# Homework 1

Alexandra Thursland

9/1/2020

**R Markdown Test**

**Question 0** R Markdown file output set to HTML. Document successfully knits. Question 0 complete.

**Working with data**

**Question 1 a)**

```
rain.df <- read.table(file="rnf6080.dat")
head(rain.df)
```

```
##   V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20 V21
## 1 60  4  1  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
## 3 60  4  3  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
## 5 60  4  5  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
##   V22 V23 V24 V25 V26 V27
## 1   0   0   0   0   0   0
## 2   0   0   0   0   0   0
## 3   0   0   0   0   0   0
## 4   0   0   0   0   0   0
## 5   0   0   0   0   0   0
## 6   0   0   0   0   0   0
```

Task complete. I loaded the data set into R and saved it as a data frame using the *read.table*() function. I verified my results by using the *head*() function.

**b)**

```
nrow(rain.df)
```

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

```
ncol(rain.df)
```

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

Task complete. There are 5070 rows and 27 columns in the rain.df data frame. I know because I used the *nrow*() and *ncol*() functions.

**c)**

```
colnames(rain.df)
```

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

Task complete. To get the column names of rain.df, I used the *colnames*() function. The names of the columns in rain.df are V1, V2, V3, ... V27.

**d)**

```
rain.df[2,4]
```

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

Task complete. To get the value of the cell at row 2, column 4, I used the following syntax: $rain.df[2, 4]$. The value of the cell at row 2, column 4 is 0.

**e)**

```
rain.df[2,]
```

```
##   V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20 V21
## 2 60  4  2  0  0  0  0  0  0   0   0   0   0   0   0   0   0   0   0   0   0
##   V22 V23 V24 V25 V26 V27
## 2   0   0   0   0   0   0
```

Task complete. To get the values of the entire second row, I used the following syntax: $rain.df[2, ]$. The content of that row is 60, 4, 2, and then 24 subsequent zeroes.

**f)**

```
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 22 23
## 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  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  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  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  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  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  0  0
```

By running the code provided and then verifying the changes to rain.df by running the *head*() function, I determined that $names(rain.df) < -c(\text{"}year\text{"}, \text{"}month\text{"}, \text{"}day\text{"}, seq(0, 23))$ changed all of the column names in the data frame. The first three columns were renamed "year", "month", and "day" respectively, and the subsequent 24 columns were named 0 through 23, indicating hours of the day.

**g)**

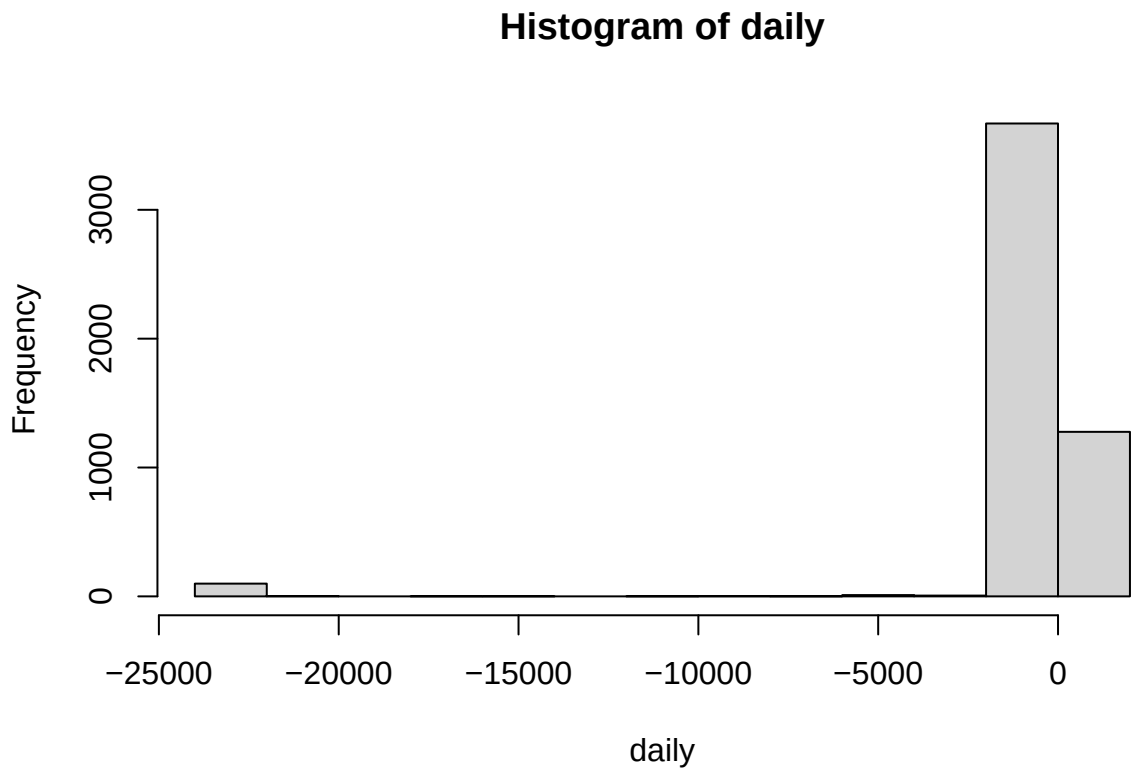
```
cols <- c(4:27)
rain.df$daily <- rowSums(rain.df[,cols])
head(rain.df)
```

```
##   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 22 23
## 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  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  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  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  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  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  0
##   daily
## 1     0
## 2     0
## 3     0
## 4     0
## 5     0
## 6     0
```

