Question 1

a.

```
# Use read.table to create dataframe from .dat file
# File path is just "rnf6080.dat" because dat is located in same dir as Rmd
rain_df = read.table("rnf6080.dat")
```

I used the `read.table` command to read in the data into the data frame.

b.

```
# Use dim function to get rows, columns of dataframe
dim(rain_df)
```

```
## [1] 5070   27
```

There are 5070 rows and 27 columns in the `rain_df`. I know this by using the `dim()` function, which gives the dimensions of a dataframe.

c.

```
# Get column names using names
names(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"
```

I used the `names()` function to get the names of the columns of `rain_df`. Because the `rnf6080.dat` file did not specify column names, R automatically set the column names to be `V1, V2, V3, ..., V27`.

d.

```
# Get value at row 2, column 4
rain_df[2, 4]
```

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

This is the command I used to get the value in the 2nd row, 4th column. The value is 0.

e.

```
# Get values in second row
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
```

The code block above contains the command I used to display the whole second row. There is a 60 in column 1, 4 in column 2, 2 in column 3, and 0 in the rest of the columns. This represents the recorded hourly rainfall on 4/2/1960 in the certain location in Canada.

f.

```
# Change column names to year, month, day, 0, 1, ..., 23
names(rain_df) <- c("year","month","day",seq(0,23))
```

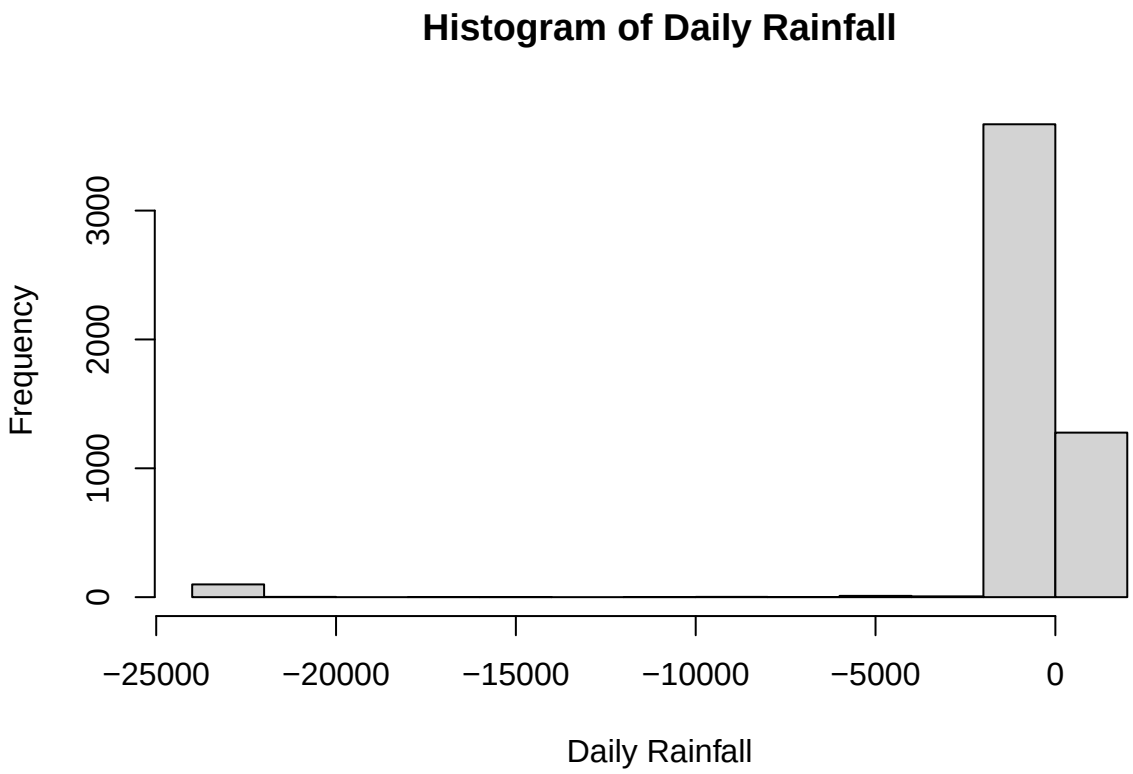
This command sets the column names of the `rain_df` dataframe to be `year` for the first column, `month` for the second column, `day` for the third column, `0` for the 4th column, `1` for the 5th column, `2` for the 6th column, ..., `23` for the 27th column.

