A Refresher on OLS; An Introduction to R; Interpreting Coefficients; Regression for Dichotomous Variables

November 19, 2011

1 An Introduction to R

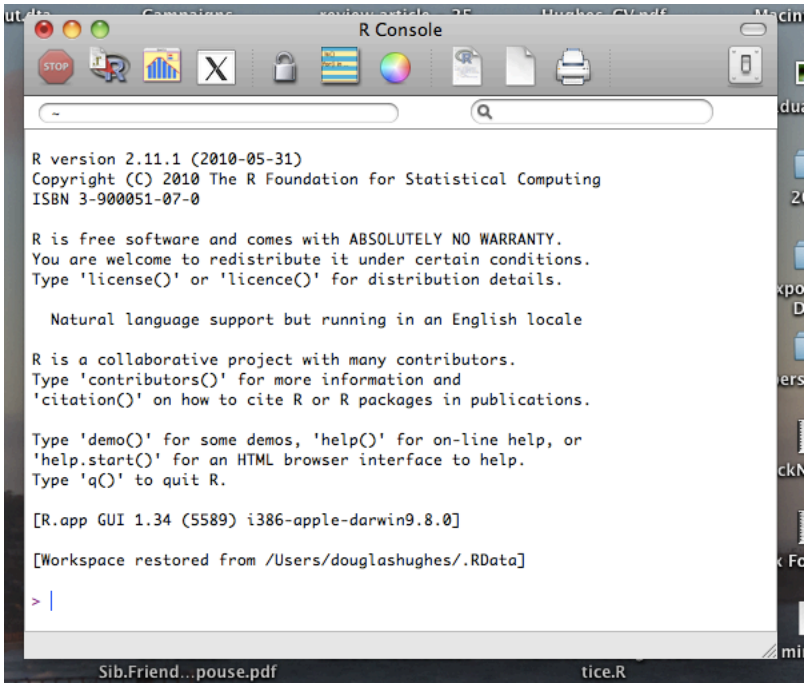
1.1 What *is* it?

R is one of two main statistics languages that Political Scientists use. The other major language in Political Science is STATA. Other programming languages include SPSS, which you used in Poli 30 and is used a lot in psychology and medicine; SAS which used to be used frequently in Business and Government, although it obsolescing; MATLAB which is used in Engineering and Physics.

I'm introducing R to you for two reasons: 1.) It is free; and, 2.) It is the package that I am most familiar with. Some people think that one is better than the other; those sorts of arguments aren't productive. Remember, your goal is to discover new things and make money. Having arguments about technology doesn't frequently accomplish either of those objectives.

1.2 Getting R

To download the program, go to this link. Download and install it just as you would iTunes or any other program. When you fire it up, you will be met with this screen.



This is the launch screen from which we will type in all of our commands.

1.3 A Command Line Language

R is a command-line language, which is different than SPSS that you used in POLI 30. Instead of navigating through a series of drop-down menus and toggle-boxes as in SPSS, in R you type all of your commands at the *command line* next to the > at the bottom of the window.

For example, if you want to add 2 & 3, type:

```
> 2 + 3
```

and press enter. When you press enter, the monkeys inside the program spring into life and pull levers. R returns to you the answer:

```
[1] 5
```

This is the basis for the program. You input "stuff," R figures it out, and returns you the answers. Try it yourself.

```
> cos(pi)
[1] -1

> sqrt(144)
[1] 12

> 4^2
[1] 16
```

While it can be tough at first to remember all the things you might want R to do, the program is pretty good at helping you out, if you ask.

```
> help("sin")
> ?sin
```

1.4 Simple Manipulation; Numbers and Vectors

1.4.1 Vectors and assignment

R operates on named *data structures*.[1] The simplest such structure is the numeric vector, which is a single entity consisting of an ordered collection of numbers. To set up a vector named x, say, consisting of five numbers, namely 10.4, 5.6, 3.1, 6.4 and 21.7, use the R command

```
> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
> x
[1] 10.4  5.6  3.1  6.4 21.7
```

Notice that the assignment operator (`<-`), which consists of the two characters `<` (less than) and `-` (minus) occurring strictly side-by-side and it points to the object receiving the value of the expression.

