HW: Week 4

36-350 – Statistical Computing

Week 4 – Fall 2020

Name: Kimberly Zhang

Andrew ID: kyz

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

Question 1

(10 points)

You are given the following matrix:

```
set.seed(505)
mat = matrix(rnorm(900),30,30)
mat[sample(30,1),sample(30,1)] = NA
```

Compute the standard deviation for each row, using `apply()` and your own on-the-fly function, i.e., a function that is defined *within* the argument list being passed to `apply()`. **Do not use the function sd()!** Realize that since there is a missing value within the matrix, you need to define your function so as to only take into account the non-missing data in each row. If your vector of standard deviations has an `NA` in it, then your function isn't quite working yet.

```
apply(mat, 1, function(x){
  y= x[is.na(x) == FALSE]
  return(sqrt(sum((y-mean(y))^2)/(length(y)-1)))})
```

```
##  [1] 1.2235111 0.9996540 0.8324186 0.7935861 0.9546933 1.1166745 1.0264495
##  [8] 0.7135952 1.0357715 0.9023740 1.2146342 0.9665977 1.1364236 0.7335094
## [15] 0.8758855 1.0529671 1.0303302 0.8857679 1.1004938 0.9636788 0.9981597
## [22] 1.1224219 1.2828417 0.9777383 0.9223948 0.8506261 0.8840344 0.6538431
## [29] 0.8304627 1.0001846
```

Question 2

(10 points)

The data frame `state.df` was defined in Q20 of Lab 4. Copy the code that created that data frame to here. Then define a function `grad.by.lit.median()` that computes the median value of the ratio of graduation rate and literacy. (Basically, define a function that does what your mutation did in Q20 of Lab 4, and returns the median value of the vector that your function derives.) Then use `split()` and `sapply()` so as to compute `grad.by.lit.median()` for each `Division` in the `state.df` data frame. Sort your output into decreasing order. (`Pacific` should be the first division output, with value 63.29626.)

```
state.df<-data.frame(state.x77, Region = state.region,
                     Division = state.division)
```

```
grad.by.lit.median<-function(x){
  median(x$HS.Grad/(100-x$Illiteracy)*100)
}

sort(sapply(split(state.df, state.df$Division), grad.by.lit.median),
     decreasing = TRUE)
```

```
##            Pacific           Mountain West North Central        New England
##           63.29626           61.56942           57.94769           57.03376
## East North Central    Middle Atlantic West South Central     South Atlantic
##           53.27952           53.08392           45.94095           45.33469
## East South Central
##           42.09705
```

Below, we read in a data table showing the fastest women's 100-meter sprint times.

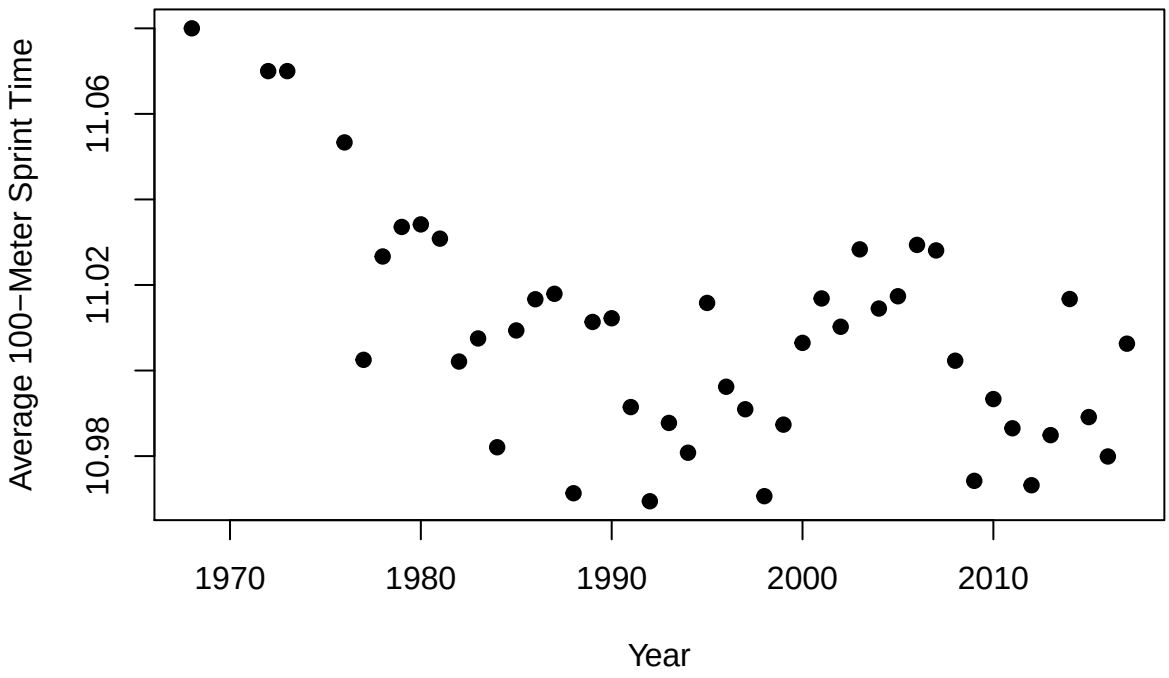
```
sprint.df = read.table("http://www.stat.cmu.edu/~pfreeman/women_100m_with_header.dat",header=TRUE)
```

