Final R Project

36-350 – Statistical Computing

Week 11 – Fall 2020

Name: Sophia Hill

Andrew ID: sghill

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

There are 200 points possible for this assignment.

The dataset that you will examine is from fivethirtyeight.com, and it provides information on 2020 presidential election polls that have been carried out since just after the midterm elections in November 2018.

The dataset, available on Canvas in the `FILES` hierarchy, is `president_polls.csv`. The only thing I will say about the contents now is that some of the columns are best treated as factors and some as strings, so *you should examine the data and try your best to preprocess all the input columns correctly.* (Note: this process may be iterative, i.e., you might determine later that you need to alter how you process the input. That's fine. That's *normal* in the real world.)

To be clear: there is generally no unique way of going about answering each question below. For instance, you may want to use base `R` sometimes, and `tidyverse` functions at other times. In the end, *I don't particularly care how you go about answering the questions, so long as you answer them correctly.* (Some of you may very well create more elegant solutions than what I have in the solution set. And that's good. Others will create coding monstrosities. . . but that's OK, as my attitude is that your coding will improve with practice. One cannot expect to leave a semester-long class with the same comfort level coding `R` as I have built up over 15+ years of nearly continuous coding. . .)

```
## -- Attaching packages ------------------ tidyverse 1.3.0 --

## v ggplot2 3.3.0     v purrr   0.3.4
## v tibble  3.0.1     v dplyr   0.8.5
## v tidyr   1.0.3     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

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

Question 1

(10 points)

Download the data file and read it into `R`. Use `read.csv()` (base R function). Pass in the file name, and one other argument whose value is a vector that specifies the classes for each column ("numeric", "character", "logical", or "factor"). Use the `R` documentation to determine the appropriate argument to use.

Now, about factors: if you look at the data, and the number of unique values in any given column is small (compared with the number of rows), and the data in that column is neither numeric or logical, you probably want to assume it's a factor variable. For instance, the `url` column is not going to be a factor variable, since there is not a small set of unique values.

```
# establish column classes
ch = "character"
n = "numeric"
l = "logical"
f = "factor"

classes = c(n, n, n, ch, n, ch, ch, ch, ch, n, ch, ch, n, ch, ch, ch,
            ch, n, ch, ch, ch, ch, ch, l, ch, f, l, l, ch, ch, ch,
            ch, n, ch, n, ch, ch, n)

# read in the data
df = read.csv("president_polls.csv", header = TRUE,
              colClasses = classes, stringsAsFactors = FALSE)
```

Question 2

(15 points)

Not all of the columns are useful, nor are all the columns complete (some contain missing data). First, show the number of rows of data and the number of columns. Then, print the output from `summary()` or a similar function to determine if any columns are wholly uninformative. (Meaning, for instance, that all the values are the same.) If any are, eliminate them. (Note that `population` and `population_full` are redundant: eliminate the latter.) Then display the proportion of missing data in each of the remaining columns, but only display these proportions when they are greater than zero. Finally, use `complete.cases()` (or an equivalent tidyverse-based scheme) to eliminate rows with missing data, and display the number of rows and columns. You should have 13983 rows and 29 columns.

```
print(nrow(df))
```

```
## [1] 13993
```

```
print(ncol(df))
```

```
## [1] 38
```

```
print(summary(df))
```

```
## question_id poll_id cycle state
## Min.   : 92078 Min.   :57025 Min.   :2020 Length:13993
## 1st Qu.:119098 1st Qu.:64518 1st Qu.:2020 Class :character
## Median :129386 Median :69464 Median :2020 Mode  :character
## Mean  :124220 Mean  :67449 Mean  :2020
## 3rd Qu.:132990 3rd Qu.:71053 3rd Qu.:2020
## Max.   :135117 Max.   :72071 Max.   :2020
##
## pollster_id pollster sponsor_ids sponsors
## Min.  : 11 Length:13993 Length:13993 Length:13993
## 1st Qu.: 744 Class :character Class :character Class :character
## Median :1193 Mode :character Mode :character Mode :character
## Mean :1056
## 3rd Qu.:1304
## Max.  :1633
##
## display_name pollster_rating_id pollster_rating_name fte_grade
## Length:13993 Min.  : 3.0 Length:13993 Length:13993
## Class :character 1st Qu.:218.0 Class :character Class :character
## Mode :character Median :324.0 Mode :character Mode :character
## Mean :291.6
## 3rd Qu.:325.0
## Max.  :628.0
## NA’s :8
## sample_size population population_full methodology
## Min.  : 88 Length:13993 Length:13993 Length:13993
## 1st Qu.: 797 Class :character Class :character Class :character
## Median : 1046 Mode :character Mode :character Mode :character
## Mean : 2982
## 3rd Qu.: 2862
## Max.  :34460
## NA’s :2
## office_type seat_number seat_name start_date
## Length:13993 Min.  :0 Length:13993 Length:13993
## Class :character 1st Qu.:0 Class :character Class :character
## Mode :character Median :0 Mode :character Mode :character
## Mean :0
## 3rd Qu.:0
## Max.  :0
##
## end_date election_date sponsor_candidate internal
## Length:13993 Length:13993 Length:13993 Mode :logical
## Class :character Class :character Class :character FALSE:13971
## Mode :character Mode :character Mode :character TRUE :22
##
##
##
##
## partisan tracking nationwide_batch ranked_choice_reallocated
## Length:13993 :8978 Mode :logical Mode :logical
## Class :character TRUE:5015 FALSE:13993 FALSE:13993
## Mode :character
##
##
##
##
## created_at notes url stage
## Length:13993 Length:13993 Length:13993 Length:13993
## Class :character Class :character Class :character Class :character
## Mode :character Mode :character Mode :character Mode :character
##
##
##
```