Notice what is happening in the assignment. The object that is getting pointed at is being made, and then stored it the computer's memory. The things that are on the "butt-end" of the assignment operator are being stored inside the object in memory. So when we say

```
> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
```

we are effectively creating something called `x` which is stored in the computer's memory that has the values `10.4 5.6 3.1 6.4 21.7`. Let's make a few more objects, one called `cats`, that has the names of all my cats, and another called `cat.ages` that has how old they are

```
y <- c("sally", "suzie", "maxwell", "robert", "stanley")
cat.ages <- c(2,5,17,25, 8)
```

1.4.2 Vector Arithmetic

Vectors can be used in arithmetic expressions, in which case the operations are performed element by element. Vectors occurring in the same expression need not all be of the same length. If they are not, the value of the expression is a vector with the same length as the longest vector which occurs in the expression. Shorter vectors in the expression are recycled as often as need be (perhaps fractionally) until they match the length of the longest vector. In particular a constant is simply repeated. So with the above assignments the command

[1] This is from The R Introduction

```
> v <- 2*x + cat.ages + 1
> foo <- c(x,y,cat.ages)
> foo1 <- data.frame(x = x, names = y, cat.ages = cat.ages)
```

generates a new vector `v` of length 11 constructed by adding together, element by element, `2*x` repeated `2.2` times, `y` repeated just once, and 1 repeated 11 times. The elementary arithmetic operators are the usual `+, -, *, /` and `^` for raising to a power.

In addition all of the common arithmetic functions are available. `log, exp, sin, cos, tan, sqrt`, and so on, all have their usual meaning. `max` and `min` select the largest and smallest elements of a vector respectively. range is a function whose value is a vector of length two, namely `c(min(x), max(x))`. `length(x)` is the number of elements in `x`, `sum(x)` gives the total of the elements in `x`, and `prod(x)` their product.

Three statistical functions are `mean(x)` which calculates the sample mean, var(x) which gives the sample variance, and `sd(x)` which gives the sample standard deviation.

1.4.3 Loading Data

For our purposes, we are interested in loading data, running a simple regression, and analyzing the output. Let's make up some data. In the next few lines I'm going to create an object called `dat` that has three columns. The first column will be called `State`, the second `Income`, and the third `Education`.

```
dat <- data.frame("State" = 1:50)
dat[,"Unemployment"] <- rnorm(50, mean = dat[,1])
dat[,"Education"] <- rnorm(50, mean = 50-dat[,1])

dat
```

Let's also load some of the data that is built into R.

```
> data(anscombe)
> anscombe
> data(faithful)
> faithful
```

Now we have some objects stored in R's memory – The first is called `dat` and has the data that we made up about States' education and income; the second is called `anscombe` and the third is called `faithful`.

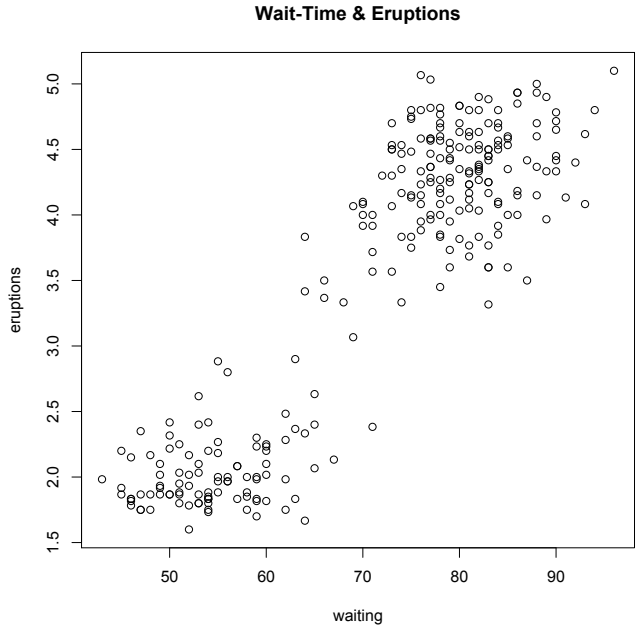
Initially, you may find it *excruciatingly* difficult to get data into any stats program in a way that the program understands. Seriously, this is going to be frustrating. Once you have the data loaded though, things get better.

