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.
I think Joe Biden will win the democratic presidential nomination. He is a front runner and has already served as the vice president. Many people already support him, especially those who supported Obama. He has a strong healthcare policy which many Americans find important.

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 https://www.realclearpolitics.com/epolls/2020/president/us/2020_democratic_presidential_nomination-6730.html (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.

Done

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.

Normalizing Our Data

1. The data is observing ways that the elections are being analyzed by looking at the different polls, dates and samples.
2. This is being translated to all of the candidates in the democratic party for the 2020 presidential election.
3. The news outlets that are reporting are all reputable sources and big news outlets

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.

4. Create a new python file for your political prediction work. Import yourutils.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.

Done

```
import pandas
import pandas as pd
import matplotlib.pyplot as plt

def loadAndCleanData(filename):
        data = pd.read_csv(filename)
        newData = data.fillna("0")
        print(newData)
        return(newData)


from Utils import *
from Poll import Poll
df= loadAndCleanData("2020nominee.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 athat 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.

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.

```
SyntaxError: invalid character in identifier
rgnt1-10-122-dhcp:problem-set-4 laneablackburn$ python main.py
                              Poll         Date  ...       Spread  Undecided
0            Economist/YouGovYouGov    3/1 – 3/3  ...     Biden +4          5
1         Morning ConsultM. Consult    3/2 – 3/3  ...     Biden +8          3
2          The Hill/HarrisXThe Hill    3/1 – 3/2  ...     Biden +5         13
3         Morning ConsultM. Consult    3/1 – 3/1  ...   Sanders +3          4
4             Reuters/IpsosReuters   2/28 – 3/2  ...  Sanders +11         11
5           Yahoo News/YouGovYouGov  2/26 – 2/27  ...   Sanders +6          3
6         Morning ConsultM. Consult  2/26 – 2/27  ...  Sanders +12         -1
7                   IBD/TIPPIBD/TIPP  2/20 – 2/29  ...   Sanders +3         14
8                   FOX NewsFOX News  2/23 – 2/26  ...  Sanders +13          5
9            Economist/YouGovYouGov  2/23 – 2/25  ...  Sanders +10          5
10         The Hill/HarrisXThe Hill  2/23 – 2/24  ...   Sanders +9          8
11            Reuters/IpsosReuters  2/19 – 2/25  ...  Sanders +12          7
12          CBS News/YouGovCBS News  2/20 – 2/22  ...   Sanders +9          5
13                  EmersonEmerson  2/16 – 2/18  ...   Sanders +7          2
14           Economist/YouGovYouGov  2/16 – 2/18  ...   Sanders +6          8
15         ABC News/Wash PostABC/WP  2/14 – 2/17  ...  Sanders +16          8
16    NBC News/Wall St. JrnlNBC/WSJ  2/14 – 2/17  ...  Sanders +12          7
17            Reuters/IpsosReuters  2/14 – 2/17  ...  Sanders +11         11
18                  SurveyUSASUSA  2/13 – 2/17  ...  Sanders +11          7
19           NPR/PBS/MaristNPR/PBS  2/13 – 2/16  ...  Sanders +12          4
20         The Hill/HarrisXThe Hill  2/14 – 2/15  ...   Sanders +3         11
21  Politico/Morning ConsultPolitico  2/12 – 2/17  ...   Sanders +8          0
22           Economist/YouGovYouGov   2/9 – 2/11  ...   Sanders +4         11
23         The Hill/HarrisXThe Hill   2/7 – 2/10  ...     Biden +3         16
24            Reuters/IpsosReuters   2/7 – 2/10  ...          Tie         17
25                MonmouthMonmouth    2/6 – 2/9  ...  Sanders +10         13
26              QuinnipiacQuinnipiac    2/5 – 2/9  ...   Sanders +8         13
27  Politico/Morning ConsultPolitico    2/4 – 2/9  ...   Sanders +3          7
28           Economist/YouGovYouGov    2/2 – 2/4  ...     Biden +5         10
29            Reuters/IpsosReuters   1/31 – 2/3  ...     Biden +6         16
30  Politico/Morning ConsultPolitico   1/27 – 2/2  ...     Biden +4          6
31              Harvard-HarrisHarris  1/27 – 1/29  ...    Biden +11         12
32    NBC News/Wall St. JrnlNBC/WSJ  1/26 – 1/29  ...   Sanders +1          7
33           Economist/YouGovYouGov  1/26 – 1/28  ...     Biden +2         11
34                  IBD/TIPPIBD/TIPP  1/23 – 1/30  ...     Biden +7         21
35              QuinnipiacQuinnipiac  1/22 – 1/27  ...     Biden +5         14
36  Politico/Morning ConsultPolitico  1/20 – 1/26  ...     Biden +6          8
37                  EmersonEmerson  1/21 – 1/23  ...     Biden +3         11
38             LA Times/USCLA Times  1/15 – 1/28  ...    Biden +16          8
39         ABC News/Wash PostABC/WP  1/20 – 1/23  ...    Biden +12         14
40         The Hill/HarrisXThe Hill  1/20 – 1/22  ...    Biden +12         21
41                  FOX NewsFOX News  1/19 – 1/22  ...     Biden +3         14
42           Economist/YouGovYouGov  1/19 – 1/21  ...     Biden +7         10
43                MonmouthMonmouth  1/16 – 1/20  ...     Biden +7         11
44                         CNNCNN  1/16 – 1/19  ...   Sanders +3         13
45  Politico/Morning ConsultPolitico  1/15 – 1/19  ...     Biden +5          6
46                  SurveyUSASUSA  1/14 – 1/16  ...    Biden +11          8
47         The Hill/HarrisXThe Hill  1/13 – 1/14  ...    Biden +10         23
```

7. Write a function in pollingData.py called plotCandidate that takes in a candidate's name 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 plotCandidate(self,candidate):
      s = self.data[candidate].plot.kde()
      plt.show()

print(poll.plotCandidate("Sanders"))
```