Question 3

(10 points)

As you did in Q7 of Lab 4, add a column dubbed `Year` to the data frame `sprint.df`, to compute a new data frame called `new.sprint.df`. Then compute the mean (or average) sprint time in each year. Do this with `tapply()`. Use `plot()` to plot the years on the x-axis and the mean time for each year on the y-axis. Also send the following arguments to `plot()`: `xlab="Year"`, `ylab="Average 100-Meter Sprint Time"`, and `pch=19`.

```
string.col<-as.character(sprint.df$Date)
date.len<-nchar(string.col)
years<-substr(string.col, date.len-3, date.len)
new.sprint.df<-data.frame(sprint.df, Year = years)
res<-tapply(new.sprint.df$Time, new.sprint.df$Year, mean)
plot(names(res),as.numeric(res), xlab = "Year",
     ylab= "Average 100-Meter Sprint Time", pch=19)
```



One thing that we did not cover in the `dplyr` notes (Notes_4D) is the concept of splitting. In base `R`, for instance, `split()` creates a list of data frames; each element of the list can then be worked with individually. To "split" a data frame in the `tidyverse`, one can use the `group_by()` function: pass in one or more variables, and the data frame will be effectively split based on these variables. I say "effectively" because you won't see visualize evidence of grouping if you just pipe to `group_by()` alone; you need to pipe the output of `group_by()` to something else.

A commonly used "something else" is `summarize()`, a function which takes the groups specified by `group_by()` and summarizes their information using one or more functions. See the documentation for `summarize` to get a sense of summary statistics that are useful.

Example: determine the number of states in each `Region` of the United States, and the mean illiteracy.

```
suppressMessages(library(tidyverse))
example.df = data.frame(state.x77,Region=state.region,Division=state.division)
example.df %>% group_by(Region) %>%
  summarize(Number.of.States=n(),Mean.Illiteracy=mean(Illiteracy))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 4 x 3
##   Region        Number.of.States Mean.Illiteracy
##   <fct>                    <int>           <dbl>
## 1 Northeast                    9            1
## 2 South                       16            1.74
## 3 North Central               12            0.7
## 4 West                        13            1.02
```

Question 4

(10 points)

Your result for Q3 should indicate that the average sprint time decreases over the years. Using a pipe stream to extract the p-value for the linear regression slope. This is a bit tricky. First you utilize `group_by()` and `summarize()` to extract the average sprint times, and pipe the results to `lm()`. You would pipe your `lm()` results to `summary()`, which prints a summary but invisibly returns a list. To get at the coefficients element of the list, you would use the `[[` function (yeah, it's a function, and you need to include the backquotes... note: don't cut-and-paste the backquotes, as cutting and pasting often leads to bad results because what you see in, e.g., the HTML rendering of this file might not be the "correct" backquote that `R` is expecting). Pass to this function the argument `"coefficients"`. At this point, your output is a matrix that has row names and column names. Extract the matrix element associated with `Year` (row) and `Pr(>|t|)` (column). (You'll need to use dot notation here, to represent the matrix, then you subset it.) Your final value should be 0.0002297436, which is less than 0.05, leading us to reject the null hypothesis that the true average time is actually constant from year to year.

```
new.sprint.df %>% group_by(Year) %>%
  summarize(Mean.Time=mean(Time)) %>%
lm(formula= .$Mean.Time~as.numeric(.$Year)) %>% summary(.) %>%
  `[[`("coefficients") %>%
.[2, "Pr(>|t|)"]
```

```
## [1] 0.0002297436
```

