistics | Nonparametric tests | Kruskall-Wallis test. This value is also highly significant supporting the conclusion that the groups are different from one another.

The exercises use a data set defined in Table 13.4. You will find that data set on the web page called Neolithic.RData

Chapter 14. Comparing Proportions of Different Samples

Drennan discusses the use of the Chi-squared test to evaluate the significance of differences in the proportions of artifacts between to sites. Table 14.1 has the data already summarized in tabular. We can enter the table directly by selection Statistics | Contingency tables | Enter and analyze two way table. A dialog box opens with sliders to choose the number of rows and columns. The default size is 2 x 2 which is what we want. Enter the column headings as Bowl and Jar and the row labels as San Pablo and San Pedro. Then enter the numbers in the cells. Finally, choose Row percentages, Chi-squared test of independence, Print expected frequencies, and Fisher's exact test. Then click OK.

Rcmdr creates a table called .Table (which is deleted after the analysis is complete). Look at the Output Window to see how the table is created and labeled. The table is printed followed by a table of row percentages, the Chi-squared test, and then the Expected values. Compare the expected values to the actual ones. They are modest and the Chi-square test results confirm that conclusion. Drennan illustrates how to evaluate the significance of a Chi-squared value using Table 14.4. The printed results in the Output Window provide the probability value directly (p = .214).

You can obtain Chi-squared probabilities directly with Rcmdr. Select Distributions | Continuous distributions | Chi-squared distribution | Chi-squared probabilities. Enter 1.5441 as the value, 1 for degrees of freedom, and select Upper tail. Click OK and check the Output Window. To find out what Chi-squared value would have been significant at the .05 probability select Distributions | Continuous distributions | Chi-squared distribution | Chi-squared quartiles. Enter .05 as the probability, 1 for degrees of freedom, and Upper tail. The result is 3.84. Our value of 1.54 is far below that value. The shape of the Chi-squared distribution changes dramatically as the degrees of freedom increases. To see how much, plot distributions for 1, 5, and 15 degrees of freedom.

To get some measures of the strength of the association between sites and ceramics you need to recreate .Table and load the `vcd` package and use the function `assocstats()`. Scroll through the Script Window until you find the line `.Table <- matrix(c(18,12,18,22), 2, 2, byrow=TRUE)`. Now select that line and the next two (the rownames and colnames lines) and click Submit. Now you have the table recreated. Type the following command on a blank line in the Script Window and Submit it:

```
assocstats(.Table)
```

This produces several measures of association including Cramer's V. As Drennan points out, the Chi-squared value can be misleading if the expected cell counts are small. The Chi-squared test in R has

two ways of addressing this possibility. One is a continuity correction. If you search through the Script Window, you will find the line `.Test <- chisq.test(.Table, correct=FALSE)`. To get the continuity correction you simply change FALSE to TRUE. Run the edited command and the following line (`.Test`) which prints the results. You can see that the Chi-squared value is even lower. The other approach is to use Monte Carlo simulation to simulate the p value (much like bootstrapping). To do this leave `correct=FALSE` and insert `simulate.p.value = TRUE`:

```
.Test <- chisq.test(.Table, correct=FALSE, simulate.p.value=TRUE)
.Test
```

For this table, the expected values are not too small and the simulated p value is quite close to the value we calculated originally.

At the end of the chapter, Drennan introduces Fisher's exact test. We computed the value earlier. You will find it in the Output Window labeled Fisher's Exact Test for Count Data. The p value is comparable to those we have already computed.

In the Postscript Drennan describes how to use Chi-square to test a one-way table. The example involves a survey in which sites are located in three different zones and we suspect the number of sites in each zone is just a reflection of the area of that zone surveyed. Table 14.7 has the relevant data. We can create a data set and perform the same analysis with Rcmdr:

```
Survey <- data.frame(Setting=c(rep("Remnant levees", 19), rep("River bottoms",
12), rep("Slopes", 7)))
```

Select the new data set in Rcmdr. Then select Statistics | Summaries | Frequency Distributions. Click OK since there is only variable. A new dialog box appears asking for the probabilities for each group. The default is to assume the same proportion in each group, but that is not what we want. Enter the percentages of area surveyed (the last column of Table 14.7). You can use 28.7, 61.0, 10.3 or .287, .610, .103. If you enter the percentages, R will convert to the probabilities for you. Then click OK again. This Chi-squared value is highly significant. The sites are not distributed according to the amount of each land area surveyed.

Table 14.10 introduces a new data set for the exercises. You will find it on the web site as OpSherds.RData.

Chapter 15. Relating a Measurement Variable to Another Measurement Variable

To illustrate relating one measurement variable to another Drennan provides data on site area and hoes in Table 15.1. You can create that data set with the following command.

```
RSHoes<-structure(list(Area = c(19, 16.4, 15.8, 15.2, 14.2, 14, 13, 12.7, 12,
11.3, 10.9, 9.6, 16.2, 7.2), Hoes = c(15L, 14L, 18L, 15L, 20L, 19L, 16L, 22L, 12L,
22L, 31L, 39L, 23L, 36L)), .Names = c("Area", "Hoes"), class = "data.frame",
row.names = c(NA, -14L))
```

We have not used this command before. I created the R data set and then used the function `dput(RSHoes)` to create a text version of the data. In this case the file is very simple, but it preserves all of the information that R stores in a data frame. If we had categorical variables, it would preserve the ordering of categories of those variables. The help list R-Help asks you to submit your data (or a portion of it if it is large) to the list in this format so that people trying to help you can easily create the data set you are working with.

