MSCS 264: Homework #11

For this assignment, you should submit a knitted pdf file to Moodle. Be sure to include all of your R code, in addition to your output, plots, and written responses. **Read each question carefully and check your assignment prior to submission.** Use what you have learned about the principles of effective statistical communication and visualization.

The data for this homework are from the Federal Election Commission (FEC) and includes all candidates who have filed with the FEC for elections in the U.S. House of Representatives, Senate or Presidency. These data include candidates who ran for office in Minnesota in 2018. (The 2019-2020 data are available here if you're curious, but they cleaned up the formatting, so it's way less exciting to practice `readr` stuff like parse!) The data set is called CandidateSummaryAction.csv and is in the Class > Data folder.

The Data folder also includes a txt file with definitions of variables.

1. (a) Read in your .csv file using `read_csv()`. Remember to examine the documentation if you are confused about a function, `?read_csv`.

```
electionStuff <- read_csv("~/Mscs 264 S23/Class/Data/CandidateSummaryAction.csv")
```

```
## Rows: 89 Columns: 50
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (43): lin_ima, can_id, can_nam, can_off, can_off_sta, can_par_aff, can_i...
## dbl  (2): can_off_dis, can_zip
## lgl  (5): off_to_fun, off_to_leg_acc, exe_leg_acc_dis, fun_dis, deb_owe_to_com
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

(b) Subset the data to include only the following variables (change the dataset name in the code below to match yours, and then remove `eval = FALSE`). Call the new dataset `fec_mn`

1 pt

```
fec_mn <- electionStuff %>%
  select(can_par_aff, can_inc_cha_ope_sea, can_nam, can_off, can_off_sta,
         can_off_dis, ind_con, par_com_con, oth_com_con, can_con, tot_loa,
         net_con, can_loa_rep, oth_loa_rep, tot_loa_rep, ind_ref,
         par_com_ref, oth_com_ref, tot_con_ref, net_ope_exp, cov_sta_dat,
       cov_end_dat)
```

2. (a) What type of variable is `cov_sta_dat`? If the class is not Date, coerce `cov_sta_dat` and `cov_end_dat` into Dates inside a `mutate` function. *Be sure to save this change to fec_mn, not just print it out!* Check the CandidateSumaryAction.dictionary.txt file in Class > Data. What do these variable mean?

```
typeof(fec_mn$cov_sta_dat)
```

```
## [1] "character"
```

Ans: The variable `cov_sta_dat` is a character variable in this dataset.

```
fec_mn <- fec_mn|>
  mutate(cov_sta_dat = mdy(cov_sta_dat), cov_end_dat = mdy(cov_end_dat))

typeof(fec_mn$cov_sta_dat)
```

```
## [1] "double"
```

```
typeof(fec_mn$cov_end_dat)
```

```
## [1] "double"
```

Ans: `cov_sta_dat` means coverage start date and `cov_end_dat` means coverage end date.

(b) Create a new variable, coverage time. (end date minus start date) What type of variable is this?

```
fec_mn <- fec_mn|>
  mutate(cov_time = cov_end_dat - cov_sta_dat)
```

Ans: the coverage time is a drtn type which is difference in time in days.

(c) Now that these columns are in date format, you can find the following FOR EACH PARTY:

- average coverage start date
- average coverage end date
- average coverage_time
- number of candidates

(Note: first identify whcih variable indicates political party!) There are MISSING VALUES in the date columns. Think about how you want to handle them; there are lots of options! Describe in words what you chose to do in order to handle the missing values.

```
fec_mn|>
  select(can_par_aff, cov_sta_dat, cov_end_dat, cov_time)|>
  drop_na()|>
  group_by(can_par_aff)|>
  summarise(avg_cov_time = mean(cov_time), avg_sta_dat = mean(cov_sta_dat), avg_end_dat = mean(cov_end_
```

```
## # A tibble: 5 x 5
##   can_par_aff avg_cov_time  avg_sta_dat avg_end_dat num_candidate
##   <chr>       <drtn>        <date>      <date>              <int>
## 1 DEM         478.5000 days 2017-06-03  2018-09-24             16
## 2 DFL         355.8125 days 2017-08-24  2018-08-15             16
## 3 IDP         287.0000 days 2018-01-01  2018-10-15              1
## 4 IND         334.5000 days 2018-01-30  2018-12-31              2
## 5 REP         497.3529 days 2017-06-17  2018-10-28             17
```

Ans: I chose to use drop_na() function to get rid of the missing values after I have selected only the desired variable columns.

(d) Find the latest coverage start dates (`cov_sta_dat`) by incumbency status (`can_inc_cha_ope_sea`).

