Volpe R Course: Day 1

Course webpage `http://bit.ly/volpeR`

Don Fisher, Dan Flynn, Jessie Yang

9/6/2017

Introduction to R

R is a free, powerful programming environment which lets you carry out a huge range of statistical tests, including ones you invent. R is the major tool used by statisticians, economists, and data scientists in all fields.

- Free: Open source, freely-distributed. This means users are constantly improving it and writing new sets of functions to make it more useful
- Powerful: The power of R is that you can write your own functions and customize the analyses, and then you can automate the analysis by re-using your code.
- Programming environment: R is command-line driven, and has it's own syntax, similar to Unix. This is less intimidating than it sounds!

Goals of this workshop include introducing R to Volpe staff, with a focus on learning the statistical software for basic data analysis, visualization, and report preparation. This course does not assume previous knowledge of statistics or programming.

What is R?

R is an open-source programming environment developed and maintained by statisticians. People are so fond of R because users are constantly writing new sets of functions for it, called packages.

The program (and all the current packages) can be downloaded from `http://cran.r-project.org`. For your own computers, you should have R installed already; if not, can download and install R by following the appropriate steps:

1. Click on Base, Download R 3.3.3 for Windows
2. Run the installation set up, saying yes to all the defaults.

Workflow in R

- Work in a **script**
- Run script to produce output
- Save output: .csv, .jpeg, .pdf, Word and Excel
- Share scripts and output
- Reproducible research!

Workflow in R

- RStudio provides a platform-independent interface and several excellent features
- Freely available on `https://www.rstudio.com/products/rstudio/download/`

Starting up R

You'll see the R console, with a bunch of official information. What you can get out of this: the program offers you a few hints on how to get started, like typing in `help()`. Try it. You'll see the window with the help documentation; not very useful now, because you don't know what to look for, but it will come in handy later.

Basic concepts

R is a command-line driven programming environment. There are two places you an enter commands:

- Console: For quick entry of single commands
- Script: For everything else! Including complex commands for preparing, analyzing, and reporting data, these are editable and sharable files ending in `.R`

Simple calculations

This section describes how to use R for simple calculations, like arithmetic and entering a series of numbers. These steps are important, because they introduce by example some of the fundamental ways that R works. First we will enter commands in the console.

```r
2 + 2
```

```
## [1] 4
```

Fortunately, that is not all R can do! R is designed to work with vectors, like columns of data in Excel.

Simple calculations on vectors

- Define variables by creating vectors using the concatenate command, `c()`.
- We might want to do this in real life when we are entering small amounts of data or testing an analysis on created data.

Let's try it by making `x` and `y` variables. The simplest way to enter numbers is simply to type `c(#, #, #)`. Try the following:

```
c(1, 2, 3)
```

```
## [1] 1 2 3
```

```
x <- c(1, 2, 3)
x
```

```
## [1] 1 2 3
```

```
print(x) # Forces R to print the output on the console
```

```
## [1] 1 2 3
```

Simple calculations on vectors

Another way to enter in a series of numbers is to use the colon to enter a continuous set of integers:

```
x <- c(1:10, 13)
```

Note: we just erased the old variable x and replaced it with a new one. Be careful when using the <- symbol that you don't erase something you wanted to keep.

Now we'll create another variable based on this one:

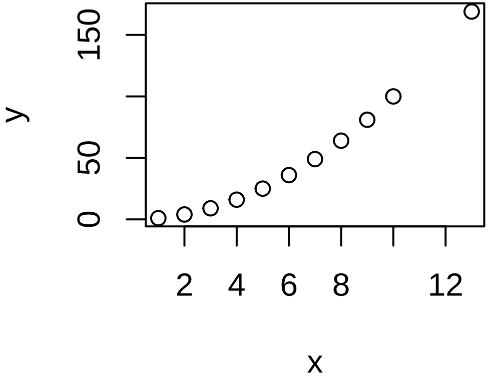
```
(y <- x^2) # Use the parentheses as a shortcut to print the output
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100 169
```