We start by looking at the data with a scatter plot (Figure 15.1). We will use Rcmdr to produce this scatterplot, but first we will use the plot() function to generate it using typed commands. The plot() function is very basic and very powerful since it has many options and the ability to add layers of information to the plot. Rcmd uses a special scatterplot() function which automatically provides many options. First, type this command:

```
plot(RSHoes$Hoes~RSHoes$Area, las=1, pch=4)
```

This is a pretty basic scatterplot. We are using the formula method to specify the plot, but we could have used `plot(RSHoes$Area, RSHoes$Hoes)` to get the same result. The `las=1` option positions the axis labels (remove it to see how they change if you leave it out). The `pch=4` option selects the X to mark the points. If we want to match the labeling on the axes, we add those options as well.

```
plot(RSHoes$Hoes~RSHoes$Area, xlab="Site Area (ha)", ylab="Number of Hoes", las=1,
pch=4)
```

As you look at the plot, it is clear that the points fall approximately along a line that decreases from left to right indicating that bigger sites have proportionally fewer hoes per 100 artifacts. Drennan discusses how this line could be expressed mathematically and then presents a way of estimating a line that minimizes the squared errors in the dependent variable (the number of hoes). Before using Rcmdr to fit the least squares line, let's use Rcmdr to create the scatterplot. Select the RSHoes data set and then select Graphs | Scatterplot. In the dialog box, select Area as the x-variable and Hoes as the y-variable. Now just click OK to get some of the kinds of information that `scatterplot()` provides. Now look at the plot. Rcmdr has added a number of things to the plot. The plot is gridded and the x and y variables are described by box-and-dot plots so that you can get an idea of how symmetrical they are. The least squares fit is the straight line in green. The red line is a smoothed line to give you an idea of how well the points follow a straight line. The smoothed line works better when there are more data points, but here they suggest that the relationship may not follow a straight line. The smoothed line shows the points decreasing steadily to a site size of about 13 hectares and then flattening out. The dashed red lines show the variability of the points around the smoothed line. The red lines help you to see how well the points follow a straight line. From Rcmdr you can also check to see if transforming the data with logarithms would improve the fit. You can plot either x or y or both on a log scale using the dialog box. In this case, it does not seem to fit the linear model better. Another transformation might do better, but the number of points is small so we should not put to much faith in the smoothed line.

Now we can use Rcmdr to fit the linear model to these data. Select Statistics | Fit models | Linear regression. Select Hoes as the Response variable (aka Dependent variable) and Area as the Explanatory variable (aka Independent variable). Rcmdr computes the results and saves them with consecutive numbers (you have just created RegModel.1). If you look just below the menu bar of Rcmdr, you will now see RegModel.1 listed next to Model:. The results match Drennan's very well (only a difference in

the third decimal place for the slope).

The first line of the output shows you the range in the residuals. This are the differences between the actual observed data and the prediction that the formula for the linear equation makes. You can see that the equation overestimates the number of hoes by 12 (-12.3) on one point and underestimates the number of hoes by 10 (+10.0) once. The next section of the output gives you the coefficients. The y-intercept or a value is listed as (Intercept) and the b value or slope is listed by the independent variable name (in this case Area). Although we are only using a single independent variable, the linear model (lm() function in R) can fit multiple independent variables to a single dependent variable so labeling the slopes by the variable name makes more sense. Each estimate is followed by a standard error, a t-value and a probability. Both probabilities are significantly different from zero. This is not surprising or terribly useful information for the the y-intercept value. If there was no linear relationship between the two variables, the slope would be zero but the intercept would be equal to the mean of the dependent variable which in this case would be 21.57 hoes. The slope estimate, -1.958, is also significantly different from zero and that is useful information. Below the information on the coefficients, the output provides the residual standard error and the multiple r-squared (0.535 which matches Drennan's results). The adjusted R square takes into account the number of independent variables being used so we don't need to look at it with only one variable. The last line provides the F-statistic (13.81)and its probability. The value suggests that we can reject the null hypothesis that there is no linear trend in the data.

The line we have estimated is subject to error. That error is expressed in the standard errors that are listed for each coefficient (7.231 for the intercept and 0.527 for the slope). Figure 15.7 shows the effect of these errors on the “true” relationship between site area and the proportion of hoes.

We can generate Figure 15.7 with a few commands:

```
plot(RSHoes$Hoes~RSHoes$Area, xlab="Site Area (ha)", ylab="Number of Hoes", las=1,
pch=4)
xp<-seq(7, 19, .1)
yp<-predict(RegModel.1,list(Area=xp),int="c")
matlines(xp,yp, lty=c(1,2,2),col="black")
```

To examine the residuals, you can add them to your data set by selecting Models | Add observation statistics to data. Unclick everything but the Fitted values and the Residuals. Click View data set and you will see that you now have two additional columns. You can generate Table 15.2 by typing the name of the data set, `RSHoes`, but if you want to round to two decimal places like the Table 15.2 use

```
round(RSHoes, 2)
```

Also we now have the information we need to produce Figure 15.6 if we want:

```
plot(RSHoes$Hoes~RSHoes$Area, xlab="Site Area (ha)", ylab="Number of Hoes", las=1,
pch=4)
abline(RegModel.1)
matlines(t(cbind(RSHoes$Area, RSHoes$Area)), t(RSHoes[,2:3]), lty=2, col="black")
```

To explore the role of soil productivity in the model, we need to add the soil productivity variable to our data ste:

```
Soil <- c(1200, 950, 1200, 600, 1300, 900, 450, 1000, 350, 750, 1500, 2300, 1650,
1700)
RSHoes <- data.frame(RSHoes[,1:2], Soil, RSHoes[,3:4])
```

We create a new vector, Soil, and then we insert it into the data set between the second and third columns. Also we should simplify the variable names of the two variables that Rcmdr inserted for us. Select Data | Manage variables in active data set | Rename variables. Select fitted.RegModel.1 and residuals.RegModel.1 (hold the Ctrl key down while selecting the variables). Click OK and then enter "Predicted" for the first variable and "Residual" for the second.

Use Graph | Scatterplot to look at the relationship between Soil (x-variable) and Resdual (y-variable). Uncheck all of the options except for the Least squares line and click OK. You can see that the relationship is a strong one and that the points fall closely along the line. Get the correlation coefficient by selecting Statistics | Summaries | Correlation test. Select Residual and Soil and click OK. The correlation coefficient is .923 and the probability is far below .05 so that we can reject the null hypothesis of no relationship (Note: The p value is so small that R uses scientific notation to represent the number. The number 2.523e-06 is 2.523×10^{-6} which is the same as .00000253).

To get the regression statistics select Statistics | Fit models | Linear regression. Residual is the response variable and Soil is the explanatory variable. The new model is named RegModel.2. The coefficients match Drennan's as expected. To put everything together we can specify a multiple regression. Navigate your way to the linear regression dialog. You want to predict Hoes from Area and Soil. Now the coefficients for both variables are in the equation. The coefficients will not match the ones you computed separately because they now account for any correlation between Area and Soil, but you can see that the R-squared is now .966.

Using transformations is straightforward in R. The simplest approach is to use Rcmdr to add transformed variables to the data set and then use them in the regression analysis.

The data in Table 15.4 is available on the web site as Yenang.RData.