Task complete.

**h)**

```
daily <- rain.df$daily
hist(daily)
```




Task complete. Created the histogram using the *hist*() function on the daily column vector.

**i)**

This cannot possibly be right because, for some reason, there are large negative values apparent on the histogram. Daily rainfall amounts cannot be negative. Upon looking more closely at the data frame, it appears that hours for which rainfall was not recorded were input as -999 rather than NA. This accounts for the error.

**j)**

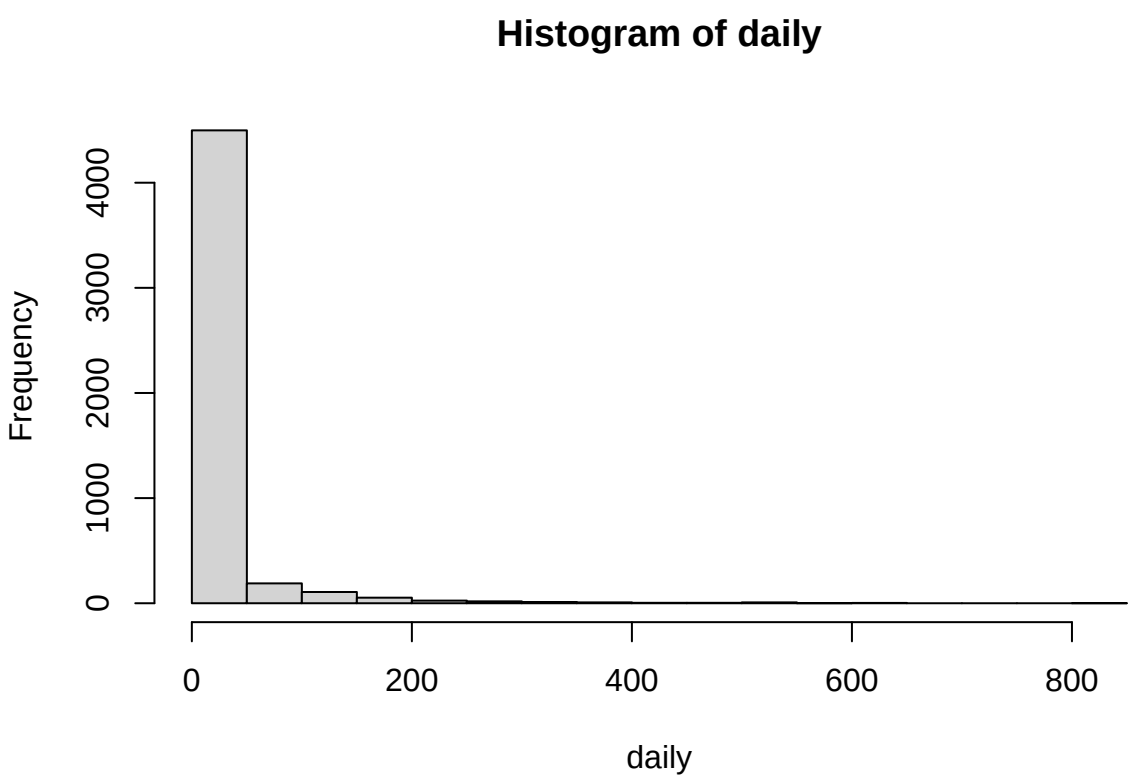
```
rain.df <- na_if(rain.df, -999)
cols <- c(4:27)
rain.df$daily <- rowSums(rain.df[,cols])
head(rain.df)
```

```
##   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 22 23
## 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  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  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  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  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  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  0  0
##   daily
## 1     0
## 2     0
## 3     0
## 4     0
## 5     0
## 6     0
```

