Statistics 251: Lab 1 Pre-reading

Objectives:

- Introduction to Software
    - Introduction to R
    - How to get help
    - Creating data
    - Mathematical and basic statistical functions
- Learn about different plots for summarizing quantitative data
    - Histograms
    - Box plots and summary statistics
    - Scatter plots
    - R functions for plotting

1. Introduction to Software

1.1 Introduction to R

We will bypass the history of R and simply tell you that it is a freeware version of a statistical programming language known as S. R is hugely popular in the statistical community, and finds use in all kinds of academic statistical work. In the industry, a separate language called SAS holds more popularity; however, R is beginning to gain traction even there. R is an object oriented programming language. An object in R has three components: information, a name and a class. You can think of the object as a jar that contains information, and the name as the label on that jar. The class can be seen as the type of jar, where different types of jars store different types of information. Types of objects include vectors, matrices, characters, numbers, etc.

Valid names are composed of letters, decimal points and numbers (just not as the first character). It is highly suggested not to use pre-defined terms to name newly created objects (eg. median or TRUE). Creating an object will change with different classes, but the general syntax is

```
 name <- function( arguments )
```

Of course, some objects don't need a function. If we want to assign a reference '`x`' to the number `7`, we can write `x <- 7`. R will create x as the corresponding object of class numerical vector. This particular vector has length 1.

```
> x <- 7
> x
[1] 7
```

1.2 Help!

If you need to figure out how a particular function works, eg. the arguments it accepts, the values it returns, then you will want to take a look into the R documentation for that function.
The help files will eventually become your best friends.

In the rare case that you have a more sophisticated question that isn't answered in the R docs, then **Google it**! You'd be surprised how many programming questions in R can be answered with a simple Google query.
To access the R documentation for a specific command, simply type

```
?command or  help(command).
```

The R documentation for that command will then be displayed.

- Type `?mean`
  What does this function do?

1.3 Creating data

1.3.1 Vectors

We already showed how to create numeric vectors of length one: `x <- 7`. We can also assign vectors of 'characters' by writing, for example, `x <- "a"`. The string `"a"` is then assigned to the reference `x`. What if we want to make vectors of a larger length? Here are some basic functions that will be helpful …

- `c()`: `c` stands for concatenate. If you write, `myVector <- c(1, 3, 7, 11)`, then R will store this numeric vector of length 4 in the reference named `myVector`.
  You can use `c()` to concatenate vectors of different lengths and scalars as well.
- `rep()`: To create a vector of the desired length containing the same value throughout. Thus, `myVector <- rep(0, 5)` creates a vector of length 5, each entry a 0.
- Sequences: We have two approaches for creating a sequence of numbers…
  - `1:n` produces a vector 1, 2, 3, …, n-1, n. The start and end values can be any number. What happens if you put in non-integer entries?
  - `seq(1,n)` is another way of producing the same vector. However, if you take a peek into the R documention for `seq`, you'll find that it accepts many more arguments as well – the lower and upper endpoints, the desired length of the vector, the amount to skip, etc. etc.

    What does the command `seq(1, 10, by = 2)` produce? Why?

Elements from vectors may be accessed via `[square brackets]`. For example, `myVector[2]` retrieves the second element from your vector `myVector`, which we set as 3.

- What output do you get if you enter myVector[c(2,3)] ?

1.3.2 Matrices

Matrices are created in R using the matrix function. The general notation for creation of a matrix is

```
matrix( input, nrow, ncol )
```

Note that the main input when creating a matrix is a vector, plus the number of rows or columns we'd like to coerce our input vector into for a matrix.

```
myMatrix <- matrix(1:12, ncol = 4)
myMatrix
```

What output do you get? How have the entries `1:12` been inserted into this matrix?

```
matrix( 0, 2, 3 )
```

Note that the value `0` has been repeated to fill a matrix of dimensions 2x2 here.
We can use `cbind()` and `rbind()` to join vectors and matrices of compatible dimensions.

```
cbind(),
```
meaning ‘column bind’, joins your vectors / matrices column-wise (side by side).
```
rbind(),
```
meaning ‘row bind’, joins your vectors / matrices row-wise (one on top of the other).
```
cbind( myMatrix, c(1,2,3) )
rbind( myMatrix, c(4, 5, 6, 7) )
```
Note the output in each case.