Chapter 16. Relating Ranks

To follow the rank correlation example, we need to create the data set in Table 16.1:

```
Soil <- LETTERS[1:17]
Prod <- c(2, 6, 3, 7, 4, 8, 8, 1, 3, 5, 1, 8, 7, 2, 4, 3, 6)
Density <- c(0.26, 1.35, 0.44, 1.26, 0.35, 2.3, 1.76, 0.31, 0.37, 0.78, 0.04,
1.62, 1.34, 0.47, 0.56, 0.48, 0.76)
KPNeolithic <- data.frame(Soil, Prod, Density)
```

You could generate the rank columns, differences and the ties columns with the following R commands:

```
ProdR <- rank(KPNeolithic$Prod)
DensityR <- rank(KPNeolithic$Density)
d <- ProdR - DensityR
d2 <- d*d
Prodt <- sapply(1:17, function(x) sum(KPNeolithic$Prod[x]==KPNeolithic$Prod))
ProdT<- (Prodt^3-Prodt)/12
Densityt <- sapply(1:17, function(x) sum(KPNeolithic$Density[x] ==
KPNeolithic$Density))
DensityT<- (Densityt^3-Densityt)/12
D2 <- sum(d2)
PT <- sum(ProdT)
DT <- sum(DensityT)
n <- length(KPNeolithic$Prod)
N <- (n^3 - n)/12
rs <- (N-PT+N-DT-D2)/(2*sqrt((N-PT)*(N-DT)))
```

But you could also just have Rcmdr computer Spearman's r for you by selecting Statistics | Summaries | Correlation test. Select Density and Prod and specify Spearmans rank correlation. R will generate a warning that because of the ties, it cannot compute exact p values. For this example p is so small that it doesn't make any difference. In fact, if you type the following command which uses the formula when there are no ties, you will see that that ties have had very little impact on the result.

```
1 - (6*D2)/(n^3 - n)
```

R also computes Kendall's tau, another correlation measure for rank data.

The data for the exercise is on the website as Teixeira.RData.

Chapter 17. Sampling a Population with Subgroups.

In this chapter Drennan discusses how to pool estimates from several sampling strata into a single estimate. The example involves site areas recorded from samples in three different landform settings. You could create a data set with the three groups of sites as we have done in previous chapters as an exercise. Here we will just focus on using R to generate the pooled estimate (so that you could use the confidence function we created earlier to produce confidence limits. To do that we only need the summary statistics:

```
N <- c(53, 76, 21)
Mean <- c(2.78, 1.71, .83)
SE <- c(.14, .15, .13)
```

The equations for the pooled mean and the pooled standard error involve weight sums of the means and the standard errors. R is particularly useful for tasks such as these because it processes vectors automatically where other programming languages would require loops. We used this feature extensively in the previous chapter to generate the columns (vectors) needed to compute Spearman's rank correlation. For example to compute the weighted sum of the means, we simply write the

command as if we were talking about single values:

```
N*Mean
sum(N*Mean)
```

The first command produces a vector of three values, one for each sample. The sum of that command is the numerator of the equation for the pooled mean. To get the denominator, we just sum the population sizes:

```
sum(N*Mean)/sum(N)
```

The pooled standard error is only slightly more complicated

```
sqrt(sum(N^2*SE^2))/sum(N)
```

With these values and the degrees of freedom (38) you can use the function we created earlier to compute confidence limits.

Chapter 18. Sampling a Site or Region with Spatial Units

Now we consider cluster sampling and the equations needed to compute proportions, means, and standard errors. The data set consists of sherd counts (cord-marked and total sherds) from ten randomly selected excavation units:

```
x <- c(10, 13, 16, 19, 17, 21, 18, 30, 19, 12)
y <- c(32, 27, 38, 73, 55, 41, 63, 81, 56, 34)
```

The expected proportion is simply the sums of the two vectors divided into one another, n is the number of units and N is 100 and is the sum of y.

```
P <- sum(x)/sum(y)
Y <- sum(y)
N <- 100
n <- length(x)
```

The equation is complicated but you could turn it into a function if you plan to use it regularly:

```
sqrt((1/n)*(sum(((x/y - P)^2*(y*n/Y)^2)/(n - 1))*(1 - n/N)))
```

We can combine these commands into a function that requires only x, y, and the total number of units:

```
SECluster <- function(x, y, N) {
     P <- sum(x)/sum(y)
     Y <- sum(y)
     n <- length(x)
     return(sqrt((1/n)*(sum(((x/y - P)^2*(y*n/Y)^2)/(n - 1))*(1 - n/N))))
}
```

```
SECluster(x, y, 100)
```

Next we turn to estimating population means from cluster samples. The equation is very similar, but it is more of a challenge to get the data in the proper order. We could just take the counts and mean lengths from Table 18.4, but it is more realistic to generate them from Table 18.3. Since we have measurements on each point, the data need to be arranged by point, not by Unit:

```
Unit <- c(7, 7, 7, 18, 29, 29, 31, 31, 31, 37, 37, 56, 72, 72, 72, 72, 72, 83, 87,
91, 91)
Length <- c(15, 19, 23, 17, 18, 23, 18, 18, 27, 18, 19, 24, 20, 21, 26, 28, 29,
16, 28, 25, 26)
length(Unit)
length(Length)
Points <- data.frame(Unit, Length)
```

In the Unit column we repeat the unit number so there is one for each point in that unit. Since we entered each column separately, the `length()` commands check to see if we have the same number of values in each. Then we combine them as Points. This gives us a data set containing one row for each point. The population mean is just `mean(Points$Length)` which is 21.81. Next we need to generate the Y and $\bar{x}$ columns:

```
Xbar <- mean(Points$Length)
y <- tapply(Points$Length, Points$Unit, length)
xbar <- tapply(Points$Length, Points$Unit, mean)
Y <- sum(y)
n <- length(y)
N <- 100
sqrt((1/n)*(sum(((xbar - Xbar)^2*(y*n/Y)^2)/(n - 1))*(1 - n/N)))
```

The final line prints the standard error and matches Drennan's calculation.

The final section of the chapter notes that if we are measuring a characteristic of the sample unit (e.g. density) then we do not need the more complicated cluster sampling equations in this chapter, but instead the simpler equations from Chapter 9.