```
##
## race_id answer candidate_id candidate_name
## Min.   :6210 Length:13993 Min.   :13253 Length:13993
## 1st Qu.:6210 Class :character 1st Qu.:13254 Class :character
## Median :6223 Mode :character Median :13256 Mode :character
## Mean :6244 Mean :13343
## 3rd Qu.:6244 3rd Qu.:13256
## Max.   :8718 Max.   :16083
##
## candidate_party pct
## Length:13993 Min.   : 0.00
## Class :character 1st Qu.:41.00
## Mode :character Median :45.55
## Mean :43.61
## 3rd Qu.:50.30
## Max.   :90.53
##
```

```r
# eliminate cycle, seat_number (values are all the same)
df$cycle <- NULL
df$seat_number <- NULL
df$population_full <- NULL
df$stage <- NULL
df$nationwide_batch <- NULL
df$ranked_choice_reallocated <- NULL
df$election_date <- NULL
df$seat_name <- NULL
df$office_type <- NULL

# show proportions of missing data
p.missing = function(x){
  return(sum(is.na(x))/length(x))
}

df.missing = apply(df, MARGIN = 2, FUN = p.missing)
print(df.missing[df.missing > 0])
```

```
## pollster_rating_id        sample_size
##       0.0005717144       0.0001429286
```

```r
# remove rows with missing data
df = df[complete.cases(df[1:nrow(df),]),]

print(nrow(df))
```

```
## [1] 13983
```

```r
print(ncol(df))
```

```
## [1] 29
```

Question 3

(10 points)

What is the range of times over which polling was done? Find the earliest date in start_date, the latest date in end_date, and determine (via code, not by hand!) how many days elapsed between these dates. (Search for the dates using code…do not make any assumptions about the earliest date being on the last line of the dataset, etc.)

```
# convert columns from character to Date class
start.dates = as.Date(df$start_date, format = "%m/%d/%Y")
end.dates = as.Date(df$end_date, format = "%m/%d/%Y")

# earliest date in start_date
print(min(start.dates))
```

```
## [1] "2018-11-12"
```

```
# latest date in end_date
print(max(end.dates))
```

```
## [1] "2020-10-27"
```

```
# days that elapsed between these dates
print(as.numeric(difftime(max(end.dates), min(start.dates))))
```

```
## [1] 715
```

Question 4

(10 points)

Create a data frame that shows the names of each (supposed) candidate and the number of times they each appear in the dataset. Sort the data frame by the number of appearances, in descending order. (Note: you might get a warning, depending on how you code this. Ignore the warning.)

```
cand.df = data.frame(sort(table(df$candidate_name),
                          decreasing = TRUE))
names(cand.df) = c("Candidate", "Frequency")

print(head(cand.df))
```

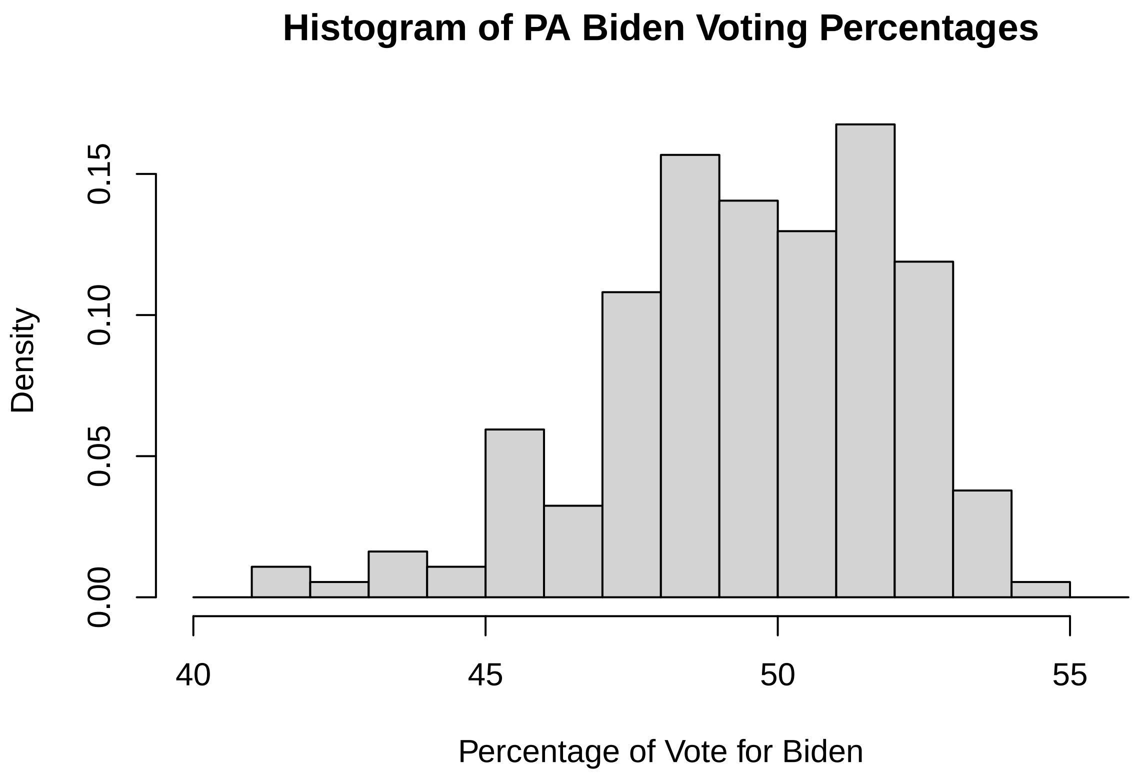
```
##              Candidate Frequency
## 1         Donald Trump      6484
## 2 Joseph R. Biden Jr.       4971
## 3      Bernard Sanders       435
## 4         Jo Jorgensen       422
## 5    Elizabeth Warren        348
## 6        Howie Hawkins       304
```

Question 5

(10 points)

Display a probability histogram (as opposed to a frequency histogram) for the percentage of the vote that Joe Biden received in each poll conducted in Pennsylvania. Give the histogram an appropriate x-axis label and title. Set the bin boundaries to a sequence from 40 to 56 in steps of 1.

