Introduction to R

The purpose of this handout is to introduce you to the statistical programming language R.

- Introduce basics of R including creating objects,
- using functions, and
- import data into RStudio.
- Build familiarity with R

1 Introduction to R

R is a sophisticated statistical programming package and a powerful graphics engine. R is considered to be a dialect of the S and S+ programming language that was created by AT&T Bell Labs. S is commercially available while R is open source and freely available through the Comprehensive R Archive Network (CRAN). R has many advantages besides being freely available. It is extremely flexible and can be used for almost any imaginable statistical needs, from simple to extremely complicated. The base program of R is relatively small (only 69 mb) and comes standard with a core of basic statistical functions. However, it is designed for custom expantion and there are over 6000 downloadable packages for different specific needs! Thus, this tutorial will only be able to scratch the surface. The goal is to introduce you to R and get you familiar with the program, not to make you an expert.

This handout and the associated videos on the "Stats at Vassar" website (http://pages.vassar.edu/statsatvassar/) will provide you with hands on experience to overcome "fear of the R prompt", or what to do when you see the R prompt:

```
>
```

You will become familiar with seeing the R prompt at the end of running a command or analysis. Because R creates objects from analyses that are stored in its memory, new users often are surprised by the fact that the results of the analyses arent immediately displayed on the screen. This is an advantage of R and a useful function. For example, if you wanted to do 100 analyses on different data sets, R can do this without opening 100 separate windows. You can store only the important info from each analysis and display all of them in a single line for comparision.

2 Why Learn R?

You might be thinking to yourself "Why do I need to learn R?" or "Seriously, I have to type everything in by hand?!" or "Can't I do this easier in another program?". There are many answers to these questions.

1. While at Vassar, it is useful for you to learn R and RStudio because if you go on to take a Statistics course (which most of you will), you will most likely use R. Thus, it is advantageous to start learning it now, it will make your life so much easier later on!

2. Because it is free and extrmely powerful, R is the only statistics program you will ever really need to know. If you go on to graduate school or into consulting or any field that deals with data, you will be able to use R. The same cannot be said with other programs like JMP, SPSS, or SAS, which are very expensive and may not be available to you at another institution.

3. Yes, you have to type everything in, but that also helps you learn what you are doing. It is very easy to click some buttons and get an answer that you don't really understand. If you have to type in the code for the statistics you are doing, you will have a better understanding of what you are doing.

4. Having some basic familiarity with "coding" is increasingly useful across a variety fo disciplines. You don't need to be a pro, but being comfortable with a computer and with typing code to achieve a result is very useful.

3 Creating Objects

As mentioned above, the fundamental unit that R operates with is called an "object". Objects allow the user to create very simple symbolic representations of simple or complex datasets or other information which allows the user to create elaborate analyses from seemingly simple code with the object stored in the computers memory. However, creating and manipulating objects are also hard to get used to for those that are accustomed to drag and drop menus or using spreadsheets to amnually manipulate and analyze data by selecting columns, etc. In the end, however, creating and manipulating objects is far more efficient than manually manipulating data, as we hope youll see by the end of the module.

Check out the video on Creating Objects on the StatsAtVassar webpage

Objects are created with the "assign" operator to assign values to an object, which is an arrow created with the less-than sign and a dash "<-".

```
> n <- 10
```

In R language, you can say that we have "assigned the value 10 to the object n". Notice that nothing seemed to happen when you typed that in and hit enter. That is good, it means that R did what you asked. Now if you type n, R will return the value assigned to it.

```
> n
[1] 10
```

Names of objects MUST start with a letter and can include letters, digits, dots (.) and underscores (_). R is case sensitive, so x and X are different objects. Remember, spelling always counts and spelling mistakes are among the most frequent "bugs" in R code. R doesn't make mistakes, you make mistakes (sorry, but it is true). If you get an error, the first thing to do is to figure out what you typed wrong.

