Lab 1: Introduction to R & Exploratory Data Analysis

Updated January, 2016.

Table of Contents

What is R?

R is an open source software programming language and environment for statistical computing and graphics. It is used widely among statisticians and data miners for developing statistical software and data analysis [1]. R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, etc.) and graphical techniques and is highly extensible. One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed [2]. Further, as we will develop in this class, R can be combined with markup language to create documents that capture both workflows and metadata.

For this class, we will use *R* to analyze/visualize environmental data and *RStudio* as the frontend to R for its programming environment, workflow and web apps.

Installing R and RStudio

You will need to install both R and RStudio. R is available for free from the R website (www.r-project.org) and RStudio from the RStudio website (www.rstudio.com). Download the relevant installer for your operating system here:

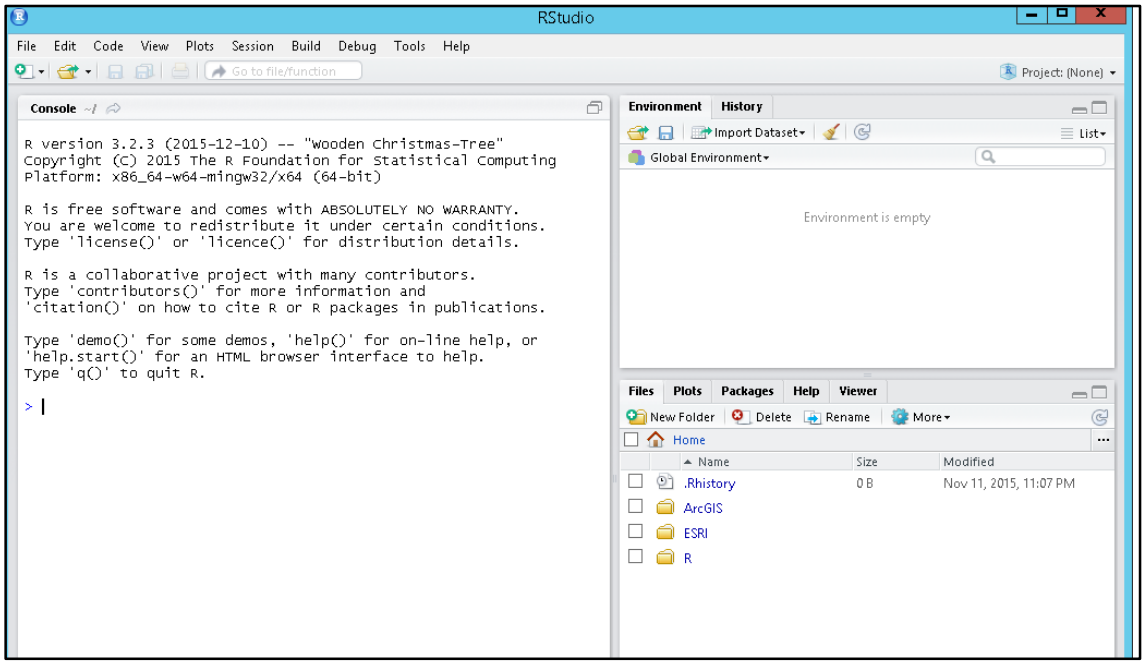
R: http://cran.cnr.berkeley.edu/
RStudio: http://www.rstudio.com/products/rstudio/download/

RStudio requires R to be installed, so install R first then install RStudio. The installers are straightforward; you may just accept the defaults during installation.

Using RStudio

At this point, RStudio is the only program you will need to use. On a Windows system, RStudio can be accessed from the start menu. Once open you are presented with a graphical user interface to RStudio tools and an R console.

RStudio Anatomy



Things to look for:

- Console - The R Console
- Environment Tab - This is your current workspace and a list of currently available variables
- History Tab - A list of past commands used in the console

- Files/Plots/Packages/Help/Viewer Tabs - Dynamic tabs that will show relevant information depending on R commands that are executed.

