# Data Wrangling in R

STA 360: Assignment 1, Fall 2020

Hugh Esterson

Due Friday August 21, 5 PM Standard Eastern Time

Today's agenda: Manipulating data objects; using the built-in functions, doing numerical calculations, and basic plots; reinforcing core probabilistic ideas.

***R Markdown Test***

0. Open a new R Markdown file; set the output to HTML mode and "Knit". This should produce a web page with the knitting procedure executing your code blocks. You can edit this new file to produce your homework submission.

***Working with data***

Total points on assignment: 10 (reproducibility) + 22 (Q1) + 9 (Q2) + 3 (Q3) = 44 points

Reproducibility component: 10 points.

```
# load packages
library(tidyverse)
```

For this assignment, I loaded the *tidyverse* package. While I mainly used Base R, I always find *tidyverse* and its code style to be helpful to use.

1. (22 points total, equally weighted) The data set **rnf6080.dat** records hourly rainfall at a certain location in Canada, every day from 1960 to 1980.

a. Load the data set into R and make it a data frame called `rain.df`. What command did you use?

```
# load & name dataset
## change data directory
rain.df <- read.table("data/rnf6080.dat")
```

I used the *read.table* command, which loads the .dat file through its corresponding file location. (Source: https://www.datafiles.samhsa.gov/faq/how-do-i-read-data-r-nid3445.)

b. How many rows and columns does `rain.df` have? How do you know? (If there are not 5070 rows and 27 columns, you did something wrong in the first part of the problem.)

```
# find dimensions of dataset
dim(rain.df)
```

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

*rain.df* has 5,070 rows and 27 columns. I found this information by using the *dim* function, which shows the dimension of an object within R.

c. What command would you use to get the names of the columns of `rain.df`? What are those names?

```
# find col 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"
```

You would use the *names* command to get the names of the columns within *rain.df*. These names are listed in the output above, showing they span $V1$ through $V27$.

d. What command would you use to get the value at row 2, column 4? What is the value?

```
# call row 2, col 4 entry
rain.df[2,4]
```

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

In order to get the entry of a specific row and column, you can call the dataframe, along with the specified coordinates. In this case, row 2, column 4 yields a value of 0.

e. What command would you use to display the whole second row? What is the content of that row?

```
# call 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
```

Similarly, you can display an entire row by calling the dataframe, with a coordinate for the row number, followed by a comma and no coordinate for the column. In this case, the second row is composed of the values 60, 4, and 2 for $V1$ through $V3$, respectively, followed by 0 from $V4$ through $V27$.

f. What does the following command do?

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

The following command renames the columns of *rain.df*. Specifically, it names the first three columns, *year*, *month*, and *day*, while naming the remaining columns 0 through 23, representing each hour of a day.

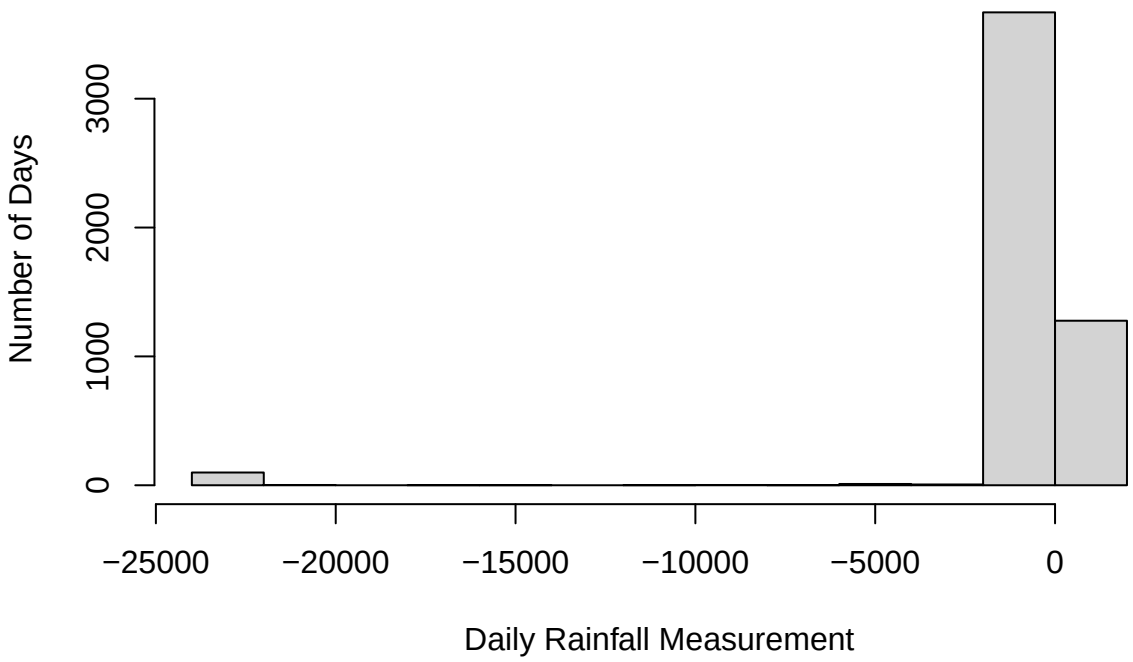
g. Create a new column called `daily`, which is the sum of the 24 hourly columns.

```
# add daily variable, with col sum
rain.df <- rain.df %>%
  mutate(daily = select(rain.df, 4:27) %>%
           rowSums(na.rm = TRUE))
```