One thing to be careful of is that it is very easy to write over and replace existing objects.

```
> n <- 15
> n
[1] 15
```

Now the object n has the value 15 assigned to it. R did not give us a warning or any indication that we were replacing the existing value of 10. Remember, R just did exactly what we told it to.

We can also use R as a calculator.

```
> 2+2
[1] 4
> 2*2+1
[1] 5
> 2*(2+1)
[1] 6
> 2^4/2+1
[1] 9
> 2^(4/2)+1
[1] 5
> 2^(4/(2+1))
[1] 2.519842
```

Check out the video on using R as a Calculator on the StatsAtVassar webpage

4 Vectors

The most common, basic, and important type of object in R is called a vector. A vector is simply a sequence of elements of the same basic type. What does that mean in lay terminology? It means that it is a group of numbers or letters or words or whatever, but within a given vector you can only have one of those types of things (numbers or letters, but not both in the same vector). We can create vectors by using the combine function, which is called using the command `c()`. The `c()` tells R that we are combining whatever is in the parentheses into a single vector. For example,

```
> a<-c(1,2,3,4,5)
> a
[1] 1 2 3 4 5
> b<-c("a","d","f","k","l")
> b
[1] "a" "d" "f" "k" "l"
```

It is important to remember that we can call the vectors anything we want. The object names a or b have no special meaning. We could have called those objects Steve or Diane. Let's make more realistic vectors. Imagine we are collecting data on seedling height (recorded in cm) in two forest plots, one where deer are allowed to browse and one where deer are not allowed because of a fence. We have measured 20 seedlings, 10 in each plot.

```
> height<-c(3.8,4.9,3.9,4.1,3.0,4.6,3.9,3.0,2.9,4.4,5.9,6.1,6.7,4.7,5.7,6.5,5.8,
  3.9,5.6,6.2)
> plot<-c(rep("deer",10),rep("no deer",10))
> height
 [1] 3.8 4.9 3.9 4.1 3.0 4.6 3.9 3.0 2.9 4.4 5.9 6.1 6.7 4.7 5.7 6.5 5.8 3.9
[19] 5.6 6.2
```

```
> plot
 [1] "deer"    "deer"    "deer"    "deer"    "deer"    "deer"    "deer"
 [8] "deer"    "deer"    "deer"    "no deer" "no deer" "no deer" "no deer"
[15] "no deer" "no deer" "no deer" "no deer" "no deer" "no deer"
```

At this point you might be wondering what the numbers in the square brackets on the left side of the output mean. We will explain this a little later in the Indexing section.

5 Functions

In the examples above we have been using several functions. The fucntion `rep()` stands for "repeat" and we have repeated a bit of text (either "deer" or "no deer") 10 times each. `rep()` returns a vector, and we used the function `c()` to combine those two vectors into a single vector, which assigned to the object "plot".

Most of the work done by `R` is going to be done by functions. Functions are prewritten programs that we use to do something to a numerical value or to an object. The first part of the function calls the appropriate function, the action is then taken on the object or numerical value in between the parentheses. For example, the function `mean()` calculates the mean of a vector of numbers.

```
> mean(c(3,3,5,6,9))
[1] 5.2
> mean(height)
[1] 4.78
```

If you want to learn more about a function, you can use the "help" function. There are several ways to get help in `R`, the easiest is to type the following:

```
> ?mean
```

This opens up a window which tells us about what the function is doing and if there are any defaults to it. For example, for the `mean()` function we can see that the help file tells us `na.rm = FALSE`, and that `na.rm` is "a logical value indicating whether NA values should be stripped before the computation proceeds.". Since na.rm is by default FALSE, that means by default NA values (missing data) will not be removed if they are present. We can change that if we need to. We can see this example if we add an "NA" to our example from above.

```
> mean(c(3,3,5,6,9,NA))
[1] NA
> mean(c(3,3,5,6,9,NA), na.rm=TRUE)
[1] 5.2
```

