```
In [ ]: # Initialize Otter
        import otter
        grader = otter.Notebook("assignment1.ipynb")
```

1 Assignment 1: 2018 US House Elections

PSTAT 134 (Spring 2020)
Due Date: Friday, April 23, 11:59 PM

1.1 Collaboration Policy

Data science is a collaborative activity. While you may talk with others about the homework, we ask that you **write your solutions individually**. If you do discuss the assignments with others please **include their names** at the top of your notebook.

Collaborators: *list collaborators here*

1.2 Direction and Goal

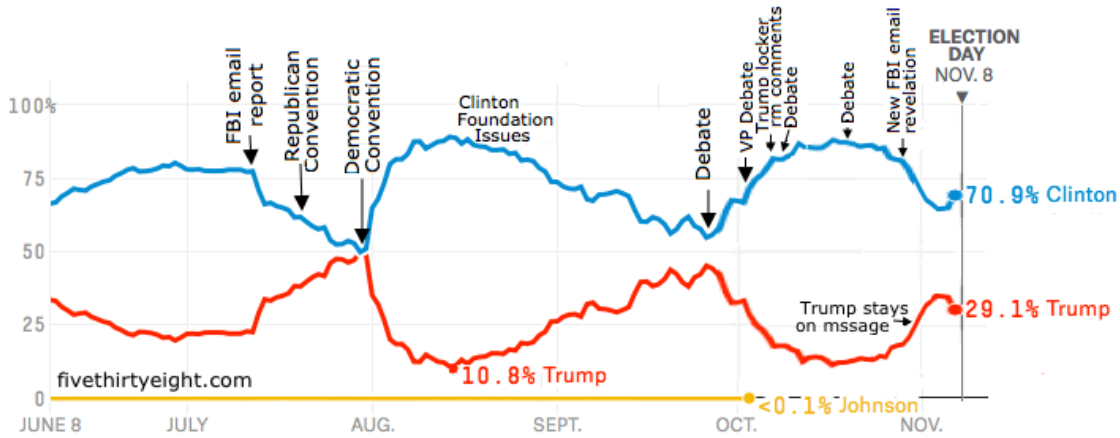


image credit

We haven't talked about predictive models, but we can still think about what makes a "good" prediction. In this assignment, we'll focus on evaluating the quality of election predictions made by the website fivethirtyeight.com. As one prominent example, fivethirtyeight predicted that Clinton had a 70.9% chance to win the election. Was their model wrong?

To gain insight into questions like this, we'll focus on US House elections predictions from 2018. Their

predictions are based predominantly on polling data but include other sources as well (state of the economy, overall favoriability of politic parties, etc).

This homework is based loosely on this article. Please read the article before beginning the assignment.

1.3 Question 1: Data Processing

1.3.1 Download Data

Command line interface is a useful tool for programmatically interacting with general functions of your computer: e.g. manipulate/manage files, download from the internet, run scripts, etc.

We will get acquainted with command line soon, but below uses command line interace commands to download the raw CSV file from fivethirtyeight's github page for this data. The `!` mark tells Jupyter notebook that the command following it is to be evaluated as a command line code.

```
In [1]: !wget -nc https://raw.githubusercontent.com/fivethirtyeight/checking-our-work-data/master/us_hou
```

```
--2021-04-11 23:12:28--  https://raw.githubusercontent.com/fivethirtyeight/checking-our-work-data/master
Resolving raw.githubusercontent.com (raw.githubusercontent.com)… 185.199.108.133, 185.199.110.133, 185.
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 24027649 (23M) [text/plain]
Saving to: 'us_house_elections.csv'

us_house_elections. 100%[===================>]  22.91M  5.09MB/s    in 4.5s

2021-04-11 23:12:33 (5.07 MB/s) - 'us_house_elections.csv' saved [24027649/24027649]
```

```
In [2]: !ls -lF ## `ls` command verify that data is downloaded in our folder
```

```
total 23548
-rw-rw-r-- 1 jovyan jovyan    68345 Oct 16 04:06 assignment1.ipynb
drwxrwxr-x 5 jovyan jovyan     4096 Oct 16 04:06 autograder/
drwxrwxr-x 2 jovyan jovyan     4096 Oct 16 03:53 images/
drwxrwxr-x 5 jovyan jovyan     4096 Oct 16 04:05 student/
-rw-rw-r-- 1 jovyan jovyan 24027649 Oct 15 19:08 us_house_elections.csv
```

