BIOS 545 Spring 2020 Homework 2

Due by 11:59 PM on Monday March 23, 2020

Instructions

There are four problems valued at 100 points total. Submit your answer sheet in a file named using the convention of `BIOS545_LastName_FirstName_HW2.Rmd` (or `BIOS545_LastName_FirstName_HW2.R`) to Canvas. You should use RStudio to create the file. Late submissions are accepted with a penalty of 15% for each day late. Submissions made later than three days from the due date will not be accepted and a grade of F will be given for that homework assignment.

Comments in the functions are strongly encouraged especially if you want partial credit. The comments will help us understand what it is you are trying to do.

All of these problems can be solved using material presented in class and the recaps. Unless otherwise indicated in the problem, you may use other R functions to help you find a solution although you cannot download additional packages to solve a problem. In some cases you might have to use the help mechanisms described in Week 1 to locate an appropriate function. We will run your R commands at the R console to verify the statements.

IMPORTANT

Short version: **Do your homework by yourself.**

Long Version: By submitting your homework you are agreeing that you and you alone are the sole author of any and all code you turn in. It is highly recommended that you do your homework alone. You may not show your code to another student nor may you view another student's code. Engaging in this activity will be considered a violation of the honor code.

Problems

1) Aggregation - 40 Points (6 pts for 1-4 and 8 pts for 5-6)

Note that you will be using **dplyr** commands such as *filter*, *select*, *group_by*, *summarize*, *left_join*, etc. You can and should pipe these together as I did in the **dplyr** lecture.

`Note that deductions will take place if you avoid using pipes`

You can also use other functions in base R along with the dplyr verbs that could help you search for strings and count number of characters.

You will need to download the .csv files that correspond to the San Francisco restaurant inspections data that is the subect of this problem. If you want to have some information on the format of these three files here is a document that provides it: https://www.dropbox.com/s/jy6vxcrvi877ezd/sf_inspections.pdf?dl=1 Here is how to read in the data.

```r
suppressMessages(library(dplyr))
url <- "https://www.dropbox.com/s/t91ejvmu1kxwcli/SFFoodProgram_Complete_Data.zip?dl=1"

download.file(url,"SFFoodProgram_Complete_Data.zip")
system("unzip -o SFFoodProgram_Complete_Data.zip")

# On Windows this probably won't work so unzip them by hand
# which usually means
# system("unzip -o SFFoodProgram_Complete_Data.zip")

bus <- read.csv("businesses_plus.csv",sep=",",stringsAsFactors=FALSE)
ins <- read.csv("inspections_plus.csv",sep=",",stringsAsFactors=FALSE)
vio <- read.csv("violations_plus.csv",sep=",",stringsAsFactors=FALSE)
```

Note that you have the answers below each question so you will know when you have the right answers.

`1.1)` Determine the total number of violations that are high risk but NOT involving vermin.

```
## [1] 6424
```

`1.2)` Determine the total number of inspections occuring between April 01, 2014 and May 01, 2014 (inclusive) AND that were of a routine nature or involved a special event.

```
## [1] 661
```

`1.3)` Find the business_ids,names, and scores of the two restaurants with the lowest inspection scores out all restaurants

```
## # A tibble: 2 x 3
##   business_id name                   min
##         <int> <chr>                <int>
## 1        7643 PACIFIC SUPER MARKET    42
## 2       74522 DICK LEE PASTRY         42
```

`1.4)` Show all distinct VALID postal codes. Note there are invalid postal codes in the business table that you have to somehow filter out. Also note that a given postal code can also be listed more than once. Start by listing all the unique postal codes and “eyeballing” which ones are non standard and perhaps include extraneous characters. Come up with ways to filter out the ones that have “junk” in them, are too long, or are all zeroes.

```
##    postal_code
## 1        94104
## 2        94124
## 3        94109
## 4        94103
## 5        94133
## 6        94118
## 7        94110
## 8        94122
## 9        94115
## 10       94131
## 11       94111
## 12       94117
## 13       94107
## 14       94108
## 15       94102
## 16       94132
## 17       94105
## 18       94134
## 19       94116
## 20       94121
## 21       94112
## 22       94127
## 23       94123
## 24       94114
## 25       94513
## 26       94545
## 27       94066
## 28       94158
## 29       95105
## 30       94140
## 31       94013
## 32       94130
## 33       92672
## 34       94120
## 35       94143
## 36       94609
## 37       94101
## 38       94188
## 39       94014
## 40       04102
## 41       94129
```

1.5) Find the total number of Mexican restaurants for each `postal_code`. For purposes of this exercise you can look for restaurant names that contain the string 'MEX'.

```
## [1] "C/C/C/C/C/en_US.UTF-8"
```

```
## # A tibble: 17 x 2
##    postal_code count
##    <chr>       <int>
##  1 94110           7
##  2 94103           4
##  3 94105           4
##  4 94114           4
##  5 94102           2
##  6 94107           2
##  7 94111           2
##   8 94112           2
##   9 94133           2
##  10 94104           1
##  11 94108           1
##  12 94117           1
##  13 94118           1
##  14 94121           1
##  15 94127           1
##  16 94132           1
##  17 94134           1
```