Task complete. From the *dplyr* package, I used the function $na_if()$ to replace all instances of the value -999 across all columns with NAs. Then I re-calculated all of the values for the daily column using the same code I produced in part g.

**k)**

```
daily <- rain.df$daily
hist(daily)
```




Task complete. By replacing the instances of -999 with NAs, we have achieved a reasonable looking histogram.

**Data types**

**Question 2 a)**

$x <- c(\text{"5"}, \text{"12"}, \text{"7"})$: No error. Produces a vector of characters "5", "12", and "7".

$max(x)$: No error. Produces "7". When $max()$ is used on a character vector, it sorts the characters lexicographically.

$sort(x)$: No error. Produces "12" "5" "7". Like $max()$, when $sort()$ is used on a character vector, it sorts the characters lexicographically in ascending order.

$sum(x)$: Produces an error. This is because the $sum()$ function can only be used to sum numeric data types, and the vecvtor x is an vector of character types (indicated by the quotes around the numbers).

**b)**

$y <- c(\text{"5"}, 7, 12)$: No error. Produces a character vector because vectors in R must all be of the same data type and the command attempted to use both character and numeric elements. Therefore, R coerces all elements in the vector into the character data type.

$y[2] + y[3]$: Produces an error. This is because, as previously stated, all the elements of vector y were coerced into characters and one cannot perform arithmetic on character data types.

**c)**

$z <- data.frame(z1 = \text{"5"}, z2 = 7, z3 = 12)$: No error. Produces a data frame with column names z1, z2, and z3, and one row with values "5", 7, and 12.

$z[1,2] + z[1,3]$: No error. Unlike with vectors, data frames can have a variety of different data types, and as long as the two values being operated on are numeric, we can perform arithmetic.

**Question 3 a)**

Reproducibility is important because it allows us to check our code for correctness, enables others to re-run our code and verify that they get the same results, and also provides a means for exactly regenerating data that has been lost and/or generating more data that matches results we already have.

*Correctness:* If your code is generating different results every time, it may be impossible to know which results (if any) are actually correct. This is important for obvious reasons.

*Outside verification:* When code is reproducible, this enables others to run it on their own machines and verify they get the same results. It contributes to verifying correctness and also generally brings ease to collaborative projects.

*Data (re)generation:* Code that can produce the same exact output every time is useful for generating and regenerating data that we might need for a project. This is important because data is the backbone of statisticsl analysis and there needs to be some way to access and/or produce it.

**b)**

There are a lot of reasons why making reproducible code is important to know for this class and our future studies. One example applicable to all statistics courses is the following: reproducibility makes it much easier for teaching assistants and professors to grade our work. If your code generates different results every time, then the results you reference in your write-up may not be the same results that a TA gets when they run the code on their own machine. This makes it harder for a TA to verify that the results you originally produced are actually correct, and might cause you to lose points. In this course, reproducibility is a part of our homework grade, so you would definitely lose points.

**c)**

**2**

I have a lot of coursework and personal project experience working with R, so I found this assignment to be pretty easy. I didn't struggle with any portion of the assignment and I completed the entire thing fairly quickly. Taking reproducibility into such strong account in my coding and write-up was definitely a little new to me, because it's something I never really thought about, but I don't think that this will be an issue now or moving forward—just something new.

**References**

Why is reproducibility important?, workshop chaired by James Perry

Replacing values with NA, documentation by Nicholas Tierney