1.3.2 Read Data into Python

Numpy and Pandas is used to read in the csv file into python.

```
In [3]: import pandas as pd
        import numpy as np
        election_data = pd.read_csv("us_house_elections.csv", low_memory=False)
```

Add column of zeros named `bin` to `election_data` (we will populate this column with meaningful data later) and `print` the first 10 rows of the `DataFrame` using `iloc`.

```
In [4]: # BEGIN SOLUTION
        election_data['bin'] = 0
        print(election_data.iloc[:10,])
        # END SOLUTION
```

```
   year office state  district special election_date forecast_date  \
0  2018  House    WY       1.0   False    2018-11-06    2018-11-06
1  2018  House    WY       1.0   False    2018-11-06    2018-11-06
2  2018  House    WY       1.0   False    2018-11-06    2018-11-06
3  2018  House    WY       1.0   False    2018-11-06    2018-11-06
4  2018  House    WY       1.0   False    2018-11-06    2018-11-06
5  2018  House    WY       1.0   False    2018-11-06    2018-11-06
6  2018  House    WV       3.0   False    2018-11-06    2018-11-06
7  2018  House    WV       3.0   False    2018-11-06    2018-11-06
8  2018  House    WV       3.0   False    2018-11-06    2018-11-06
9  2018  House    WV       3.0   False    2018-11-06    2018-11-06

  forecast_type party            candidate  projected_voteshare  \
0          lite     D          Greg Hunter             33.29836
1          lite     R           Liz Cheney             61.18835
2        deluxe     D          Greg Hunter             31.37998
3        deluxe     R           Liz Cheney             63.10673
4       classic     D          Greg Hunter             31.33293
5       classic     R           Liz Cheney             63.15379
6          lite     D  Richard Neece Ojeda             47.56779
7          lite     R  Carol Devine Miller             52.43221
8        deluxe     D  Richard Neece Ojeda             45.14111
9        deluxe     R  Carol Devine Miller             54.85889

   actual_voteshare  probwin  probwin_outcome  bin
0               NaN  0.00134               0    0
1               NaN  0.99866               1    0
2               NaN  0.00020               0    0
3               NaN  0.99980               1    0
4               NaN  0.00032               0    0
5               NaN  0.99968               1    0
```

```
6               NaN  0.27966               0    0
7               NaN  0.72034               1    0
8               NaN  0.09662               0    0
9               NaN  0.90338               1    0
```

Fivethirtyeight has three different prediction models: `lite`, `classic` and `deluxe`, which roughly incorporate an increasing number of assumptions. In this assignment lets focus on evaluting the quality of the `classic` predictions. You can read more about how the prediction models work here.

Fivethirtyeight continuously updated their predictions as more polling data became available for each of the races. Let's focus on the predictions a few months before the election, on August 11th, and on the morning of election day, November 6th.

1.3.3 Question 1a: Subset Data

Create a new pandas dataframe called `election_sub` by filtering to include only rows in which the `forecast_type` is "classic", and the date of the foreceast (`forecast_date`) is 8/11 or 11/6.

Using `query` method seems well-suited. Note you can make two (or more) calls to `query` by chaining calls to `query` like this: `election_data.query(...).query(...)`. Output of one query will be used as an input to the second query.

```
In [5]: election_sub = election_data.query("forecast_type == 'classic'").query('forecast_date =="2018-1
        """ # BEGIN PROMPT
        # Fill-in ...
        election_sub = election_data.query("forecast_type ...").query('forecast_date ... | forecast_dat
        """; # END PROMPT
```

```
In [ ]: grader.check("q1a")
```

1.3.4 Question 1b: Filtering Data

In previous question, data was subset for two forcast dates: 2018-11-06 and 2018-08-11. Presumably, there *should be* two rows (predictions) for each candidate; however, you will see that some candidates are missing one of the two predictions and not all name entries are valid.