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 avgInPoll(self,candidate):
      return self.data[candidate].mean()

print(poll.avgInPoll("Sanders"))
```

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?

Poll	RCP Average
Date	2/13 - 2/22
Sample	--
Sanders	29.3
Biden	17.2
Bloomberg	15.3
Warren	13.2
Buttigieg	9.8
Klobuchar	6.3
Steyer	2.2
Gabbard	1.4
Spread	Sanders +12.1
undecided	5.3

```
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.). 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 "SampleType" 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.

```
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
```

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

```
cleanSample(demNoms)
```

(file name)

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 divide 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.

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

computePollWeight(demNoms, 'EmersonEmerson')

computePollWeight(demNoms,'CNN,CNN')

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, data):
    weighted = []
    for poll in data["Poll"].unique():
        x = sum(data[data["Poll"] == poll][candidate])
        y = computePollWeight(data, 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?

```
weightedStatsPerCandidate("Sanders", demNoms)
weightedStatsPerCandidate("Biden", demNoms)
weightedStatsPerCandidate("Bloomberg" , demNoms)
weightedStatsPerCandidate("Warren", demNoms)
```

Based on these weighted stats I think that Biden will win.

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. Which candidates are most correlated? Which are least correlated?

```
def computeCorrelation(candidate1, candidate2, data):
    print(candidate1, "and", candidate2, " have a correlation of", data[candidate1].corr(data[candidate2]))
    return data[candidate1].corr(data[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)

```
computeCorrelation("Biden", "Bloomberg", demNoms)
computeCorrelation("Biden", "Warren", demNoms)
computeCorrelation("Biden", "Buttigieg", demNoms)
computeCorrelation("Biden", "Klobuchar", demNoms)
computeCorrelation("Biden", "Steyer", demNoms)
```

```
computeCorrelation("Bloomberg", "Warren", demNoms)
computeCorrelation("Bloomberg", "Buttigieg", demNoms)
computeCorrelation("Bloomberg", "Klobuchar", demNoms)
computeCorrelation("Bloomberg", "Steyer", demNoms)
```

```
computeCorrelation("Warren", "Buttigieg", demNoms)
computeCorrelation("Warren", "Klobuchar", demNoms)
computeCorrelation("Warren", "Steyer", demNoms)
```

```
computeCorrelation("Buttigieg", "Klobuchar", demNoms)
computeCorrelation("Buttigieg", "Steyer", demNoms)
```

```
computeCorrelation("Klobuchar", "Steyer", demNoms)
```

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

```
repeats = []
for candidate1 in candidateList:
    for candidate2 in candidateList:
        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,
demNoms))
            repeats.append([candidate1, candidate2])
```

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(data, candidateList):
    sandersST = []
    bidenST = []

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

        for candidate in candidateList:
            if candidate != "Sanders" and candidate != "Biden":
                bidenCorr = computeCorrelation("Biden", candidate, data)
                sandersCorr = computeCorrelation("Sanders", candidate, data)

                if abs(bidenCorr) > abs(sandersCorr):
                    bidenCount += x[candidate]
                else:
                    sandersCount += x[candidate]
        bidenST.append(bidenCount)
        sandersST.append(sandersCount)
    data["bidenST"] = bidenST
    data["sanersST"] = sandersST
```

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(demNoms, candidateList)
print("Sanders mean is", int(demNoms["sandersST"].mean()))
print("Bidens mean is", int(demNoms["bidenST"].mean()))
# the weighted mean is looking at the significance of all the canidates
print("Sanders weighted mean is", int(weightedStatsPerCandidate("Sanders", demNoms)))
print("Bidens weighted mean is", int(weightedStatsPerCandidate("Biden", demNoms)))
```

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`).

```
Import numpy as np

Import scipy.stats
```

```
def getConfidenceIntervals(column):
    npArray = 1.0 * np.array(column)
    standardError = scipy.stats.sem(npArray)
    n = len(column)
    return standardError * scipy.stats.t.ppf((1+.95)/2.0,n-1)
```

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?