HW: Week 5

36-350 – Statistical Computing

Week 5 – Spring 2022

Name: Ethan Vertal

Andrew ID: Ethan Vertal

You must submit **your own** lab as a PDF file on Gradescope.

HW Length Cap Instructions

- If the question requires you to print a data frame in your solution e.g. `q1_out_df`, you must first apply **head(q1_out_df, 30)** and **dim(q1_out_df)** in the final knitted pdf output for such a data frame.
- Please note that this only applies if you are knitting the `Rmd` to a `pdf`, for Gradescope submission purposes.
- If you are using the data frame output for visualization purposes (for example), use the entire data frame in your exploration
- The **maximum allowable length** of knitted pdf HW submission is **30 pages**. Submissions exceeding this length *will not be graded* by the TAs. All pages must be tagged as usual for the required questions per the usual policy
- For any concerns about HW length for submission, please reach out on Piazza during office hours

---

```
suppressWarnings(library(tidyverse))
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.8
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Question 1

(20 points)

An alternative to `read.table()` and such is the `scan()` function. The `scan()` function is *very* handy, particularly when someone gives you weirdly formatted text data files. (Maybe groups of unequal-length rows map to one record, etc., etc.) In this problem, use `scan()` to read in `simple.txt` (which you downloaded for Lab 5) and then post-process what you've read in to create a data frame with correct column names and correct data types (`character` for the `name` column and `double` for all the other columns). Your final step will be to print out the data frame. Look at the documentation for `scan()` and pay particular attention to the `what` argument. Once you've scanned the data, use a combination of, e.g., `matrix()` and `data.frame()` to bend the data to your will, and then cast the data in columns 2 through 8 to `numeric`. Hint: `t()` transposes a matrix. Also, pass `stringsAsFactors=FALSE` as an argument to `data.frame()`.

```
types <-  list(name = "character", u = numeric(), g = numeric(), r = numeric(),
               i = numeric(), z = numeric(), y = numeric(), redshift = numeric())
simple_df <- data.frame(scan("simple.txt", what = types, skip = 1))
simple_df
```

```
##       name       u       g       r       i       z       y redshift
## 1 galaxy.A 17.8313 16.9077 16.4431 16.2099 16.0613 15.8732 0.038356
## 2 galaxy.B 19.0731 17.7448 16.9789 16.5288 16.2551 15.9531 0.058309
## 3 galaxy.C 21.6380 21.0106 20.8286 20.6283 20.6552 20.5280 0.063701
## 4 galaxy.D 20.5474 19.5542 19.2387 19.0568 19.0887 18.9865 0.059006
## 5 galaxy.E 21.2378 20.6876 20.5661 20.4371 20.4799 20.4503 0.063202
## 6 galaxy.F 22.4627 21.4597 21.0484 20.8274 20.7639 20.6385 0.057773
## 7 galaxy.G 23.8221 22.8950 22.5779 22.3543 22.3225 22.2038 0.061548
## 8 galaxy.H 23.0491 22.1536 21.8791 21.6889 21.7044 21.6381 0.063769
## 9 galaxy.I 23.6742 23.0346 22.7857 22.6116 22.5813 22.5462 0.061427
```

Question 2

(20 points)

Let's up the ante a bit here. Download `branch.txt` from the `DATA` directory on Canvas. Examine it with an external viewer. This one's a bit of a mess. (Welcome to real-world data.) Construct a data frame from these data. Assume all the columns are character (there is no need in this exercise to do a final cast of the numeric columns to numeric type). To read in the data themselves, I'd advise you to use `scan()` while skipping the first line and using "|" as the separator. (See the documentation for `scan()`.) To make the data frame, you could use a combination of `matrix()` and `data.frame()` as in Q1, but before doing do, clean up your strings: replace all tab symbols (`\t`) with empty strings, and replace any leading spaces and trailing spaces with empty strings. (Hint: `gsub()`.) Note that the data comprise 14 columns and 39 rows (not including the header).

Getting the column names is a bit trickier: they are separated by `|_.`, which `scan()` cannot handle. So I'd advise you to use `scan()` to read in *just the first line* (use `\n` as a separator; see the argument `n`), then use `strsplit()` to split the line into 14 column names. You might have to "escape" (i.e., apply double backslashes) some or all of the characters used in splitting. Again, clean things up: get rid of `\t` symbols and trailing spaces.

In the end, display the first four columns and first six rows of your beautiful data frame, rising like a phoenix from the ashes of the terribly formatted ASCII file that you began with.