Using Pandas, remove any NaN names and any candidate that does not have two predictions.

Finally, overwrite `election_sub` with the filtered data.

There are different ways of doing this. I found the following functions useful:

- pandas.DataFrame.isnull
- pandas.DataFrame.groupby
- pandas.core.groupby.DataFrameGroupBy.filter
- pandas.DataFrame.shape

When using the documentation, make sure to use the correct version. You can check by running `pd.__version__`.

```
In [8]: election_sub = election_sub[~election_sub['candidate'].isnull()].groupby('candidate').filter(la
        """ # BEGIN PROMPT
        # Fill-in ... and ###some task###
        election_sub = election_sub[  ###filter-out NaN names###  ].groupby(...).filter( ###lambda func
        """; # END PROMPT
```

```
In [ ]: grader.check("q1b")
```

1.3.5 Question 1c: Transform Data

We want to check whether events predicted by 538 to occur with probability *close to* X% actually occured about X% of the time. To do this, we have to define *close*.

First, we'll define the `cut_points` as 20 equally spaced numbers between 0 and 1 using `np.linspace`. Then we'll group the predicted probabilities into the `19` equally spaced bins determined by those cut points. Define the bin for each observation using the `pd.cut` function on the `probwin` variable. Then, assign the result to column `bin` of `election_sub`. Use `include_lowest=True` when calling `pd.cut`.

Note: Can you spot the strange behavior of `include_lowest=True`*? Despite the output,* `pd.cut` *seems to work correctly*

```
In [12]: cut_points = np.linspace(0, 1, 20) # SOLUTION
         election_sub['bin'] = pd.cut(election_sub['probwin'], cut_points, include_lowest=True) # SOLUT
```

```
In [ ]: grader.check("q1c")
```

1.4 Question 2: Looking for Insights

1.4.1 Question 2a: Calculate Change in Support

Let's see if we can find the candidates whose standings change the most between August 11 and November 6: one with largest improvement and another with largest decrease in win-probability. First, use the `agg` function calculate the difference.

Following functions have been useful for me:

- `numpy.diff`
- `pandas.DataFrame.sort_values`
- `pandas.DataFrame.groupby`
- `pandas.DataFrame.agg`: especially, different functions to columns

Save the resulting DataFrame from `agg()` to a variable, `probwin_change`.

```
In [15]: probwin_change = election_sub.sort_values(by=['forecast_date','candidate']).groupby('candidate
         """ # BEGIN PROMPT
         # Fill-in ...
         probwin_change = election_sub.sort_values(by=[...]).groupby(...).agg({... : ...})
         """; # END PROMPT
```

```
In [ ]: grader.check("q2a")
```

1.4.2 Question 2b: Looking for Largest Changes

Now, save the name of the candidates to string variables `rising_candidate` (largest increase) and `falling_candidate` (largest decrease).

- `pandas.DataFrmae.idxmax`
- `pandas.DataFrmae.idxmin`

```
In [20]: rising_candidate = probwin_change.idxmax().values[0]  # SOLUTION
         falling_candidate = probwin_change.idxmin().values[0] # SOLUTION
```

```
In [ ]: grader.check("q2b")
```

1.4.3 Question 2c: Verify Outcome

Did the candidate win or lose the election? Verify with election outcome.

Type your answer here, replacing this text.