```r
# get pecentage of votes for Biden
pa.bid = df %>%
  filter(., state == "Pennsylvania" & answer == "Biden") %>%
  select(., pct)

pa.bid = as.numeric(unlist(pa.bid))

hist(pa.bid, freq = FALSE,
     breaks = seq(40, 56, 1),
     xlab = "Percentage of Vote for Biden",
     main = "Histogram of PA Biden Voting Percentages")
```



Question 6

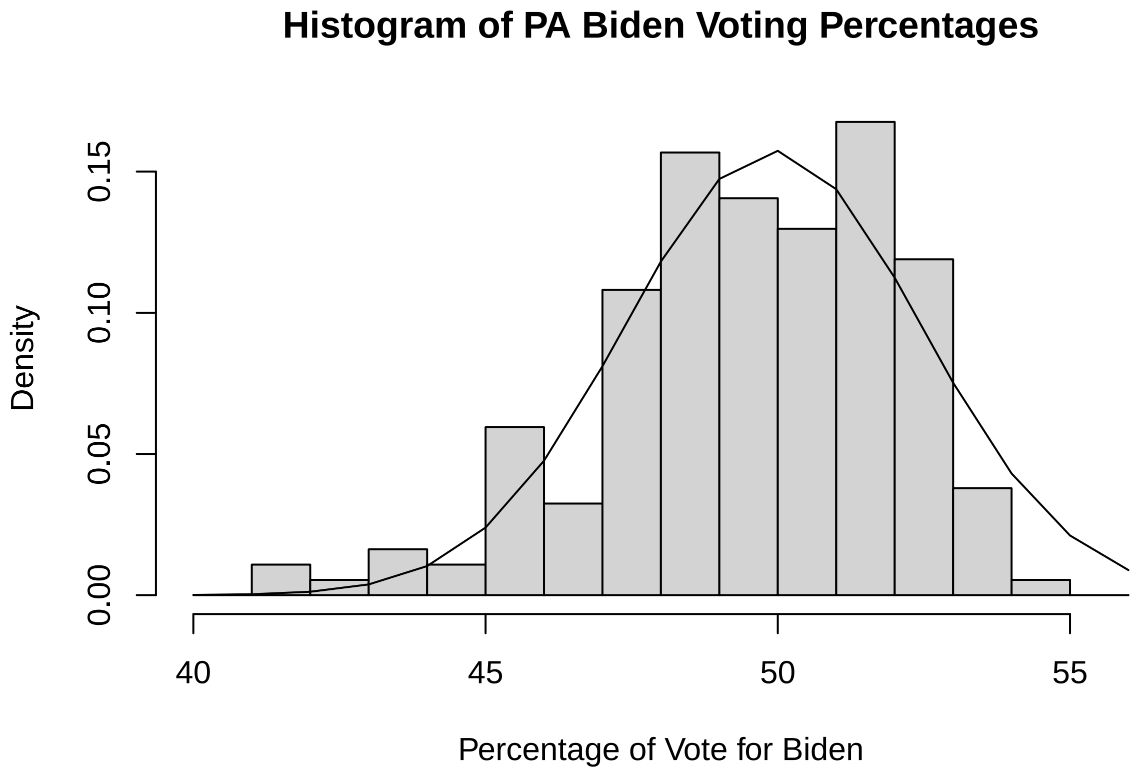
(10 points)

Biden's support in Pennsylvania has an approximately normal shape. (Yeah, there's a lower tail and all, but let's go with it.) Fit a normal distribution to these data using an appropriate optimizer. (You need not include the gradient here.) Display the optimized values of the `mean` and `sd` parameters. Redisplay your histogram from Q5 with the optimized normal pdf superimposed. Don't expect the model to be a "good" one. Hint: if you have a hard time finding the optimum value, try plotting a few times with lines superimposed with different values of the `rate` parameter. This will help build your intuition.

```r
set.seed(6)

# returns negative log-liklihood for a Normal dist
my.fit.fun = function(my.par, my.data){
  return(-sum(log(dnorm(my.data, mean = my.par[1], sd = my.par[2]))))
}

# guesses for mean and sd
my.par = c(50, 2)
# optimization
optim.out = optim(my.par, my.fit.fun, my.data = pa.bid,
                  method = "Nelder-Mead")