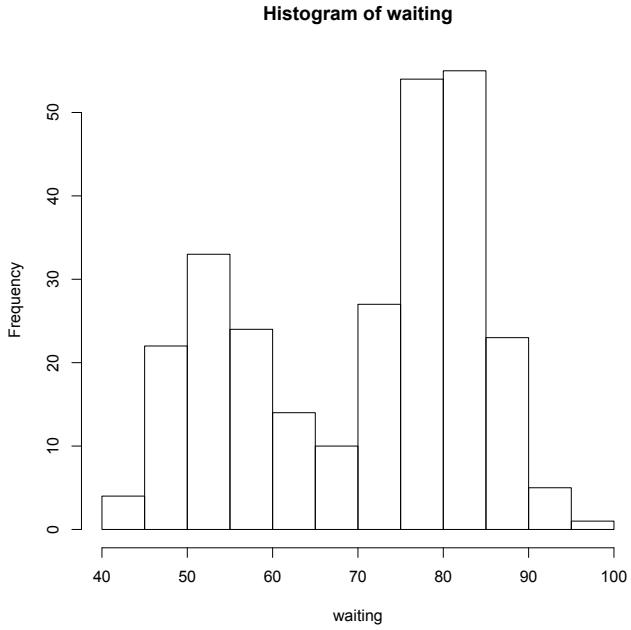
1.4.4 Simple Plots

Typically, the first thing you want to do is to look at the data in a scatter plot. To do this, tell R to plot what you want:

```
> attach(faithful)
> plot(x = waiting, y = eruptions, main = "Wait-Time & Eruption Strength")
> histogram(waiting)
```

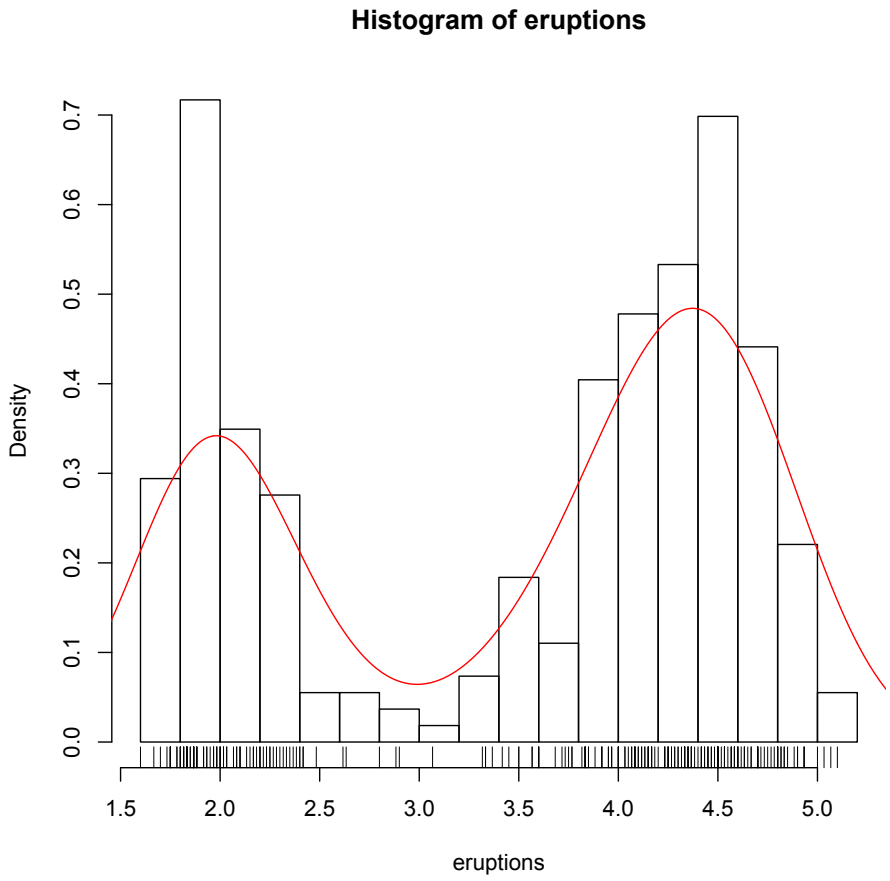
R will present you with a scatter plot and a histogram. Cool! Looking at the scatter plot, there seems to be a positive relationship between the amount of time between eruptions of Old Faithful and the strength of eruptions. Looking at the histogram of wait time between eruptions, the wait time seems to be distributed into two types – long wait times, and short wait times.