```
fec_mn|>
  select(can_inc_cha_ope_sea, cov_sta_dat)|>
  group_by(can_inc_cha_ope_sea)|>
  slice_max(cov_sta_dat, n = 1)
```

```
## # A tibble: 3 x 2
## # Groups:   can_inc_cha_ope_sea [3]
##   can_inc_cha_ope_sea cov_sta_dat
##   <chr>                <date>
## 1 CHALLENGER           2018-05-01
## 2 INCUMBENT            2017-12-15
## 3 OPEN                 2018-06-08
```

(e) You probably noticed something weird. The code below uses the function `which` to identify the row of the data that is the problem, then we can use View() to see that row plus the one above and below it. Describe what you think happened (why does the data look like this?)

```
which(fec_mn$can_inc_cha_ope_sea == "5440 MOREHOUSE DRIVE #4000")
View(fec_mn[17:19, ])
```

Ans: Somebody entered the address of something instead of putting the incumbency status of the candidate in variable `can_inc_cha_ope_sea`. It looks like a fake record in the dataset.

Since row 18 mostly has missing data anyway, we will remove it for now:

```
fec_mn <- fec_mn %>% slice(-18)
```

(f) Now that you have removed the unusual row, let's find the latest start by incumbency status again. This time, print the entire row of information so we can see the candidate's name! (Hint: slice_max instead of summarize!!!)

```
fec_mn|>
  #select(can_inc_cha_ope_sea, cov_sta_dat)|>
  group_by(can_inc_cha_ope_sea)|>
  slice_max(cov_sta_dat, n = 1)
```

```
## # A tibble: 3 x 23
## # Groups:   can_inc_cha_ope_sea [3]
##   can_par_aff can_inc_~1 can_nam can_off can_o~2 can_o~3 ind_con par_c~4 oth_c~5
##   <chr>       <chr>      <chr>   <chr>   <chr>     <dbl> <chr>   <chr>   <chr>
## 1 REP         CHALLENGER BEY, N~ S       MN            0 <NA>    <NA>    <NA>
## 2 DEM         INCUMBENT  SMITH,~ S       MN            0 $7,313~ <NA>    $1,500~
## 3 DEM         OPEN       ANDERS~ H       MN            5 $510,1~ <NA>    $24,20~
## # ... with 14 more variables: can_con <chr>, tot_loa <chr>, net_con <chr>,
## #   can_loa_rep <chr>, oth_loa_rep <chr>, tot_loa_rep <chr>, ind_ref <chr>,
## #   par_com_ref <chr>, oth_com_ref <chr>, tot_con_ref <chr>, net_ope_exp <chr>,
## #   cov_sta_dat <date>, cov_end_dat <date>, cov_time <drtn>, and abbreviated
## #   variable names 1: can_inc_cha_ope_sea, 2: can_off_sta, 3: can_off_dis,
## #   4: par_com_con, 5: oth_com_con
## # i Use `colnames()` to see all variable names
```

3. (a) What type of variable is `ind_con`? Why did R treat it as this variable type?

```
typeof(fec_mn$ind_con)
```

```
## [1] "character"
```

Ans: `ind_con` is a character variable. It was treated as a character variable because it has dollar sign ($) in front of each dollar amount.

(b) Convert the ind_con variable to numeric, and then find the maximum individual contributions (`ind_con`) by party. Calcuate the max both with na.rm = TRUE and with na.rm = FALSE. How does max behave with missing values? Is this what you would have expected?

```
fec_mn|>
  mutate(ind_con = parse_number(ind_con))|>
  group_by(can_par_aff)|>
  summarise(max_con = max(ind_con))
```

```
## # A tibble: 8 x 2
##   can_par_aff max_con
##   <chr>         <dbl>
## 1 DEM              NA
## 2 DFL              NA
## 3 IDP              NA
## 4 IND           17239
## 5 LMN              NA
## 6 MGP              NA
## 7 OTH              NA
## 8 REP              NA
```

```
fec_mn|>
  mutate(ind_con = parse_number(ind_con))|>
  #drop_na(ind_con)|>
  group_by(can_par_aff)|>
  summarise(max_con = max(ind_con, na.rm = TRUE))
```

```
## Warning in max(ind_con, na.rm = TRUE): no non-missing arguments to max;
## returning -Inf

## Warning in max(ind_con, na.rm = TRUE): no non-missing arguments to max;
## returning -Inf

## Warning in max(ind_con, na.rm = TRUE): no non-missing arguments to max;
## returning -Inf

## # A tibble: 8 x 2
##   can_par_aff  max_con
##   <chr>          <dbl>
## 1 DEM         7313957.
## 2 DFL         4889906.
## 3 IDP             318.
## 4 IND           17239
## 5 LMN            -Inf
## 6 MGP            -Inf
## 7 OTH            -Inf
## 8 REP         3829946.
```

Ans: When there are missing values after group_by then the max function inside summarise creates missing values for the maximum values per group. I would expect max to produce missing values because it has to do some calculations with the missing values to produce the max value. But any calculations with missing values produce missing values. But when we use na.rm = TURE then some of the party does not have any record that does not have non-missing values. So wehn trying to calculate max the function sets -inf as the initial max and that max value is never changed so that's why we see the -inf for some parties.

(c) Note that NA values in the ind_con variable indicate that no individual contributions were made (e.g. $0). How should we calculate the mean? This code adds two columns to the table: What is Mean_narm doing? What is Mean_sum_over_n doing? Which is a better way to calculate the mean?