Question 5

(10 points)

Using `state.df` from above, display the sample mean and sample standard deviation of incomes in each defined `Region-Division` pair. (Here you can use `sd()`.) Arrange your results by descending sample mean.

```
state.df %>% group_by(Region, Division) %>%
  summarize(Mean=mean(Income), Sd= sd(Income)) %>%
  arrange(., desc(Mean))
```

```
## # A tibble: 9 x 4
## # Groups:   Region [4]
##   Region        Division            Mean    Sd
##   <fct>         <fct>              <dbl> <dbl>
## 1 West          Pacific            5183.  654.
## 2 Northeast     Middle Atlantic    4863   396.
## 3 North Central East North Central 4669   272.
## 4 North Central West North Central 4570.  305.
## 5 Northeast     New England        4424.  600.
## 6 West          Mountain           4402.  493.
## 7 South         South Atlantic     4355.  632.
## 8 South         West South Central 3774.  376.
## 9 South         East South Central 3564.  321.
```

Question 6

(10 points)

Repeat Q5, but display the 5th and 95th percentiles for income. Also display the difference between the two,

and arrange your table in descending order of that difference. See the documentation for `quantile()` to determine how to get a single-number summary out (you won't get this by default).

```
state.df %>% group_by(Region, Division) %>%
  summarize(Mean=mean(Income), Sd= sd(Income),
            Quantile5 = quantile(Income, probs= 0.05, names = FALSE),
            Quantile95 = quantile(Income, probs = 0.95, names= FALSE),
            Diff = Quantile95- Quantile5) %>%
  arrange(., desc(Diff))
```

```
## # A tibble: 9 x 7
## # Groups:   Region [4]
##   Region        Division            Mean    Sd Quantile5 Quantile95  Diff
##   <fct>         <fct>              <dbl> <dbl>     <dbl>      <dbl> <dbl>
## 1 South         South Atlantic     4355.  632.     3623.      5130. 1506.
## 2 Northeast     New England        4424.  600.     3747.      5200. 1452.
## 3 West          Pacific            5183.  654.     4701.      6075. 1374.
## 4 West          Mountain           4402.  493.     3748.      5056. 1308.
## 5 North Central West North Central 4570.  305.     4193.      4963.  770.
## 6 South         West South Central 3774.  376.     3403.      4157.  754.
## 7 Northeast     Middle Atlantic    4863   396.     4494.      5204.  709.
## 8 South         East South Central 3564.  321.     3177.      3805.  628.
## 9 North Central East North Central 4669   272.     4460       5036.  576.
```

The following code replaces the `Date` column in `new.sprint.df` with `Day`, `Month`, and `Year`.

```
if ( exists("new.sprint.df") == TRUE ) {
  newer.sprint.df = new.sprint.df %>% separate(col=Date,into=c("Day","Month","Year"),sep="\\.",convert='
}
```

Question 7

(10 points)