# optimized mean and sd
print(optim.out$par)
```

```
## [1] 49.920001  2.534351
```

```r
b = seq(40, 56, 1)
hist(pa.bid, freq = FALSE, breaks = b,
     xlab = "Percentage of Vote for Biden",
     main = "Histogram of PA Biden Voting Percentages")
lines(b, dnorm(b, mean = optim.out$par[1],
               sd = optim.out$par[2]))
```



Question 7

(10 points)

Estimate the uncertainty for the `mean` parameter via bootstrapping. Display a histogram showing the estimated values of `mean` and display the 2.5% and 97.5% quantiles. Hint: if you get warnings about "NaNs produced", wrap the calls to optim with `suppressWarnings`. Hint II: set the random number seed for reproducibility.

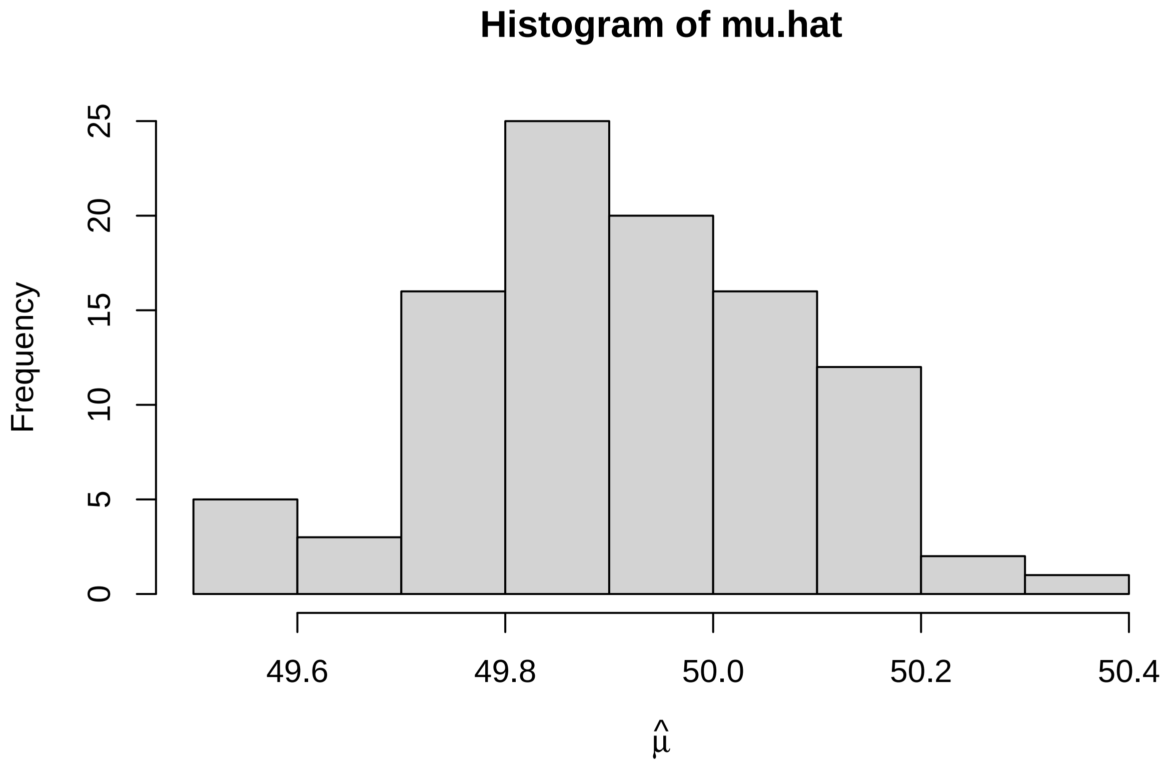
```
set.seed(7)

B = 100
indices = sample(length(pa.bid), B*length(pa.bid), replace = TRUE)
data.array = matrix(pa.bid[indices], nrow = B)

f = function(x){
  optim.out = suppressWarnings(optim(c(40, 56),
                                     my.fit.fun, my.data = x))
  return(optim.out$par)
}

apply.out = apply(data.array, 1, f)
mu.hat = apply.out[1,]

# uncertainty for the mean parameter
print(sd(mu.hat))
```

```
## [1] 0.1632243
```

```
# 2.5% and 97.5% quantiles
quantile(mu.hat, c(.025, .975))
```

```
##     2.5%    97.5% 
## 49.57697 50.19532
```

```
hist(mu.hat, xlab = expression(hat(mu)))
```

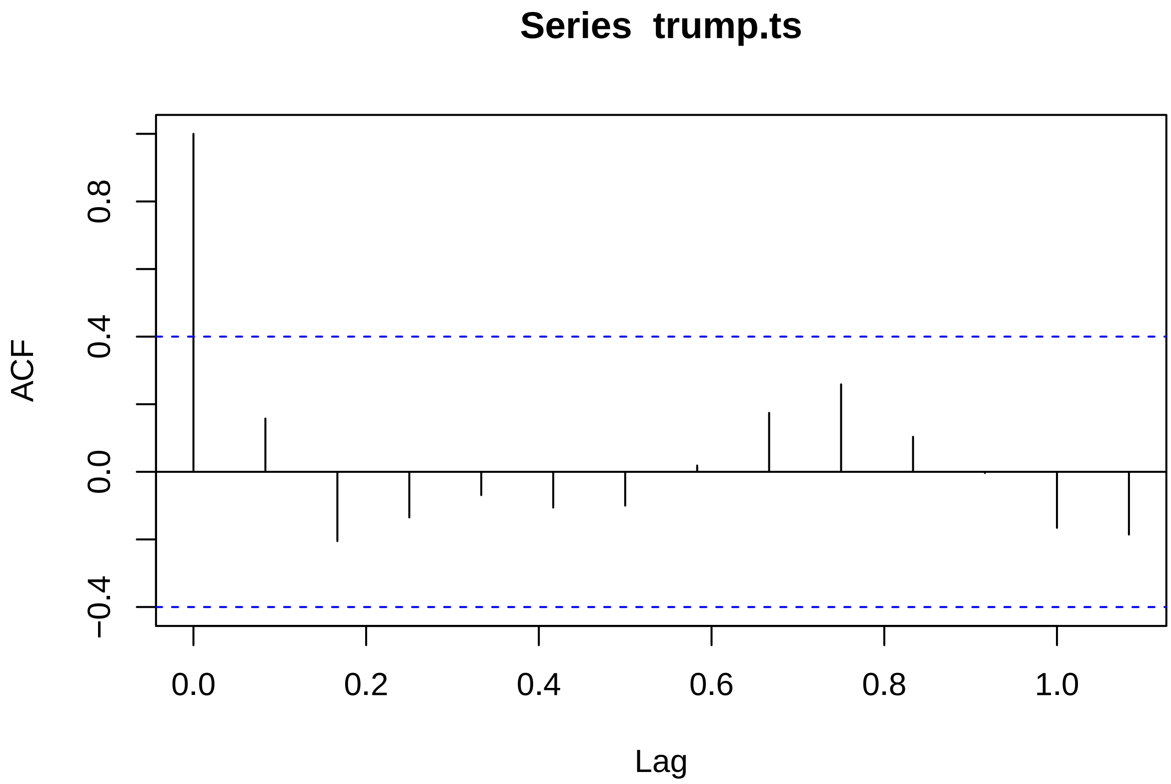


## Question 8 *(20 points)*

Compute the per-month average value of `pct` for Donald Trump for all polls in the dataset. In other words, compute the average value of `pct` for Trump in November 2018, then December 2018, etc., up to October 2020. Plot the autocorrelation function for your resulting vector of averages. If you observe local minima at lags 2 and 13, a local maximum at lag 9, and no significant values (other than lag 0, which doesn't count), you've probably coded everything correctly. (Note: this is 20 points because constructing the code to extract `pct` on a month-by-month basis will take a bit of thought. Realize that `Date` objects are good to work with if you are trying to go through the data in temporal order.) For the dates of the polls: use `end_date`.