Note: We have not actually modified the value of the ‘`myMatrix`’ reference! If you type in `myMatrix`, you will find it is still the same matrix you generated at the start. In R, the values of a reference will only change if you explicitly re-assign them again.
```
myMatrix <- cbind( myMatrix, c(1, 2, 3) )
```

Now, the reference `myMatrix` will contain the new vector `c(1, 2, 3)` as well.
You can access elements in a matrix with the syntax `myMatrx[ row, column ]`.
```
myMatrix[2,2]
```

You can also extract entire rows or columns by leaving a specific entry in the square brackets blank. The syntax is a bit odd, but once you are used to it, it is nice shorthand for accessing whole columns / rows. For example,
```
myMatrix[2,]
```

This command can be read as, “Give me the elements in row 2, from all of the columns available in `myMatrix`”. Or, more plainly, “Give me all of the elements in row 2”. Similarly,

```
myMatrix[,2]
```

will return all the entries in column 2.

There are more objects that can be created and used in R. For example, a data frame is a very useful object type. If you are interested, you can read about data frames in R online.

1.4 Mathematical and Basic Statistical Functions

1.4.1 Mathematical functions

Mathematical operations are simple and resemble almost every other programming language you might have already encountered, or might encounter in the future. We’ll start with vectors since most mathematical operations are done on these.

Consider `myVect <- 1:14`. Because this is a vector, we can perform most basic mathematical functions on it like adding, subtracting, multiplying, or dividing. Suppose we type in:
```
myVect + 1
```

we notice that 1 is added element-wise to the vector 1:14, and the resulting output is 2:15. Similarly, if we type in
```
myVect * 2
```
then each element in the vector is multiplied by 2.

What happens if you type `myVect * c(1, 2)` ?
Can you explain the resulting output?
What if you type `myVect * c(1, 2, 3)`?

What does the warning message mean?

So, we have '+' for addition, '-' for subtraction, '*' for multiplication, and '/' for division. We also have more complex functions such as,

```
log(myVect) ## takes the logarithm, base e
myVect^2 ## takes each element in myVect, and puts it to the power of 2
sqrt(myVect) ## Square root of each element in myVect
exp(myVect) ## e^( each element in myVect )
```

Note: the ## can be used to write comments alongside your code. A line of input that is preceded by # will be ignored by R if passed to the command line. **Always comment your code!**

R will also obey the rules of **BEDMAS** – that is, it performs the operations in order such that items within brackets are computed first, then exponentiation is done, then division/multiplication, and finally addition/subtraction.

1.4.2 Basic Statistical functions

Some of the more common statistical functions we'll use throughout the course…

Sum of elements in a vector / matrix: `sum()`
Average of elements in a vector / matrix: `mean()`
Median of elements in a vector / matrix: `median()`
Variance: `var()`
Standard deviation: `sd()`
Maximum value in a vector / matrix: `max()`
Minimum values in a vector / matrix: `min()`
Range = (min, max) of a vector / matrix: `range()`
Quantiles of a vector / matrix (formal definition coming later) `quantile()`
Summary of the values in your vector / matrix: `summary()`

It is highly recommended you keep the following reference card on hand for whenever necessary:
http://cran.r-project.org/doc/contrib/Short-refcard.pdf

2. Summarizing Quantitative Data.

Visualizing your data with different kinds of figures is of utmost importance in being able to understand the distribution of your data. In this lab, you will focus on using R to generate different kinds of statistical plots that will be useful as summaries of the data.

In particular, we will focus on visualizing **quantitative data**.

