2022 MIDTERM ELECTION PREDICTION PROJECT

ISYE 7406 – Data Mining & Statistical Learning

Miguel Asse (GT ID: 903755722; email: XYZ)
Martin Huerta (GT ID: 903657152; email: mhuerta9@gatech.edu)
November 27th, 2022
Group #94

Abstract

Using data to predict political outcomes has been used for decades with the modern political polling apparatus that we know today being invented in the 1940s, and gained popularity in 2012 with the rise of FiveThirtyEight. Political predictions using data can be done with a variety of statistical methods and data sources.

As part of our project, we will be taking several data sets and combining them to predict the 2022 midterm state senate political party winners. The prediction will be treated as a classification problem, and will aim to answer an important question: based on a series of factors, **is a candidate predicted to win their race**?

We will be using the following statistical learning methods:

- Logistic Regression
- Decision Trees
- Random Forest
- KNN
- KNN with a Grid Search for optimal parameter tuning
- Boosting, specifically the AdaBoostClassifier from scikitlearn

Models will be evaluated based on the mean error rate of predictions and the F1 score. The definitions are as follows:

- **Mean Error Rate** is the average error rate of misclassifications between the predictions and actual values.
- **F1 Score** is the harmonic mean between precision and recall. It attempts to minimize false positives and false negatives. The final score ranges from 0 to 1, and the higher (closer to 1) is, the better.

Ultimately based on the above criteria, **Decision Trees** were considered the best model, given it had the 2nd lowest mean error rate and highest F1 score.

Introduction

For our project, we will be taking datasets from across the U.S. Census Bureau, the MIT Election & Data Science Lab, and several others to see if socio-economic factors can be used to predict if a senate candidate will win their senate race. This was made famous in the US by FiveThirtyEight correctly using polling data to predict all 50 states and the District of Columbia (D.C.) during Obama's 2012 presidential campaign.

We were curious if we could use various statistical learning methods, combining with political and socio-economic data, to make predictions about whether a senate candidate would win. We also had a unique ability to test our results against the 2022 Midterm Election Results given they were wrapping up during the course project timeline.

Our process followed several structured steps:

1. **Data Preparation / Cleaning -** We had to gather our data across all our various sources, clean it, understand how we could merge it together and create new features based off of what the data contained
2. **Model Building & Tuning -** After our data was clean, we wanted to structure our training data into training and test sets, and tune our models optimally against our testing data
3. **Evaluation -** Afterwards, we evaluated our results against FiveThirtyEight's predictions, as well as the actual 2022 midterm election results.

Data Mining Challenges mainly including the data preparation work, including:

1. **Account for multiple rows of the same candidate** - A candidate can have multiple rows in the same state and year if they were endorsed by multiple political parties

2. **Repeat Candidates** - A candidate for senate could be nominated across multiple years

3. **Lack of additional features** - No additional socio-economic data was included outside of the senators, the state, their party details, and the total votes they received

Beyond the challenges, this project provided interesting lessons for our group that we can take with us as we apply statistical learning to other problems. In particular, two important general takeaways were:

1. **When in doubt, go granular** - One sacrifice our team had to make in the interest of time for our project was using national socio-economic indicators vs state specific indicators like national unemployment versus state-specific unemployment. With more time and resources, we would prefer to go more granular and have state specific indicators (i.e. population breakdowns, education levels) to try and see if the level of granularity can have an impact on predictive power).

2. **Optimize your models** - We found one of the strongest performing models by using GridSearch & iterating through 38,220 combinations of KNN models. Ultimately this showed us the power of GridSearch (and by extension, RandomSearch) and its ability to find an optimal model in a way no human could.

Problem Statement or Data Sources

Several datasets were used for the project, including:

1. Historical Senate results - MIT Election Data and Science Lab

2. Unemployment Statistics 1951-2021 from the Current Population Survey - US Bureau of Labor Statistics

3. [Unemployment Statistics for 2022](#) - U.S. Bureau of Labor Statistics

4. [Historical Educational Achievement from 1940 - Present](#) - US Census Bureau

5. [2022 Senate Candidates by State and Party](#) - Ballotpedia

6. [FiveThirtyEight's Senate Forecast](#) - FiveThirtyEight

7. [Historical data for all races and for Hispanic origin (1610–2020)](#) - Wikipedia