```
trump.pct = df %>%
  filter(., answer == "Trump") %>%
  select(., pct, end_date)
```

```
# put dates in Date format
trump.pct$end_date = as.Date(trump.pct$end_date, format = "%m/%d/%Y")
# add column with just month and year
trump.pct$m.y = format(trump.pct$end_date, "%Y-%m")
# remove column that includes days
trump.pct$end_date <- NULL

# get average pct for each month
t.mean = trump.pct %>%
  group_by(., m.y) %>%
  summarize(mean.pct = mean(pct))

# create time series object
trump.ts = ts(t.mean$mean.pct, start = c(2018, 11), frequency = 12)

acf(trump.ts)
```



Question 9

(10 points)

Display a table that shows the *percentage* of online polls that achieve each possible polling grade. (See the columns `methodology` and `fte_grade`.) Only include online polls that were graded! (For the others, the grade is the empty string "“.) You will need to reorder the output from table so that”A+" is shown first, then “A”, “A-”, “A/B”, “B+”, etc. (There is no magic way to do this: type out the vector that has the order you need and apply that.) Round each percentage to the nearest tenth of a percent. If you do everything correctly, you should find, e.g., that 7.6% of graded online polls get a “B-” rating.

```
# get onlne poll grades that are not the empty string
p.g = df %>%
  filter(., methodology == "Online" & fte_grade != "") %>%
  select(., fte_grade)

# get grade percentages
p.g.per = round((table(p.g)/sum(table(p.g)))*100, digits = 1)

g.order = c("A, A/B, B, B-, B/C, C+, C, C-, C/D, D-")
# sorted table
print(p.g.per[order(match(p.g.per, g.order))])
```

```
## p.g
##    A  A/B    B   B-  B/C    C   C-  C/D   C+   D-
##  4.0  0.8  5.0  7.6 22.1  4.9  2.8  1.5  1.6 49.8
```

Question 10

(10 points)

Compute the mean and the standard error of the mean of the value of `pct` for each unique value of `candidate_party`, but display your result for Republicans (`REP`) and Democrats (`DEM`) only.

```
# standard error function
se = function(x){
  return(sd(x)/sqrt(length(x)))
}

df %>%
  group_by(., candidate_party) %>%
  select(., pct) %>%
  summarize(., pct.mn = mean(pct), pct.se = se(pct)) %>%
  filter(., candidate_party == "DEM" | candidate_party == "REP") %>%
  print(.)
```

```
## Adding missing grouping variables: ‘candidate_party‘
```

```
## # A tibble: 2 x 3
##   candidate_party pct.mn pct.se
##   <chr>            <dbl>  <dbl>
## 1 DEM               48.6 0.0974
## 2 REP               44.1 0.0892
```

Question 11

(10 points)

Create a new data frame in which each unique value of `poll_id` only appears twice. Call this `df.sub`. If you do this correctly, the number of rows in `df.sub` should be 2790 (representing 1395 separate polls). Display the number of rows. Hint: playing with `table()` and its names attribute may help you.

```r
p.t = table(df$poll_id)
p.t.2 = p.t[p.t == 2]

df.sub = df %>%
  filter(., poll_id %in% names(p.t.2))

print(nrow(df.sub))
```

```
## [1] 2790
```

Question 12

(10 points)

The data frame `df.sub` has results for 1395 two-candidate polls. Some of these polls involve "registered voters" (`rv` in the `population` column) and some involve "likely voters" (`lv`). Are the means of the absolute values of the differences in the percentages for each candidate significantly different for polls of registered voters versus polls of likely voters? To determine this, compute the absolute difference in the percentages for each group (it will be a vector of differences for each group) and use a two-sample t-test to test the null hypothesis that the means of differences for each group are equal. (For the alternative hypothesis that the means are not equal, I get a p value of 9.057×10^{-4}.)

```r
# for each rv poll, pct that voted for biden
# trump rv is 100 - biden pct
rv.biden = df.sub %>%
  group_by(., poll_id) %>%
  filter(., population == "rv" & answer == "Biden") %>%
  select(., pct)
```

```
## Adding missing grouping variables: ‘poll_id‘
```

```r
rv.trump = data.frame(rep(100, nrow(rv.biden))) - rv.biden$pct
rv.dif = abs(rv.biden$pct - rv.trump)

# repeat for lv
lv.biden = df.sub %>%
  group_by(., poll_id) %>%
  filter(., population == "lv" & answer == "Biden") %>%
  select(., pct)
```

```
## Adding missing grouping variables: ‘poll_id‘
```

```r
lv.trump = data.frame(rep(100, nrow(lv.biden))) - lv.biden$pct
lv.dif = abs(lv.biden$pct - lv.trump)

# t-test for difs
print(t.test(rv.dif, lv.dif))
```

```
##
##  Welch Two Sample t-test
##
## data:  rv.dif and lv.dif
## t = 5.8907, df = 422.25, p-value = 7.861e-09
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  1.639903 3.282339
## sample estimates:
## mean of x mean of y
##  7.466944  5.005823
```

Question 13

(10 points)

How many times does the name of each state appear in the polling URLs? (See the column `url`.) Create a data frame that shows how often each of the 40 states with a *one-word name* appear. (In the next question we'll deal with two-word names.) A few things to keep in mind. Note that you don't need to type out the state names…use `state.name`, a built-in `R` object you utilized earlier in the semester. To eliminate the two-word state names for now, search for instances of a space. (Also note that URLs are repeated within the rows associated with each poll. Only analyze *unique* instances of each URL.)