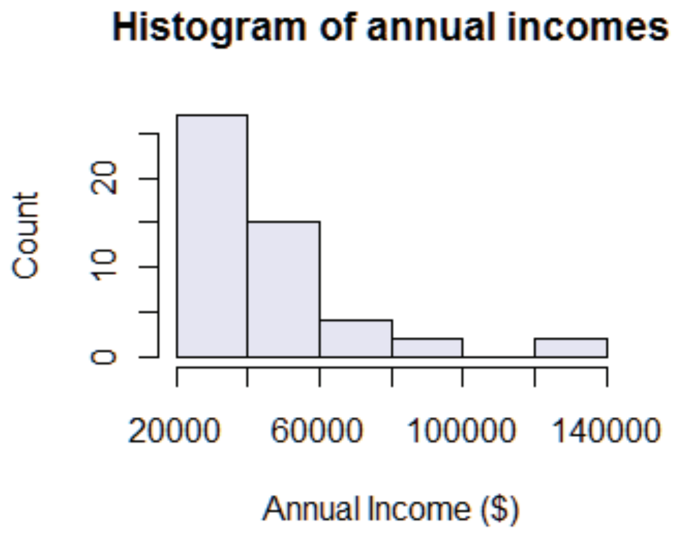
2.1 Histogram

A histogram allows you to visualize the **distribution** of a set of observations collected on a **quantitative variable.** Histograms are used to assess:

- The shape of the distribution (symmetric, left-skewed, right-skewed),
- Whether the data set contains outliers (data points that are 'far' from the other data),
- Modality of the data (unimodal = one peak, bimodal = two peaks, and so on)

To generate a histogram, first the data must be **binned**, and then the **counts of observations in each bin** are plotted.

Eg: Suppose you have data on annual incomes for 50 people.



In our set of data, most people have an income in the range $20000-$40000; however, the number of people with large incomes is smaller.

- The distribution is **right skewed** (long right tail),
- There are <u>**maybe**</u> some outliers in the right tail (this is a bit subjective!)
- The distribution is uni-modal (only one peak).

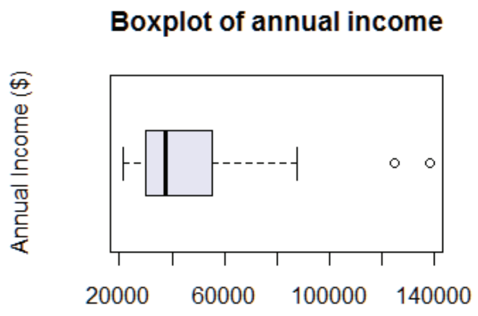
In R histogram of data may be obtained using `hist()` command. Please note that for the lab you will have to build histogram of a particular column in a table. For that, you can use dollar sign reference as in `hist(table_name$column_name),` were `table_name` is the name under which your table is saved and `column_name` is the name of the column inside this table.

2.2 Boxplots

The boxplot is a 'simplified' histogram that focuses on visualization of the **five-number summary** for a set of **quantitative** data, as well as a cutoff for **outliers** in the data. The five number summary is made up of the **minimum, 1st quartile, median, 3rd quartile, and maximum.**

	Minimum	1st Quartile	Median (2nd quartile)	3rd quartile	Maximum
% data lying to the left of…	0%	25%	50%	75%	100%

What would a boxplot of the previous set of income data look like?



Min	1st Quantile	Median	3rd Quantile	Max
21390	30220	37510	54020	138400

Can you see the similarity between this boxplot and the histogram plotted above? The **box** is the shaded region of the boxplot, and is contained between the 1st and 3rd quantiles. It **captures the middle 50% of the data.** The **whiskers**, or dotted lines in the plot, extend up to the largest / smallest observations not deemed to be outliers.

