Lab #1 Activities

Adejare Taiwo

The R code we have covered in class is available on the lecture section UM Learn page, under **Content** > **Course Material**.

Question 1:

(a) In R, create a vector (call the vector **x**) of the five values 2, 5, 1, 4 and 7. Use R's built-in function `mean()` to find the mean of these five values. (We say we are *passing in* the **x** vector to the `mean` function.)

```
x = c(2, 5, 1, 4, 7)
mean(x);
```

```
## [1] 3.8
```

(b) Suppose we have surveyed four respondents to collect their responses on some variable of interest. Only three of them reply, so we have a missing value for the fourth person. Missing values are coded as NA in R. We can create the vector of responses: `y = c(2,5,1,NA)`. What does R tell you is the mean of this vector **y**?

```
y = c(2, 5, 1, NA)
mean(y)
```

```
## [1] NA
```

(c) We can find the mean of the non-missing values for the vector in part (b) using extra options to R's `mean` function. The extra options are called **arguments**. To see what arguments are available for a function, we need to access the help documentation. In the R console, type `?mean` or `help(mean)` and press enter (or return). The documentation will open up in the bottom right corner of RStudio. Write the R code that will compute the mean of **y**, omitting missing values, using the proper argument to the `mean` function.

```
y = c(2, 5, 1, NA)
mean(y, na.rm = TRUE)
```

```
## [1] 2.666667
```

Notice that typing `mean(0.75,0.25)` in R will not return 0.5 (the mean of 0.75 and 0.25). That is because R is interpreting only the value 0.75 as belonging to the data set of values of which you want the mean. It is interpreting 0.25 as the value of the **trim** argument that is shown in the help documentation (we will not be working with the **trim** argument). In order to get the mean of 0.75 and 0.25, the proper code is `mean(c(0.75,0.25))` or we must store 0.75 and 0.25 in a vector and then find the mean of that vector, as we did in part (a).

Question 2:

(a) There are two datasets built-in to R named **state.area** and **state.name** referring to the 50 U.S. states. Type these names at the R console to see the data they contain. The datasets are linked, so, for example, the first component of **state.area** (which is the value 51609) gives you the area in square miles of the first state in **state.name** (which is Alabama).

Observing the output of **state.name**, we see that California is the 5th state. Therefore, we can access California's area (in square miles) by accessing the 5th component of **state.area**:

```
state.area[5]
```

```
## [1] 158693
```

Write the R code that obtains the area in square miles of Nevada.

```
state.area[28]
```

```
## [1] 110540
```

(b) A third related dataset is named **state.region**, for which the order of the components also corresponds to the order in **state.name**. Write the R code that obtains the region of the U.S. that the state of Iowa is in, based on this dataset. (When you extract the appropriate region, you may also get a list of all regions, which is okay.)

```
state.region[15]
```