```r
# remove state names with spaces
greg = gregexpr("[[:alpha:]]+ [[:alpha:]]+", state.name)
states = state.name[!(regmatches(state.name, greg) %in% state.name)]

# only use unique urls
urls = unique(df$url)

# get count for how many times each state appears in the urls
ct = unlist(lapply(states,
                   FUN = function(x){length(grep(x, urls,
                   ignore.case = TRUE))}))

# data frame of state counts
df.states = data.frame(states, ct)
names(df.states) = c("State", "Count")

print(df.states)
```

```
##             State Count
## 1        Alabama     2
## 2         Alaska     5
## 3        Arizona    34
## 4       Arkansas     1
## 5     California    10
## 6       Colorado    10
## 7    Connecticut     1
## 8       Delaware     2
## 9        Florida    38
## 10       Georgia    29
## 11        Hawaii     1
## 12          Idaho     1
## 13       Illinois     0
## 14        Indiana     1
## 15           Iowa    19
## 16         Kansas     7
## 17       Kentucky     8
## 18      Louisiana     0
## 19          Maine     8
## 20       Maryland     1
## 21 Massachusetts     3
## 22       Michigan    43
## 23      Minnesota    14
## 24    Mississippi     0
## 25       Missouri     2
## 26        Montana    12
## 27       Nebraska     3
## 28         Nevada     8
## 29           Ohio    19
## 30       Oklahoma     1
## 31         Oregon     1
## 32   Pennsylvania    31
## 33      Tennessee     0
## 34          Texas    34
## 35           Utah     9
## 36        Vermont     1
## 37       Virginia     3
## 38     Washington    17
## 39      Wisconsin    29
## 40        Wyoming     0
```

Question 14

(10 points)

Now, take the list of character vectors you generated in the last question, and paste contiguous words together (with a space separating them!). For instance, if you have a vector of strings `c("a","b","c")`, you should create the vector `c("a b","b c")`. (You might actually create `c("a b","b c","c ")` depending on how you create the vector, but that's fine.) Then use code similar to what you used above to count up instances of two-word state names. Put these into a data frame, like above, and merge that data frame with the one you created in the last question…and reorder the state name column to be in alphabetical order. Display your final result. I find that Arizona appears 34 times and New Hampshire 7 times.

```
# get df of two word states
states.2 = state.name[!(state.name %in% states)]
states.split = unlist(strsplit(states.2, split = " "))
states.2.df = data.frame(nrow = length(states.2), ncol = 2)

for (i in seq(2, length(states.split), 2)){
  state.vec = c(states.split[i - 1], states.split[i])
  states.2.df[i/2,1] = state.vec[1]
  states.2.df[i/2,2] = state.vec[2]
}

states.2.df$count = integer(length(states.2.df))
```

```r
names(states.2.df) = c("First", "Second", "Count")

# for each url
for (i in 1:length(urls)){
  # check if each state is in it
  for (j in 1:nrow(states.2.df)){
    v = c(states.2.df[j,1], states.2.df[j,2])
    if (sum(sapply(v, grepl, urls[i], ignore.case = TRUE)) == 2){
      states.2.df[j,3] = states.2.df[j,3] + 1
    }
  }
}

# combine count vectors
df.states.2 = unite(states.2.df, "State", c(1, 2), sep = " ")
state.counts = rbind(df.states, df.states.2)

# order the data frame alphabetically
state.counts = state.counts[order(state.counts$State),]

print(state.counts)
```

```
##            State Count
## 1        Alabama     2
## 2         Alaska     5
## 3        Arizona    34
## 4       Arkansas     1
## 5     California    10
## 6       Colorado    10
## 7    Connecticut     1
## 8       Delaware     2
## 9        Florida    38
## 10       Georgia    29
## 11        Hawaii     1
## 12         Idaho     1
## 13      Illinois     0
## 14       Indiana     1
## 15          Iowa    19
## 16        Kansas     7
## 17      Kentucky     8
## 18     Louisiana     0
## 19         Maine     8
## 20      Maryland     1
## 21 Massachusetts     3
## 22      Michigan    43
## 23     Minnesota    14
## 24   Mississippi     0
## 25      Missouri     2
## 26       Montana    12
## 27      Nebraska     3
## 28        Nevada     8
## 41 New Hampshire     7
## 42    New Jersey     5
## 43     New Mexico     3
## 44       New York     1
## 45 North Carolina    39
## 46   North Dakota     3
## 29           Ohio    19
## 30       Oklahoma     1
## 31         Oregon     1
## 32   Pennsylvania    31
## 47   Rhode Island     0
## 48 South Carolina     7
## 49   South Dakota     1
## 33      Tennessee     0
## 34          Texas    34
## 35           Utah     9
## 36        Vermont     1
## 37       Virginia     3
## 38     Washington    17
## 50  West Virginia     0
## 39      Wisconsin    29
## 40        Wyoming     0
```

Question 15

(10 points)

Write a function that takes in a candidate’s surname (e.g., “Biden”) and the full polling data frame and returns a set of names that represents the candidate’s unique “opponents”. This involves getting the set of `poll_id` values associated with the candidate, and outputting all names associated with those values (but without the original name). For instance, if “Trump” only appears in polls that also include “Rubio” and “Bush”, your function should output “Rubio” and “Bush”. (Note: outputting full names is fine.) Your function should return `NULL` if the name you provide is not in the list of candidates. It should also return null if there is a space in the name you provide. Your function should work with capitalized and uncapitalized input. Your function should return vector of type “character”, not a factor variable. Test your function using “Smith”, “Nancy Pelosi”, “Cuomo”, and “trump”.