Below is an example of what our main dataset from MIT looked like:

	year	state	state_po	state_fips	state_cen	state_ic	office	district	stage	special	candidate	party_detailed	writein	mode	candidatevotes	total
0	1976	ARIZONA	AZ	4	86	61	US SENATE	statewide	gen	False	SAM STEIGER	REPUBLICAN	False	total	321236	74
1	1976	ARIZONA	AZ	4	86	61	US SENATE	statewide	gen	False	WM. MATHEWS FEIGHAN	INDEPENDENT	False	total	1565	74
2	1976	ARIZONA	AZ	4	86	61	US SENATE	statewide	gen	False	DENNIS DECONCINI	DEMOCRAT	False	total	400334	74
3	1976	ARIZONA	AZ	4	86	61	US SENATE	statewide	gen	False	ALLAN NORWITZ	LIBERTARIAN	False	total	7310	74
4	1976	ARIZONA	AZ	4	86	61	US SENATE	statewide	gen	False	BOB FIELD	INDEPENDENT	False	total	10765	74

Proposed Methodology

Our methodological approach was as follows:

1. **Train, Test & Split** - We split our data into 80% & 20% training and test sets
    a. For KNN, we standardized the features due to KNN being sensitive to variable scale

2. **Model Implementation** - We implemented a variety of models to determine winners, including Logistic Regression, Decision Trees, Random Forests, K-Nearest Neighbors (KNN), KNN with GridSearch, Random Forest, and the AdaBoost Classifier

3. **Compare Results** - After model implementation and running the individual models across training and test sets, we compared the results across the various classifiers using the Mean Error Rate and F1 Score.

4. **Evaluate against FiveThirtyEight & Actual 2022 Senate Results** - We evaluated our predictions against FiveThirtyEight's final predictions and the actual final senate race outcomes

We chose the above statistical learning methods in order to use a wide breadth of different models that had varying degrees of flexibility based on whether the data seemed to be more parametric or non-parametric, and to see if any model in particular had superior performance, so we could understand if that indicated a trend in our dataset we could exploit for more accurate model predictions.

Analysis and Results

To evaluate the models, we chose to use a train-test split and measure the performance of each model based on the Mean Error Rate and F1 Score. In summary, KNN had the lowest Mean Error Rate but all the models had a similar result. Interestingly, KNN also had one of the lowest F1 Scores. When compared to regular KNN, the optimized GridSearch KNN had a higher F1 Score. Finally, Decision Trees had the second lowest Mean Error Rate and highest F1 Score. For this reason, we chose Decision Trees as our best performing model. These results are shown on the table below:

Table 1. Model Performance Summary

Model	Mean Error Rate	F1 Score
Logistic Regression	0.2602	0.6247
Decision Trees	0.2267	0.6303
Random Forest	0.2282	0.6123
KNN	0.2122	0.3364
KNN - Grid Search	0.2282	0.4852
Boosting	0.2471	0.2478

For our next set of results, we shifted our focus to how our top performing model would fare against actual 2022 election results and FiveThirtyEight, as well as comparing FiveThirtyEight again 2022 election results.

Our best model, Decision Trees, had a Mean Error Rate of 0.22 and an F1 Score of 0.42 against actual winners of the race. This was in line with our test error rate of 0.22, and based on the classification matrix as shown below.

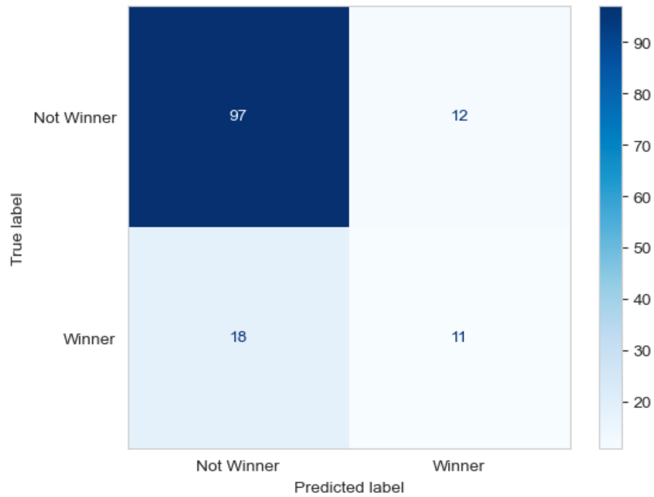

Figure 2. Classification Matrix for Decision Trees Model Results vs 2022 Race Winners

We derived the following metrics for the analysis using the classification matrix:

- **Precision -** (TP / TP + FP) = (11 / (11 + 12)) = 0.48
- **Sensitivity -** (TP / TP + FN) = (11 / (11 + 18)) = 0.38
- **Specificity -** (TN / TN + FP) = (97 / (97 + 12)) = 0.89

This show that our model performs well when predicting who will not win, but had mediocre results otherwise.

