Problem set #4

Spencer Fairbairn

Gathering Polling Data

1. We all have our own biases and expectations that may color the way that we interpret data. Let's start by acknowledging those biases such that we can actively consider them as we explore our data. Who do you think will win the nomination? Why do you think they will win? If you have not been following the race, note that Ballotpedia (https://ballotpedia.org/List_of_registered_2020_presidential_candidates

(Links to an external site.)) has a running list of active candidates and details about those candidates including their policy positions.

While reading through news articles we can see that the top democreatic candidate would have to be Joe Biden, He has a clear path to be the democratic leader since there has been a lot of candidates dropping out.

2. Most major news outlets collect their own polling data about who the public thinks will win an election. Real Clear Politics is a news agency that aggregates data about different polls. If you visit their website, you'll notice that they have data from a large number of polls (e.g., https://www.realclearpolitics.com/epolls/2020/president/us/2020_democratic_presidential_nomination-6730.html

(Links to an external site.)

), but no way to directly download that data. We're going to pull down their polling data using their API. Details about this API are available at https://pypi.org/project/realclearpolitics/

(Links to an external site.)

. This API allows us to write python scripts to pull data from specific polls, but we can also use its command line tool to pull down entire datasets. To do this, we first need to install the API. From your command line, run the following command:

```
pip install realclearpolitics
```

Once that's installed, you can download data from the RCP polls using the command:

```
rcp
```

(Links to an external site.)

(Links to an external site.)

Make sure this datafile is in your final submission. You may wish to rename the datafile for simplicity.

From rcp import get_polls, get_poll_data

From pprint import pprint

Import pandas as pd

Import pprint as pprint

Import matplotlib.pyplot as plt

3. Now open the data file using a program of your choice. Delete the row of the data from the Poll "RCP Average": this is an average computed by RCP that may accidentally skew our own computations. The value in each candidate's column corresponds to the percentage of polled voters voting for each candidate. Make three qualitative observations about the data.

1. ***Gabbard clearly has the lowest number of votes.***
2. ***Sanders and Biden have the Highest number of votes***
3. ***Sanders has the Highest spread***

Normalizing Our Data

One thing you may notice about the data is that rows don't always sum to 100%, yet ideally, we have data about 100% of voters. Those remaining voters are largely composed of "undecided" voters (people who are still trying to figure out who they will vote for) while a few are voting for candidates not in the poll. It may be useful to consider these voters as we explore our data. The next problems will help us to preprocess the data and integrate those voters into our analysis. **Note:** Having a list of candidate names to iterate over might be helpful!

4. Create a new python file for your political prediction work. Import your`utils.py` file from Problem Set Three and use this file to load in the polling data using `loadAndCleanData`. Note: You may wish to copy the `utils.py` file into your current directory to make this import easier.

def loadAndCleanData(fileName):

data = pd.read_csv(fileName)

newData = data.fillna("0")

print(newData)

return newData

cleanData = loadAndCleanData(“democraticnominations.csv”)

5. Create a new file called `pollingData.py` and import that into your main python file. In `pollingData.py`, create a function called `normalizeData` that will take a dataframe as an argument. This function should add a new column to your dataframe called "Undecided" and fill that column such that the sum of all candidates polling numbers plus this column sums to 100. **Note:** you may need to manually correct values in your dataset for this to work correctly. Specifically, look for "--." There are ways of doing this programmatically by using the replace function and then cleaning each column,

(Links to an external site.)

but sometimes it is easier to fix your data directly.

Def normalizedata(data):

data.append(column)

Name = “Undecided”

Def normalizeData(data):

undecided_list = [ ]

For i, v in data.iterrows( ):

undecided_list.append((100 - (v[3] + v[4])))

data[“Undecided”] = undecided_list

Return data

6. In your main file, run `normalizeData` on your current dataframe to update your dataframe. Print the dataframe to confirm that the function worked correctly.

7. Write a function in `pollingData.py` called `plotCandidate` that takes in a candidate's name and a dataframe and generates a scatterplot based on that candidate's data, with the Poll value on the x-axis and the candidate's rating in each poll (i.e., the data in their column) on the y-axis.

CLARIFICATION: Note that the `plt.scatter` function might come in handy here https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html#matplotlib.pyplot.scatter

(Links to an external site.)