```
branch <- suppressMessages(scan("branch.txt", what = "", skip = 1, sep = "|"))
branch <- gsub("\t+", "", branch, useBytes = TRUE)
branch <- gsub("^[ ]+", "", branch, useBytes = TRUE)
branch <- gsub("[ ]+$", "", branch, useBytes = TRUE)
better.branch <- data.frame(matrix(branch, nrow = 39, ncol = 14, byrow = TRUE))

names(better.branch) <- sapply(strsplit(scan("branch.txt", what = "", n = 1, sep = "\n"), "\\|_\\."),
                               FUN = function(x) { x <- gsub("\t+", "", x, useBytes = TRUE)
                                                   x <- gsub(" +(\\|)?", "", x, useBytes = TRUE)
                                                   return (x)})
head(better.branch, c(6, 4))
```

```
##    Subm_ID Score Sigma_s Detection_image
## 1 A_SP_0.0  80.9    0.25              No
## 2 A_SP_0.1 100.3    0.25              No
## 3 A_SP_0.4 579.8     1.0              No
## 4 A_SP_1.0 120.4    0.25              No
## 5 A_SP_1.7  78.5    10.0              No
## 6 A_SP_1.9 939.1     1.0              No
```

Question 3

(20 points)

Read in data from `https://download.bls.gov/pub/time.series/ap/ap.data.0.Current`, which are housed at the Bureau of Labor Statistics. Note before you start that the data are *tab delimited*, and you might find it helpful to remember that a tab is denoted `\t` in a string. The data may not read in cleanly with a simple function call; you may need to skip the header, in which case you will need to provide column names yourself. Also, the parser may misidentify column types, so you may have to set those too. And…you may have to cast data in some columns to be of proper type, after the reading in of the data is done. (Data wrangling is a messy business.) Once everything is read in and cast to (if necessary) proper type, display the mean and standard deviation of the data in the value column for every year *after* 2009 (i.e., 2010 and later). The tidyverse will help you here. Hint: `group_by()`.

```
# read delim. Summarize data using tidyverse. Remove NAs. Summary() etc.
if ( require(tidyverse) == FALSE ) {
  install.packages("tidyverse",repos="https://cloud.r-project.org")
  suppressWarnings(library(tidyverse))
}
df <- read.delim('https://download.bls.gov/pub/time.series/ap/ap.data.0.Current')
df$value <- suppressWarnings(as.double(df$value))
df <- df[!is.na(df$value), ]
df_new <- df %>% group_by(year) %>% summarise_at(vars(value), list(value_mean = mean,
                                                         value_sd = sd))
tail(df_new, 13)
```

```
## # A tibble: 13 x 3
##     year value_mean value_sd
##    <int>      <dbl>    <dbl>
##  1  2010       14.8     30.2
##  2  2011       15.0     29.7
##  3  2012       14.5     27.9
##  4  2013       9.66    22.5
##  5  2014       3.07     1.90
##  6  2015       2.81     2.00
##  7  2016       2.61     1.90
##  8  2017       2.71     1.89
##  9  2018       2.75     1.83
## 10  2019       2.70     1.81
## 11  2020       2.64     1.93
## 12  2021       3.16     2.16
## 13  2022       3.42     2.23
```

Question 4

(20 points)

Download `planets.csv` from the Canvas site. It is in the Week 5 directory. Use an external viewer (your choice) to look at the file. Then apply an appropriate function to read the file's contents into `R`. Your goal: to determine what proportion of the columns have data in at least 20% of their rows. (In other words, step from column to column and see if the proportion of `NA`'s is less than 80%. Then determine the proportion of the columns that fulfill this condition.) Your final answer should be 82.86% [or 0.8286].