Simple calculations on vectors

And we can plot this using plot(x, y).

```
plot(x, y)
```



Simple calculations on vectors

What if we always want to consider these variables x and y together, like they are a data set? One way to combine vectors is to use the `data.frame()` command. Two useful tricks for taking a quick look at your data set are `summary()` and `head()`. Type in (hitting return after each line):

```
example <- data.frame(x, y)
summary(example)
```

```
##        x                y
##  Min.   : 1.000   Min.   :  1.00
##  1st Qu.: 3.500   1st Qu.: 12.50
##  Median : 6.000   Median : 36.00
##  Mean   : 6.182   Mean   : 50.36
##  3rd Qu.: 8.500   3rd Qu.: 72.50
##  Max.   :13.000   Max.   :169.00
```

```
head(example)
```

```
##   x  y
## 1 1  1
## 2 2  4
## 3 3  9
## 4 4 16
## 5 5 25
## 6 6 36
```

DIY Statistics

Let's use our fake data to illustrate how to do basic statistics with home-made methods, and write our own little program.

Firsts, try summing the using `sum(x)`. Ok, now try `mean(x)`. Another: `length(x)` tells you how many numbers there are in the vector x; you can also think of this as the n number of samples in your data set.

Now, let's combine them to start doing some statistics. Remember that the sum of squares an important value in statistics, which is the sum of the squared differences between each data point and the mean. In symbols:

$SS = \sum(x - \overline{x})^2$

DIY Statistics

We can make R do this by typing a series of commands nested inside each other, adding the parentheses to keep things straight. First, the difference between each x and the mean value for all x's:

```
x-mean(x)
```

Now square it:

```
(x-mean(x))^2
```

And sum them all:

```
sum((x-mean(x))^2)
```

DIY Statistics

Great. Now what can we do with this number? We might want to know the variance of the data, which is the sum of squares divided by n-1:

$$s^2 = \frac{\sum(x-\overline{x})^2}{n-1}$$

We can tell R to do this by adding one more bit to our last command (hint: use the up arrow to restore the last command without having to type it in):

```
(sum((x-mean(x))^2))/(length(x)-1)
```

What if we want to calculate this for a lot of variables, and don't enjoy cutting and pasting all that much? We can write a function to make our lives easier. It's much easier to do this in an Editor window than in the R console. Go to File > New File > R Script.

DIY Statistics

In the new window that opens up, type this:

```
variance <- function(x){
    sum(((x-mean(x))^2)/(length(x)-1))
}
```

And test it out:

```
variance(x)
```

Congratulations. You just wrote your first R function. In fact, your work has already been done by others: R has a built-in function to calculate variances, sensibly called `var()`. Check `var(x)` and `var(y)` to see that your function works perfectly.

DIY Statistics

What if you are already confused about what all variables you have to work with? Look at the list of objects in your workspace using `ls()` (without typing anything in the parentheses). You'll see `variance`, `x`, `y`, and `example` in there. Remove `x` and `y`, and variance by the command

```r
rm(x, y, variance)
```

Getting you data in

R looks for the working directory to read data files and to write output. Where is this directory? Type in `getwd()` to see.

It will probably be located somewhere inconvenient. Let's change this with `setwd()`. First, go to the directory you would like to work in, and make a new folder called "R". Find the path for that directory, and paste it in as follows: `setwd(<New R Directory>)`

Dataframes

Nearly all of the time you will be working with dataframes; think of a spreadsheet in Excel. A dataframe is an object with rows and columns. The rows contain different observations from your study, while the columns are different variables. The values in the body of the dataframe can be numbers, text (e.g. the names of factor levels for categorical variables, like ‘male’ or ‘female’ in a variable called ‘gender’), calendar dates, or logical variables.