Write a function called `day_of_year()` that converts an input day and month (integers both) into the day of the year. For instance, passing in day=31 and month=12 (December 31st) would yield 365. Usually. Also pass in the year; if the year is divisible by 4 (i.e., if year%%4 == 0) *and* the year is not 2000 *and* the month is March or later, add a day. . . because you are dealing with a leap year. Test your function by sending in June 1st, 1996, and then June 1st, 1997, and then June 1st, 2000. The outputs should be 153, 152, and 152 respectively. Once you've written your function, use `mutate()` and your `day_of_year()` function to define a new `DayOfYear` column for `newer.sprint.df`, then output just the `Day`, `Month`, `Year`, and `DayOfYear` columns arranged in ascending values of `DayOfYear`. Just show the first six rows. Your `DayOfYear` values should range from 56 (first row) to 93 (sixth row). Hint: it may be useful to define a vector giving the number of days in each month, and to use `cumsum()` to define another vector giving the cumulative number of days through the end of a month (e.g., 31 for January, 59 for February, etc.)

```
day_of_year<- function(day, month, year){
              month.days<-c(31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)
              cum.days<-cumsum(month.days)
              res<-ifelse(month == 1, day, cum.days[month-1] + day)
              res[which((year%%4 == 0) & (year != 2000) & (month >= 3))] =
                res + 1
              return(res)}
day_of_year(1, 6, 1996)
```

```
## [1] 153
```

```
day_of_year(1, 6, 1997)
```

```
## [1] 152
```

```
day_of_year(1, 6, 2000)
```

```
## [1] 152
```

```
newer.sprint.df %>% mutate(DayOfYear = day_of_year(Day, Month, Year)) %>%
  select(Day, Month, Year, DayOfYear) %>%
  arrange(DayOfYear) %>%
  head()
```

```
## Warning: Problem with `mutate()` input `DayOfYear`.
## i number of items to replace is not a multiple of replacement length
## i Input `DayOfYear` is `day_of_year(Day, Month, Year)`.
```

```
## Warning in res[which((year%%4 == 0) & (year != 2000) & (month >= 3))] <- res + :
## number of items to replace is not a multiple of replacement length
```

```
##   Day Month Year DayOfYear
## 1  25     2 1998        56
## 2   8     3 1982        67
## 3  14     3 2015        73
## 4  26     3 1999        85
## 5   3     4 2015        93
## 6   3     4 2015        93
```

Question 8

(10 points)

Who was the oldest person included in the sprint table for the year 2011? In the end, just show the first and last name, and the two-digit birth year. Hint: utilize `separate()`, an example usage of which is given above, to separate birthdates into day, month, and two-digit year, and go from there.

```
newer.sprint.df %>%
separate(col=Birthdate,
         into=c("Bday","Bmonth","Byear"),sep="\\.",convert=TRUE) %>%
  arrange(., Byear) %>% select(., First.Name, Last.Name, Byear) %>%
  head(., 1)
```

```
##   First.Name Last.Name Byear
## 1     Wyomia      Tyus    45
```

Below we read in the data on the political economy of strikes that you examined in Lab 4.

```
strikes.df = read.csv("http://www.stat.cmu.edu/~pfreeman/strikes.csv")
```

Question 9

(10 points)

Using `split()` and `sapply()`, compute the average unemployment rate, inflation rates, and strike volume for each year represented in the `strikes.df` data frame. The output should be a matrix of dimension 3×35. (You need not display the matrix contents...just capture the output from `sapply()` and pass that output to `dim()`.) Provide appropriate row names (see `rownames()` to your output matrix. Display the columns for 1962, 1972, and 1982. (This can be done in one line as opposed to three.)

```
res<-strikes.df %>% split(., strikes.df$year) %>% sapply(., function(x){
  list(Mean.unemployment= mean(x$unemployment, na.rm = TRUE),
       Mean.inflation= mean(x$inflation, na.rm= TRUE),
       Mean.strike.vol=mean(x$strike.volume, na.rm= TRUE))})
dim(res)
```

```
## [1]  3 35
```

```
res[, c("1962", "1972", "1982")]
```

```
##                   1962     1972     1982
## Mean.unemployment 2.127778 2.705556 6.805882
## Mean.inflation    3.738889 6.238889 9.594118
## Mean.strike.vol   214.5556 387.1111 227.8824
```

Question 10

(10 points)

Utilize piping and `group_by()`, etc., to compute the average unemployment rate for each country, and display that average for only those countries with the maximum and minimum averages. To be clear: your output should only show average unemployment for Ireland and Switzerland, and nothing else. (Hint: remember `slice()`, a less-often-used `dplyr` function.) Hint: arrange your output in order of descending average unemployment, then note that `n()` applied as an argument to the right function will return the last row.

```
strikes.df %>% group_by(country) %>%
  summarize(Avg.unemploy=mean(unemployment)) %>%
  arrange(., desc(Avg.unemploy)) %>%
  slice(c(1, n()))
```

```
## # A tibble: 2 x 2
##   country     Avg.unemploy
##   <fct>              <dbl>
## 1 Ireland             7.77
## 2 Switzerland         0.329
```