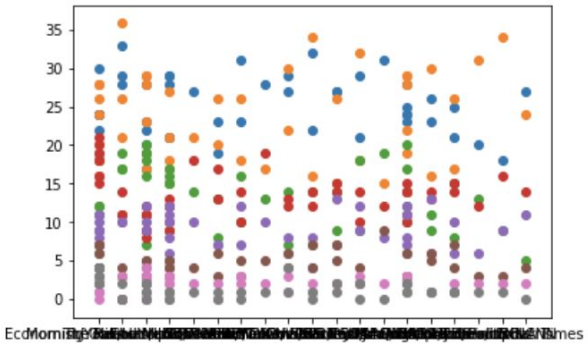
. Here's a nice blog for using this function https://chrisalbon.com/python/data_visualization/matplotlib_scatterplot_from_pandas/

(Links to an external site.)

.

Create several plots of your data based on these polling numbers in your main python file using this function (one plot per candidate). Who do you think will win based on the raw polling numbers and why? Note that it may be helpful to create an object representing each candidate to save yourself some work. Note that if you use this option, any function taking a candidate's name will simply take `self`.

```
def Candidateplot(candidate,self):
        plt.scatter(y=self[candidate], x=self["Poll"])
san = Candidateplot("Sanders", df)
biden  = Candidateplot("Biden", df)
bloom = Candidateplot("Bloomberg", df)
warren = Candidateplot("Warren", df)
butt = Candidateplot("Buttigieg", df)
klob = Candidateplot("Klobuchar", df)
stey = Candidateplot("Steyer", df)
gab = Candidateplot("Gabbard", df)
```



I believe that biden will win

8. In your `pollingData.py` file, write a function called `statsPerCandidate` that takes a dataframe and a candidate's name and computes and returns the average of polling numbers for a given candidate across all polls. Note that pandas' `mean` function might come in handy here (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.mean.html (Links to an external site.)).

def statsPerCandidate(candidate, self):

    averagePoll = self[candidate].mean()

    return averagePoll

bide = statsPerCandidate("Biden", df) 23.520833333333332

sand = statsPerCandidate("Sanders", df) 24.75

warr = statsPerCandidate("Warren", df) 13.458333333333334

bloomb = statsPerCandidate("Bloomberg", df) 12.6875

butti = statsPerCandidate("Buttigieg", df) 8.354166666666666

klobu = statsPerCandidate("Klobuchar", df) 4.3125

steye = statsPerCandidate("Steyer", df) 2.0

gabb = statsPerCandidate("Gabbard", df) 1.3125

print(bide, sand, warr, bloomb, butti, klobu, steye, gabb)

By looking at the data above we can see that bernie has the highest score meaning that he is in the lead

9. In your main file, run `statsPerCandidate` for each candidate in your dataset (include undecideds). Print these averages out as you compute them. What are the averages for each candidate? Who is polling highest on average?

candidateList = []

for candidate in demNoms.columns:

  if candidate not in ['Poll', 'Date', "Sample", 'Spread', 'Undecided']:

    candidateList.append(candidate)

    print(candidate, statsPerCandidate(candidate, demNoms))

Weighting Predictions by Sample Quality

By now, you've probably noticed that different polls may lead to drastically different predictions. Next we're going to look at the quality of each poll and use that to take a more nuanced look at our polling data.

10. The "Sample" column of our dataframe is labeled using either "RV" or "LV" This indicates whether the poll reflects registered voters or likely voters. You can get more information on this distinction from the polling agency Gallup here: https://news.gallup.com/poll/110287/what-difference-between-registered-voters-likely-voters.aspx (Links to an external site.)

```
def cleanSample(data):
    LV = []
    RV = []
    for i in data["Sample"]:
        LV.append(i[-2:])
        RV.append(i[:-2])
    for i in range(len(RV)):
        if RV[i] == '':
            RV[i] = 0
        else:
            RV[i] = int(RV[i])
    data["Sample Type"] = LV
    data["Sample Size"] = RV
    data = data.replace('', None)
    return data
```

. For now, we need to separate these labels from the sample size to figure out how we might weight our samples. In `pollingData.py`, write a function called `cleanSample` that takes a dataframe as an argument. The function should add a column called "Sample Type" and one called "Sample Size" to your dataframe. It should then take the "Sample" value and fill the "Sample Size" column with the number of people

sampled (i.e., the number before either LV or RV) and put the kind of voters sampled (LV or RV) in the SampleType column.

11. In your main file, run this function on your dataframe.

cleansample(demNoms)

12. In the same way that we can compute a weighted sum, we can also compute a *weighted average*, that is, an average that considers how important each datapoint is to understanding the average. Write a function in your `pollingData.py` file called `computePollWeight` that takes a dataframe and a poll name as arguments and returns a weight based on that poll. You can compute this weight by first computing the number of people sampled in all polls. You then return the weight of the given poll divided by the weight of all polls. This will give you the percentage of all polled respondents that this poll reflects. Test your function in your main python file by using it to print out the weight of a few popular polls.