6 Data frames

Most of the time when we want to analyze data, we are going to be working with a data frame. Data frames are the most analagous object in `R` to a spreadsheet like you might be used to from Microsoft Excel or another program. We can create a new data frame within R by using the function `data.frame()`. For example,

```
deer_data<-data.frame(plot,height)
```

Now we have an object called `deer_data`. We can view our data frame by typing the name of the data frame (but this might be a bad idea if we have a lot of data!) or we can use the `str()` function to view the structure of the data.

```
> deer_data
      plot height
1     deer    3.8
2     deer    4.9
3     deer    3.9
4     deer    4.1
5     deer    3.0
6     deer    4.6
7     deer    3.9
8     deer    3.0
9     deer    2.9
10    deer    4.4
11 no deer    5.9
12 no deer    6.1
13 no deer    6.7
14 no deer    4.7
15 no deer    5.7
16 no deer    6.5
17 no deer    5.8
18 no deer    3.9
19 no deer    5.6
20 no deer    6.2

> str(deer_data)
'data.frame':  20 obs. of  2 variables:
 $ plot  : Factor w/ 2 levels "deer","no deer": 1 1 1 1 1 1 1 1 1 1 ...
 $ height: num  3.8 4.9 3.9 4.1 3 4.6 3.9 3 2.9 4.4 ...
```

Notice that the `str()` function just shows us a very abbreviated version of the data frame, which is nice. It is a compact and easy way to look at your data. We have 20 observations of 2 different variables called plot and height. Notice the $ before each variable name. The $ is the primary way you access objects within a data frame. For example,

```
> deer_data$plot
 [1] deer    deer    deer    deer    deer    deer    deer    deer    deer    deer
[10] no deer no deer no deer no deer no deer no deer no deer no deer no deer no deer
Levels: deer no deer
```

We type the name of the data frame, then the $, and then the name of a variable (or column).

7 Indexing

By now you have undoubtedly noticed the numbers in the square brackets on the left side of the output. Each of the elements in a vector comes with a number, a place in the order of the vector from first to last. We can call out any one or multiple elements of a vector by using the index function `[]` (square brackets). Square brackets are actually a type of function and using indexing is a fundamental and key part of understanding how to use `R` to access and manipulate objects. For example, lets say we want the first element of `height`.

```
> height[1]
[1] 3.8
```

We can do this for any element in the vector, or multiple elements by specifying them using the range desired and a colon. The example below returns objects 5 through 10 in vector height.

```
> height[5:10]
[1] 3.0 4.6 3.9 3.0 2.9 4.4
```

When we have a 2-dimensional object such as a data frame, we can still use the `[]`'s to index, but now we have to provide row and column information, in the form of `[row,column]`. If we wanted the the same values of seedling height in the data frame, we can type the following, which says "return rows 5 through 10, in the second column".:

```
> deer_data[5:10,2]
[1] 3.0 4.6 3.9 3.0 2.9 4.4
```

If we want to return both columns, we can leave the column part blank.

```
> deer_data[5:10,]
   plot height
5  deer    3.0
6  deer    4.6
7  deer    3.9
8  deer    3.0
9  deer    2.9
10 deer    4.4
```

We can also use indexing to subset the data frame based on particular values we are looking for. For example, maybe we want to know which plants were taller than 6 cm? We can type the following, which says "in the data frame `deer_data`, return any row where height is greater than 6, and return both columns".

```
> deer_data[deer_data$height>6,]
      plot height
12 no deer    6.1
13 no deer    6.7
16 no deer    6.5
20 no deer    6.2
```

