```
> ## ERIC AGYEMANG
> ## IT 497  Lab #1
>
> # R is a statistical programming language
> # and much more!!!
>
> # We will start with the basics
>
> # You can use R like a calculator
>
> x <- 1
> y <- 2
> z <- x + y
> z
[1] 3
> str(z)
 num 3
>
> # First, let’s read in heart attack payment data.
>
> # This data comes from The United States Department of Health and Human
> # Services (HHS)
> # http://www.healthdata.gov/dataset/heart-attack-payment-national
>
> # I have downloaded a copy of the data to ISU's sever to make it simple
>
> df <- read.table("http://www.itk.ilstu.edu/faculty/jrwolf/hacosts.csv",
+                  header = TRUE, sep = ",")
> # In the above df is the name of our data frame. A data frame is used for
storing
> # data tables.
>
> # It is a list of vectors of equal length.
>
> # We can display the contents of our data frame by typing its name and
selection
> # run.
> df
   State     Cost
1     AK 20987.60
2     AL 21850.32
3     AR 21758.00
4     AZ 22690.62
5     CA 22707.45
6     CO 21795.30
7     CT 22712.70
8     DC 22292.43
9     DE 22655.83
10    FL 22719.42
11    GA 21305.17
12    HI 20850.50
13    IA 21045.11
14    ID 21476.89
15    IL 22553.83
16    IN 22191.22
17    KS 22215.52
18    KY 22262.04
```

```
19    LA 21591.36
20    MA 22452.12
21    MD 21472.65
22    ME 21002.10
23    MI 21742.48
24    MN 21505.48
25    MO 22166.10
26    MS 21759.93
27    MT 21066.00
28    NC 20794.19
29    ND 22343.14
30    NE 23022.83
31    NH 21982.56
32    NJ 23754.00
33    NM 20721.09
34    NV 23548.79
35    NY 21953.64
36    OH 22628.79
37    OK 21257.56
38    OR 20926.61
39    PA 22224.06
40    RI 22133.50
41    SC 21583.75
42    SD 21085.50
43    TN 21472.25
44    TX 22725.69
45    UT 22896.21
46    VA 21416.67
47    VT 20399.90
48    WA 21509.73
49    WI 22185.89
50    WV 21890.88
51    WY 22916.50
> 
> # The top line of the table is called the header.
> # The header contains the column names.
> 
> names(df)
[1] "State" "Cost"
> head(df, 10)
   State     Cost
1     AK 20987.60
2     AL 21850.32
3     AR 21758.00
4     AZ 22690.62
5     CA 22707.45
6     CO 21795.30
7     CT 22712.70
8     DC 22292.43
9     DE 22655.83
10    FL 22719.42
> tail(df)
   State     Cost
46    VA 21416.67
47    VT 20399.90
48    WA 21509.73
49    WI 22185.89
```

```
50    WV 21890.88
51    WY 22916.50
> tail(df, 8)
   State     Cost
44    TX 22725.69
45    UT 22896.21
46    VA 21416.67
47    VT 20399.90
48    WA 21509.73
49    WI 22185.89
50    WV 21890.88
51    WY 22916.50
> head(df, 23)
   State     Cost
1     AK 20987.60
2     AL 21850.32
3     AR 21758.00
4     AZ 22690.62
5     CA 22707.45
6     CO 21795.30
7     CT 22712.70
8     DC 22292.43
9     DE 22655.83
10    FL 22719.42
11    GA 21305.17
12    HI 20850.50
13    IA 21045.11
14    ID 21476.89
15    IL 22553.83
16    IN 22191.22
17    KS 22215.52
18    KY 22262.04
19    LA 21591.36
20    MA 22452.12
21    MD 21472.65
22    ME 21002.10
23    MI 21742.48
> nrow(df)
[1] 51
> ncol(df)
[1] 2
> str(df)
'data.frame':   51 obs. of  2 variables:
 $ State: Factor w/ 51 levels "AK","AL","AR",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ Cost : num  20988 21850 21758 22691 22707 ...
>
> # We can access specific data frame columns by name
>
> df$State # When accessing columns by name, we use the data frame name
followed by the $ symbol and the column name
 [1] AK AL AR AZ CA CO CT DC DE FL GA HI IA ID IL IN KS KY LA MA MD ME MI MN
MO MS MT NC ND NE NH NJ NM NV NY OH
[37] OK OR PA RI SC SD TN TX UT VA VT WA WI WV WY
51 Levels: AK AL AR AZ CA CO CT DC DE FL GA HI IA ID IL IN KS KY LA MA MD ME
MI MN MO MS MT NC ND NE NH ... WY
> df$Cost
```

```
 [1] 20987.60 21850.32 21758.00 22690.62 22707.45 21795.30 22712.70 22292.43
22655.83 22719.42 21305.17 20850.50
[13] 21045.11 21476.89 22553.83 22191.22 22215.52 22262.04 21591.36 22452.12
21472.65 21002.10 21742.48 21505.48
[25] 22166.10 21759.93 21066.00 20794.19 22343.14 23022.83 21982.56 23754.00
20721.09 23548.79 21953.64 22628.79
[37] 21257.56 20926.61 22224.06 22133.50 21583.75 21085.50 21472.25 22725.69
22896.21 21416.67 20399.90 21509.73
[49] 22185.89 21890.88 22916.50
> mean(df$Cost) # We can perform operations on specific columns
[1] 21925.53
> min(df$Cost)
[1] 20399.9
> max(df$Cost)
[1] 23754
> sd(df$Cost)
[1] 752.1785
>
> # Each horizontal line below the header is called a data
> # row. Each data element of a row is called a cell.
