HW: Week 4

36-350 – Statistical Computing

Week 4 – Fall 2020

Name: Kyle Wagner

Andrew ID: kowagner

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) {return(sqrt(sum((x - mean(x, na.rm = TRUE))^2 , na.rm = TRUE)/ (length(x)-1)))
```

```
##  [1] 1.2235111 0.9996540 0.8324186 0.7797836 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.)

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

```
## -- Attaching packages --------------

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.1     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

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

```r
state.df = as.data.frame(state.x77)
state.df$region = state.region
state.df$division = state.division

grad.by.lit.median = function(df) {
  val <- mutate(df,GradLit = 100 * ‘HS Grad‘ / (100 - Illiteracy))  %>%
  {median(.$GradLit)}
  c = ("GradLit" = val)
  return(c)
}
state.df.by.div = split(state.df,f = state.df$division)
sort(sapply(state.df.by.div,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.

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

Question 3

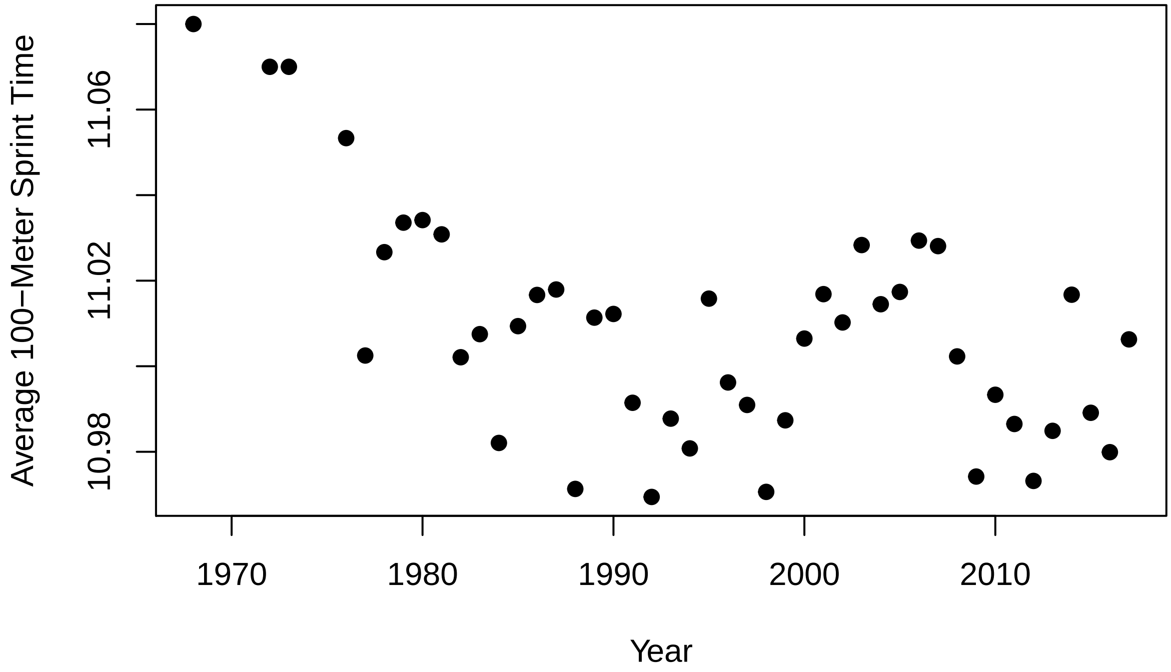
(10 points)

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`.

```
extract <- substr(as.character(sprint.df$Date),7,11)
new.sprint.df <- sprint.df
new.sprint.df$Year <- extract

vals <- tapply(new.sprint.df$Time,INDEX = new.sprint.df$Year,mean)

plot(names(vals),vals,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 9.969597e-05, 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. (Depending on how you process the data, the answer that you get may be 0.0002297436. If you get this, that's OK. It's not strictly correct, but for our purposes we won't worry about the different result.)

```
new.sprint.df %>%
  group_by(Year) %>%
  summarize(time= mean(Time)) %>%
  lm(time ~ as.numeric(Year),.) %>%
  summary() %>%
  `[[`("coefficients")
```

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

##                      Estimate   Std. Error   t value     Pr(>|t|)
## (Intercept)      13.155485368 0.5005989171 26.279492 4.048047e-28
## as.numeric(Year) -0.001076322 0.0002509463 -4.289054 9.969597e-05
```

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.

```
#res <- tapply(state.df$Income,INDEX = list(state.df$region,state.df$division),sd)
#res1 <- tapply(state.df$Income,INDEX = list(state.df$region,state.df$division),mean)
res <- state.df %>%
  group_by(state.df$region,state.df$division) %>%
  summarize(sds = sd(Income),m = mean(Income)) %>%
  arrange(.,desc(m))
```

```
## ‘summarise()‘ regrouping output by ’state.df$region’ (override with ‘.groups‘ argument)
```

```
res
```