All values of the same variable must go in the same column. If you had an study with a control, treatment1 and treatment2, and two measurements per treatment, this might seem logical:

control	treatment1	treatment2
6.1	6.3	7.1
5.9	6.2	8.2

Getting your data in

That was not a good dataframe for R, because values of the response variable appear in 3 different columns. The better way to enter these data is to have two columns (vectors): one for the response and one for the levels of the experimental factor:

response	treatment
control	6.1
control	5.9
treatment1	6.3
treatment1	6.2
treatment2	7.1
treatment2	8.2

Getting your data in

If you already had data in vector format, you could create a data frame like so:

```
response <- rep(c("control", "treatment1", "treatment2"),
                each =  2)
values <- c(6.1, 5.9, 6.3, 6.2, 7.1, 8.2)

(mydataframe <- data.frame(response, treatment = values))
```

```
##     response treatment
## 1    control       6.1
## 2    control       5.9
## 3 treatment1       6.3
## 4 treatment1       6.2
## 5 treatment2       7.1
## 6 treatment2       8.2
```

Mini-exercise:

What is the `rep()` function? What happends if `each` is changed to `times`? Look it up in the help.

Getting your data in

Now that we know what format we want the data, we are ready to read data in.

R doesn't read Excel files directly (not easily). Save your data as comma-delimited text files: `File > Save As...` then from the 'Save as type' options choose `Comma Separated Values (.csv)`. This file can then be read into R directly as a dataframe, using the `read.csv()` function. Tips:

Find the `Motor Trend Car Road Tests.csv` file and put it in your newly created R folder (your working directory). Then type in:

```
setwd("H:/R")
cars <- read.csv("Motor Trend Car Road Tests.csv")
```

Once the file has been imported to R we want to use attach to make the variables accessible by name within the R session and use names to get a list of the variables:

```
names(cars)
```

```
##  [1] "Model" "mpg"   "cyl"   "disp"  "hp"    "drat"  "wt"    "qsec"
##  [9] "vs"    "am"    "gear"  "carb"
```

Taking a first look at your data

Now were ready to start poking around at our data. Use summary() to make sure things seem to be in order:

```r
summary(cars)
dim(cars) # Number of rows and columns
```

```
##                  Model         mpg             cyl
##  AMC Javelin        : 1   Min.   :10.40   Min.   :4.000
##  Cadillac Fleetwood: 1   1st Qu.:15.43   1st Qu.:4.000
##  Camaro Z28         : 1   Median :19.20   Median :6.000
##  Chrysler Imperial  : 1   Mean   :20.09   Mean   :6.188
##  Datsun 710         : 1   3rd Qu.:22.80   3rd Qu.:8.000
##  Dodge Challenger   : 1   Max.   :33.90   Max.   :8.000
##  (Other)            :26
```

```
## [1] 32 12
```

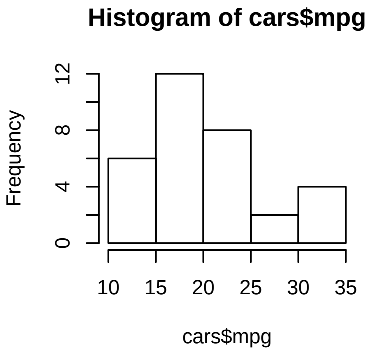
We can see this is a data set of road test results from 32 car models. See data(mtcars) for more details on this built-in dataset.

Taking a first look at your data

To get a sense of how variable of the continuous variables are, make the following histograms:

Note that R is case sensitive; `mpg` is not the same as `MPG` or `Mpg`

```
hist(cars$mpg)
hist(cars$cyl)
```



Taking a first look at your data

Now let's look at one of these variables. First look at miles per gallon:

```
mean(cars$mpg)
```

```
## [1] 20.09062
```

```
sd(cars$mpg)
```

```
## [1] 6.026948
```

Ok, interesting but not too exciting. What about the mean values for a given number of cylinders?

Taking a first look at your data

Note we have been using the dollar sign $ to pick out one vector of a dataframe. We can also use a trick called indexing, which locates a value or set of values within a data frame. R uses brackets `[ ]` for indexing.