** R determines points to be more than 1.5 x IQR (*explained later*) away from the appropriate quantile to be outliers.

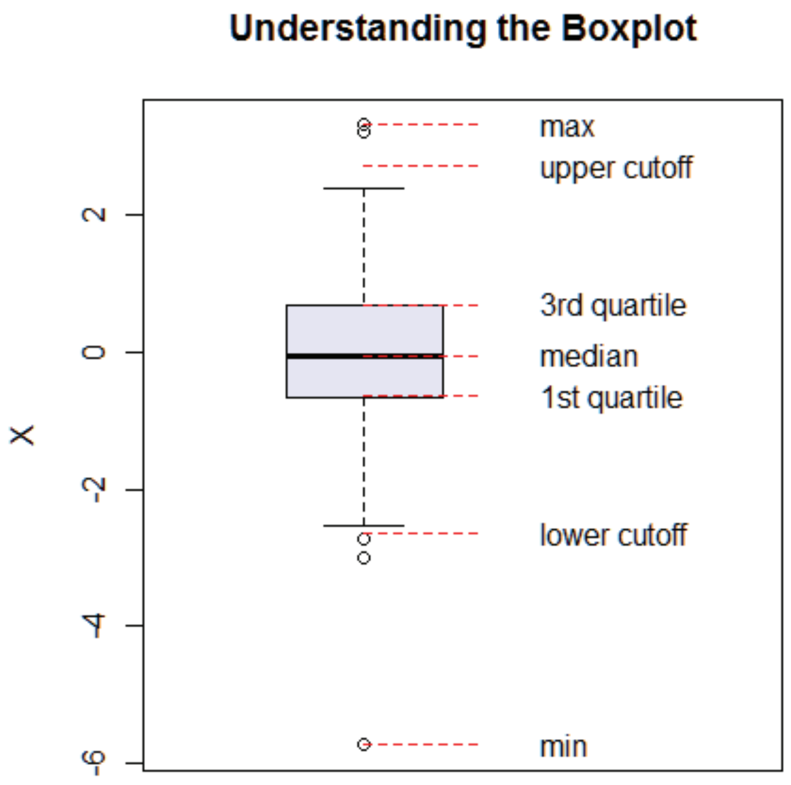
Lower limit: 30220 – 1.5 * (54020 – 30220) = -5480,
Upper limit: 54020 + 1.5 * (54020 – 30220) = 89720

In this case, we only have two points lying outside the upper limit, which R prints as outliers. **However, please refer to lecture notes for further comments on outliers.**

Using R, the summary statistics of the data can be found using `summary()` command. Boxplot is built using `boxplot()`. Please note that you can reference particular column of the table for each of these functions using the spelling shown in previous section.

2.2.1 Understanding the Boxplots

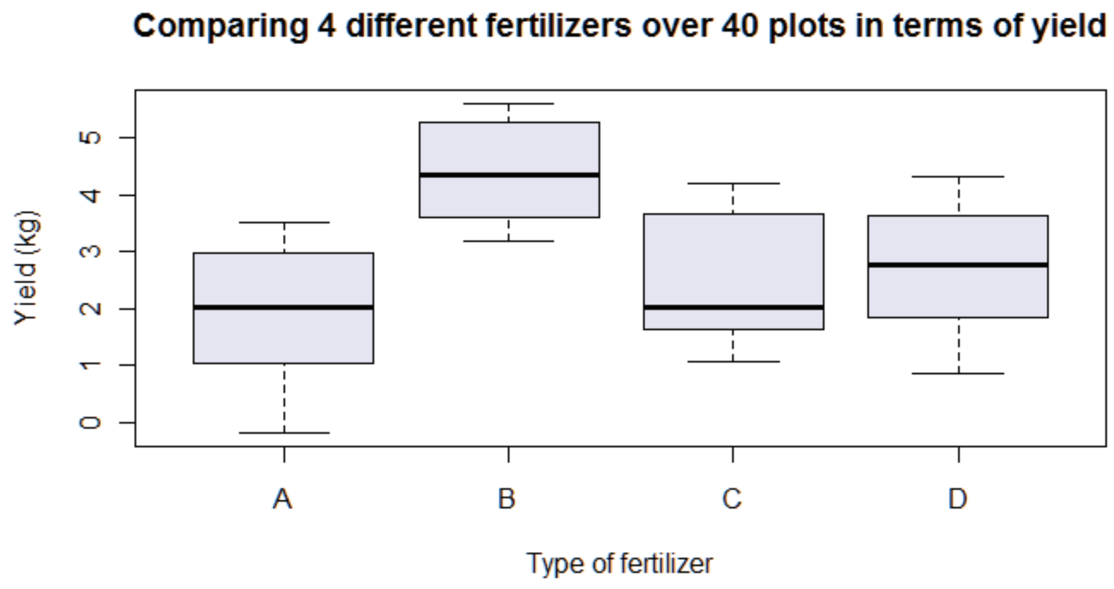
Note the difference between the actual lower and upper cutoffs (typically not plotted), and the smallest/largest data point that lies within the cutoffs (which IS plotted). The IQR is hence the length of the box, or the distance between the 3rd and 1st quartiles.