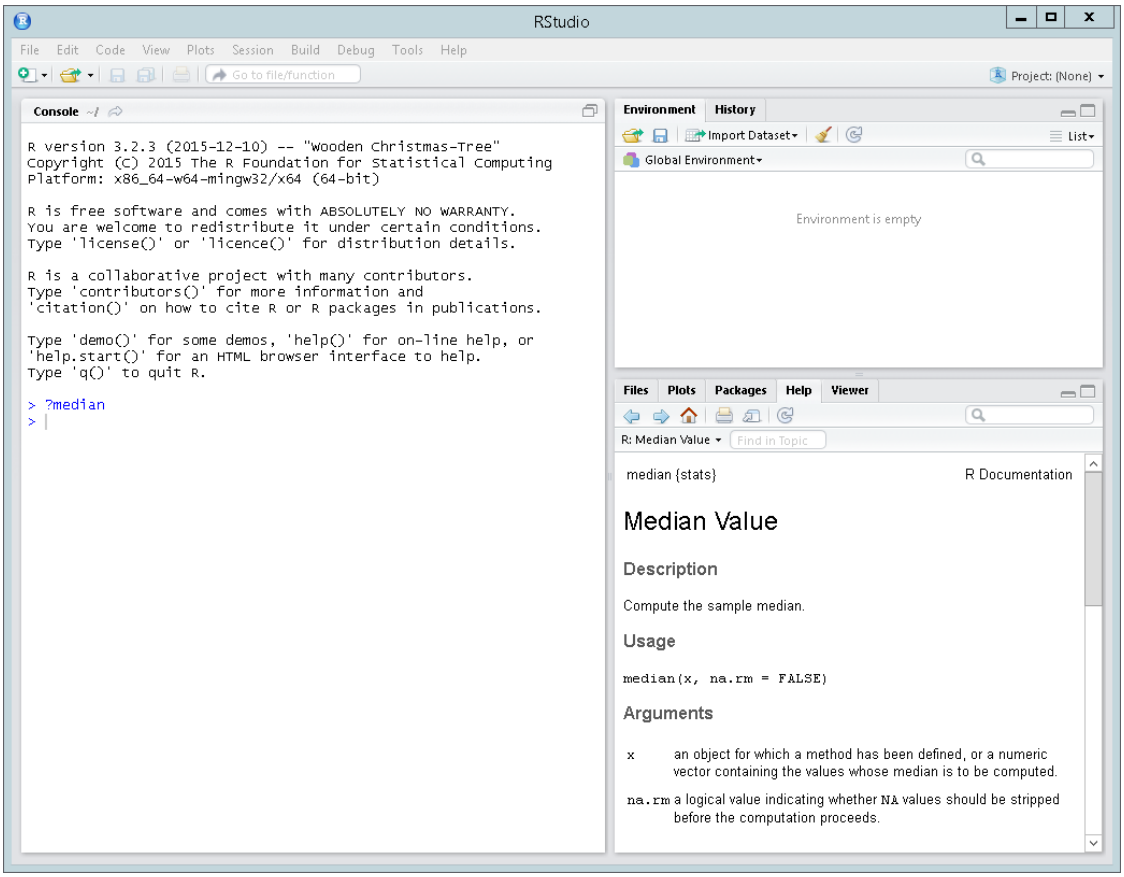
Using the R Console

The R console is an interpreter that *directly executes* any valid command typed into the interface. This can be useful for obtaining help on a function or testing a function before implementing it into your larger script. As an example we'll store a matrix and recall one of its elements in the console.

Help

Help for any function can be accessed by typing a question mark and then the function name or `help()`. Get help on the `median` function by typing the following in the console:

```
?median
```



Notice the Help tab will come forward and display the median function's usage. It shows the parameters associated with the function and their default values (if you do not give a particular parameter while executing a function, it will try to use the default value).

If you don't know the exact name of a command, you can search using keywords by using a double question mark. Type the following and press Enter to find relative information on standard deviation:

```
??"standard deviation"
```

Other helpful commands include the following (try them out):

```
print("hello world") # print something to the console
data() # lists available datasets
library() # lists available libraries
ls() # lists objects in your session
rm() #removes an object/variable from your R environment
class(a) # object type of a
q() # is to quit, but don't do this yet!
```

(Note the comments are preceded by the `#` symbol. You can use this at anytime to comment your own code.)