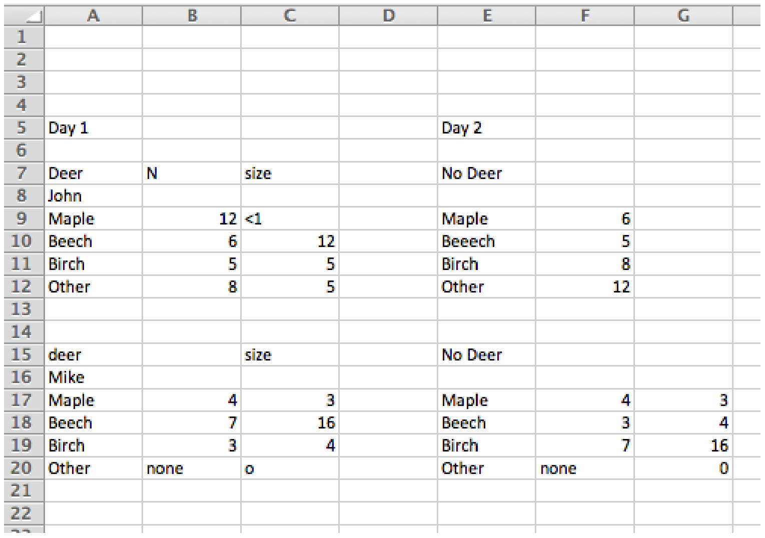
We can see that there were only four seedlings that were larger than 6 cm tall. Similarly, we can subset the data frame to just deer or no deer treatments.

```
> deer_data[deer_data$plot=="no deer",]
      plot height
11 no deer    5.9
12 no deer    6.1
13 no deer    6.7
14 no deer    4.7
15 no deer    5.7
16 no deer    6.5
17 no deer    5.8
18 no deer    3.9
19 no deer    5.6
20 no deer    6.2
```

Obviously in this case we knew that the no deer treatment was observations 1120, but you can imagine the usefulness of this with a larger dataset!

8 Organizing your data

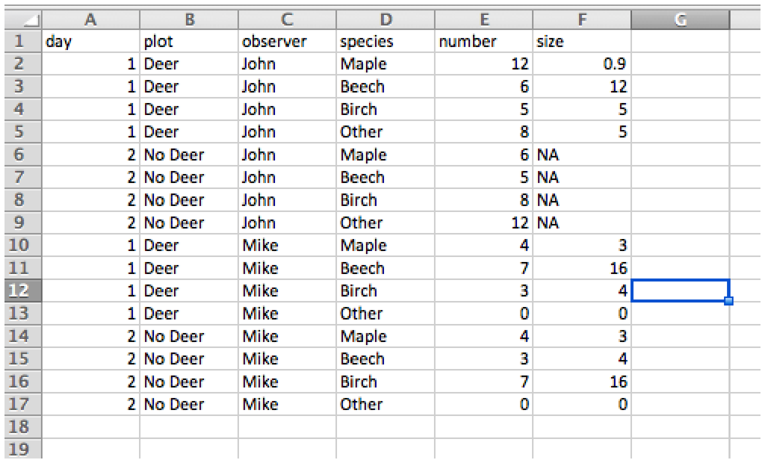
Spreadsheets (MS Excel, etc) are useful for entering data, but not necessarily for analyzing data. Their flexible nature enables the user to enter all sorts of different pages with a variety of notes and a way to store those data. Many people are terrible at organizing their data so that it can be analyzed, most times spreadsheets are simple a place to dump information. For example, look at the screenshot below.



	A	B	C	D	E	F	G
1							
2							
3							
4							
5	Day 1				Day 2		
6							
7	Deer	N	size		No Deer		
8	John						
9	Maple	12	<1		Maple	6	
10	Beech	6	12		Beeech	5	
11	Birch	5	5		Birch	8	
12	Other	8	5		Other	12	
13							
14							
15	deer		size		No Deer		
16	Mike						
17	Maple	4	3		Maple	4	3
18	Beech	7	16		Beech	3	4
19	Birch	3	4		Birch	7	16
20	Other	none	o		Other	none	0
21							
22							

Figure 1: An example of the wrong way to organize your data.