g.

```
# Create a new column called daily_rain_fall that is the sum of the columns 0-23
rain_df$daily_rain_fall <- rowSums(rain_df[, as.character(0:23)])
```

h.

```
# Create a histogram of daily rain fall frequencies
hist(rain_df$daily_rain_fall,
     main = "Histogram of Daily Rainfall",
     xlab = "Daily Rainfall")
```



i.

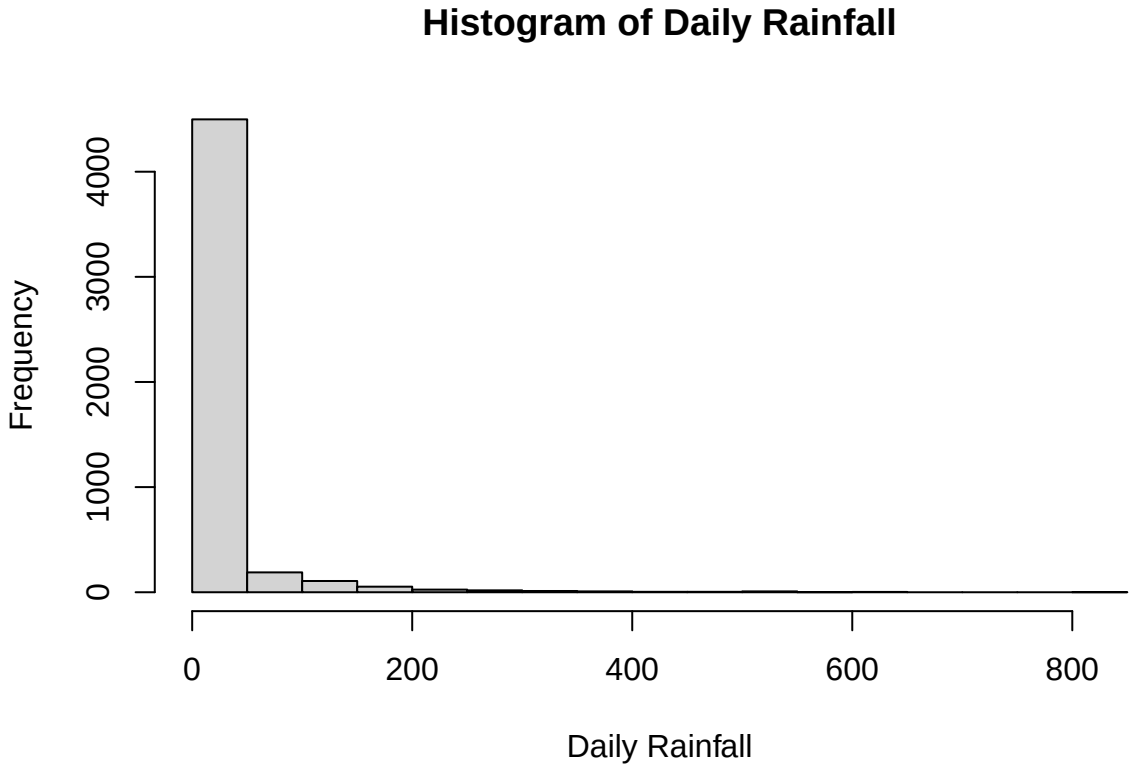
The histogram above cannot be right. This is because the histogram indicates that there are days in which the daily rainfall was negative, which is impossible. The minimum daily rainfall is 0, as there is no way for it to rain a negative amount. This likely means that there is some incorrect data or representation of missing data as negative values, which is causing the histogram to show that there are days with negative rainfall.