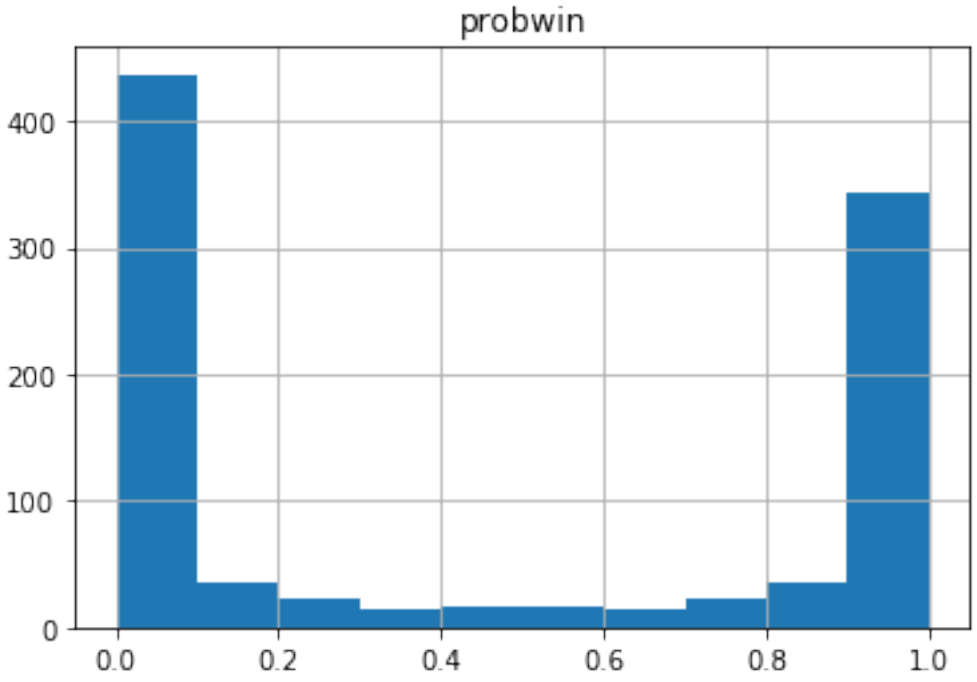
SOLUTION

1.5 Prediction vs Actual Outcomes

1.5.1 Question 3a: Prediction Histogram

Make a histogram showing the predicted win probabilities *on the morning of the election.* Again, restrict yourself to only the `classic` predictions.

```
In [24]: # BEGIN SOLUTION
         election_sub.query("forecast_date=='2018-11-06'").hist('probwin');
         # END SOLUTION
```



1.5.2 Question 3b: Prediction difficulty

Are most house elections easy to forecast or hard to forecast? State your reasoning.

Type your answer here, replacing this text.

SOLUTION

1.5.3 Question 4a: Compute Actual Outcomes

Now we've grouped the observations into a discrete set of bins according to the predicted probability, `probwin`. Within each bin, we now want to compute the actual fraction of times the candidates won.

If 538 did a good job, it will be close to the predited probabilities. You'll need to use the `groupby` function to compute the mean of `probwin_outcome` (1 is a win and 0 is a loss) within each bin. Once again you can use `agg` method here.

Save the fraction of actual wins in each bin in a list called `fraction_outcome`.

```
In [25]: fraction_outcome = election_sub.groupby('bin').agg({'probwin_outcome':np.mean}) #SOLUTION
```

```
In [ ]: grader.check("q4a")
```

1.5.4 Question 4b: Preparing to Present Results

For this problem we'll make a plot of the predicted probabilities and actual fraction of wins in each bin. We've already computed the actual fraction of wins; all that remains is to plot it against the predicted value assocaited with each bin.

For the predicted value in each bin, using the midpoint of the bin would make sense. Compute the midpoints of each bin from `cut_points`.

```
In [30]: midpoints = (cut_points[1:] + cut_points[:-1]) / 2 # SOLUTION
```