Chapter 19. Sampling without Finding Anything

In this chapter Drennan considers how we can use the absence of an artifact in a sample to conclude that it is absent or very rare at the site. He summarizes the discussion in Table 19.1 The following function generates a line of the table when given a sample size. It uses a feature of R that we have not discussed before. The ability to add labels with `names()`. Here we put the column labels from Table 19.1 above each confidence level.

```
Nothing<-function(x) {
     tmp <- round(1-c(.999, .995, .99, .98, .95)^x, 3)
     names(tmp) <- c("0.1%", "0.5%", "1.0%", "2.0%", "5.0%")
     return(tmp)
```

```
}
```

Chapter 20. Sampling and Reality

Drennan summarizes his discussion about sampling and its role in archaeological research. The chapter focuses on how we might use the material of the previous chapters to conduct archaeological research and reach conclusions based on different kinds of samples and different ways of viewing the population from which those samples come. The chapter does not introduce new methods or data sets.

Chapter 21. Multivariate Approaches and Variables

Drennan introduces multivariate approaches to archaeological data with a new data set on households at Ixcaquixtla. The data set consists of several kinds of measures, but most of them are ratio scale measurements (percentages, ratios, or counts). There is one rank variable, Energy Invested in Burials, that is coded as 1, 2, or 3). Presence/absence variables are coded as 1 and 0. The data set is available on the website as Housholds.RData and will be used in the next four chapters. Currently the variables are all numeric in the R data set. In addition to representing variables as numbers, R can also represent variables as factors. Factors are categorical variables that are represented by numbers, but numbers provide an index to the name of the category. If we change the Households data set by changing the 0 an 1 coding for Platforms and Mace Heads to “Absent” and “Present” before importing the data set (e.g. changing the numbers to characters in a spreadsheet before importing them), R will convert them to factors by replacing “Absent” with 1 and “Present” with 2. R would not compute the mean of a factor unless you converted it to a numeric value first. But R would let you use the factor to divide the data set into categories.

If you load the data into Rcmdr, you can compute descriptive statistics of all the variables, but you cannot easily get Rcmdr to divide the data set into groups and compute the statistics on each group because Rcmdr uses a dialog box that lists the factors in the dataset. This data set has no factors. If you are typing commands, this generally is not an issue because R will convert the numeric field to a category if it can (and sometimes with surprising results if you are not careful). You can convert MaceHeads and Platform to factors using Data | Manage variables in active data set | Convert numeric variables to factors. Then you can select MaceHeads and Platform and to factors and specify that 0 should be Absent and 1 should be Present. View the modified data set and you will see the 0's and 1's have been replaced with Absent and Present. Now you can compute statistics on subgroups of data defined by these variables. If you want, you can also convert EnergyB, but just leave the 1, 2, 3 coding since we don't have text labels for each category. Then you can break the data into subgroups on this variable as well. Once you have done this use the command str(Households) to see the structure of the data frame. Most of the variables are labeled “num” for numeric, but the ones you created are now labeled as Factor. If you look at the data listing, you will see that the factor levels are coded as 1 and 2 for the Absent/Present factors and 1, 2, and 3 for the Energy factor.

There are no missing data in this dataset, but R uses NA to identify missing values. If you have missing data, you will usually have to tell a function what to do about it. There are several options ranging from aborting the analysis, leaving out cases with missing data, or working around the missing data. There

are also several packages in R providing access to the most recent approaches to dealing with missing data. It is much more common to use these approaches when you are fitting models to your data (such as regression models, than if you are exploring the data looking for potentially significant patterns. Confirming those patterns or hypotheses might then involve the kinds of models that can benefit from imputing missing data.

Chapter 22. Similarities between Cases

Identifying patterns in multivariate datasets often involves reducing the number of dimensions since we can plot two dimensions easily, but when the number of dimensions (variables, columns) exceeds 3 it becomes difficult to impossible. Multivariate methods designed to help us find patterns in our data often use methods to consolidate information in a smaller number of dimensions. One method is to measure the similarity or difference between two cases (rows) or variables (columns).

If the data consist of numeric measurements, Euclidean distance is a standard dissimilarity measure. The computation is straightforward and any statpack will be able to provide it. In R the function `dist()` provides Euclidean distance matrices. Table 22.1 provides measurements on four projectile points:

```
PPoints<- data.frame(Length=c(4.3, 4.5, 4.4, 2.3), Width=c(1.2, 1.4, 1.1, 0.9),
Thickness=c(0.35, 0.55, 0.37, 0.3), Weight=c(75, 80, 80, 75))
dist(PPoints)
```

These commands create the data set and compute the Euclidean distance matrix (Table 22.2). The `dist()` function can compute several other distance measures using the `method=` option. We can standardize the variables with the mean and standard deviation first (Table 22.3) and compute the distances again (Table 22.4):

```
scale(PPoints)
dist(scale(PPoints))
```

If your data consist of presence/absence (dichotomous) variables, there are a number of similarity measures that have been developed. The data in Table 22.6 provide the example:

```
Sherds <- data.frame(Slip=c("Present", "Absent", "Absent", "Present", "Present",
"Present", "Present"), Red.Paint=c("Absent", "Present", "Present", "Absent",
"Present", "Absent", "Absent"), Incising=c("Absent", "Present", "Absent",
"Present", "Absent", "Present",  "Present"), Punctations=c("Absent", "Absent",
"Absent", "Absent", "Present", "Absent", "Absent"), Quartz=c("Absent", "Absent",
"Absent", "Absent", "Absent", "Present", "Present"), Mica=c("Absent", "Absent",
"Present", "Present", "Absent", "Absent", "Absent"))
Sherds
str(Sherds)
```

The str(Sherds) command shows you the structure of the data set. All of the variables are stored as factors with “Absent” stored as 1 and “Present” stored as 2. This is the way the data is presented, in the

table, but the similarity and dissimilarity functions in R want the data in the form of 0 and 1 integers. To get it in that format, we need to create a new dataset:

```
Sherds2 <- data.frame(sapply(Sherds, as.numeric)-1)
```

This command creates a new data set Sherds2 by converting the Absent/Present values to 1/2 and then subtracting 1 to give us 0/1 values. Now we can compute Tables 22.7 and 22.8. There are several different packages that we can use, but `simba` is simple and straightforward and has 56 different similarity measures for binary (dichotomous) data. You will have to install this package:

```
install.packages("simba")
library(simba)
sim(Sherds2, method="simplematching")
sim(Sherds2, method="jaccard")
```

These commands install the package, load it and use the `sim()` function to compute the simple matching coefficient and Jaccard's coefficient (Tables 22.7 and 22.8). Note that these are similarity measures, not distance measures. Make sure you know which one you need for a particular analysis. You can convert either to a distance measure just by inserting 1 – in front of the `sim()` function:

```
1 - sim(Sherds2, method="simplematching")
1 - sim(Sherds2, method="jaccard")
```

When the data have a mixture of dichotomous, categorical, rank, and metric data, Drennan suggests Gower's and Anderberg's coefficients. R has several packages that compute Gower's coefficient, but none that compute Anderberg's. The reason seems to be that Anderber's coefficient does not seem to be regularly used anymore. Gower's coefficient is available in the `cluster` package.