Executing a Function and Accessing Data Frame Elements

A function is executed by typing its name followed by parameters inside parentheses and pressing Enter. Store a vector into the variable "`fish.ID`" by executing the following command in the console:

```
fish.ID <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

A vector is a name for data structure in R composed of a sequence of data elements of the same type. In R, a number is a vector of length 1. Here we use the `c()` function to *combine* a series of numbers to create a 10-member vector. We can also make other vectors:

```
fish.lengths <- c(18, 61, 89, 21, 93, 86, 107, 131, 103, 117)
```

We can even make vectors of character strings or logical values

```
fish.species <- c("ONCL", "ONCL", "ONCL", "ONMY", "ONMY", "ONCL",
"ONMY", "ONCL", "ONMY", "ONCL")
fish.isJuvenile <- c(TRUE, FALSE, FALSE, TRUE, FALSE, FALSE, FALSE,
FALSE, FALSE, FALSE)
```

You may notice that these all seem like columns in a table. Data frames are common data structures used in R. A data frame is a list of vectors of equal length. We can make a data frame using the `data.frame()` function:

```
fish.data <- data.frame(fish.ID, fish.lengths, fish.species,
fish.isJuvenile)
```

Once stored, you can recall the variable by simply typing the variable name into the console and pressing enter.

```
fish.data

##    fish.ID fish.lengths fish.species fish.isJuvenile
## 1        1           18         ONCL            TRUE
## 2        2           61         ONCL           FALSE
## 3        3           89         ONCL           FALSE
## 4        4           21         ONMY            TRUE
## 5        5           93         ONMY           FALSE
## 6        6           86         ONCL           FALSE
## 7        7          107         ONMY           FALSE
## 8        8          131         ONCL           FALSE
## 9        9          103         ONMY           FALSE
## 10      10          117         ONCL           FALSE
```

This gives you a visual of the "`fish.data`" data frame. It is easy to see how to access an individual element from this visualization. For instance, the weight of fish #5 is located at [5,2]. In R, we address elements in any tabular data structure (data frame, matrix, table, etc,) with the following syntax: `variable[row,column]`

To access this element you would use the following:

```
fish.data[5,2]
```

You can also access an entire column or row. To access an entire 2nd column from this data frame, use the following:

```
fish.data [,2]
```

For the entire 1st row, use:

```
fish.data [1,]
```

The columns of data frames in particular can also be accessed by the column name:

```
fish.data$fish.lengths
```

However to, to access multiple columns (here, columns 2-4), it is more convenient to use the bracket notation. Here, we use the colon operator, which is equivalent to the `seq()` function, which produces a sequence of numbers from the starting value to the ending value in increments of 1:

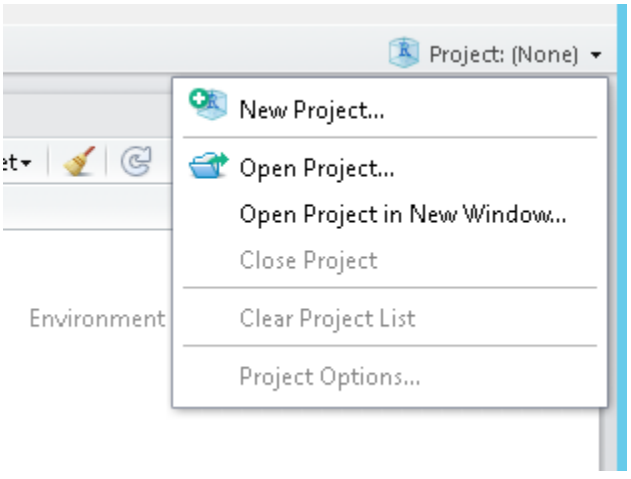
```
fish.data [,2:3]
```

R Markdown

R Markdown is an authoring format that enables easy creation of dynamic documents, presentation and reports from R [3]. We will use it for demonstrating your workflow and outputting results (both of which you will turn in).

Creating a Project and Saving R Markdown Files

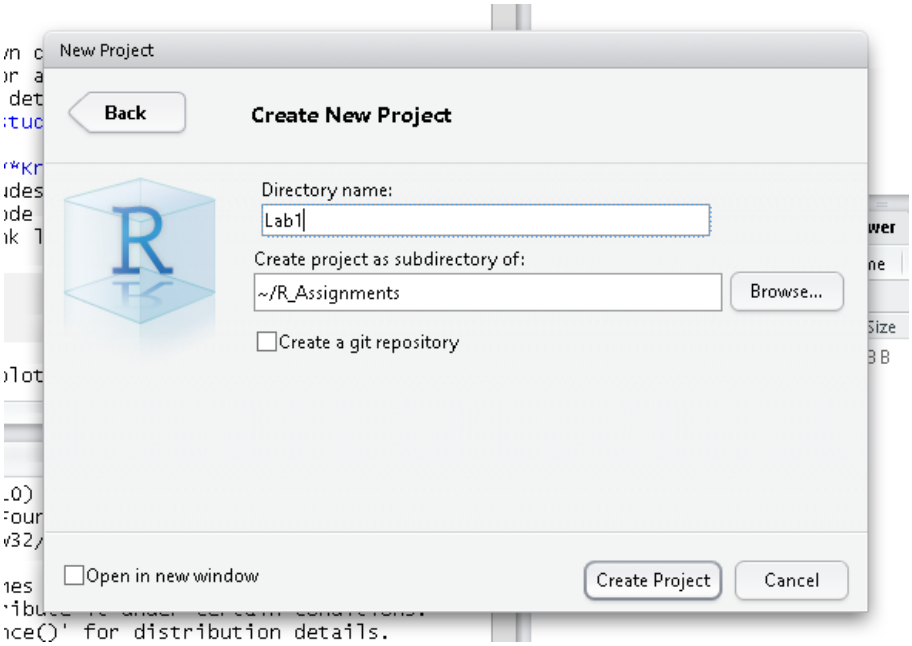
Your workspace should be a dedicated directory for your project that will contain all your relevant data. This should always be set before starting your exercises. There are multiple ways to choose where your R environment data should be saved. When using RStudio, it's easiest to use the built-in Projects feature.
To create a new project, click on the Project icon at the upper right corner of your RStudio window.



Select `New Project…` and then select `New Directory → Empty Project` in the following window.

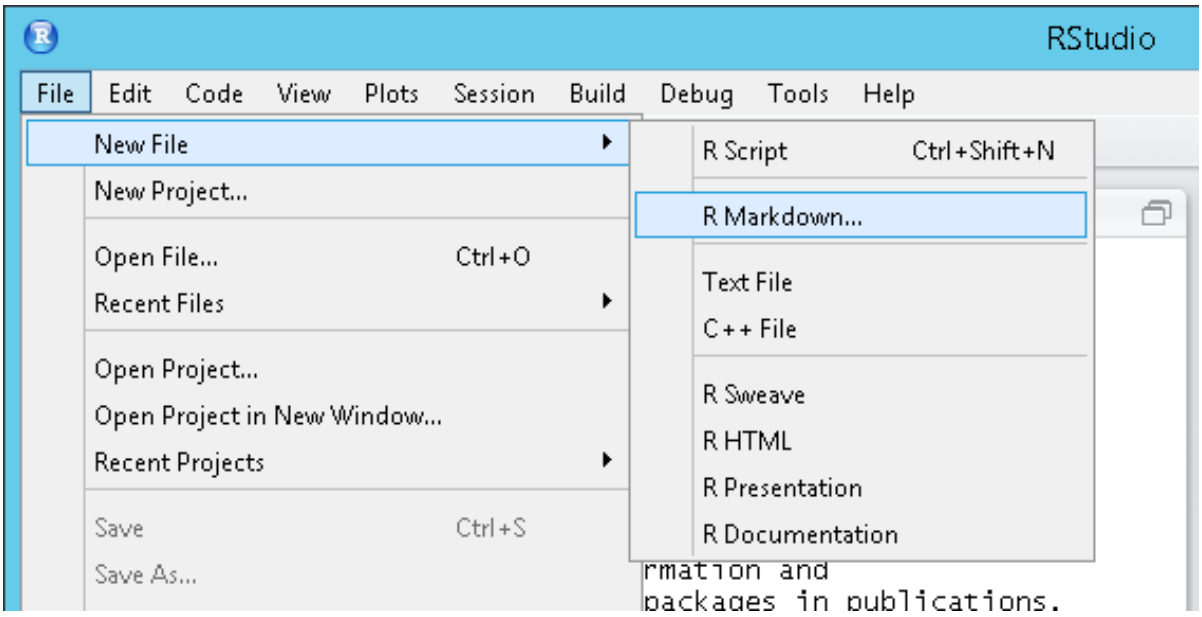
In the following screen, click the browse button, and navigate to where you would like your project to be saved. RStudio will create a new folder in the location that you choose. Back at the Create New Project window, choose a `Directory name`. This will be both the name of the folder that RStudio creates and the name of your project (under the drop-down menu in the upper-right corner of RStudio).

Finally, create your project.



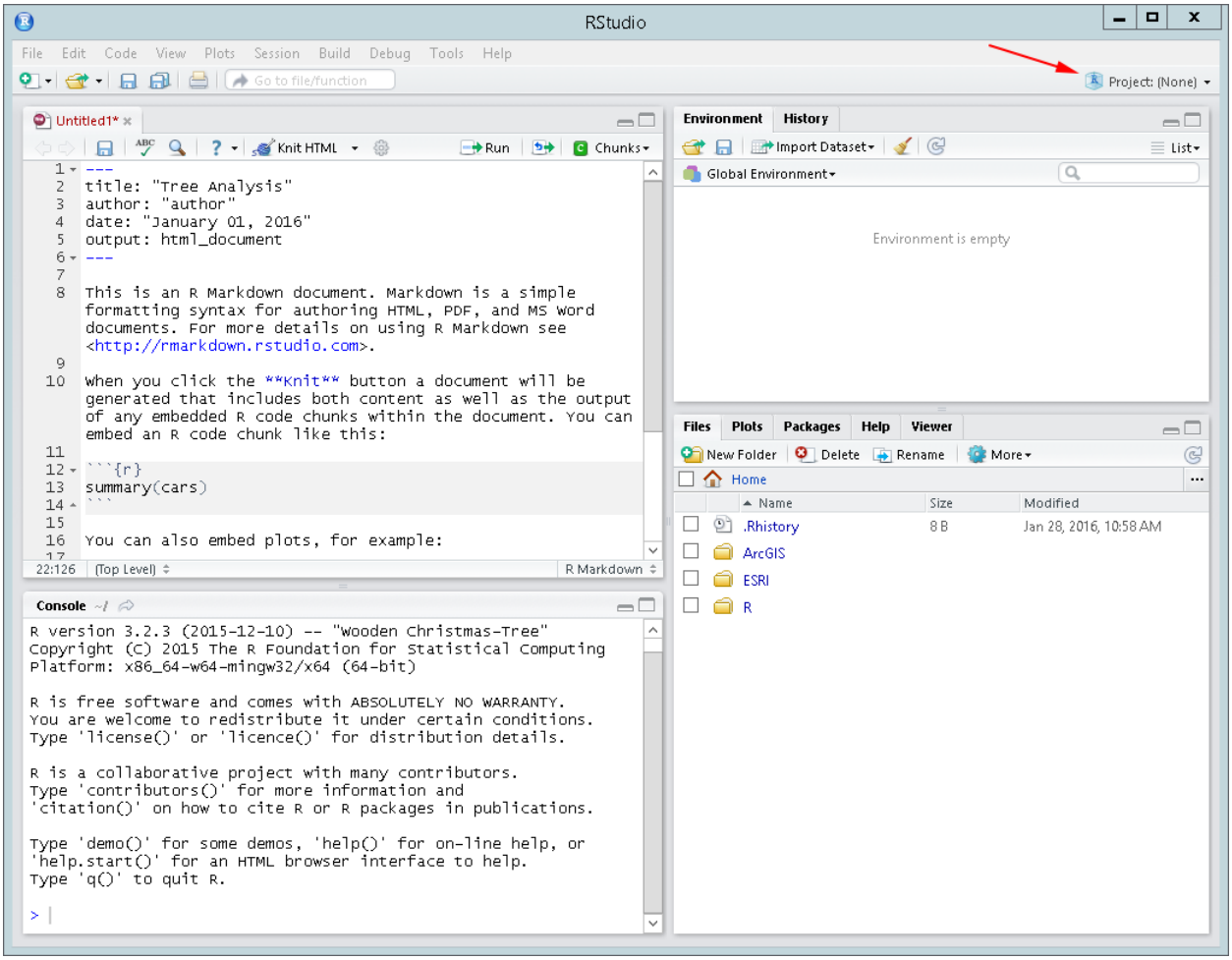
Creating a new R Markdown Document

To create a new R Markdown document, go to `File → New File → R Markdown...`