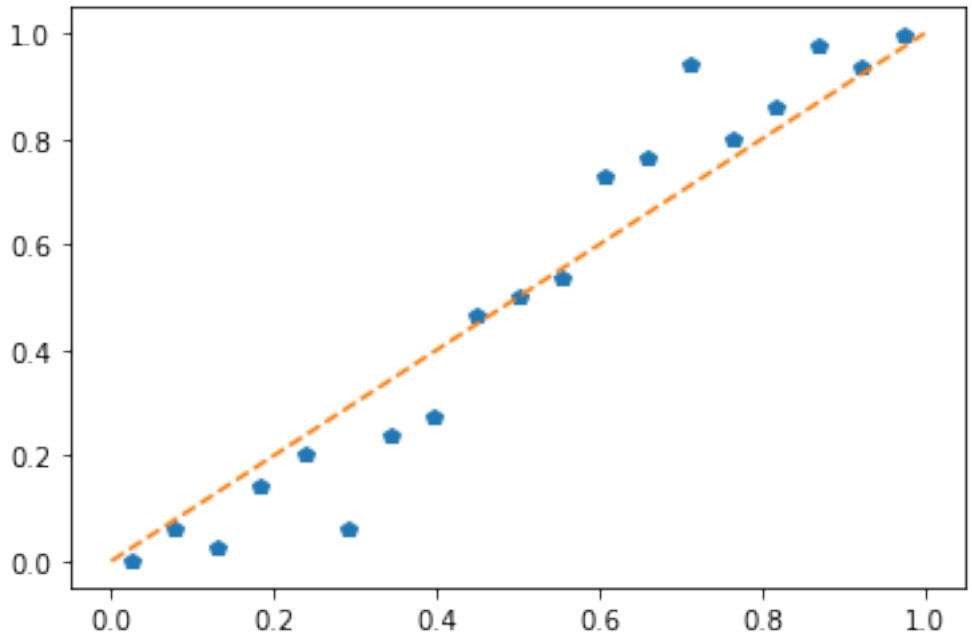
```
In [ ]: grader.check("q4b")
```

1.5.5 Question 4c: Visualize Results

Now make a scatterplot using `midpoints` as the x variable and `fraction_outcome` as the y variable. Draw a dashed line from `[0,0]` to `[1,1]` to mark the line y=x.

```
In [33]: # magic for showing figures inline
         %matplotlib inline
         import matplotlib.pyplot as plt

         # BEGIN SOLUTION
         plt.plot(midpoints, fraction_outcome, 'p')
         plt.plot([0, 1], [0, 1], '--');
         # END SOLUTION
```



1.6 Quantifying Uncertainty

1.6.1 Question 5a: Model-based Error Estimation

If you did things correctly, it should look like fivethirtyeight has done "pretty" well with their forecasts: the actual fraction of wins tracks closely with the predicted number.

But how do we decide what's "good enough"? Consider this example: I correctly predict that a coin is fair (e.g. that it has a 50% chance of heads, 50% chance of tails). But if I flip it 100 times, I can be pretty sure it won't come up heads exactly 50 times. The fact that heads didn't come up exactly 50 times doesn't make my prediction incorrect.

To assess how reasonable the predictions are, I need to quantify the uncertainty in my estimate. It's reasonable to assume that within each bin, k, the observed number of wins, $Y_k \sim \text{Binomial}(n_k, p_k)$, where n_k is the number of elections and p_k is the predicted win probability in bin k.

Classical results tell us that the obseved fraction of wins in bin k, $\hat{p} = \frac{Y_k}{n_k}$ has variance $\text{Var}(\hat{p}_k) = \frac{p_k(1-p_k)}{n_k} \approx \frac{\hat{p}_k(1-\hat{p}_k)}{n_k}$. The standard deviation of the Binomial proportion then is $\hat{\sigma}_k \approx \sqrt{\frac{\hat{p}_k(1-\hat{p}_k)}{n_k}}$.

If we use the normal approximation to generate a confidence interval, then the 95% interval has the form $\hat{p}_k \pm 1.96\hat{\sigma}_k$.

Create a new "aggregated" dataframe named `election_agg`. Take `election_sub`, group by `bin` and compute both the average of the `probwin_outcome` (`mean`) and the number of observations in each bin (`count`) using the `agg` function. Call this new data frame, `election_agg`.

