STAT 206 Homework 1

Due Monday, October 9, 5:00 PM

General instructions for homework: Homework must be completed as an R Markdown file. Be sure to include your name in the file. Give the commands to answer each question in its own code block, which will also produce plots that will be automatically embedded in the output file. Each answer must be supported by written statements as well as any code used. (Examining your various objects in the "Environment" section of RStudio is insufficient – you must use scripted commands.)

1. The data set at [http://www.stats.uwo.ca/faculty/braun/data/rnf6080.dat] records hourly rainfall at a certain location in Canada, every day from 1960 to 1980.
   a. First, we need to load the data set into R using the command `read.table()`. Use the help function to learn what arguments this function takes. Once you have the necessary input, load the data set into R and make it a data frame called `rain.df`.

```
www = "http://www.stats.uwo.ca/faculty/braun/data/rnf6080.dat" #creating a object
#variable to hold the string of the website with the data to pass to the
#read.table function as the first argument
rain.df = read.table(www) #creating the rain.df object name for the dataframe
```

   b. How many rows and columns does `rain.df` have? (If there are not 5070 rows and 27 columns, something is wrong; check the previous part to see what might have gone wrong in the previous part.)

```
dim(rain.df)[1] # calls the dim function to retrieve the number of
```

```
## [1] 5070
```

```
#rows based on the argument [1]
dim(rain.df)[2] # calls the dim function to retrieve the number
```

```
## [1] 27
```

```
#of rows based on the argument [2]
```

   c. What are the names of the columns of `rain.df`?

```
colnames(rain.df) #calls the function colnames to retrieve the column
```

```
##  [1] "V1"  "V2"  "V3"  "V4"  "V5"  "V6"  "V7"  "V8"  "V9"  "V10" "V11"
## [12] "V12" "V13" "V14" "V15" "V16" "V17" "V18" "V19" "V20" "V21" "V22"
## [23] "V23" "V24" "V25" "V26" "V27"
```

```
#names from rain.df. In this case the column names are generic and not descriptive.
```

   d. What is the value of row 5, column 7 of `rain.df`?

```
#the brackets [] are used to extract specific subsets of data from vectors,
#dataframes, matrices and arrays

rain.df[5,7]
```

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

   e. Display the second row of `rain.df` in its entirety.

```
#again since we are extracting data (a subset) from the dataframe we will
#use brackets[] - convention is for rows first, then columns, omiting the
#column after the comma returns all columns,

rain.df[2,]
```

```
##   V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20
```

```
## 2 60  4  2  0  0  0  0  0  0   0   0   0   0   0   0   0   0   0   0   0
##   V21 V22 V23 V24 V25 V26 V27
## 2   0   0   0   0   0   0   0
```

f. Explain what this command does:
`names(rain.df) <- c("year","month","day",seq(0,23))`
by running it on your data and examining the object. (You may find the display functions `head()` and `tail()` useful here.) Is it clear now what the last 24 columns represent?

```
names(rain.df) <- c("year","month","day",seq(0,23))
head(rain.df)
```

```
##   year month day 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
## 1   60     4   1 0 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0
## 2   60     4   2 0 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0
## 3   60     4   3 0 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0
## 4   60     4   4 0 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0
## 5   60     4   5 0 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0
## 6   60     4   6 0 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0
##   22 23
## 1  0  0
## 2  0  0
## 3  0  0
## 4  0  0
## 5  0  0
## 6  0  0
```

```
tail(rain.df)
```

```
##      year month day 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
## 5065   80    11  25 0 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 5066   80    11  26 0 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 5067   80    11  27 0 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 5068   80    11  28 0 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 5069   80    11  29 0 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
## 5070   80    11  30 0 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0
##      21 22 23
## 5065  0  0  0
## 5066  0  0  0
## 5067  0  0  0
## 5068  0  0  0
## 5069  0  0  0
## 5070  0  0  0
```

```
# the given fuction and inputs for name are to set the names of all the
#columns, the head and tail functions make this clear. The last 24 columns
#most likely represent the hours of the day
```