Clarification: If there are multiple polls from the same outlet, use the average of the poll numbers to compute the poll weight.

```
def computePollWeight(self, pollName):
    return sum(self[self['Poll'] == pollName]["Sample Size"]) / sum(self["Sample Size"])

morn = computePollWeight(df, "Morning ConsultM. Consult")

hill = computePollWeight(df, "The Hill/HarrisXThe Hill")

print(morn, hill)
```

13. Write a second function in your `pollingData.py` file called `weightedStatsPerCandidate` that takes in a candidate name and dataframe and returns the weighted average of that candidate's polling data. Use your `computePollWeight` function to compute this number.

```
def weightedStatsPerCandidate(candidate, self):
    weighted = []
    for poll in self["Poll"].unique():
        x = sum(self[self["Poll"] == poll][candidate])
        y = computePollWeight(self, poll)
```

```
        weighted.append(x*y)

    return sum(weighted)/len(weighted)
```

14. Run `weightedStatsPerCandidate` in your main python file to compute the weighted average for each candidate. Print this average. Based on these new numbers, who do you think will win the nomination and why?

```
bid1 = weightedStatsPerCandidate("Biden", df)
san1 = weightedStatsPerCandidate("Sanders", df)
bloom1 = weightedStatsPerCandidate("Bloomberg", df)
war1 = weightedStatsPerCandidate("Warren", df)
butt1 = weightedStatsPerCandidate("Buttigieg", df)
klob1 = weightedStatsPerCandidate("Klobuchar", df)
stey1 = weightedStatsPerCandidate("Steyer", df)
gab1 = weightedStatsPerCandidate("Gabbard", df)

print(bid1, san1, war1, bloom1, butt1, klob1, stey1, gab1)
```

Statistics Over Time:

15. You might note that the polls in the dataset occur over various timepoints. One question you might have is whether the polling numbers for any sets of candidates are correlated. This often implies that sentiment improving or declining towards a set of policies changes the numbers for all candidates. It can also capture how an increase in one candidate's numbers suggests a decrease in another's (e.g., the candidates are *anticorrelated*, meaning their correlation is negative). Write a function called `computeCorrelation` that computes the correlation between any two candidates. The function should take as input two candidates and a dataframe and return the correlation between those candidates.

```
def computeCorrelation(candidate1, candidate2, self):

    print(candidate1, "and", candidate2, " have a correlation of",
self[candidate1].corr(self[candidate2]))

computeCorrelation("Sanders", "Biden", demNoms)

computeCorrelation("Sanders", "Bloomberg", demNoms)

computeCorrelation("Sanders", "Warren", demNoms)
```

computeCorrelation("Sanders", "Buttigieg", demNoms)

computeCorrelation("Sanders", "Klobuchar", demNoms)

computeCorrelation("Sanders", "Steyer", demNoms)

Sanders and Biden have a correlation of -0.553284587148943

Sanders and Bloomberg have a correlation of 0.5001912408750656

Sanders and Warren have a correlation of -0.2481059643176889

Sanders and Buttigieg have a correlation of 0.384610202616565

Sanders and Klobuchar have a correlation of 0.27366268597430043

Sanders and Steyer have a correlation of -0.11304817626267137

computeCorrelation("Biden", "Bloomberg", df)

computeCorrelation("Biden", "Warren", df)

computeCorrelation("Biden", "Buttigieg", df)

computeCorrelation("Biden", "Klobuchar", df)

computeCorrelation("Biden", "Steyer", df)

Biden and Bloomberg have a correlation of -0.47549215276831625

Biden and Warren have a correlation of 0.21541823902261228

Biden and Buttigieg have a correlation of -0.7359534870307493

Biden and Klobuchar have a correlation of -0.6166637124965957

Biden and Steyer have a correlation of -0.12346180389199973

Biden and Bloomberg have a correlation of -0.47549215276831625

Biden and Warren have a correlation of 0.21541823902261228

Biden and Buttigieg have a correlation of -0.7359534870307493

Biden and Klobuchar have a correlation of -0.6166637124965957

Biden and Steyer have a correlation of -0.12346180389199973

Bloomberg and Warren have a correlation of -0.5661999129408489

Bloomberg and Buttigieg have a correlation of 0.21990524046491924

Bloomberg and Klobuchar have a correlation of 0.02016132856034869

Bloomberg and Steyer have a correlation of 0.05464122874169334

computeCorrelation("Warren", "Buttigieg", df)

computeCorrelation("Warren", "Klobuchar", df)

computeCorrelation("Warren", "Steyer", df)