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

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).

```
res <- state.df %>%
  group_by(state.df$region,state.df$division) %>%
  summarize(fifth = quantile(Income,.05),ninetyfifth = quantile(Income,.95),dif = quantile(Income,.95)-
  arrange(.,desc(dif))
```

```
## ‘summarise()‘ regrouping output by ’state.df$region’ (override with ‘.groups‘ argument)
```

```
res
```

```
## # A tibble: 9 x 5
## # Groups:   state.df$region [4]
##   ‘state.df$region‘ ‘state.df$division‘ fifth ninetyfifth   dif
##   <fct>             <fct>               <dbl>       <dbl> <dbl>
## 1 South             South Atlantic      3623.       5130. 1506.
## 2 Northeast         New England         3747.       5200. 1452.
## 3 West              Pacific             4701.       6075. 1374.
## 4 West              Mountain            3748.       5056. 1308.
## 5 North Central     West North Central  4193.       4963.  770.
## 6 South             West South Central  3403.       4157.  754.
## 7 Northeast         Middle Atlantic     4494.       5204.  709.
## 8 South             East South Central  3177.       3805.  628.
## 9 North Central     East North Central  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.)

Note: the use of an `if` statement may lead to suboptimal results when you pass in vectors of days, months, and years. Consider using, e.g., `which()` instead. (There are other options as well.)

```
month_day = c(31,28,31,30,31,30,31,31,30,31,30,31)
md_cum = cumsum(month_day)

day_of_year = function(day, month) {
  initial = md_cum[month]
  sub = month_day[month]
  add = sub - day
  total = initial + add
  return(total)
}
newer.sprint.df = mutate(newer.sprint.df,
                         DayOfYear = day_of_year(newer.sprint.df$Day,newer.sprint.df$Month))
```

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(.,Birthdate,into = c("other","BirthYear"),sep = 6) %>%filter(.,Year == 201
```

```
##   First.Name Last.Name BirthYear
## 1 Schillonie   Calvert        88
```

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.)

```
my.fun = function(x) {
  avg_unemploy = mean(x$unemployment)
  inflation_rate = mean(x$inflation)
  avg_strike = mean(x$strike.volume)
  res = c(avg_unemploy,inflation_rate,avg_strike)
  return(res)
}
inter <- split(strikes.df,strikes.df$year)
sapply(inter,my.fun)
```

```
##            1951       1952       1953       1954       1955       1956
## [1,]   3.088889   3.683333   3.594444   3.505556   3.044444   3.033333
## [2,]  13.088889   5.794444   1.333333   1.833333   1.294444   3.705556
## [3,] 359.222222 588.666667 211.944444 139.333333 215.277778 561.944444
##            1957       1958       1959       1960       1961       1962
## [1,]   3.055556   3.422222   3.094444   2.555556   2.333333   2.127778
## [2,]   3.255556   3.472222   1.377778   1.955556   2.355556   3.738889
## [3,] 216.111111 145.611111 239.444444 123.111111 230.111111 214.555556
##            1963       1964       1965       1966       1967       1968
## [1,]   2.144444   1.861111   1.800000   1.838889   2.177778   2.394444
## [2,]   3.366667   4.116667   4.216667   3.938889   3.772222   3.983333
## [3,] 220.388889 200.055556 169.000000 253.500000 183.611111 217.500000
##            1969       1970       1971       1972       1973       1974
## [1,]   2.138889   2.161111   2.388889   2.705556   2.472222   2.633333
## [2,]   4.150000   5.488889   6.288889   6.238889   8.577778  13.161111
## [3,] 408.833333 394.888889 304.777778 387.111111 407.277778 400.444444
##            1975       1976       1977       1978       1979       1980
## [1,]   3.844444   4.322222   4.605556   4.883333   4.577778   4.716667
## [2,]  12.683333  10.466667   9.733333   7.194444   8.250000  11.494444
## [3,] 342.722222 411.555556 304.888889 257.833333 435.055556 367.277778
##            1981       1982       1983       1984       1985
## [1,]   5.647059   6.805882   7.823529   7.582353   7.323529
## [2,]  11.405882   9.594118   6.629412   5.582353   5.464706
## [3,] 261.647059 227.882353 195.470588 274.176471 217.764706
```

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(rate_unemploy = mean(unemployment)) %>%
  arrange(.,desc(rate_unemploy)) %>%
  filter(country == "Ireland" | country =="Switzerland")
```

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

```
## # A tibble: 2 x 2
##   country     rate_unemploy
##   <chr>               <dbl>
## 1 Ireland             7.77
## 2 Switzerland         0.329
```

```
#  lm(time ~ as.numeric(Year),.) %>%
#  summary() %>%
#  ‘[[‘("coefficients")
```