1.6) With respect to all the Mexican restuarants, what are the 10 restaurants with the worst scores? Present the result starting with the lowest / worst score.

```
##    business_id                                 name Score
## 1         1005          CHAVITA'S MEXICAN RESTAURANT    69
## 2         3229             TOMMYS MEXICAN RESTAURANT    70
## 3         5140                  EL FARO MEXICAN FOOD    72
## 4         1702 EL TOREADOR FONDA MEXICANA RESTAURANT    73
## 5        74593            LA PLACITA MEXICATESSEN    73
## 6         1702 EL TOREADOR FONDA MEXICANA RESTAURANT    74
## 7        70019        EL METATE CON SABOR AMEXICO LLC    74
## 8         7770              DOMINGUEZ MEXICAN BAKERY    75
## 9        74593            LA PLACITA MEXICATESSEN    76
## 10        1005          CHAVITA'S MEXICAN RESTAURANT    77
```

2) Tabulation (20 points)

Write a function called *mytable* that, given an input vector, produces the same output as would the *table* function that is built into R. This implies of course that you cannot use the actual *table* function or other existing counting or aggregation functions (e.g. **xtabs**) to solve this problem. Of course you can use the *table* function to verify that the output from your *mytable* function matches the output of *table*.

```
mytable <- function(somevec) {

# A function to sort a numeric vector in ascending or descending order
#
# INPUT: "somevec" an unsorted numeric vector
#
# OUTPUT: a named vector that presents the total number of times that each
#         unique value in the vector appears

   # Your code goes here
}
```

Let's try it on an example vector:

```
set.seed(123)
somevec <- as.integer(runif(30,1,10))

mytable(somevec)
```

```
## 1 2 3 4 5 6 7 8 9
## 3 1 4 1 5 4 3 2 7
```

```
table(somevec)
```

```
## somevec
## 1 2 3 4 5 6 7 8 9
## 3 1 4 1 5 4 3 2 7
```

Another example:

```
set.seed(321)
somevec <- as.integer(runif(30,1,10))

mytable(somevec)
```

```
## 1 2 3 4 5 6 7 8 9
## 1 2 4 5 3 7 2 1 5
```

```
table(somevec)
```

```
## somevec
## 1 2 3 4 5 6 7 8 9
## 1 2 4 5 3 7 2 1 5
```

Here is some general pseudocode:

- Determine the unique values in the input vector `somevec`
- Sort the resulting vector
  - For each unique value, determine how many times it appears in `somevec` - use functions you've seen previously
- After processing each unique value then assign names to the return vector
- The names and arrangement of your counts should match those from the output of the *table* function
- Return a vector with your computed counts

3) Data Frame Interrogation (20 points)

Read in the following data frame which tracks the prices of six "crypto currencies" over the past year. This data frame has some rows with missing values and the default format may require some transformation and/or the creation of additional columns so that you can answer the questions below. How you do this is up to you although you have seen several functions in class that will help.

Please execute the following commands, exactly as presented, to read in the data frame for this problem.

```
url <- "https://www.dropbox.com/s/g1f519txobjxstj/crypto.csv?dl=1"
download.file(url,"crypto.csv")
colClasses <- c("character", "numeric", "numeric", "numeric", "numeric",
                "character", "character", "character")
crypto <- read.table("crypto.csv",colClasses=colClasses,header=TRUE)
```

a) Find the number of records for each crypto currency category. Present the answer in sorted order from highest to lowest

```
##  42_bit bitcoin   bitbc     pbt    idex     rmc
##     365     364     359     210      34      30
```

b) What is the highest closing price occurring on any Monday within the data set?

```
## [1] 99368.9
```

c) In what month (January, February, March, . . .) and on what day of the week (Monday, Tuesday, . . .), did the lowest Opening price occur?

```
## [1] "March"    "Thursday"
```

d) On what day of the week (Monday, Tuesday, . . .) did the largest **absolute** difference between Opening and Closing price occur and how much was that difference? Note the Closing price could be higher or lower than the opening price.

```
## [1] "Monday"
```

```
## [1] 20361
```

e) What is the average Volume for each crypto currency Symbol?

```
##       42_bit        bitbc      bitcoin         idex          pbt 
## 5.876389e+03 3.253687e+04 4.004901e+09 5.670841e+04 3.214692e+05 
##          rmc 
## 4.904807e+04
```

4) Separate Columns in a Data Frame (20 points)

Lots of times we get a data frame that has a column that encodes different bits of information within that one column. Here is an example. We have three patients who came to the clinic. We capture the dates on which they came. This is recorded in the `dates` column. In the second column, called `measures`, we also record their gender and blood pressure (systolic and diastolic). These measurements are separated by a colon character. While we could leave the column intact we could also separate those individual pieces of information into separate columns for more clarity in our data.

```r
mydf <- data.frame(dates = c("2010-05-28", "2010-06-10", "2010-06-12"),
                   measures = c("M:132:94", "F:140:99", "M:128:82"),
                   stringsAsFactors = FALSE)
mydf
```

```
##        dates measures
## 1 2010-05-28 M:132:94
## 2 2010-06-10 F:140:99
## 3 2010-06-12 M:128:82
```