Warren and Buttigieg have a correlation of -0.12778779441556246

Warren and Klobuchar have a correlation of 0.13149952402665743

Warren and Steyer have a correlation of -0.39679125868104903

computeCorrelation("Buttigieg", "Klobuchar", df)

computeCorrelation("Buttigieg", "Steyer", df)

Buttigieg and Klobuchar have a correlation of 0.385532320381382

Buttigieg and Steyer have a correlation of 0.282527112224575

computeCorrelation("Klobuchar", "Steyer", df)

Klobuchar and Steyer have a correlation of -0.06322027015090681

16. Use `computeCorrelation` to test the correlation between all candidates. Which candidates are most correlated? Which are least correlated?

```
repeats = []
for candidate1 in can_list:
    for candidate2 in can_list:
        if candidate1 != candidate2:
            if [candidate1,candidate2] not in repeats and [candidate2, candidate1] not in repeats:
                print(candidate1, "and", candidate2, "have a correlation of", computeCorrelation(candidate1, candidate2, df))
                repeats.append([candidate1, candidate2])
```

Biden and Sanders have a correlation of -0.553284587148943

Biden and Sanders have a correlation of None

Biden and Bloomberg have a correlation of -0.47549215276831625

Biden and Bloomberg have a correlation of None

Biden and Warren have a correlation of 0.21541823902261228

Biden and Warren have a correlation of None

Biden and Buttigieg have a correlation of -0.7359534870307493

Biden and Buttigieg have a correlation of None

Biden and Klobuchar have a correlation of -0.6166637124965957

Biden and Klobuchar have a correlation of None

Biden and Gabbard have a correlation of 0.014038694898209522

Biden and Gabbard have a correlation of None

Biden and Steyer have a correlation of -0.12346180389199973

Biden and Steyer have a correlation of None

Biden and Undecided have a correlation of 0.21134293602355686

Biden and Undecided have a correlation of None

Sanders and Bloomberg have a correlation of 0.5001912408750656

Sanders and Bloomberg have a correlation of None

Sanders and Warren have a correlation of -0.2481059643176889

Sanders and Warren have a correlation of None

Sanders and Buttigieg have a correlation of 0.384610202616565

Sanders and Buttigieg have a correlation of None

Sanders and Klobuchar have a correlation of 0.27366268597430043

Sanders and Klobuchar have a correlation of None

Sanders and Gabbard have a correlation of -0.07709255859630605

Sanders and Gabbard have a correlation of None

Sanders and Steyer have a correlation of -0.11304817626267137

Sanders and Steyer have a correlation of None

Sanders and Undecided have a correlation of -0.6932034302859948

Sanders and Undecided have a correlation of None

Bloomberg and Warren have a correlation of -0.5661999129408489

Bloomberg and Warren have a correlation of None

Bloomberg and Buttigieg have a correlation of 0.21990524046491924

Bloomberg and Buttigieg have a correlation of None

Bloomberg and Klobuchar have a correlation of 0.02016132856034869

Bloomberg and Klobuchar have a correlation of None

Bloomberg and Gabbard have a correlation of -0.21698108831718554

Bloomberg and Gabbard have a correlation of None

Bloomberg and Steyer have a correlation of 0.05464122874169334

Bloomberg and Steyer have a correlation of None

Bloomberg and Undecided have a correlation of -0.44045699180378706

Bloomberg and Undecided have a correlation of None

Warren and Buttigieg have a correlation of -0.12778779441556246

Warren and Buttigieg have a correlation of None

Warren and Klobuchar have a correlation of 0.13149952402665743

Warren and Klobuchar have a correlation of None

Warren and Gabbard have a correlation of 0.3028703909442035

Warren and Gabbard have a correlation of None

Warren and Steyer have a correlation of -0.39679125868104903

Warren and Steyer have a correlation of None

Warren and Undecided have a correlation of -0.12117466459590887

Warren and Undecided have a correlation of None

Buttigieg and Klobuchar have a correlation of 0.385532320381382

Buttigieg and Klobuchar have a correlation of None

Buttigieg and Gabbard have a correlation of 0.01128645409361886

Buttigieg and Gabbard have a correlation of None

Buttigieg and Steyer have a correlation of 0.282527112224575

Buttigieg and Steyer have a correlation of None

Buttigieg and Undecided have a correlation of -0.32438974612281196

Buttigieg and Undecided have a correlation of None

Klobuchar and Gabbard have a correlation of 0.1203839305725848

Klobuchar and Gabbard have a correlation of None

Klobuchar and Steyer have a correlation of -0.06322027015090681

Klobuchar and Steyer have a correlation of None

Klobuchar and Undecided have a correlation of -0.16834706710668232

Klobuchar and Undecided have a correlation of None

Gabbard and Steyer have a correlation of 0.05770297860846152