j.

```
# Set the rainfall values that were negative to NA
rain_df[rain_df < 0] = NA
# Update the daily rain fall total sums
rain_df$daily_rain_fall <- rowSums(rain_df[, as.character(0:23)])
```

k.

```
# Create a histogram with the updated daily rain fall totals
hist(rain_df$daily_rain_fall,
     main = "Histogram of Daily Rainfall",
     xlab = "Daily Rainfall")
```



This histogram is much more reasonable than the previous histogram. There are no more days represented in the histogram in which the daily rainfall was negative, which is impossible. Now we can see a much more realistic picture of what the frequency of daily rainfall looks like. From the histogram, we can see that there is the largest frequency of days in which the daily rainfall is closer to 0, which makes sense because from some basic research it doesn't rain much more than it does rain in Canada based on (https://www.ef.edu/blog/faq/whats-canadas-climate-like/). As a result, we would expect the highest frequency of days to have no rainfall.

Question 2

a.

For `x <- c("5","12","7")`, I would not expect an error because we are simply creating a vector with 3 character types.

For `max(x)`, while it may seem like it could be erroneous, an error would not be thrown because from reading the `max` function documentation, character comparisons will simply return the larger character lexicographically. As a result, "7" should be the return value because "7" is the largest string lexicographically in `x`, as `x` is a character vector. This is because of the quotes around each value when `x` was initialized.

For `sort(x)`, there is no error because characters/strings can be sorted as well as numbers. It simply will sort the values in `x` lexicograpically, resulting in "12", "5", "7".

For `sum(x)`, there will be an error. This is because `sum` is expecting an input type that is either numeric, complex, or logical, but `x` is a character vector. As a result, we will get an invalid type for the argument.

b.

For `y <- c("5",7,12)`, there will be no error. This is because `R` will type cast the values in an atomic vector to be consistent. Because the character type in the initialization ("5") has precedence over numeric types, the 7 and 12 will be converted to characters as well, and `y` will be equivalent to `("5", "7", "12")`.

For `y[2] + y[3]`, there will be an error. This is because `y[2]` and `y[3]` are now character types, and the `+` operator is not supported for character types, only numeric.

c.

For `z <- data.frame(z1="5",z2=7,z3=12)`, there will not be an error. This will create a dataframe with one row, with columns `z1, z2, z3`. The value in the singular row for the `z1` column will be "5", for `z2` it will be 7, and for `z3` it will be 12. This will not cause any errors because different columns in a dataframe can have different types.

For `z[1,2] + z[1,3]`, there will not be an error. This is because we are getting the value in row 1, column 2 of the dataframe `z`, and the value in row 1, column 3 of the dataframe. These are both numeric types, so using `+` will work. It will add the values 7 and 12, returning 19.

Question 3

a.

Reproducible code is valuable for ensuring that analyses can be consistently replicated and verified by others. It enhances transparency and credibility by allowing anyone to understand and reproduce the results. This is particularly important in research, where reproducibility is a key aspect of the scientific method. Tools like RMarkdown and version control help document the entire process, making it easier for others to build upon previous work and for you to revisit and extend your own analyses.

b.

In this class, making code reproducible is essential because it ensures that the data transformations and cleaning processes you apply can be reliably repeated. For example, if you make a script that cleans and preprocesses a dataset by handling missing values, normalizing variables, and doing something like entity resolution, having reproducible code allows you to apply the same process to updated datasets or similar datasets in the future. Having consistency is important in data mining, where small changes in preprocessing can lead to significant differences in the outcomes of the analysis.

Another example of the importance of reproducible code in this class and other classes is regarding collaborative work. Imagine working in a group where multiple team members are contributing to different parts of the code. If each team member writes reproducible code, it ensures that everyone can run the same scripts on their own machines and get identical results. This is crucial when integrating different parts of the project, such as combining datasets that have been independently cleaned or ensuring that the final analysis reflects the same processing steps. Without reproducible code, inconsistencies could arise, leading to conflicting results or difficulties in troubleshooting errors.

c.

This assignment was a 4/10 difficulty.