In order to create a new column that summed the hourly rain totals, I used a *tidyverse* and *dplyr* approach. I have become familiar with this package and style from previous classes, especially STA 323, which had a full lecture on the multiple, helpful uses of *dplyr*. This approach was also shown on the linked StackOverflow thread and selects the fourth through 27th columns, which are the hourly rain totals in the original dataframe. (Source: https://stackoverflow.com/questions/29006056/efficiently-sum-across-multiple-columns-in-r).

h. Give the command you would use to create a histogram of the daily rainfall amounts. Please make sure to attach your figures in your .pdf report.

```
# plot simple histogram
hist(rain.df$daily,
     main = "Daily Rainfall Measurements in Canadian Location, \n1960-1980",
     xlab = "Daily Rainfall Measurement",
     ylab = "Number of Days")
```




Using the *hist* command, you can call a variable from a dataframe to create a histogram of that variable's values. I have also added axis and graph titles that correspond to the dataset, as explained in the homework assignment. While this approach uses Base R, a similar plot could be created with the package *ggplot*.

i. Explain why that histogram above cannot possibly be right.

The histogram above cannot possible be right because there is a small bin of the histrogram at a negative value of *daily*, or daily rainfall measurement. In context, this would mean that there was negative daily rain total, which is not possible.

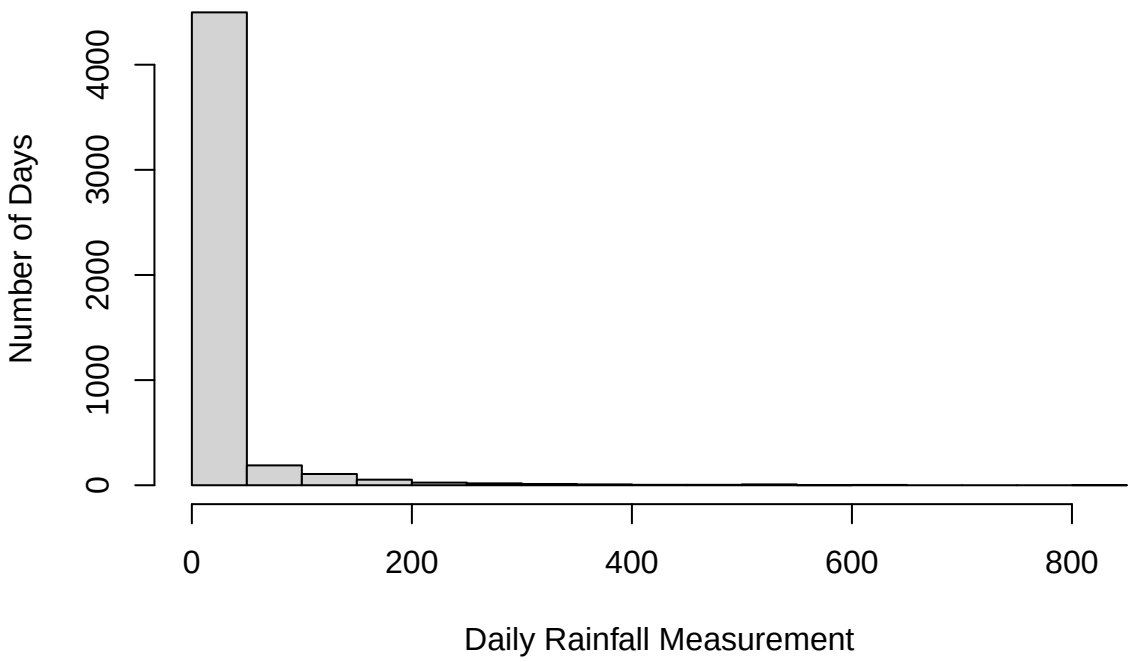
j. Give the command you would use to fix the data frame.

```
# replace negative daily values with NA
rain.df[rain.df < 0] <- NA
```