> # We can also access columns, rows and even cells by
> # location
>
> df$Cost[2] # This returns the cost cell in the second row
[1] 21850.32
> df$Cost[2:5] # This returns the cost cells in rows 2 through 5
[1] 21850.32 21758.00 22690.62 22707.45
> df[5,] # This returns all of the rows in column 5
  State     Cost
5    CA 22707.45
> df[1:5,] # This returns all of the rows in columns 1 through 5
  State     Cost
1    AK 20987.60
2    AL 21850.32
3    AR 21758.00
4    AZ 22690.62
5    CA 22707.45
> df[5,1] # This returns the data in columns 5 row 1
[1] CA
51 Levels: AK AL AR AZ CA CO CT DC DE FL GA HI IA ID IL IN KS KY LA MA MD ME
MI MN MO MS MT NC ND NE NH ... WY
> df[5,2] # This returns the data in columns 5 row 2
[1] 22707.45
>
> # We can find the minimum and maximum values in each column
>
> which.min(df$Cost) # this returns the row of the minimum value
[1] 47
> which.max(df$Cost) # this returns the row of the maximum value
[1] 32
>
> df[which.min(df$Cost), ] # this returns the minimum value
   State    Cost
47    VT 20399.9
> df[which.max(df$Cost), ] # this returns the maximum value
   State  Cost
32    NJ 23754
```

```
>
> # We can sort the data in our data frame by column
> df <- df[order(df$Cost), ] # the default is ascending
> head(df)
   State     Cost
47    VT 20399.90
33    NM 20721.09
28    NC 20794.19
12    HI 20850.50
38    OR 20926.61
1     AK 20987.60
>
> high <- df[1:5,] # This assigns the data from the first 5 rows to a data
frame named high
>
> df <- df[order(-df$Cost), ] # to sort in descending order
> head(df)
   State     Cost
32    NJ 23754.00
34    NV 23548.79
30    NE 23022.83
51    WY 22916.50
45    UT 22896.21
44    TX 22725.69
>
> low <- df[1:5,] # This assigns the data from the first 5 rows
>                 # to a data frame named low
>
> ################################
> # Let’s move a bit faster
>
> # read in fitbit data
> df <-read.csv("http://www.itk.ilstu.edu/faculty/jrwolf/fitbitstats.csv",
stringsAsFactors=F)
>
> # Convert the Steps to numeric
> df$Steps <- as.numeric(df$Steps)
> # Look at the structure of your data frame
> str(df)
'data.frame':  1052 obs. of  4 variables:
 $ id      : int  1 2 3 4 5 7 8 9 10 11 ...
 $ Steps   : num  7073 4175 5287 36905 8862 ...
 $ Distance: num  3 1.8 2.2 17.4 4.1 6 3.5 2.3 5 3.2 ...
 $ Scale   : chr  "miles" "miles" "miles" "miles" ...
>
> # Check minimums and maximums
>
> which.min(df$Steps)
[1] 844
> which.max(df$Steps)
[1] 12
>
> df[which.min(df$Steps), ]
     id Steps Distance Scale
844 881     6        0 miles
> df[which.max(df$Steps), ]
   id Steps Distance Scale
```

```
12 13 49771     21.7 miles
>
>
#############################################################################
###############
>
> # Now using the fitbit data answer the following
>
> # 1. Look at the first 6 rows of data
> head(df, 6)
  id Steps Distance Scale
1  1  7073      3.0 miles
2  2  4175      1.8 miles
3  3  5287      2.2 miles
4  4 36905     17.4 miles
5  5  8862      4.1 miles
6  7 12281      6.0 miles
>
> # 2. Look at the last 6 rows of data
> tail(df, 6)
       id Steps Distance Scale
1047 1094 17313      8.2 miles
1048 1095  9634      4.3 miles
1049 1096   969      0.4 miles
1050 1097 14201      6.4 miles
1051 1098 13091      6.2 miles
1052 1099 16155      6.3 miles
>
> # 3. Find the maximum and minimum number of steps taken by FitBit users in
our data,
> # maximum number of steps taken by FitBit users in our data
> df1<-df[which.max(df$Steps), ] # This returns the row of the maximum number
of steps reguired
> df1
   id Steps Distance Scale
12 13 49771     21.7 miles
> df1$Steps # This returns the maximum number of steps required as chosen
from the returned row of the maximum number of steps above.
[1] 49771
> #As the required maximum number of steps
> ## Or
> df[12, 2]
[1] 49771
> #As the required maximum number of steps
>
> # minimum number of steps taken by FitBit users in our data
> df2<-df[which.min(df$Steps), ] # This returns the row of the minimum number
of steps reguired
> df2
     id Steps Distance Scale
844 881     6        0 miles
> df2$Steps # This returns the minimum number of steps required as chosen
from the returned row of the minimum number of steps above.
[1] 6
> #As the required minimum number of steps
> ## Or
> df[844, 2]
```

```
[1] 6
> #As the required minimum number of steps
>
> # 4. Find the total number or rows in the data
> nrow(df)
[1] 1052
>
> # 5. Send me your code and your answers via Reggienet
```