# Stat 290: Lab 2

## Introduction to R/S-Plus

### Lab Objectives

1. To introduce basic R/S commands
2. Exploratory Data Tools

### Assignment

Work through the example on your own and fill in numerical answers and graphs. Nothing need to be turned in. I will describe all commands using R, but everything will work with S-Plus except where noted.

### Background - VA Performance Monitors

There are 80 VA performance monitors which are used to assess changes in quality care and efficiency, resource allocation, operational cost effectiveness, etc. The outcomes in the monitor areas represent $y$ annual numbers of individuals who failed to return for an outpatient visit within 30 days of discharge, out of the total number of annual discharges $n$. Monitor 21 here is in the Substance Abuse Psychiatric care. High non-return rates are indicative of poor "quality" in these specific care areas. This subsample of a much larger data sets provides information on years 1992 and 1993 for the 159 hospitals in the VA system.

The data also include $x$, a measure of the VA's case-mix adjustment factor that, for each hospital in each year, is supposed to be a predictor of the fraction of returns. This plays the role of a covariate in models for the outcomes.

Our first step will be exploratory data analysis.

### Getting Data

From the course datasets web page (`http://www.stat.duke.edu/courses/Fall02/sta290/datasets.html`), download the datasets on VA hospital quality for 1992 and 1993 (they are under the first bullet) to your sta290 directory. If you look at these files, you will see that the first several lines contain information about the data set. The lines with a # at the beginning are comments, which R will ignore. The line with `y x n` is a header line which contains variable names.

### Starting R

Start up R either within emacs (`M-x R`) or at the shell (`R`). *If you are running S and want to work in a subdirectory, make sure that you have run* `Splus CHAPTER` *from Unix before starting S. To run Splus on the command line enter* `Splus -e` *or from within emacs* `M-x S`. If you are running R/S from within emacs you will need to specify a starting directory. The directory where you start emacs is the default, so edit if needed, then hit return. You should see information about the program in the `*R*` buffer, and finally a prompt `>` .

## Reading in Data

Read the 1992 data into an object called `va92`, by entering the following after the prompt `>`

```
va92 <- read.table(file="va92.orig.dat", header=T)
```

The assignment character is `<-` or equivalently the underscore _, so, please remember to never use these as part of an object name. However, periods are legal in object names (e.g., you can have an object named y.hat, but not y_hat). The `header=T` option indicates that the first row of the file contains variable names. By default it is False (F) so include this only when there is a header line. The command `names` returns the names of the variables in the dataframe,

```
names(va92)
```

Let's look at the data. Type in `va92` and hit enter. R will show you the whole dataset, but it will likely 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 `va92[2,3]`. To see the first ten rows, use `va92[1:10,]`. By leaving out the column argument, you tell S to use all columns. The colon signifies a sequence. `dim(va92)` will tell you the dimensions of `va92`, number of rows first, then number of columns. Columns in a dataframe can also be accessed by their names, `va92$x` is equivalent to `va92[,2]`.

Now read in the 1993 data.

```
va93 _ read.table(file="va93.orig.dat", header=T)
```

You can check on what objects you have created with `ls()`, which is just like the unix command, except that all function calls in R 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. Just as `rm` removes objects in Unix, `rm(obj1, obj2)` will remove obj1, obj2 in R.

## Graphics

While older versions of R and S required that you open a graphics window by commands such as `X11()` (R and S) or `motif()` (S only), S-Plus 6 and R will open a graphics window automatically when a plot or graphics command is issued. The `plot` command is very versatile. `plot(va92$y)` (with a single argument) will plot the number of failures by case index number (1 to 159). `plot(va92$n,va93$n)` 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(va92$y, ylab= "Number of Non-Returns")
plot(va93$y, ylab= "Number of Non-Returns")
plot(va92$y, va93$y, xlab= "Number of Non-Returns (92)",
     ylab="Number of Non-Returns (93)")
abline(0,1)
plot(va92$n,va93$n,  xlab= "Number of Discharges (92)",
    ylab="Number of Discharges (93)")
abline(0,1)
par(mfrow=c(1,1))
```

The `abline` command adds a line to the plot, in this case, 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. The xlab and ylab options allow you to override the default text for the axes labels.

What we are really interested in the fraction of cases that fail to return. R will do simple arithmetic, element-wise arithmetic on vectors and matrices, and matrix arithmetic. The default for vectors and matrices is element-wise. We can assign the results to a new object, or add it to our dataframes. To not clutter up the directory, and since this is not just a temporary calculation, we will store it in the dataframe:

```
va92$f <- va92$y/va92$n; va92$f <- va93$y/va93$n
```

the semi-colon allows us to enter two commands on the same line. 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(va92$f,ylab="Fraction of Non-Returns (92)", 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. Besides scatter plots, 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(va92$f, main="Histogram of Fraction of Non-Returns 1992",
     xlab="Fraction"))
hist(va92$f,nclass=15,  main="Histogram of Fraction of Non-Returns 1992",
     xlab="Fraction"))
hist(va92$f,nclass=30,prob=T,
     main="Histogram of Fraction of Non-Returns 1992", xlab="Fraction"))
plot(va92$f,va93$f,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(va92`$f, va93$`f)` (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 `va92[c(100,102),]; va92[c(100,102),]`. You could assign the output of identify to a variable, say `different`

```
different <-  identify(va92$f,va93$f)
va92[different,]
va93[differnet,]
```

which can then be used to extract the rows of the dataframes with the cases found by identify.

Side-by-side boxplots are another way to compare to distributions:

```
boxplot(va92$f, va93$f,  names=c("1992", "1993"))
title("Fraction of Non-Returning Cases")
```

and may be useful for identifying potential outliers. What interpretations can we draw from the boxplot?

To save plots in a postscript file, we need to first shutdown any other graphics devices using the dev.off() function. Then issue the postscript command with a filename (and other options)

```
dev.off()
postscript("vaplots.ps")
par(mfrow=c(1,1))
boxplot(va92$f, va93$f,  names=c("1992", "1993"))
title("Fraction of Non-Returning Cases")
dev.off()
```

followed by any plotting commands. The second dev.off() command says to close down the graphics device. See `help(postscript)` for more options.

### Numerical Summaries

Numerical summaries of quantitative variables are also useful. `range(x)` gives the range of the x, `mean(x)` and `median(x)` give the mean and median, respectively `summary(x)` 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(x,prob=c(0.05,.5,0.95))`. If summary is given a dataframe as an argument, summary returns the summary statistics for each variable.

```
summary(va92)
summary(va93)
```

### Binomial Distribution Functions

R and S have 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 proportion of non-returns in the VA92 data is about 0.42. 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) with a common $p$ is to compare the quantiles of the VA data to the quantiles of a binomial. Why are we not using the mean of the fractions? To get the 0.05 and 0.95 quantiles of a binomial(400,0.42):

```
qbinom( c(0.05,0.95), 400, 0.42)
```

For the proportions, we need to divide by $n$: `qbinom( c(0.05,0.95), 400, 0.42)/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,va92$n,rep(0.42,159))/va92$n
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 0.42 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. Are the 1992 VA data 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(1,1))
x <- 0:100
hist(rbinom(10000,100,0.42), prob=T, nclass=50, xlim=range(x))
lines(dbinom(x,100,0.42))
```

Note that in `hist`, the argument `prob=T` tells `hist` to use a density scale on the $y$-axis instead of frequency. `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, rbeta, rpoisson, etc.

## Finishing Up

When you are done in R/S, the command to quit is `q()`. Make sure you quit before exiting from emacs or logging off, or you may spawn a runaway process which hangs around and gobbles up memory and computer cycles!