Install package `cluster` and then use the function daisy to compute Gower's coefficient. Package `cluster` has a number of useful functions to conduct cluster analyses so it produces only distance or dissimilarity measures. That means we have to subtract the distance matrix from 1 to get a similarity matrix:

```
install.packages("cluster")
library(cluster)
1-daisy(Households[, -1], metric="gower", type=list(ordratio=3, asymm=c(4, 6)))
```

This command will produce a similarity matrix that matches the one in Table 22.9. It requires a bit of explanation. First, we use Households[, -1] to drop the first column (Household Unit) from the data because it simply identifies each row of data. The function that computes distance matrices is daisy and we want Gower's coefficient. The `type=list()` tells the function what kinds of variables we are using. We list types and their column numbers starting with the second column (since we dropped the first one). So Energy Invested in Burials (EnergyB) is to be treated as an ordinal (rank) variable even though it is a numeric (ratio) variable in the data set. Columns 4 and 6 (Mace Heads and Platform) are treated as asymmetric binary numbers. This means that joint presence is more important than joint absence (see Drennan's discussion of Jacard's coefficient).

Chapter 23. Multidimensional Scaling

Multidimensional scaling takes a distance matrix and tries to fit the inter-point distances to a space that has fewer dimensions than the original data so that it is easier to visualize. To follow Drennan's presentation, we need to load the Households data set and compute a distance matrix using Gower's coefficient (just as we did in the previous chapter):

```
library(cluster)
HHGDist <- daisy(Households[, -1], metric="gower", type=list(ordratio=3,
asymm=c(4, 6)))
```

We do not subtract from 1 because the R functions that perform multidimensional scaling use a distance matrix not a similarity matrix. Drennan illustrates how the stress decreases as we go from fitting a single dimension to 5. We can perform the same analysis and produce the plot in Figure 23.1 as follows:

```
Stress <- sapply(1:5, function(x) isoMDS(HHGDist, k=x)$stress)
plot(1:5, Stress, xlab="Dimensions", type="b")
```

The stress at each iteration is displayed in the Output Window. The function `isoMDS()` computes non-metric multidimensional scaling. It displays stress as a percentage rather than a proportion so the values are 100 times the ones that Drennan has. Once that is taken into account, the values are very close, but R seems to get slightly lower stress for each dimension with its configuration. Looking at the plot we decide that 3 dimensions fits the distance matrix well enough.

There is another way we can compare the results. We can plot the distances that we have computed with Gower's coefficient to the distances between the households as computed by the multidimensional scaling:

```
plot(HHGDist, dist(isoMDS(HHGDist, k=1)$points), main="One Dimension", ylab="MDS
Distance", xlab="Gower's Distance")
```

The points fall along a line, but there is quite a bit of noise. Let's try two through 5 dimensions:

```
plot(HHGDist, dist(isoMDS(HHGDist, k=2)$points), main="Two Dimensions", ylab="MDS
Distance", xlab="Gower's Distance")
plot(HHGDist, dist(isoMDS(HHGDist, k=3)$points), main="Three Dimensions",
ylab="MDS Distance", xlab="Gower's Distance")
plot(HHGDist, dist(isoMDS(HHGDist, k=4)$points), main="Four Dimensions", ylab="MDS
Distance", xlab="Gower's Distance")
plot(HHGDist, dist(isoMDS(HHGDist, k=5)$points), main="Five Dimensions", ylab="MDS
Distance", xlab="Gower's Distance")
```

Two dimensions is pretty good, but three is even better. Adding four and five does not improve the fit very much and adds additional complexity when it comes to interpreting the results.

Lets save the three dimension result so we can examine it more closely:

```
HHScale <- isoMDS(HHGDist, k=3)$points
HHScale
```

Compare our results with Table 23.1. They don't seem very similar. Multidimensional scaling results are not unique. Since we are only interested in the distances between the points there are many ways any given distance can be represented. Scaling differences, rotational differences (spinning the dimensions), and reflection (reversing a dimension) can all represent the same set of differences. To show that the differences are more apparent than real, load the data set HHMDS.RData from the website. The data set contains the values from Table 23.1. First let's see if Drennan's dimensions correlate with our dimensions.

```
cor(HHMDS, HHScale)
```

Drennan's dimensions are labeled Dim1, Dim2, and Dim2 while our dimensions are labeled [,1], [,2], and [,3]. The diagonal of the correlation matrix gives the correlations between Dim1 and [,1], Dim2 and [,2], and Dim3 and [,3]. Each correlation is .99 showing that the differences we see between the two results are mostly due to scaling differences. Notice also that the correlations for the second and third dimensions are negative. In other words, these dimensions are flipped between our results and Drennan's. If we invert our second and third dimensions then our plot will look almost exactly like the ones in the book. We can also add the Households data to create a single data set to use for plotting:

```
Households2<- data.frame(Households, X1= HHScale[,1], X2= -HHScale[,2],
     X3=-HHScale[,3])
```

Figures 23.3 – 23.12 involve plotting two dimensions at a time with circles of various sizes representing another variable. R has a plotting function that can handle this called `symbol()`. Since is a lot of repetition between the commands to create each figure, we will create a simple function to make things easier.

```
PlotMDS <- function(x, artvar, artlabel) {
colvar<-which(colnames(x)==artvar)
oldpar <- par(mfrow=c(1, 3))
bubble<-x[,colvar]/diff(range(x[,colvar]))/100
symbols(x$X2, x$X1, xlab="Dimension 2", ylab="Dimension 1", circles=bubble,
     inches=FALSE, bg="dark gray")
symbols(x$X1, x$X3, main=artlabel, xlab="Dimension 1", ylab="Dimension 3",
     circles=bubble, inches=FALSE, bg="dark gray")
symbols(x$X2, x$X3, xlab="Dimension 2", ylab="Dimension 3", circles=bubble,
     inches=FALSE, bg="dark gray")
par(oldpar)
}
```

