RStudio Module #1 - Basic Introduction

Introduction to RStudio

The purpose of this document s to introduce you to some basic functions of R programming using the RStudio user interface.

In this short tutorial, we will cover the following topics:

1. Creating and Altering Objects
2. Creating and Altering Vectors
3. Further Functions to Use with Numerical Vectors

Creating and Altering Objects

The following items will guide you through the creation of "objects" in RStudio and then how to modify them after they have been created. Throughout these tasks, you will continually see the assign command (<-) which tells RStudio to assign some value(s) on the right-hand side of the operator to a named "object" on the left-hand side of the operator.

1. Create a new object called x and assign it a value of 25. Then create another object - y - and assign it a value of 35.

```
x<-25
y<-35
```

2. Take a look at the environment pane in the top right - you will see there are new objects - x & y - that you have just created. If you just type the name of the object in the console pane it will print the value you have assigned to it. Do so now.

```
x
```

```
## [1] 25
```

```
y
```

```
## [1] 35
```

3. Since these objects only contain single values, you can easily use the calculator functions in RStudio to perform mathematical operations (and if necessary, true/false questions) using them. Please note that RStudio is case-sensitive so typing X or Y in the console pane will yield nothing, since these objects do not exist.

4. You can also create new objects in RStudio using existing ones. As an example, create a new object - z - from objects x & y using the following code:

```
z<-(x*y)+(x^2)+sqrt(y)
z
```

```
## [1] 1505.916
```

5. Further, names for objects can be as descriptive as you like, but there are some important rules to follow. First, you may not begin an object's name with a number or have the object's name be from a set of reserved names for RStudio commands (e.g., if, for, next, break, in, etc. . .). Second, unless you place the name inside quotations you may not use spaces in object names (e.g., "variable 1"). Third, in place of spaces you may use periods or underscores (e.g., variable.1, variable_1).

```
variable.1<-(z+x)*y
"variable 1"<-(z+x)*y
variable_1<-(z+x)*y
"this name is far too long to be useful"<-(z+x)*y
```

6. You may also modify existing values of objects, and this merely involves replacing the contents with something else on the right-hand side of the assignment operator while referencing an existing object on the left-hand side of the operator. Below, we will convert the values of x and y to percentages, then store these values in preexisting objects.

```
x<-(x/(x+y))*100
y<-(y/(x+y))*100
x
```

```
## [1] 41.66667
```

```
y
```

```
## [1] 45.65217
```

It is important to note that this will override the original values!!! If you wanted to still use the original values of these variables, you would need to re-run the code in the first item of this tutorial to get the variables back to their original values.

Creating and Altering Vectors

Typically, you will want to assign several values to an object, if not very, very many. In this case, vectors become a useful tool to store data in an R object (but data frames are even more useful – more on this later on). A necessary command for creating vectors is c(), which stands for "combine" or "concatenate". This command will incorporate the various values you assign a vector object into a single row or column, that can be further indexed by its position in the vector (e.g., 1st position [1], 2nd position [2], 3rd position [3], etc. . .).

1. We will begin by entering the number of homicides in the city of Charlotte for the past several years. Begin by assigning a name to the vector – homicide.count – and follow the assignment operator (<-) with the following values – 60, 44, 58, 52, 55, 59, 55, 83.

```
homicide.count<-c(60,44,58,52,55,59)
```

2. The homicide.count object now appears in the environment pane, and it will contain 6 elements.
3. Prnt the content of homicide.count by typing the following into the console pane (or an R script that you have opened, preferably).

```
homicide.count
```

```
## [1] 60 44 58 52 55 59
```

4. Now, let's say we want to reference the third item in homicide.count - I can do so by typing the following into the console pane:

```
homicide.count[3]
```

```
## [1] 58
```

5. Or, we can reference the fifth element by typing the following into the console pane:

```
homicide.count[5]
```

```
## [1] 55
```

6. Let's say we wanted to sort homicide counts in ascending order. We can do so by typing the following into the console pane:

```
sort(homicide.count)
```

```
## [1] 44 52 55 58 59 60
```

Note that this does not alter the actual order of the elements in the stored object, it merely prints them out in ascending order. If we wanted to store this sort order, we would have to use an assignment operator along with the function typed above (do not do this right now!).

7. Finally, let's say that we found that we had made a mistake in entering the data and the third element is actually supposed to be 60 and not 58. We can make that adjustment by referencing the third element in the homicide.count object and, using an assignment operator, we can assign a new value for just that element.

```
homicide.count[3]<-60
homicide.count
```

```
## [1] 60 44 60 52 55 59
```

8. Further, we forgot to add values for prior years and we need to add them now. You can add values to a vector object by referencing new elements to be added after existing ones like so:

```
homicide.count[7:8]<-c(42,55)
homicide.count
```

```
## [1] 60 44 60 52 55 59 42 55
```

Further Functions to Use with Numerical Vectors

We will rarely be dealing with vectors containing text or "string" characters (and even then, text values will have an underlying numerical code) so it is helpful to know some additional operations you can use on numeric vectors.

1. The min( ) function ("Minimum"): This will return the minimum value in a numeric vector. Let's use this function now to find the minimum number of homicides in Charlotte over the past 10 years:

```
min(homicide.count)
```

```
## [1] 42
```

2. The max( ) function ("Maximum"): This will return the maximum value in a numeric vector. Let's use this function now to find the minimum number of homicides in Charlotte over the past 10 years:

```
max(homicide.count)
```

```
## [1] 60
```

3. The mean( ) function ("Average"): This will return the average value in a numeric vector. Let's use this function now to find the average number of homicides per year in Charlotte over the past 10 years:

```
mean(homicide.count)
```

```
## [1] 53.375
```

4. The median( ) function ("Median"): This will return the value at the 50th percentile of the numeric vector so here somewhere between the 5th and 6th values when they are placed in ascending order. Let's use this function now to find the median homicide count for Charlotte over the past 10 years:

```
median(homicide.count)
```

```
## [1] 55
```

5. The sum( ) function ("Total"): This will return the total cumulative value in a numeric string. Let's use this function now to find the total number of homicides in Charlotte over the past 10 years:

```
sum(homicide.count)
```

```
## [1] 427
```

6. The table( ) function ("Frequency"): This will return a frequency table containing all the unique values in the homicide.count vector and the number of times they appear. Let's use this function now to see the frequency table for homicides in Charlotte over the past 10 years:

```
table(homicide.count)
```

```
## homicide.count
## 42 44 52 55 59 60
##  1  1  1  2  1  2
```

7. The range( ) function ("Range"): This will return the minimum and maximum values within the vector at the same time, so it is the same as running both the min( ) and max( ) functions at once. Let's use this function now to see the range of homicide values in Charlotte over the past 10 years:

```
range(homicide.count)
```

```
## [1] 42 60
```

8. The sd( ) function (“Standard Deviation”): This will return the standard deviation about the mean for values within a numeric vector. The standard deviation captures the expected difference from the mean you could expect any given observation to be, on average, within a particular set of data or a numeric vector. Let’s use this function now to see the standard deviation of homicide counts in Charlotte over the past 10 years:

```r
sd(homicide.count)
```

```
## [1] 7.008923
```

Final note - if you remember your measures of central tendency from your statistics course, we are missing one here - the mode. This is because R has no pre-defined function to compute the mode. However, you can easily identify the mode using the table() function or you can write your own function to compute the mode. Since the mode is not a very helpful measure of central tendency (for a variety of reasons) we can stick with identifying it using the table() function for now.