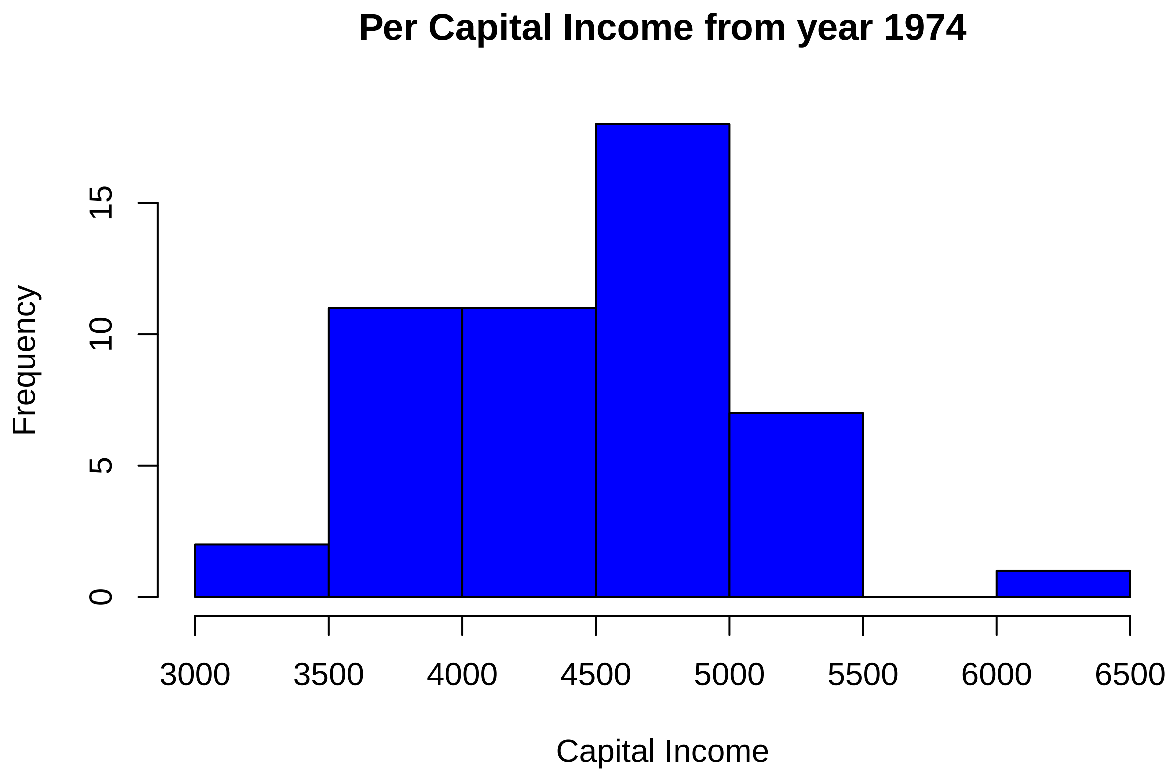
```
## [1] North Central
## Levels: Northeast South North Central West
```

(c) A fourth related dataset is named **state.x77**. Type `state.x77` at the R console to see the data. The data is longer, so you can scroll back up to see the beginning of the data. This data is stored in a matrix. We will extract the per capita incomes (data is from the year 1974) from this dataset with the following code. We are extracting the second column of **state.x77**, where the incomes reside. (The state names are not in any column; rather, they are the names of the rows.)

```
incomes = state.x77[,2]
```

Create a histogram of the `incomes` object. Type `?hist` at the console to see arguments that can be added to the `hist()` function to enhance your histogram. Add at least one enhancement (change the title of the histogram, an axis label, or the colour of the bars).

```
?hist
hist(incomes, col = "blue", main = "Per Capital Income from year 1974", xlab = "Capital Income")
```



Question 3:

(a) Create a vector in R with 7 components: three TRUEs and four FALSEs. (The order of the TRUEs and FALSEs does not matter). Pass in this vector to the `sum()` function and report the output.

```
vt = c(TRUE, TRUE, TRUE, FALSE, FALSE, FALSE)
sum(vt);
```

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

(b) Create a vector in R with 5 components: two TRUEs and three FALSEs. (The order of the TRUEs and FALSEs does not matter). Pass in this vector to the `sum()` function and report the output.

```
vt1 = c(TRUE, TRUE, FALSE, FALSE, FALSE)
sum(vt1)
```

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

(c) Create a vector in R with 3 components: one TRUE and two FALSEs. (The order of the TRUEs and FALSEs does not matter). Pass in this vector to the `sum()` function and report the output.

```
vt2 = c(TRUE, FALSE, FALSE)
sum(vt2)
```

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

(d) Create a vector in R with 4 components: zero TRUEs and all four FALSEs. Pass in this vector to the `sum()` function and report the output.

```r
vt3 = c(FALSE, FALSE, FALSE, FALSE)
sum(vt3)
```

```
## [1] 0
```

(e) Based on these results, what does it seem the `sum()` function is doing when it is given a vector of TRUEs and FALSEs?

Base on the above result i see that sum adds up all the true values and return the result omitting the false object.