The function plots all three panels for a particular variable. We want each figure to be roughly square so we open a plotting window that is long and narrow. The `windows()` function is for Windows computers. The `record=` option saves the plots so you can page back through them. If you are using Mac OS you should look up the `quartz()` function or if Linux, the `X()` function. The other four commands plot a one of the four figures:

```
windows(10, 4, record=TRUE)
PlotMDS(Households2, "PctBowls", "Percent Bowls")
PlotMDS(Households2, "EnergyB", "Energy in Burials")
PlotMDS(Households2, "PctDec", "Percent Decorated Ceramics")
PlotMDS(Households2, "Fauna2Sherd", "Ratio of Fauna to Sherds")
PlotMDS(Households2, "Platform", "Platform")
PlotMDS(Households2, "MaceHeads", "Mace Heads")
PlotMDS(Households2, "Shell2Sherd", "Marine Shell to Sherd Ratio")
PlotMDS(Households2, "PctObsidian", "Percent Obsidian")
PlotMDS(Households2, "PctWasters", "Percent Wasters")
PlotMDS(Households2, "PctDebitage", "Percent Debitage")
```

Finally we can produce the base figures that Drennan annotates with his analysis in Figure 23.13:

```
oldpar <- par(mfrow=c(1, 3))
with(Households2, plot(X2, X1, type="n"))
with(Households2, text(X2, X1, Unit))
with(Households2, plot(X1, X3, type="n"))
with(Households2, text(X1, X3, Unit))
with(Households2, plot(X2, X3, type="n"))
with(Households2, text(X2, X3, Unit))
par(oldpar)
```

Finally, it is possible to use Rcmdr to look at all three dimensions in a window that you can drag to rotate in three dimensions. Select the Households2 data set and then select Graphs | 3d Graph | 3d scatterplot. Select X1 as the Response variable, and X2 and X3 as the Explanatory variables. Unclick Linear least-squares and then click OK. The window will be very small. Use your mouse to drag the lower right corner to make it bigger. Now you can rotate the figure to view it from various angles. Close the window when you are done. If you convert Platform, MaceHeads, EnergyB to factors, you can plot the each group in a different color. If you edit the `scatter3d()` function in the Script Window you can use shades of gray to indicate a fourth variable. First use the command:

```
colors<-gray(1:20/25)
```

To create a variable that contains shades of gray ranging from black to white. We are creating 20 shades, but we divide by 25 so that the `colors[20]` will be very light gray, but not invisible (white). Now add an option to the scatter3d function so that it looks like this:

```
scatter3d(Households2$X2, Households2$X1, Households2$X3, surface=FALSE,
  point.col=colors[rank(-Households2$PctBowls)],
  residuals=TRUE, bg="white", axis.scales=TRUE, grid=TRUE, ellipsoid=FALSE,
  xlab="X2", ylab="X1", zlab="X3")
```

the part in bold is the part you are adding. This changes the color of the point according to the rank of the variable you are studying (in this case PctBowls). We put a minus sign in front of the variable name because R ranks go from lowest (rank = 1 is the lowest value) to the highest, but the gray shades go from darkest to lightest. We want high values to be shaded as dark gray and lower values to be shaded

as light gray. Select the whole command and Submit. You could edit this command to look at the other variables as well.

There is yet another way to look at the results to see what they might tell us about the households using the concepts we learned in Chapter 15 and anticipating some of the concepts of the next chapter. We can look at the correlations between the variables we have for each household and the coordinates that we calculated with multidimensional scaling:

```
cor(Households2[,2:11], Households2[,12:14])
```

Looking down the first column (X1) we can see that PctBowls, PctDec, EnergyB, and Fauna2Sherd are all highly negatively correlated (ranging from -.78 to – 84). As the X1 coordinate increases, these variables decrease. In the second column (X2) MaceHeads, Platform, and Shell2Sherd are highly negatively correlated. In the third column PctWasters and PctObsidian have high negative correlations and PctDebitage has a high positive correlation. As the X3 coordinate increases, PctDebitage increases, but PctWasters and PctObisdian decrease.

You can visualize this relationships using the 3d scatterplot. For example, select PctBowls as the response and then select X1 and X2 as the explanatory variables (chose the two that have the strongest correlations). Leave the Linear least-squares option checked and R will plot a regression plane for the response variable.

Chapter 24. Principal Components Analysis

Principal components finds linear combinations of variables that summarize much of the variability in the original data in a few principal components. It creates a new coordinate system for looking at the data. The results of the analysis are a summary of how much variability each component explains (the eigenvalues), a description of the strength of each variable on each component (the loadings), and how each row of data can be represented by the new coordinate system (the scores). Rcmdr can compute principal components for us but the menu approach has a few drawbacks. Once the results are displayed and the principal component scores are (optionally) added to your data, they are deleted. R has a number of implementations of principal component analysis. One is the traditional eigenvalue analysis, `princomp()` which Rcmdr uses and a second approach using singular value decomposition, `prcomp()`. The second approach is generally more stable, especially if the variables are highly correlated. In many cases, they give comparable results. We'll start with princomp() and then re-run the analysis with `prcomp()` to show they give the same results. When you work with your own data, the general recommendation is to use `prcomp()`. Run the following commands to compute the principal components:

```
PC <- princomp(Households[, 2:11], cor=TRUE)
str(PC)
round(cbind(Eigenvalues=PC$sdev^2, Variance=PC$sdev^2/length(PC$sdev)), 4)
PC$loadings
AdjLoadings <- sweep(PC$loadings, 2, PC$sdev, "*")
round(AdjLoadings[,1:5], 3)
round(AdjLoadings[c(3, 5, 1, 2, 6, 4, 7, 8, 10, 9),1:5], 3)
```

The first command generates the principal components and stores them in the variable PC. The second commands inspects PC to see what is stored in it. It is a list of 7 elements. The standard deviations of the components (PC$sdev). When we square these we get the variances (eignenvalues) that are displayed in Table 24.1. The component loadings (PC$loadings) reflect the contribution of each of the original ten variables to the component. These loadings are standardized so that the sum of the squared loadings equals 1. While this approach is standard in many fields, archaeologists invariably use loadings that are standardized by the variance (eigenvalue) of the component. Then the means and standard deviations of the original variables (PC$center and PC$scale). Then the number of observations and the component scores (PC$scores). The scores give us a way of plotting the original data in a reduced space (fewer than the 10 dimensions represented by the original data). Finally the command we used for the analysis (PC$call).