Gabbard and Steyer have a correlation of None

Gabbard and Undecided have a correlation of -0.2190108946645316

Gabbard and Undecided have a correlation of None

Steyer and Undecided have a correlation of 0.06827859884132419

Steyer and Undecided have a correlation of None

17. Correlations can indicate shared policies: if two candidates' polling numbers increase together, it may indicate a preference towards a certain kind of policy. Post-Super Tuesday, Biden and Sanders are the front runners. We will use that knowledge plus their correlation with other candidates to guess how polling numbers might turn out based on historical data and correlations.

```
def superTuesday(self, can_list):
    sandersST = []
    bidenST = []

    for i, x in df.iterrows():
        bidenCount = x["Biden"]
        sandersCount = x["Sanders"]

        for candidate in can_list:
            if candidate != "Sanders" and candidate != "Biden":
                bidencor = computeCorrelation("Biden", candidate, self)
                sanderscor = computeCorrelation("Sanders", candidate, self)

                if abs(bidencor) > abs(sanderscor):
                    bidenCount += x[candidate]
                else:
                    sandersCount += x[candidate]
        bidenST.append(bidenCount)
        sandersST.append(sandersCount)
```

To do this, we'll assign polling numbers from other candidates to Biden or to Sanders based on their correlation in historical polls. One common theory used in political prediction is that each candidate that has dropped out is likely to be more correlated with either Biden or Sanders and their voters are likely to vote for the candidate with the most in common with their original candidate's viewpoints. This is a simplification on how this all works, but let's test the theory.

Write a function called `superTuesday` in `pollingData.py`that takes in a dataframe. The function should add two new columns to your dataset, "BidenST" and "SandersST." For each candidate that has dropped out of the race, if that candidate is more correlated with Biden, add their total to Biden's polling numbers. If they are more correlated with Sanders, add their total to Sanders' polling numbers. BidenST will store the new total for Biden and SandersST will store the new total for Sanders.

18. Run `superTuesday` on your dataset. Use it to compute the mean and weighted mean for Biden and Sanders. Who do you think will win based on this data?

superTuesday(df, can_list)

print("Sanders mean is", int(df["sandersST"].mean()))

print("Bidens mean is", int(df["bidenST"].mean()))

print("Sanders weighted mean is", int(weightedStatsPerCandidate("Sanders", df)))

print("Bidens weighted mean is", int(weightedStatsPerCandidate("Biden", df)))

Biden and Bloomberg have a correlation of -0.47549215276831625

Sanders and Bloomberg have a correlation of 0.5001912408750656

Comparing Candidates:

19. Means alone are not the best way to compute differences between candidates: we also need to consider uncertainties. Write a function called `getConfidenceIntervals` in `utils.py` that takes in a data column and returns a 95% confidence interval on that column. Print the results of that function and use it to estimate whether or not there is a difference between Biden and Sanders using the raw

polling numbers and the aggregated polling numbers (i.e., those computed with `superTuesday`).

20. Let's more directly compare the two candidates. Write a function called `runTTest` in `utils.py` that runs a t-test between two data columns. In your main file, use that function to compare Biden's numbers and Sander's numbers and then compare their aggregated numbers. Print the results Do your conclusions stay the same as they did with the confidence intervals?

21. Data post-Super Tuesday may provide more insight into our theory about correlation and voters. Let's pull down the newest RCP datafile. **First, rename your current datafile so that it does not get overridden.** Then, repeat the rcp command in Problem 2 to get a new datafile. Remove the RCP Average row and any data that is pre-Super Tuesday (March 3) and zero out any "--" values (it is ok to do this either manually or programmatically). Include this file in your submission.

22. Rerun your analysis on the new data (hint: the candidate names may be different in the new dataset). Do you notice any major differences in the predictions made by the data? Specifically, compare the aggregated predictions (mean, weighted mean, and confidence intervals) from your old data with the raw means, weighted means, and confidence intervals on your new data. Was your Super Tuesday aggregation a good predictor of the real polls? What about the data leads you to this conclusion? What does this say about the correlation theory from Problem 17?