Upon further review of the dataframe, it looks as though erroneous readings were inputted as a value of "-999." This led to negative daily rain totals, which does not make sense contextually. While it is up to the data scientist, I would treat negative values, and especially those of "-999" as faulty entries. Thus, I will look to replace any negatives with a value of $NA$, as to keep the integrity and structure of the dataframe. (Note: I experimented using a variant of the *pmax* function for this problem, setting an absolute lower bound of 0, but decided to use this more efficient approach.)

k. Create a corrected histogram and again include it as part of your submitted report. Explain why it is more reasonable than the previous histogram.

```
# replot histogram of daily rainfall
hist(rain.df$daily,
     main = "[Updated] Daily Rainfall Measurements in Canadian Location, \n1960-1980",
     xlab = "Daily Rainfall Measurement",
     ylab = "Number of Days")
```




Repeating the *hist* function call on the *daily* variable, one can see that this histogram is more reasonable. There are positive negative values of daily rain total along the horizontal axis, with the majority of days showing little to no rain (skewed right), which is to be expected in most climates. This change was made through replacing all the faulty, negative rainfall readings to $NA$ within the dataframe.

***Data types***

2. (9 points, equally weighted) Make sure your answers to different parts of this problem are compatible with each other.

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

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

The *max* command will return "7" as a non-erroneous result. This is because the factor is built of type character, and not numerical values. Thus, the string with the highest first digit will be returned as the maximum.

The *sort* command will return a non-erroneous result. It will be "12," followed by "5," followed by "7." This again will sort from the smallest to largest first digit of the respective strings.

The *sum* command will return an error, as you cannot sum multiple characters.

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

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

The *c* command will build a vector of three elements, coerced to type character: "5," "7," and "12." This is non-erroneous, although one should note that all parts of this vector will be of type characters.

The second command will return an error, as you cannot compute arithmetic on character values.

c. For the next two commands, either explain their results, or why they should produce errors.

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

The *data.frame* command will build a dataframe with $z1$, $z2$, and $z3$ as the column names, with $z1$ being of type character and $z2$ and $z3$ being of type double. The corresponding values of 5, 7, and 12 will be placed as the values within this dataframe. This code works, generally, because a dataframe can have columns with different types, although within a column itself, all entries will be of one type.

The second command will sum the second and third columns of the first row. Since these are both of type double, arithmetic will lead to a non-erroneous result. In this case, the sum of the two entries will be 19.

3. (3 pts, equally weighted).

a.) What is the point of reproducible code?

The point of reproducible code is to ensure that your code can be not only run, but also understood, by others. Keeping reproducibility in mind, you can essentially allow your code to be analyzed anywhere, at anytime, by anyone. While this idea might sound a bit cliche, it is important for someone to be able to reproduce your code to verify your results. This type of transparency can allow other peers to cross-check your work and ensure that the work upholds standards of the industry or field in which your work falls.

b.) Give an example of why making your code reproducible is important for you to know in this class and moving forward.

I think it is important for me to know the power of reproducible code because it is a standard practice in coding-based work and academic environments. For this class and others I have taken, reproducible code can allow a classmate, instructor, or TA to quickly and easily find bugs or points of trouble in my code. In a work environment, reproducible code is important for working on group projects, where other programmers will need to be able to understand and manipulate your code quickly. This is especially true in a growingly "open-source," scientific setting.

c.) On a scale of 1 (easy) – 10 (hard), how hard was this assignment. If this assignment was hard ($> 5$), please state in one sentence what you struggled with.

For this first assignment, I would rate its difficult as a 6. To me, most of the trouble initiated with creating my own workflow and not the actual subject matter of this homework. In previous Statistics classes at Duke (e.g. 210, 323), the GitHub workflow was a bit different, with each student receiving their own repo as part of the class' GitHub "organization." In this case, I struggled with cloning/creating a private repo from Prof. Steort's master repository. I also used RStudio Cloud, through Duke OIT, which seemed to work decently. As far as subject matter went, I think this assignment was a helpful review. I hope to clean up my own workflow and procedures for the next assignment, and I'm looking forward to it.

# Notes & References

- For this assignment, I referred to Prof. Steort's "Intro to R" materials. This served as a helpful refresher and guide as to which functions to review and utilize in this homework.
- I also referred to, but did not explicitly copy portions of, previous assignments I had completed in STA 210 and STA 323. These resources also helped me to remind myself of coding styles I had used in the past.
- Lastly, as per the course's policies, I discussed portions of the assignment with my classmate Calleigh S. and Sara B. We each used our own, independent approaches, but found it useful to discuss the material.