```
planets <- read.csv("planets.csv", skip = 73)
props <- planets %>% is.na %>% colSums %>% sapply(FUN = function(x) {return(x/nrow(planets))})
sum(props < 0.8) / length(props)
```

```
## [1] 0.8285714
```

Question 5

(20 points)

Make a data frame that is in essence a "dictionary" for the data in the `planets.csv` file. What this means is: extract those lines of the file that contain variable names and corresponding definitions, and from those lines extract the variable names into a vector called `variable` and the definitions into a vector called `definition`. Output the first six rows only! (Hint: in your call to `data.frame()`, set the argument `stringsAsFactors` to `FALSE`. This changes the column contents to character strings rather than factor variables.) Hint: let's say you do an `strsplit()` to split the variable from the definition in each line. The output will be a list, with one list element for each line that contains two strings, one for the variable and one for the definition. A handy way to extract all of the variables would be, e.g., sapply(

,[[,1). That `[[` function is really useful.

```
planets_defs <- read.csv("planets.csv", skip = 2, nrows = 69)
planets_defs <- sapply(planets_defs, FUN = function(x) { gsub("# COLUMN ", "", x) })
h <- sapply(planets_defs, FUN = function(x) { strsplit(x, ": +") })
variable <- sapply(h, '[[', 1)
definition <- sapply(h, '[[', 2)
vd.df <- data.frame(list(variable = variable, definition = definition), stringsAsFactors = FALSE)
row.names(vd.df) <- NULL
head(vd.df, 6)
```

```
##        variable                       definition
## 1   pl_hostname                        Host Name
## 2     pl_letter                    Planet Letter
## 3 pl_discmethod                 Discovery Method
## 4       pl_pnum      Number of Planets in System
## 5     pl_orbper             Orbital Period [days]
## 6 pl_orbpererr1 Orbital Period Upper Unc. [days]
```

Question 6

(20 points)

Extract the 2020 Major League Baseball standings from the web site given below and put them into a *single* data frame that contains all 30 MLB teams, with the first column being the team name, the second column being the number of wins, and the third column being the number of losses. Order the data frame by decreasing number of wins. Use `rvest` functions to extract any tables you need, which are of class `data.frame`, and then process the data frames until you get a single one as described above.

```
if ( require(rvest) == FALSE ) {
  install.packages("rvest",repos="https://cloud.r-project.org")
  library(rvest)
}
```

```
## Loading required package: rvest

##
## Attaching package: 'rvest'

## The following object is masked from 'package:readr':
##
##     guess_encoding
```

```
site = read_html("https://www.baseball-reference.com/leagues/MLB-standings.shtml")
df <- site %>% html_nodes("table") %>% html_table()
new_df <- rbind(df[[1]], df[[2]])
new_df <- rbind(new_df, df[[3]])
new_df <- rbind(new_df, df[[4]])
new_df <- rbind(new_df, df[[5]])
new_df <- rbind(new_df, df[[6]])
new_df <- new_df[,1:3]
suppressWarnings(new_df[order(-new_df[,2]),])
```

```
## # A tibble: 30 x 3
##    Tm                        W     L
##    <chr>                 <int> <int>
##  1 San Francisco Giants    107    55
##  2 Los Angeles Dodgers     106    56
##  3 Tampa Bay Rays          100    62
##  4 Houston Astros           95    67
##  5 Milwaukee Brewers        95    67
##  6 Chicago White Sox        93    69
##  7 Boston Red Sox           92    70
##  8 New York Yankees         92    70
##  9 Toronto Blue Jays        91    71
## 10 Seattle Mariners         90    72
## # ... with 20 more rows
```