g. Create a new column in the data frame called `daily`, which is the sum of the rightmost 24 columns. W

```r
#creating a subset of the data which will have all rows but only the
#columns for the hourly data

hourly = rain.df[,4:27]
```

```
hourly_means = rowMeans(hourly) #calculating the mean by row gives the average per day

  #The newly created vector will be appended on to the rain.df using the cbind function

rain.df = cbind(rain.df, hourly_means)

# hist(rain.df[hourly_means])

#The histogram cannot be displayed because there is a mix with negative subscripts
#the function above has been commented out to export pdf
```
h. Create a new data frame `rain.df.fixed` that takes the original and fixes it for the apparent flaw y

```r
#the negative values from the hourly means does not make sense. There is no such
#thing as negative rainfall. These values will be addressed with the following code

hourly_means_fixed = hourly_means[hourly_means < 0] <- NA

rain.df.fixed = cbind(rain.df, hourly_means_fixed)


hist(hourly_means)
```


2. Syntax and class-typing.
    a. For each of the following commands, either explain why they should be errors, or explain the non-erroneous result.

```
vector1 <- c("5", "12", "7", "32")
max(vector1)
sort(vector1)
sum(vector1)
```

```
#The first line creates a vector but assigns it string values instead of numberical values
#The max function will still work but will not return the largest values as expected
#The sort function will return an order but it will not be in a numerically logical order
#The sum function will simply error out because it cannot add strings together.
#It will not even concatenate it like you would see as possible in Python (i.e "a" + "b" yields "ab")
```

b. For the next series of commands, either explain their results, or why they should produce errors.

```
vector2 <- c("5",7,12)
vector2[2] + vector2[3]

dataframe3 <- data.frame(z1="5",z2=7,z3=12)
dataframe3[1,2] + dataframe3[1,3]

list4 <- list(z1="6", z2=42, z3="49", z4=126)
list4[[2]]+list4[[4]]
list4[2]+list4[4]
```

```
#Vector two is stored with a mix of characters and strings as a result the
#vector class is "character" vector[2] and vector[3] are set to  "character"
```

```
#as well there for they cannot be added
#as stated above the second line of code will error out because it is not
#adding numeric values

#dataframes can carry a mix of strings and numeric values therefore this
#is not a 1x3 matrix but a 1x3 dataframe
#from the dataframe the 2nd and 3rd values are extracted and added together
#which is fine, resulting in 19

#lists can also have a mix of strings and values. Therefore we can store multiple
#classes to a list type
#in the second line of code we are able to add the numbers together because we are
#extracting the numeric values within the list using the brackets []
#in the 3rd line we cannot add these together because we cannot do arithmitic ON
#lists, we can only do arithmatic WITH elements of the list. For example  we cannot
#have list4 + 1
```

3. Working with functions and operators.
   a. The colon operator will create a sequence of integers in order. It is a special case of the function `seq()` which you saw earlier in this assignment. Using the help command `help(seq)` to learn about the function, design an expression that will give you the sequence of numbers from 1 to 10000 in increments of 372. Design another that will give you a sequence between 1 and 10000 that is exactly 50 numbers in length.

```
#using the seq function we assign a vector to first as a sequence of numbers
#FROM 1, TO 10000 BY an increment of 372

first = seq(1, 10000, by = 372)
#first

#to specify the length of the vector using seq we use the argument "length.out"
second = seq(1, 10000, length.out = 50)
#second
```

   b. The function `rep()` repeats a vector some number of times. Explain the difference between 'rep(1:3, times=3) and rep(1:3, each=3).

```
#the times argument repeates the sequence of numbers a given amount of times.

rep(1:3, times=3)
```

```
## [1] 1 2 3 1 2 3 1 2 3
```

```
#the each argument repeats each numeric value a number of times before moving
#on to the next number in the sequence and repeating that the same number of times

rep(1:3, each=3)
```

```
## [1] 1 1 1 2 2 2 3 3 3
```