```
fec_mn %>%
 mutate(ind_con_num = parse_number(ind_con)) %>%
  group_by(can_par_aff) %>%
  summarize(Mean_narm = mean(ind_con_num, na.rm = TRUE),
            n= n(),
            Mean_sum_over_n = sum(ind_con_num, na.rm = TRUE)/n)
```

```
## # A tibble: 8 x 4
##   can_par_aff Mean_narm     n Mean_sum_over_n
##   <chr>           <dbl> <int>           <dbl>
## 1 DEM          1168498.    21         890284.
## 2 DFL           841186.    29         406090.
## 3 IDP              318.     2            159.
## 4 IND            11358.     2          11358.
## 5 LMN               NaN     1              0
## 6 MGP               NaN     1              0
## 7 OTH               NaN     2              0
## 8 REP           753985.    30         402125.
```

Ans: If the NA values correspond to $0 contribution then while counting the mean contribution we need to include the NA values as $0 contribution so the mean is correct. The variable Mean_narm is calculating the mean contribution of only the candidates who have more than $0 contribution and dropping the NA containing records all together. On the other hand Mean_sum_over_n is calculating the sum of contributions excluding the NA (i.e. $0 contribution) and then dividing the sum by the number of candidates in each group including the candidates with NA (i.e. $0) contribution which calculates the more accurate mean that includes all candidates irrespective of their contribution. Therefore, Mean_sum_over_n is a better way to calculate the mean.

4. Your final tibble should be 88 x 23 after selecting variables in (1c), mutating dates in (2a), creating a new variable in 2b, and removing a row in (2e). Write this tibble to your student folder as a csv file. Show that when you read it back to R, the column specifications are maintained. (Use write_csv to write the file; then read_csv to read it back in; then print out the dataset!)

```
write_csv(fec_mn, "~/Mscs 264 S23/Submit Section A/Intisar/fec_mn.csv")
```

```
fec_mn <- read_csv("~/Mscs 264 S23/Submit Section A/Intisar/fec_mn.csv")
```

```
## Rows: 88 Columns: 23
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (19): can_par_aff, can_inc_cha_ope_sea, can_nam, can_off, can_off_sta, ...
## dbl   (2): can_off_dis, cov_time
## date  (2): cov_sta_dat, cov_end_dat
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
typeof(fec_mn$cov_end_dat)
```

```
## [1] "double"
```

```
typeof(fec_mn$cov_sta_dat)
```

```
## [1] "double"
```

```
typeof(fec_mn$cov_time)
```

```
## [1] "double"
```

Ans: The column specifications are maintained as we can see the date variables are stored as double/date format.

```
print(fec_mn)
```

```
## # A tibble: 88 x 23
##    can_par_aff can_inc~1 can_nam can_off can_o~2 can_o~3 ind_con par_c~4 oth_c~5
##    <chr>       <chr>     <chr>   <chr>   <chr>     <dbl> <chr>   <chr>   <chr>
##  1 DEM         OPEN      ABDULA~ H       MN            5 $129,1~ <NA>    $2,613~
##  2 DFL         OPEN      AKZAM,~ H       MN            1 <NA>    <NA>    <NA>
##  3 DEM         CHALLENG~ ALI, A~ S       MN            0 <NA>    <NA>    <NA>
##  4 DEM         OPEN      ANDERS~ H       MN            5 $510,1~ <NA>    $24,20~
##  5 REP         CHALLENG~ ANDERS~ S       MN            0 $5,240~ <NA>    <NA>
##  6 REP         CHALLENG~ ANDERS~ S       MN            0 <NA>    <NA>    <NA>
##  7 REP         CHALLENG~ ANDERS~ S       MN            0 <NA>    <NA>    <NA>
##  8 DEM         OPEN      AUSTIN~ H       MN            1 $5,818~ <NA>    <NA>
##  9 REP         CHALLENG~ BARNHE~ S       MN            0 <NA>    <NA>    <NA>
## 10 REP         CHALLENG~ BEY, N~ S       MN            0 <NA>    <NA>    <NA>
## # ... with 78 more rows, 14 more variables: can_con <chr>, tot_loa <chr>,
## #   net_con <chr>, can_loa_rep <chr>, oth_loa_rep <chr>, tot_loa_rep <chr>,
## #   ind_ref <chr>, par_com_ref <chr>, oth_com_ref <chr>, tot_con_ref <chr>,
## #   net_ope_exp <chr>, cov_sta_dat <date>, cov_end_dat <date>, cov_time <dbl>,
## #   and abbreviated variable names 1: can_inc_cha_ope_sea, 2: can_off_sta,
## #   3: can_off_dis, 4: par_com_con, 5: oth_com_con
## # i Use ‘print(n = ...)‘ to see more rows, and ‘colnames()‘ to see all variable names
```