You may be asked to update some packages, respond “Yes” (make sure to have an active connection to the internet as this will retrieve packages from an online source). You will then be presented with a dialog in which you can select the type of Markdown you wish to create, a title, author and output format. For now, select Document, title it “Tree Analysis”, put your name under author and Word as the output format (Note that you can change these parameters after

you start the new Markdown). Click OK. A new window within RStudio will be created which will help you demonstrate your workflow. You can use the example code that is generated as a template for completing the lab exercises.



To save your Markdown file, select `File→Save` and a Save Dialog will appear. You should already be in your workspace directory, save it there. Give it a name (example `Lab1`) and press save. This will generate a file with the `.Rmd` extension. Make sure to periodically save your work and also after you are finished with the lab, before you submit it.

Generating the Output Document

At this point, you can create a preview of the output document. To generate the output document, click on the "`Knit HTML/PDF/Word`" button above your code. You can click the little triangle to the right of the Knit button to choose the format of your output document. When you press the knit button, another dialog will appear showing you a preview of your current output. To close the preview, press the close button.

Note: The production of PDF outputs requires a TeX installation. Installing LaTex via MikTeX or TeX Live is recommended for students who may have the need to perform sophisticated mathematical typesetting. However, on all installations, installing LaTeX consumes a fair amount of disk space (the installer is ~2GB).

Formatting R Code

As with most programming languages, there exists a convention for formatting statements in R. Adhering to style conventions helps make your R code easier to read, share, and verify.

In the previous example:

```
fish.ID <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

Notice how <- was used to assign the variable, A instead of the equals sign, even though an equals sign could be used (and is traditionally used in most programming languages). More examples can be found in Google's R Style Guide [4].

Homework Exercise (3 Points total)

These exercises rely on the lab data found in the resource section. Download `Trees.csv` from CatCourses and save to your R workspace.

Your document should have the following sections, and provide written explanations formatted in R Markdown that explains your code, output and graphics in the following format:

NAME
CLASS
DATE

Homework Assignment 1

Objective Statement: [What are you trying to accomplish?]

Methods: [In general terms, what analyses are you doing?]
- **Data**: [What are the data and where did they come from?]
- **Code**: [In specific terms, what is the code that was used to conduct the analysis?]

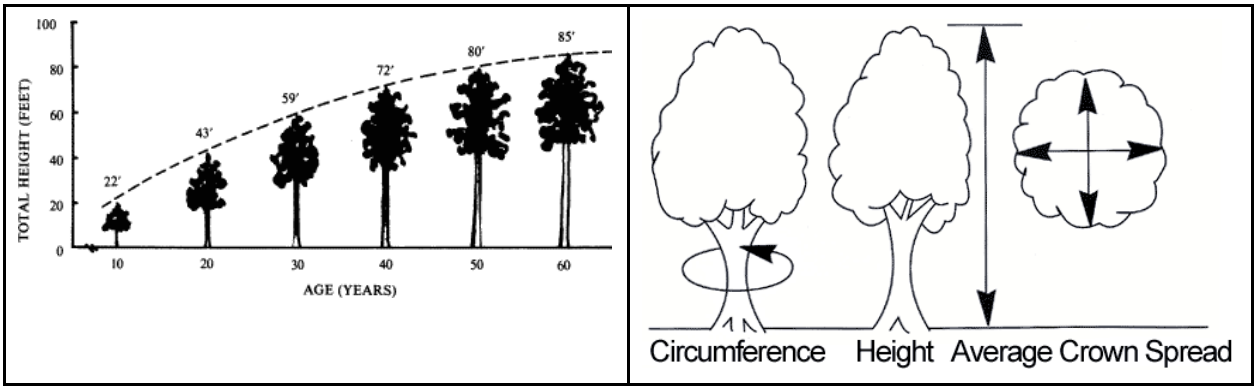
Results: [What do the results show? Numerical evidence and graphic evidence are required.]

Discussion: [What do the results mean?]

Limitations: [What are the limitations, caveats, and assumptions of the analysis?]

OBJECTIVE

Understanding the dimensional relationships among objectives is key to making inferences about unmeasured characteristics. An emerging area of environmental data analysis is in Carbon Markets to offset GHG emissions. To better estimate standing carbon in forests, allometric equations are used to estimate biomass (ie volumes) of trees. Your objective is to understand the nature of conifers in the Sierra Nevada using measurements extracted from LiDAR (tree crown heights and radii). Your objective is report out the descriptive statistics of the dataset and determine if tree heights are positively or negatively correlated with crown radii. Test the null hypothesis that there is zero correlation. Advanced users are challenged to create a map that shows the relative position and sizes of tree crowns [Pro tip: use the `ggplot2` package].



Step 1 - DEVELOP WORKFLOW and PSEUDOCODE; IMPORT DATA FRAME

Critical to any successful data analysis is sketching out your workflow. Develop pseudocode as appropriate. What are the critical steps? Use the template to form an outline and comment your steps as you go using `#comment`.

