Stat 290: Lab 2

Introduction to S-Plus

Lab Objectives

1. To introduce basic S-Plus commands

Assignment

There is nothing to be turned in for this lab.

S-Plus

Getting the Data

From the course datasets web page (`http://www.stat.duke.edu/courses/Fall01/sta290/datasets.html`), download the datasets on VA hospital quality for 1992 and 1993 (they are under the first bullet). If you look at these files, you will see that they contain header information. Read the header for information on the data, but note that you need to remove the header lines (e.g. in emacs) before you can read them into S.

Manipulating Objects

Start up S either within Emacs (`M-x S`) or at the shell (`Splus`). If you are running S-Plus 5 and want to work in a subdirectory, make sure that you have run `Splus CHAPTER` before starting S.

Read the 1992 data into an S object called `x` with

```
x _ read.table(file="va92.orig.dat")
```

Note that the underscore is the assignment character, so please remember to never use an underscore as part of an object name. However, periods are legal in object names (e.g., you can have an object named x.hat, but not x_hat). Also, the letters `c`, `q`, `s`, and `t` are all built-in commands, so S will be very unhappy if you try to name an object with one of these letters (but it will let you do it, and then it may cause all sorts of problems later on).

Let's look at the data. Type in `x` and hit enter. S will show you the whole dataset, but it will scroll off the screen. The data are stored in a matrix-like object called a *dataframe*. To view the value of the second row, third column, type in `x[2,3]`. To see the first ten rows, use `x[1:10,]`. By leaving out the column argument, you tell S to use all columns.

The colon signifies a sequence. `dim(x)` will tell you the dimensions of `x`, number of rows first, then number of columns.

We don't need all of the columns, but for now we are just interested in the annual counts (failures and total counts) by hospital, so we will save these into separate vectors:

```
y92<-x[,1]; n92<-x[,3]
```

Note that you can use a semi-colon to separate commands on the same line. Now read in the 1993 data. Since we don't need the rest of the 1992 dataset anymore, we can overwrite the object `x`:

```
x _ read.table(file="va93.orig.dat")
y93<-x[,1]; n93<-x[,3]
```

You can check on what objects you have created with `ls()`, which is just like the unix command, except that all function calls in S must put the arguments in parentheses after the name of the function. Since `ls()` requires no arguments, you must put empty parentheses after its name. We don't need the object `x` anymore, so you can delete it with

```
rm(x)
```

Note that `rm()` is also a function call, but this time you put in the parentheses the name of the S object that you with to delete.

Graphing

Open a graphics window with `motif()`. Normally this will be the first thing you want to do after starting S. The `plot` command is S is very versatile. `plot(y92)` will plot the number of failures by hospital index number (1 to 159). `plot(n92,n93)` plots the number of total cases by hospital in 1992 and 1993. You can make multiple plots on the same screen with `par(mfrow=c(i,j))` (i rows, j columns):

```
par(mfrow=c(2,2))
plot(y92); plot(n92)
plot(n92,n93)
abline(0,1)
par(mfrow=c(1,1))
```

The `abline` command adds a line to the plot, in this case, one with intercept zero and slope one, so that you can compare the plotted points to see how much they differ from the line y=x. The last `par` command resets the display to show one graph at a time.

What we are really interested in the fraction of cases that fail to return. S will do simple arithmetic, element-wise arithmetic on vectors and matrices, and matrix arithmetic. The default for vectors and matrices is element-wise:

```
f92_y92/n92; f93_y93/n93
```

When we plot the fractions, we want the scale to be from zero to one, since that is the possible range of fractions, so we can use the `ylim` parameter:

```
plot(f92,ylim=c(0,1))
```

The notation `c(0,1)` means a vector whose elements are zero and one. The function `c` (think of as concatenate, combine, or column), takes any number of arguments (separated by commas) and concatenates them into a vector.