2.2.2 Why use Boxplots?

The boxplot is kind of a simplified histogram, with the only major benefit we might see (so far) is that it points out the outliers for us. Of course, whether or not a data point should be judged an outlier is a bit subjective; however, there are certain criteria we can use.

The main strength of boxplots comes in comparing **multiple quantitative variables**, or **a single quantitative variable measured over different groups**.



Based on the plot, it's easy to see that fertilizer B seems to perform the best overall, while fertilizers A, C and D are relatively comparable, with perhaps A doing the worst. Note that the boxplot provides a nice visual representation of both **location** and **spread** of the data.

Food for thought:

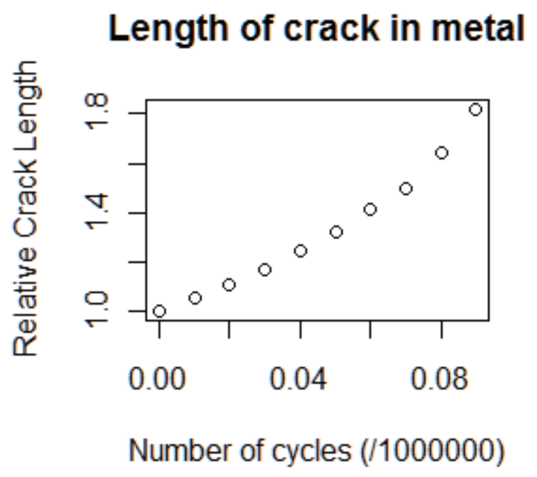
Do you believe, based on this plot, fertilizer B is **always** the best fertilizer? Or do you believe it might be the best **on average?**

The R commands for obtaining the side-by-side boxplots are given in section 4.4.

2.3 Scatterplot

The scatterplot is used when **two quantitative variables** are measured over **the same set of sampled items / people**. It helps as a way of visualizing the relationship between these two variables.

Eg: We will focus on measuring the length of a crack in metal after it is subjected to stress. An initial notch of length 0.90 inches was made on a unit, which then was subjected to several thousand test cycles. After every 10,000 test cycles the crack length was measured. Testing was stopped if the crack length exceeded 1.60 inches, defined as a failure, or at 120,000 cycles.



How might you describe this plot?

- As the number of cycles increases, the relative crack length increases,
- The change in relative crack length seems to increase as we get further into the experiment,
- The relationship is approximately linear, with positive deviations occurring near the end of the experiment.

2.4 Getting Comfortable with R Functions for Plotting

Type

```
x <- rnorm(20); y <- rep(1:4,each=5); z <- y + rnorm(20)
```

into your console to get started. Note that `rnorm()` is a function that generates random data from a standard Normal distribution – we will delve more into these types of functions later. Confirm that you have stored a set of data for each of `x, y, z` after running that code.

Display your data all at once in the R console by typing:

```
cbind( x, y, z )
```

What is the length of each vector of data you generated?

Boxplots

1) Use the `summary` function to calculate the 5 number summary of your `x, y, z` data.
2) Type `boxplot( x )`. What do you see?
3) Now, type `boxplot( x ~ y )`. What do you get?
4) What about `boxplot( z ~ y )` ? What do you think the tilde `(~)` is doing then?

Histograms

1) Type `hist( x, col = “grey” )`. What do you see? What does the `‘col’` argument do?

Scatterplots

1) Type `plot( x, y)`. What do you see? Describe the relationship.