R has many built in functions for importing data, the easiest of which is loading 'comma-separated value' text files. These data are often denoted by `.csv` file extensions. Load `Trees.csv` into a matrix using the `read.csv` function. (Hint: You can get help with the `read.csv` function using the help flag `?`) Print the dimensions `dim()` and structure `str()` of the data frame. Do the data seem to correctly formatted?

Step 2 - RUN DATA DIAGNOSTICS

R has a built in summary function that is useful for obtaining a quick summary of a data set, but to better understand how these built in functions work, step through some diagnostics on your own. The following are some hints:

```
#What are your data?
data(Trees)
dim(Trees)
ncol(Trees)
nrow(Trees)
```

```
str(Trees)
head(Trees)
tail(Trees)
#Explore Trees[,2:5] columns as x
#min(x), mean(x), median(x), max(x), range(x), iqr(x), etc.
```

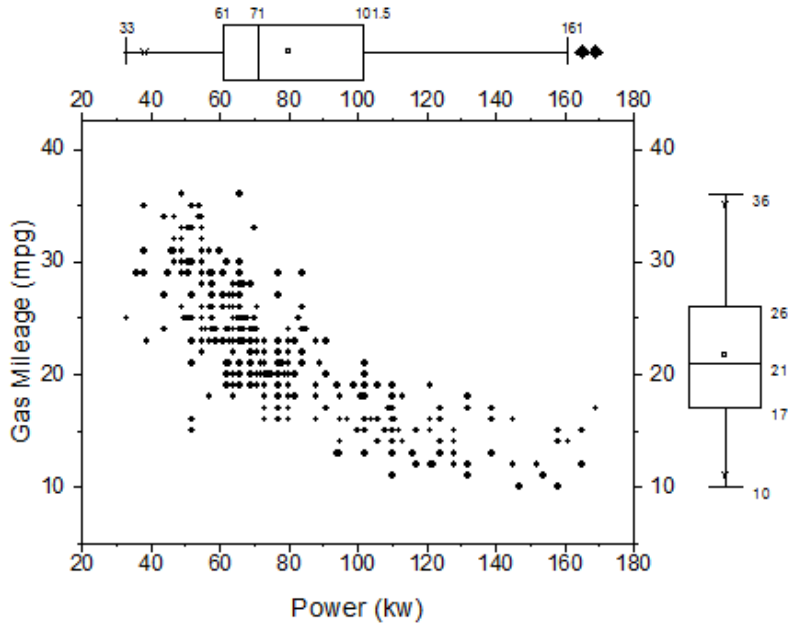
R also has a histogram function for visualizing data distributions.

```
hist(Trees[,1]) #first column is just ID, but you get the point...
```

Create data summaries, data distribution histograms for tree heights and crown radii. Does the data appear to meet assumptions of normality? Create log transformed histograms and reexamine.

Step 3 - EXAMINE CORRELATION BETWEEN VARIABLES

Find the (or "Pearson") correlation for tree heights and crown radii and create a pairwise plot for tree heights and crown radii. The plot should have a xy scattergram as the main figure, and boxplots on margins.



Example plot:

What is the `r` value? Use `cor.test` to determine the probability of rejecting the null hypothesis of no correlation between the two variables.

Step 4 - CREATE A FUNCTION TO SPEED UP ANALYSIS

You can create your own functions in R. Use the following template to create a function:

```
function_name = function(x) {
    #do something, such as square a passed value of x
    sqr = x^2
    return(sqr) #if applicable, return something
}
```

Create a function similar to the summary function called EDA (exploratory data analysis) that accepts a list of values as an input and outputs the following in a human readable way:

- Minimum
- Mean
- Median
- Maximum
- Range
- Standard Deviation
- Coefficient of Variation

Additional functionality, such as including plots and transformations could embed this function within an R script. We'll save R scripts for next time.

Resources

Data

- `Trees.csv` on CATCOURSES (`/ES207/Data/`)

References

- R Reference Card (`R-Short-Refcard.pdf`)
- R Operators and a few Functions (`R-ReferenceSheet.pdf`)

Citations

[1] http://en.wikipedia.org/wiki/R_(programming_language)
[2] http://www.r-project.org/
[3] http://rmarkdown.rstudio.com/
[4] https://google.github.io/styleguide/Rguide.xml

Keywords

R, RStudio, RMarkdown, `cor, hist, knitr`