Good practice should have a consistent use of capitals and be careful of spelling (for example, see "Beeech" above) as R will treat both of these names separately. Remember, R will only do what you tell it to, so if you tell it there is a tree named "Beech" and a tree named "Beeech" it will see those separately. What we are going for is one observation per row, and placing data into a long format. As far as summary data goes (means etc) it is advisable to include only raw data in some kind of master data file. R can create summary data for you faster than you or excel. Remember to use NA for missing data.



	A	B	C	D	E	F	G
1	day	plot	observer	species	number	size	
2	1	Deer	John	Maple	12	0.9	
3	1	Deer	John	Beech	6	12	
4	1	Deer	John	Birch	5	5	
5	1	Deer	John	Other	8	5	
6	2	No Deer	John	Maple	6	NA	
7	2	No Deer	John	Beech	5	NA	
8	2	No Deer	John	Birch	8	NA	
9	2	No Deer	John	Other	12	NA	
10	1	Deer	Mike	Maple	4	3	
11	1	Deer	Mike	Beech	7	16	
12	1	Deer	Mike	Birch	3	4	
13	1	Deer	Mike	Other	0	0	
14	2	No Deer	Mike	Maple	4	3	
15	2	No Deer	Mike	Beech	3	4	
16	2	No Deer	Mike	Birch	7	16	
17	2	No Deer	Mike	Other	0	0	
18							
19							

Figure 2: An example of a much better way to organize your data.

9 Types of data in R

Data are stored in four types of modes, typically. These are "numeric", "character", "complex", or "logical" (TRUE or FALSE). It is useful to know the mode of an object or a vector for several reasons. First, R tries to be smart about determining if data that are entered are numerical, integer, logical, or characters. But R is only as smart as you are. Often times if you have mistakes in a dataset, for example, letters where numbers should be, R will interpret that one mistake as representative of the entire column and treat the vector as a character vector.

There are a number of different types of objects in R. Understanding how each of these work can be important.

How do you know what kind of data you have, or how R has interpreted or imported your data? All R objects have two characteristics, they have a length and a mode. The length is how many objects are in a vector and the mode tells you the type of data of an object.

```
> mode(height)
[1] "numeric"
> length(height)
[1] 20
> mode(plot)
[1] "character"
```

In this example, we can see that our variable plot is considered character data, whereas height is known to be numeric.

Table 1: Some various types of data objects available in `R`

Object	**Modes**	**Several modes possible?**
vector	numeric, character, logical or complex	No
factor	numeric or character	No
array	numeric, character, logical or complex	No
matrix	numeric, character, logical or complex	No
data frame	numeric, character, logical or complex	Yes
ts	numeric, character, logical or complex	No
list	numeric, character, logical, complex, function, expression...	Yes

10 Other Useful Functions

It is impossible to try and provide a comprehensive list of the functions and could use. Here is a brief list of our favorites and/or most used.

- Functions for manipulating data frames or other objects
  - `ncol()` - tells you the number of columns.
  - `nrow()` - tells you the number of rows.
  - `head()` - shows you the top 6 rows.
  - `tail()` - shows you the last 6 rows.
  - `names()` - shows you the column names and allows you to change them.
  - `colSums()` - calculate the sum of a column or columns in a data frame.
  - `colMeans()` - calculate the mean of a column or columns in a data frame.
- General functions for creating or manipulating objects
  - `seq()` - Create a sequence of numbers.
  - `rep()` - Repeat something a set number of times.
  - `trunc()` - Truncate a number to just the integer.
  - `round()` - Round a number to a set number of decimal places.
  - `rbind()` - Bind two vectors together as rows.
  - `cbind()` - Bind two vectors togethr as columns.
  - `paste()` - Stick two bits of text together.
  - `rm()` - Remove an item from the memory.
- Mathematic functions
  - `mean()` - Calculate the mean of a vector of numbers.
  - `sum()` - Calculate the sum of a vector of numbers.
  - `max()` - Find the max value in a vector.
  - `min()` - Find the min value in a vector.
  - `sd()` - Calculate the standard deviation of a vector.