We learned in class that there is a function called *separate* in the **tidyr** package, which can easily separate the gender, systolic, and diastolic measurements into three separate columns in the data frame. Here is how it would look:

```r
library(tidyr)  # You have to install it first of course
separate(mydf, 2, into = c("gender", "sys", "dia"), sep = ":")
```

```
##        dates gender sys dia
## 1 2010-05-28      M 132  94
## 2 2010-06-10      F 140  99
## 3 2010-06-12      M 128  82
```

Pretty cool huh? Notice that it adds the three new columns specified by the `into` argument and it also removes the original `measures` column since we no longer need it. Your assignment is to write your own version of the *separate* function. Obviously you cannot use the actual *separate* function to help you. We will make it easier for you to do this as the *separate* function can work in general cases which can be quite complex.

Write a function called *myseparate* that effectively does the same thing as the *separate* function. Here is a shell that describes the inputs and outputs:

```r
myseparate <- function(df, colnum = 2, into = c("a", "b", "c"), sep = ":") {

# Function to split a column of a data frame into new columns
#
# INPUT: df - a data frame
#        colnum - a numeric indicator of what column you want to separate
#        into - a vector that contains the desired names of the new columns
#        sep - A character delimeter by which to separate
#
# OUTPUT: retdf - a data frame that has the new columns as specified by
#                 the "into" argument. It will not contain the original
#                 column number specified by the "colnum" argument

  return(retdf)

}
```

Here is some pseudocode:

- Use `colnum` to access the specified column of `df`. This will be a character vector. Check to see if the `sep` character appears in the column number specified by `colnum`.
- Split `df` using the specified separator - use one of the functions you learned about in class to split characters. Store the results into a list called `hold`.
- Initialize three vectors which will eventually contain the results of the split.
- For each list element in `hold`:
  - Take the first element value and store it in the first of the three vectors.
  - Take the second element value and store it in the second of the three vectors.
  - Take the third element value and store it in the third of the three vectors.
- Create a temporary data frame out of the three vectors from above. Use the idiom:
  - `tmpdf <- data.frame(cbind(a = a, b = b, c = c), stringsAsFactors = FALSE)`
  - `a`, `b`, `c` represent the three vectors mentioned before.
- Make sure the names of the columns of this data frame match the values of the `into` arguemnt.
- Next bind this data frame to the original data frame, Also remove the column number specifed in the call to your function (e.g. 2 or 3).
- Return the data frame.

Here is how this would work with the example data frames:

```
mydf
```

```
##        dates measures
## 1 2010-05-28 M:132:94
## 2 2010-06-10 F:140:99
## 3 2010-06-12 M:128:82
```

```
myseparate(mydf, 2, into = c("gender", "sys", "dia"), sep = ":")
```

```
##        dates gender sys dia
## 1 2010-05-28      M 132  94
## 2 2010-06-10      F 140  99
## 3 2010-06-12      M 128  82
```

```
# Download a table3
url <- "https://www.dropbox.com/s/40bc0ymg2zmy38c/table3.csv?dl=1"

download.file(url,"table3.csv")
(table3 <- read.table("table3.csv", stringsAsFactors = FALSE))
```

```
##       country year                  rate
## 1 Afghanistan 1999       745/19987071/898
## 2 Afghanistan 2000     2666/20595360/3490
## 3      Brazil 1999   37737/172006362/349
## 4      Brazil 2000  80488/174504898/2984
## 5       China 1999 212258/1272915272/875
## 6       China 2000 213766/1280428583/984
```

```
myseparate(table3, 3, into = c("hiv", "hep", "turb"),sep = "/")
```

```
##       country year    hiv        hep turb
## 1 Afghanistan 1999    745   19987071  898
## 2 Afghanistan 2000   2666   20595360 3490
## 3      Brazil 1999  37737  172006362  349
## 4      Brazil 2000  80488  174504898 2984
## 5       China 1999 212258 1272915272  875
## 6       China 2000 213766 1280428583  984
```

Make sure that the separator is actually relevant. So in the previous example with the `table3` data frame you see that the 3 column is three numbers separated by a "/" character. So if a user specifies, for example, a ":" then it wouldn't make sense and the code would fail. So before you do very much within your *myseparate* function you might first check to see if the specified `sep` character even exists in the column indicated by the `colnum` argument. As an example - if I pass a "sep" character of ":" it should fail with an error message such as:

```
Error in myseparate(table3, 3, into = c("hiv", "hep", "turb"), sep = ":") :
Sorry but : doesn't occur in specified column
```

hint: you can use 'grepl()' or 'grep()' to check if separator is relevant.