Exploratory Data Analysis

The first stage of any data analysis is EDA (exploratory data analysis). Graphing is a good start to EDA, but we also need some numerical summaries. `range(f92)` gives the range of the data, `mean(f92)` and `median(f92)` give the mean and median. `summary(f92)` is a useful command that combines these, as well as giving the first and third quartiles. To get arbitrary quantiles, you can use the `quantile` command, as in `quantile(f92,prob=c(0.05,.5,0.95))`.

Histograms are useful for looking at the data, although the number of bins can have a big effect on the look of the histogram:

```
par(mfrow=c(2,2))
hist(f92)
hist(f92,nclass=15)
hist(f92,nclass=30,prob=T)
plot(f92,f93,xlim=c(0,1),ylim=c(0,1))
abline(0,1)
```

In the last scatterplot, you may notice two observations in the lower right corner, far away from the y=x line. These are two hospitals with high failure rates in 1992 that did much better in 1993. To see which they are, you can use the `identify` command. Type in `identify(f92,f93)` (you need to give `identify` the same x and y arguments that you gave to the `plot` command) and left click on these two points (and any other interesting ones). When you are done, middle click anywhere in the graphics window to terminate the identify mode. (Note that if you had used a `par` command after the last plot, identify won't work.) You should see that the two marked observations are numbers 100 and 102. To see what the data values are, look at `f92[100]; f93[100]; f92[102]; f93[102]`.

Binomial Distribution Tools

S has a number of built-in functions for various standard probability distributions. For the binomial distribution, `rbinom(m, n, pr)` draws m random samples from a binomial(n, pr), `dbinom(x, n, pr)` evaluates the probability mass function at x, `pbinom(q, n, pr)` gives the cdf at quantile q, and `qbinom(p, n, pr)` gives the quantile with probability p (i.e., it is the inverse-cdf function).

Recall that the mean proportion of the VA data is about 0.44. An ad-hoc way of comparing the distribution of the VA data to a binomial (i.e., to see if it would be plausible that the VA data are a random sample from a binomial distribution) is to compare the quantiles of the VA data to the quantiles of a binomial. To get the .05 and .95 quantiles of a binomial(400,.44):

```
qbinom( c(0.05,0.95), 400, 0.44)
```

For the proportions, we need to divide by n: `qbinom( c(0.05,0.95), 400, 0.44)/400`. But the VA data have a different n for each hospital, so we need to account for this. Instead of getting the quantiles directly, we will get a random sample of binomials of appropriate sizes, and compare those quantiles. This is a Monte Carlo experiement. We can repeat it several times to get an idea of whether we think it is reasonble that the VA data come from binomial distributions.

```
bin92_rbinom(159,n92,rep(0.44,159))/n92
quantile(bin92,prob=c(0.05,.5,0.95))
```

Note that you can feed `rbinom` a vector of sizes (n92). When you do this, you also have to give it a vector of probabilities. Since we want all the probabilities to be the same, we can use the `rep` command to repeat the value 0.44 for 159 times. Afterwards, we divide by the respective total counts to transform to proportions. Repeat the generation of random binomials and quantiles several times to see how much it varies between random samples. Notice that the 1992 VA data is considerably more dispersed than would normally be observed by random chance from a binomial model. How about the 1993 data?

We can plot the pmf of a binomial, and compare it to a histogram of many random draws:

```
par(mfrow=c(2,1))
x_0:100
plot(dbinom(x,100,.4))
hist(rbinom(10000,100,0.4), prob=T, nclass=20, xlim=range(x))
par(mfrow=c(1,1))
```

Note that in `hist`, the argument `prob=T` tells `hist` to use a probability scale on the y-axis instead of counts. `xlim=range(x)` forces the limits on the x-axis to be the range of `x`.

These random variable tools are available for a number of different distributions, e.g., rnorm, rgamma, rt, etc.

Sorting

The function `sort` will sort a vector. `order` gives the indices in order of ascending size of the values in the vector. Thus `sort(n92)` is equivalent to `n92[order(n92)]`. If you look at `order(n92)[1]`, you will see that hospital 97 is the smallest.

Finishing Up

There is no assignment for this lab. When you are done in S, the command to quit is `q()`. Make sure you quit S before logging off, or you may spawn a runaway process!