Inside the square brackets, you can tell R to pick out one element from any R object (vector, dataframe, matrix, or list, which will be explained later!).

First, look at all the MPG measurements:

```
cars$mpg
```

Now pick out only the first one with

```
cars$mpg[1]
```

Mini-exercise:

What is the last value in the vector `mpg`?

Indexing

We can pick out all the mpg measurements which are from 4-cylinder cars using the logical symbol ==. This will only return the results for which the condition after the double equals sign is true:

```
cars$mpg[cars$cyl == 4]
```

Taking a first look at your data

Now we get only the 11 values for 4-cylinder cars We can take the mean and standard deviation of this set:

```
mean(cars$mpg[cars$cyl == 4])
sd(cars$mpg[cars$cyl == 4])
```

But this could get old quickly. Let's use the table apply command `tapply()` to get the means for all cars at once:

```
tapply(cars$mpg, cars$cyl, mean)
```

```
##        4        6        8
## 26.66364 19.74286 15.10000
```

Much better. Do the same for `sd()`.

Excercise 1:

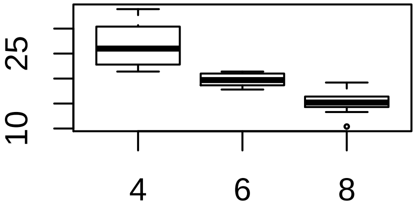
Write a function to calculate the minimum, median, and standard deviation of a vector, and show the result on the console.

Try yourself first, then work with your neighbor.

Basic plots

Let's look at these data in two ways. First, let's examine how MPG varies by number of cylinders. We can do this a few ways, but first make a box plot:

```
boxplot(mpg ~ cyl, data = cars)
```

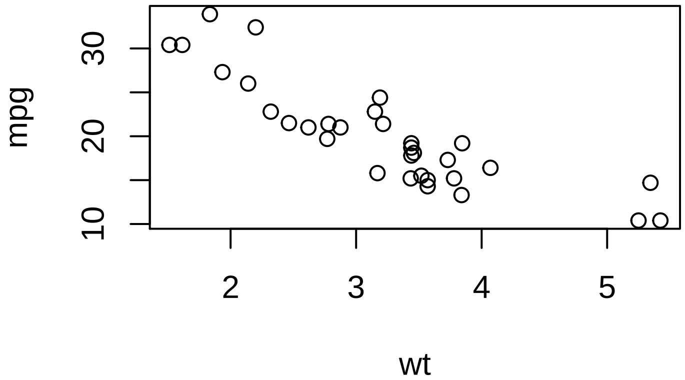


The trick here is the ~, which says "as a function of". To the left of it is the dependent variable, and to the right is the independent variable.

Basic plots

Next, look at the relationship between MPG and vehicle weight.:

```
plot(mpg ~ wt, data = cars)
```



How to get help

Learning to use R effectively takes patience. But while you'll find that many little things may seem frustrating, others have been there before you, and your questions are probably answered somewhere. See the list of URL's attached for ideas. Within R, you can always type in `?<command>` or `help(command)` to get help on a specific command. Use the search box for full-text searching of the help files.

Supporting resources:

- Stackoverflow
- Cross Validated
- R-specific search engine: RSeek
- Quick-R (Recommended supplement to this course!)

Homework!

1. Find a dataset of interest for this course! If you don't have one handy, consider the following sources:

- Bureau of Transportation Statistics
- NHTSA Research and Data
- Hubway Data

2. Write an R script to do the following:

- Read in one data file
- Produce summary statistics
- Produce at least one exploratory figure

2. Go to the course website: `http://bit.ly/volpeR`

- Upload your .R script in the **Homework 1** folder.
- Upload the data file your script uses
- Use the file naming convention `Homework 1 <Lastname>.R` and `Homework 1 <Lastname>.csv` for your script and your data file, respectively. Please use the spacing