```r
q.15 = function(x, df){
  # make name uppercase if it’s not already
  x = paste(toupper(substr(x, 1, 1)),
            substr(x, 2, nchar(x)), sep = "")

  # check if not is not in list of candidates or
    # if there is a space in the name
  cands = unique(df$answer)
  if (!(x %in% cands) | (grepl(" ", x))){
    return(NULL)
  }

  # get poll_ids with candidate
  ids = df %>%
    filter(., answer == x) %>%
    select(., poll_id)

  # for each poll_id, get other candidates
  # then append this to the list of opponents
  opps = character(0)
  for (i in 1:nrow(ids)){
    op = df %>%
      filter(., poll_id == ids[i,] & answer != x) %>%
      .$answer

    opps = append(opps, op)
  }

  # only return unique opponents
  return(unique(opps))
}

print(q.15("Smith", df))
```

```
## NULL
```

```
print(q.15("Nancy Pelosi", df))
```

```
## NULL
```

```
print(q.15("Cuomo", df))
```

```
## [1] "Biden"   "Trump"   "Clinton"
```

```
print(q.15("trump", df))
```

```
##  [1] "Biden"         "Jorgensen"     "Hawkins"       "West"
##  [5] "Blankenship"   "Simmons"       "Pierce"        "Pence"
##  [9] "Harris"        "De La Fuente"  "La Riva"       "Kennedy"
## [13] "Hornberger"    "Cuomo"         "Clinton"       "Obama"
## [17] "Amash"         "Sanders"       "Warren"        "Bloomberg"
## [21] "Buttigieg"     "Klobuchar"     "Gabbard"       "Steyer"
## [25] "Yang"          "Booker"        "Castro"        "O’Rourke"
## [29] "Haley"         "Bullock"       "Delaney"       "Gillibrand"
## [33] "Williamson"    "Messam"        "Bennet"        "de Blasio"
## [37] "Winfrey"       "Inslee"        "Hickenlooper"  "Gravel"
## [41] "Moulton"       "Rapinoe"       "Swalwell"      "Ryan"
## [45] "Schultz"       "Brown"         "Pelosi"        "Schumer"
## [49] "Ocasio-Cortez"
```

Question 16

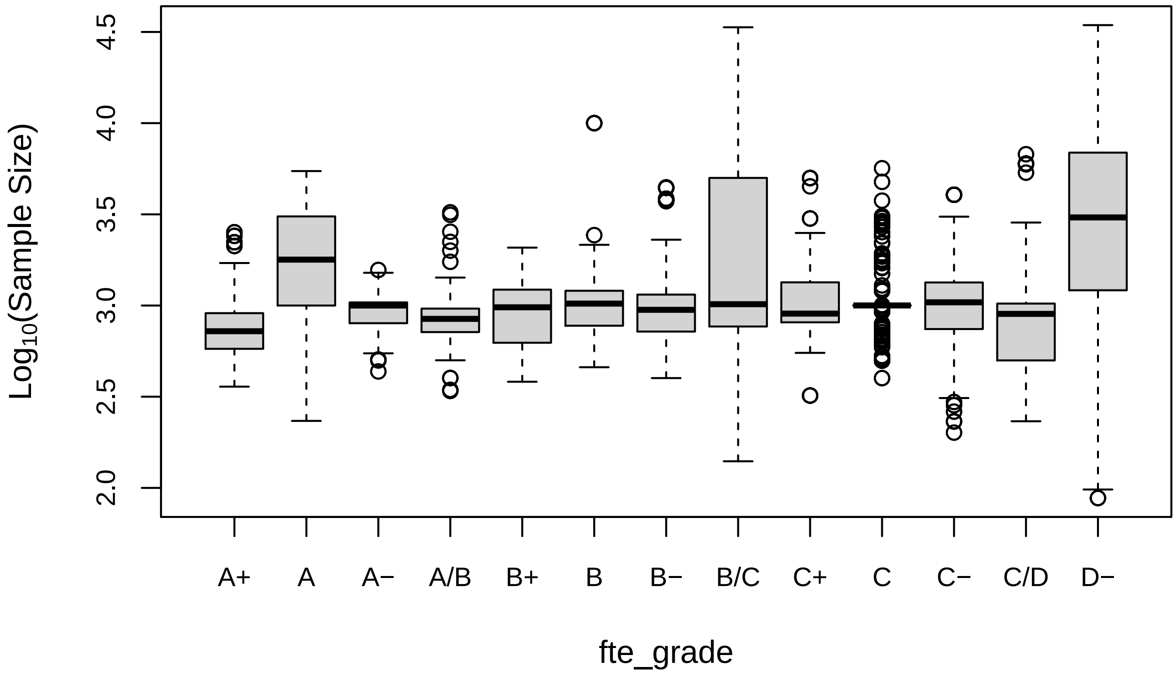
(10 points)

We might expect that the higher a grade a polling firm receives, the larger the sample size of its polls. Maybe, maybe not. Visually test this hypothesis by making side-by-side boxplots showing the distribution of log-base-10 of the sample size versus polling grade. Your plot should have data in the order “A+”, “A”, “A-”, “A/B”, etc., from left to right. Any data for which there is no grade should be excluded. This means you’ll have to drop `""` (or the empty string) as a factor level (hint: see `droplevels()`). For the plot: use base-R boxplotting, and change the y-axis label to be “Log”, followed by a subscript “10”, followed by “(Sample Size)”. You’ll need to Google how to do this: the function you are looking for is `expression()`.