The next command squares the standard deviations and lists them along with the proportion of the total variance in the original data that each component explains. The output matches Table 21.1. Next we list the principal component loadings. R doesn't print very small loadings so you can see the larger ones. Also it lists the variables in their order in the data set, not the order used in Table 24.2. Further, the loading values do not match. For Comp.1, PctBowls the value is -0.485 not -0.909. The summary at the bottom of the listing indicates that each component has a sum of squares (SS loadings) of 1.0, not the original variance (which for component 1 is 3.511. We need to rescale the loadings and create AdjLoadings which we do in the next command. We print those loadings and the are closer to Table 24.2 but the order of the variables is not the same. Table 24.2 is organized so that for each component, the strongest loadings (whether positive or negative) are grouped together and sorted. For the first component, Energy in Burials has the strongest loading so it comes first. We could create a function to sort the components this way, but we'll take the simpler approach of just telling R in what order to list the components.

Now we are very close to Table 24.2 but there are still differences. The principal component solution is only defined to reflections of the components so two different programs might reverse the positive and negative signs of the loading, but they should do so consistently on each component. We match Table 24.2 on Comp.1 perfectly and the loadings have the same values on the rest of the components, but the patterns of positive and negative loadings do not match. In fact Table 24.2 has only positive loadings on components 2 through 5. This is surely a misprint in the book as having all positive loadings would be extremely unlikely.

R has a function to do varimax rotation. If we rotate the adjusted loadings, round them to 3 decimal places and print in the same order as Table 24.3, you can see that the results match very well which confirms our suspicion that Table 24.2 is misprinted.

In addition to the numerical results, there we can generate graphical summaries as well. A biplot displays both the variables and the observations in a single plot. The variables are displayed as arrows (vectors) originating from the (0, 0) point on the graph. The position of the arrow reflects the component loadings on the two variables plotted. The observations are plotted using the principal component scores.

```
biplot(PC)
biplot(PC, choices=c(1, 3))
```

The first plot clearly shows the clusters of variables that have high loadings on the first and second components. They form a roughly 90 degree angle indicating that they the two clusters are uncorrelated with one another (not a surprise since each principal component must be uncorrelated with all the other components). Longer arrows indicate variables that have more of their variance explained by the components displayed. Shorter arrows (e.g. PctDebitage) are not well explained by the first two components. The households form either a hairpin shape or three clusters depending on how you look at it, but household 14 seems to be different from the others. It has both a platform and mace heads like households 3, 11, and 18 over in the lower left corner, but it also has high values on several of the variables that define the first component. Much less clustering is apparent when we look at components 1 and three although household 14 is still separated from the rest. With 5 components there are 10 different combinations to plot.

You could add the principal component scores to your data set and use them to plot the data just as we did in the last chapter. Rcmdr makes that easy to do if you select Statistics | Dimensional analysis | Principal components analysis. Select the 10 variables (don't include Unit), leave Analyze correlation matrix checked and check Add principal components to data set. Rcmdr will ask how many components to add. Then you can use 3dscatter to look at three components at a time.

Finally, the other function for principal components analysis is `prcomp()`. The function works the same way:

```
PC <- prcomp(Households[, 2:11], retx=FALSE, center=TRUE, scale=TRUE)
str(PC)
round(cbind(Eigenvalues=PC$sdev^2, Variance=PC$sdev^2/length(PC$sdev)), 4)
PC$rotation
AdjLoadings <- sweep(PC$rotation, 2, PC$sdev, "*")
round(AdjLoadings[,1:5], 3)
round(AdjLoadings[c(3, 5, 1, 2, 6, 4, 7, 8, 10, 9),1:5], 3)
```

We specify the principal components analysis a bit differently. The function `prcomp()` does not save the number of observations (it is easily obtained from other information) or the call. Also loadings is called rotation and scores are called x. The results for these data are the same except that the first component is reversed (negative correlations are positive and vice versa).

Chapter 25. Cluster Analysis

Drennan illustrates the basic features of cluster analysis with a simple example that he presents in Table 25.1. The data is presented as a matrix of similarity coefficients for seven cases. The cluster analysis functions in R work entirely with distance (or dissimilarity) matrices so we have to convert this one before we can use it. We simply subtract each value from one and convert it to a object of type dist. If you run the following line it will create a distance matrix that you can use:

```
Example <- a<-structure(c(0.66, 0.05, 0.31, 0.13, 0.88, 0.14, 0.78, 0.96, 0.1,
0.85, 0.24, 0.89, 0.25, 0.63, 0.68, 0.37, 0.04, 0.41, 0.73, 0.57, 0.51), Size =
7L, class = "dist", Diag = FALSE, Upper = FALSE)
```

```
Example
```

Notice that the diagonal of 1's (actually 0's after we subtract them from one) is missing because R knows that an object's distance from itself is zero so it doesn't bother to store the information. Now we can follow Drennan's analysis except that where he is looking for the largest similarity, we are looking for the smallest distance. Looking at the matrix you should be able to follow the steps for single linkage clustering. To let R run that analysis use the following commands:

```
Out<-hclust(Example, method="single")
str(Out)
Out$merge
plot(Out)
plot(as.dendrogram(Out), horiz=TRUE)
```

We run the analysis with `hclust()` and save the results in Out. We look at the structure of Out to see what it contains. The array Out$merge contains the information about how the clustering proceeded. The other variables contain the information needed to plot the dendrogram. We list the contents of Out$merge to see if `hclust` worked the same way that Drennan did. Row one contains a -4 and a -6. Recall that Drennan combined cases 4 and 6 in his first step. The minus signs indicate that both 4 and 6 are individuals not clusters. The next step shows that 1 and 3 are combined and then 2 and 5. In step 4 the cluster defined in step 2 (1, 3) is merged with the cluster defined in step 3 (2, 5). Then 7 is merged with the cluster defined in step 4 (1, 2, 3, 5). Finally the cluster defined in step 1 is merged with the cluster defined in step 5. The commands above produce two dendrograms. The first shows the merging process clearly but it does not resemble the traditional dendrogram that Drennan presents in Figure 25.1. The second command produces a dendrogram that is more similar, although they are pointing in different directions and the axis in Figure 25.1 scales similarities rather than distances. The ordering of the individual cases is irrelevant, it is the merging process that we are interested in. Notice that in comparison to Figure 25.1 our dendrogram flips the positions of case 7 with cases 2 and 5. But the branching is the same: 1 with 3, 2 with 5 then those two groups merge before 7 merges with the other four. The grouping process agrees in the two figures even though the cases are listed in a different order.