We can also make more complicated, and pretty plots for the histogram.

```
> histogram(eruptions, breaks = 20, prob = TRUE)
> lines(density(eruptions), col = "red")
> rug(waiting)
```



1.4.5 OLS Regression

We are interested in exactly what is the relationship between time between eruptions and the strength of the eruption, so we do some **Ordinary Least-Squares Regression**. In R, the call to fit an OLS model is `lm()` – which stands for `Linear Model`. To set up a linear model we place the IV and DV inside the `lm()` function call. The first term is the DV which is separated from the V by a tilde (∼).

We want to store the results of this regression in R's memory, so just like when we made a the simple vector of numbers, we are going to use the assignment vector (`<-`) to create an object and place the results of the regression into that object.

```
> out <- lm(eruptions ~ waiting, data = faithful)
> out

Call:
lm(formula = eruptions ~ waiting, data = faithful)

Coefficients:
(Intercept)      waiting
   -1.87402      0.07563
```

Hmm...that result wasn't very informative. To get a more complete result reported to us, we need to ask for a summary of the regression.

```
> summary(out)
Call:
lm(formula = eruptions ~ waiting, data = faithful)

Residuals:
     Min       1Q   Median       3Q      Max
-1.29917 -0.37689  0.03508  0.34909  1.19329

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.874016   0.160143  -11.70   <2e-16 ***
waiting      0.075628   0.002219   34.09   <2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.4965 on 270 degrees of freedom
Multiple R-squared: 0.8115,Adjusted R-squared: 0.8108
F-statistic:  1162 on 1 and 270 DF,  p-value: < 2.2e-16
```

That looks pretty good! But what have we got up there?

The `Call:` line tells us what regression we ran. We ran `lm(formula = eruptions ~ waiting, data = faithful)`.

`Estimate` is the estimate of the regression coefficients. The estimated intercept if -1.8, and the estimated `waiting` coefficient is 0.07

`Std.Error` is the standard error of the regression coefficient. This is one of the **MAJOR** benefits of using a stats-program, because calculating standard errors of regression coefficients by hand is a monumental pain in the butt. Just like inference for hypothesis tests that you covered in Poli 30, the standard error is critical in determining if the coefficient you found could have happened by chance, or if it is actually different from zero.

`t value` is the regression coefficient divided by the standard error. In a hypothesis-test framework, we would compare this t-value to the critical value given our confidence level and degrees of freedom. (Recall, if we want a confidence level of 95% and have a lot of observations this critical value is 1.96.)

`Pr(>|t|)` is the p-value, or likelihood that a regression coefficient of that size occurred by random error. For both the intercept and the coefficient on `waiting` it is very unlikely that coefficients of this magnitude occurred by chance.