```
# formula = log-base-10(sample size) ~ polling grade
df.g = df %>%
  group_by(., fte_grade) %>%
  filter(., fte_grade != "") %>% # remove empty grades
  select(., sample_size)
```

```
## Adding missing grouping variables: ‘fte_grade‘
```

```
# create col of log-base-10 sample size
df.g$samp.log = log10(df.g$sample_size)

# sort grades as factors
g.order = c("A+", "A", "A-", "A/B", "B+", "B", "B-", "B/C",
            "C+", "C", "C-", "C/D", "D-")
df.g$fte_grade = as.factor(df.g$fte_grade)
df.g$fte_grade = factor(df.g$fte_grade, levels = g.order)

# boxplot
boxplot(samp.log ~ fte_grade, data = df.g,
        ylab = expression(paste("Log"[10], "(Sample Size)")),
        par(cex.axis = .8))
```

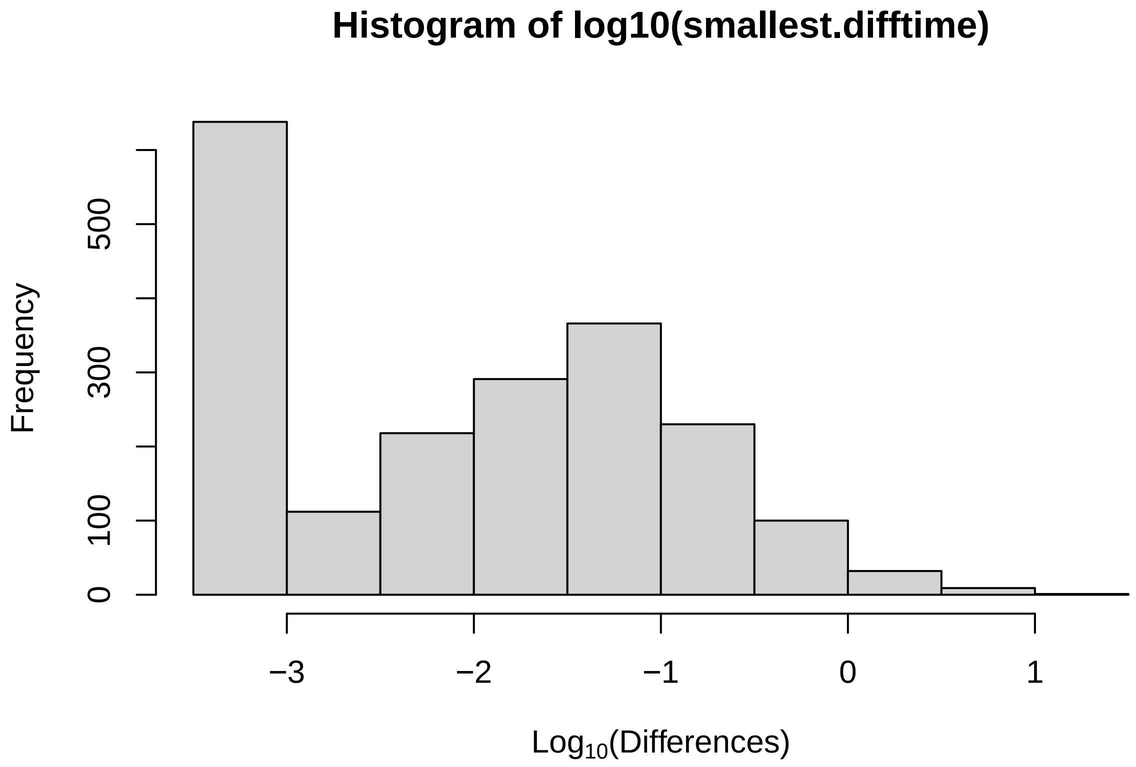


Question 17

(15 points)

For each poll, compute the absolute values of the amount of time between it and all other polls, as judged by the values in the created_at column, and retain the *smallest* value. (If my polls are at times 4, 5, and 8, I would retain the values 1, 1, and 3, where the first 1 is the time between 4 and 5, etc.) Histogram the result, and log-transform the values along the x-axis, using base-10. (Use expression() again for proper labelling.) Note that getting this right is a bit complicated. First, retain only the unique values of created_at. Second, you need to convert dates like 2/9/19 to 02/09/2019; a combination of strsplit(), sapply(), and paste() would work here. (This is not strictly necessary if you can insert "20" at the beginning of each year by another method.) Third, convert the data to POSIXlt format. Fourth, initialize a vector called smallest.difftime that will hold all the differences. Fifth, utilize a for-loop and the difftime() function with units days, and save the smallest difference to smallest.difftime. (If you just compute differences without difftime, the units can change from computation to computation. . . which is bad.) Then plot. If you have 2000 data, you will compute 2000 time differences; you'd want to sort these values and take the second sorted value, since the first sorted value is 0 (time difference between a datum and itself). Enjoy.

```r
# put times in POSIXlt format
times = unique(as.POSIXlt(df$created_at, format = "%m/%d/%Y %H:%M"))

# find time differences
smallest.difftime = integer(0)
for (i in 1:length(times)){
  small = min(abs(difftime(times[i], times[-i], units = "days")))
  smallest.difftime = append(small, smallest.difftime)
}

# get numeric values and plot
smallest.difftime = as.numeric(smallest.difftime)
hist(log10(smallest.difftime),
     xlab = expression(paste("Log"[10], "(Differences)")))
```



Question 18

(10 points)

Edit your file `dark_and_stormy.R` in your 36-350 `Git` repo so that it prints "It was a dark and stormy night so I stayed in to complete my R project and contemplate the future." Then push your change to `GitHub` and use `source_url()` from the `devtools` package to run the code in the chunk below.

```r
devtools::source_url("https://raw.githubusercontent.com/sophiahill/36-350/main/dark_and_stormy.R")
```

```
## SHA-1 hash of file is bd337a30377fdf65c16541e877b05571f08dbe0e
```

```
## [1] "It was a dark and stormy night so I stayed in to complete my R project
and contemplate the future."
```

And with that, you're done.