We can use the same commands to run through the complete linkage example with one alteration:

```
Out<-hclust(Example, method="complete")
Out$merge
plot(Out)
plot(as.dendrogram(Out), horiz=TRUE)
```

The first three steps are identical to the single linkage example. Then case 7 is added to the cluster defined in step 1 (4, 6). Clusters defined in steps 2 (1, 3) and 3 (2, 5) get merged next. Finally the two remaining clusters are merged.

Comparing the dendrogram to Figure 25.2 indicates an error in the book. The figure shows case 7 merging with cases 1 and 3 instead of 4 and 6 as the text on the previous page indicates. Now let's follow the process for average linkage.

```
Out<-hclust(Example, method="average")
Out$merge
plot(Out)
plot(as.dendrogram(Out), horiz=TRUE)
```

The first three steps are the same, but now case 7 merges with the cluster defined in step 3 (2, 5). Then clusters defined in step 2 (1, 3) and 4 (2, 5, 7) are merged and then the two remaining clusters merge. The dendrograms match Figure 25.3.

To try cluster analysis on the Households data set we need to recreate the distance matrix that we first created in Chapter 22:

```
library(cluster)
HHGDist <- daisy(Households[, -1], metric="gower", type=list(ordratio=3,
asymm=c(4, 6)))
```

Drennan does a single linkage cluster analysis so we'll do the same and add some supplementary information to help evaluate the utility of the result. After plotting the first graph, use the History menu tab on the graph window and select recording so you can page back and forth between the graphs.

```
ClustSL<-hclust(HHGDist, method="single")
plot(19:1, ClustSL$height, xlab="Number of Clusters", ylab="Height", las=1,
type="b")
grid()
plot(as.dendrogram(ClustSL), horiz=TRUE)
abline(v=seq(.05, .25, .05), lty=3, col="gray")
```

The first plot shows the number of clusters against height and here we used the grid() function to add a grid to the plot. We are interested in places where the height jumps steeply. For example going from 19 to 18 clusters, from 12 to 11 clusters, from 7 to 6 clusters, and from 5 to 4. In general we want to identify clusters just before the increase. Overall, the plot is pretty linear and that is not a good sign. The height measure is growing pretty steadily. With 20 households, cluster solutions of 19 and 11 are useless. We should probably focus on either 5 or 7 clusters. The second plot prints the dendrogram and we've added vertical lines along the x axis to help judge the number of clusters at different stages of the clustering. For example, if we cut the tree at a height of .2 we will have seven clusters (the number of horizontal lines that the dotted line crosses), but five of these clusters have a single household and another has only two households. All the rest of the households (13) are in the final cluster.At a height of about .22 we have five clusters with sizes of 1, 1, 2, 3, and 13. This suggests that we will have problems interpreting the results. Single linkage is the simplest clustering method, but it has a tendency to “chain” meaning that single members are progressively added to the same cluster as it grows larger and larger. We might get better results with another method. Although Drennan does not discuss the results, we can try average and complete linkage to see if they give us better results.

```
ClustSL<-hclust(HHGDist, method="average")
plot(19:1, ClustSL$height, xlab="Number of Clusters", ylab="Height", las=1,
type="b")
grid()
plot(as.dendrogram(ClustSL), horiz=TRUE)
```

```
abline(v=seq(.1, .4, .1), lty=3, col="gray")
```

The plot of height by number of clusters is similar to the single linkage one although there is a distinct jump going from 5 to 4 clusters. But that solution still puts most of the households (13 of 20) in a single cluster. Let's try complete linkage.

```
ClustSL<-hclust(HHGDist, method="complete")
plot(19:1, ClustSL$height, xlab="Number of Clusters", ylab="Height", las=1,
type="b")
grid()
plot(as.dendrogram(ClustSL), horiz=TRUE)
abline(v=seq(.1, .7, .1), lty=3, col="gray")
```

Now we have a very distinct jump from 5 to 4 clusters. We will focus on the 5 cluster solution which falls at about height .35. The dendrogram is much more balanced and the 5 cluster solution includes clusters of 2, 2, 3, 5, and 8. Note that we are preferring this solution because it divides the households more evenly, but if the households come from a highly stratified society where there are a few high ranking households and many lowly ranking households, the unbalanced cluster solutions may be more realistic. We'll use this cluster solution to see if it seems to make sense. Keep in mind that cluster analysis always returns a solution. We should always try to use information that is not included in the analysis itself to verify that the solution is a meaningful or useful one. On the website you will find a data set called Households3. It includes all of the data in Households, plus the multidimensional scaling coefficients and the principal component scores. We'll add the cluster membership to that data set:

```
Clust5 <- cutree(ClustSL, k=5)
Households3 <- data.frame(Households3, Clust5)
Households3$Clust5 <- as.factor(Households3$Clust5)
```

Now you can use Rcmdr to generate numerical summaries by cluster to see how the clusters vary in terms of the 10 variables that were used to define. You can also use scatterplot to plot the first two multidimensional scaling dimensions and the first two principle components by Cluster5.

The last analysis of the chapter is a cluster analysis of the variables from the Households data set using the correlation matrix as the input. We will cluster the variables using the correlation matrix and single linkage clustering.

```
HVDist<-as.dist(1-cor(Households[,2:11]))
VarClus <- hclust(HVDist, method="single")
plot(9:1, VarClus$height, xlab="Number of Clusters", ylab="Height", las=1,
type="b")
grid()
oldpar <- par(mar=c(5.1, 4.1, 4.1, 5.1))
plot(as.dendrogram(VarClus), horiz=TRUE, )
abline(v=seq(.1, .7, .1), lty=3, col="gray")
par(oldpar)
```

Note that we subtract the correlation coefficients from 1 to convert them from a singularity measure to a distance measure. This makes a +1 correlation a 0, a 0 correlation a 1, and a -1 correlation a 0. As Drennan points out, variables with a high negative correlation could be seen as very similar so it may

be preferable to use the absolute value of the correlation to make +1 and -1 correlations the same (highly similar). The clustering plot does not show sharp breaks except at 3 clusters. To produce the dendrogram we need to increase the right side margin space to make room for the variable names. Then we plot the dendrogram and add the gridlines. The 3cluster solution has the group of variables that were linked in the principle components analysis clearly identified. Percent debitage is alone and the rest of the variables form the third group.