THen, use the `mean` and `count` columns of `election_agg` to create a new column of `election_agg` titled `err`, which stores $1.96 \times \hat{\sigma}_k$ in each bin k.

```
In [34]: election_agg = election_sub.groupby('bin')['probwin_outcome'].agg(['mean', 'count']) # SOLUTIO
         election_agg['err'] = 1.96 * np.sqrt(election_agg['mean'] * (1 - election_agg['mean']) / elect
```

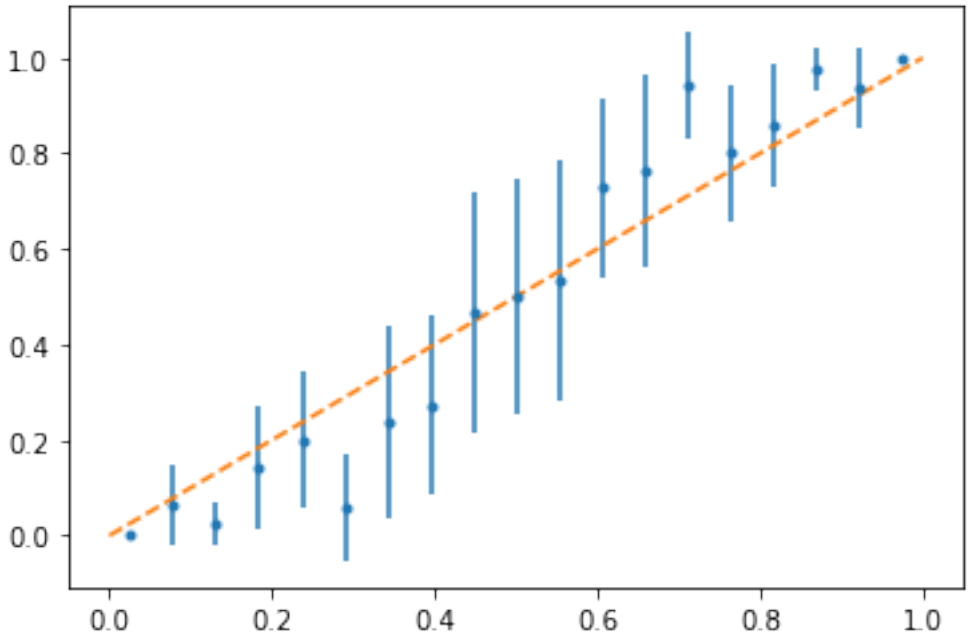
```
In [ ]: grader.check("q5a")
```

1.6.2 Question 5b: Visualize Error Bars 1

Use `plt.errorbar` to create a new plot with error bars associated with the actual fraction of wins in each bin. Again add a dashed y=x line. Set the argument `fmt='.'` to create a scatterplot with errorbars.

```
In [38]: # Plotting code below

         # BEGIN SOLUTION
         plt.errorbar(midpoints, election_agg['mean'].values, yerr=election_agg['err'].values, fmt='.')
         plt.plot([0, 1], [0, 1], '--');
         # END SOLUTION
```



1.6.3 Question 5c: Computing Coverage

If our intervals were true 95% confidence intervals, then we would expect about 95% of them to cover the midpoint of the bin (i.e. overlap with the y=x line).

What fraction of the 95% confidence intervals cover the bin midpoint? Create a variable, `upper`, to be the `mean + err` and another, `lower`, to be `mean - err` (both `upper` and `lower` should pandas series). Next, compute `frac_covering` as the fraction of midpoints between `lower` and `upper`.

```
In [39]: upper = election_agg['mean'] + election_agg['err'] # SOLUTION
         lower = election_agg['mean'] - election_agg['err'] # SOLUTION

         frac_covering = ((upper > midpoints) & (lower < midpoints)).mean() # SOLUTION
```

```
In [ ]: grader.check("q5c")
```

1.6.4 Question 5d: Understanding Confidence Intervals

Are the 95% confidence intervals generally larger or smaller for more confident predictions (e.g. the predictions closer to 0 or 1). What are the factors that determine the length of the confidence intervals?

Type your answer here, replacing this text.

SOLUTION

Intentionally Blank

To double-check your work, the cell below will rerun all of the autograder tests.

```
In [ ]: grader.check_all()
```

1.7 Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit. **Please save before exporting!**

```
In [ ]: # Save your notebook first, then run this cell to export your submission.
        grader.export()
```