We see similar results against FiveThirtyEight's predictions, but with a slightly lower sensitivity rate.

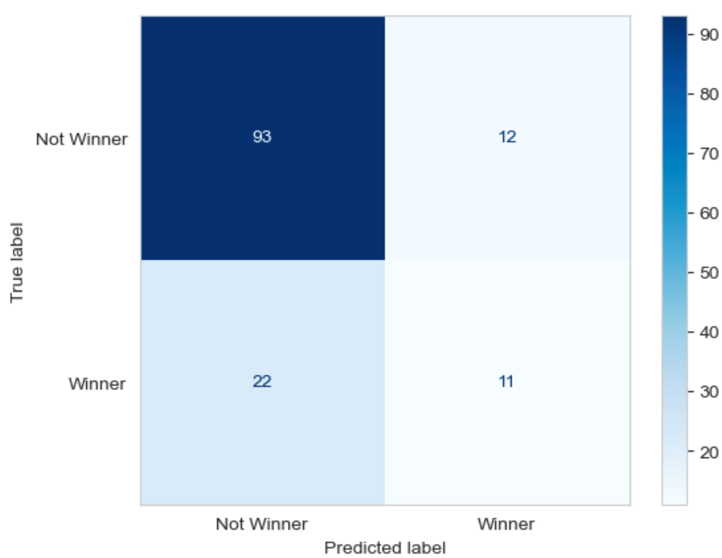

Figure 3. Classification Matrix for Decision Trees Model Results vs FiveThirtyEight's Predictions

- **Precision - (TP / TP + FP) = (11 / (11 + 12)) = 0.48**
- **Sensitivity - (TP / TP + FN) = (11 / (11 + 22)) = 0.33**

- **Specificity - (TN / TN + FP) = (93 / 93 + 12) = 0.89**

Finally, we wanted to see how FiveThirtyEight's results compare to actual results. This would help us gauge the performance of our model against an established elections prediction entity using advance statistical methods. Ultimately, FiveThirtyEight itself had strong results against the actual winners, and was much more precise.

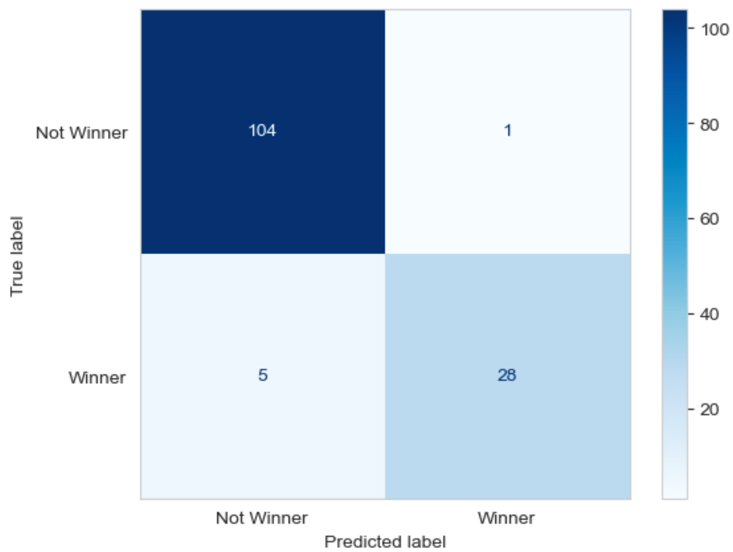

Figure 4. Classification Matrix for FiveThirtyEight's Predictions vs 2022 Race Winners

- **Precision - (TP / TP + FP) = (28 / (28 + 1) = 0.97**
- **Sensitivity - (TP / TP + FN) = (28 / (28 + 5)) = 0.85**
- **Specificity - (TN / TN + FP) = (104 / 104 + 1) = 0.99**

Conclusions

Our main conclusion is that predicting political winners is challenging. Our models were able to do relatively well predicting who would not win (i.e. high specificity), however, it was more challenging predicting the winners, with a 0.50 precision rate. FiveThirtyEight's model using polling was more accurate.

Furthermore, we realized national indicators are not very predictive. While we tried to gather comprehensive socio-economic data, we think the features were not granular enough to help our models predict the correct outcome. Further analysis could include state or county level data if available.

Finally, The model picked isn't everything. We tried several iterations of models across our dataset, and even did an optimized GridSearch for KNN, despite the variety of models we attempted against the problem, all models had error rates ranging from 0.21 - 0.26. We ultimately think Decision Trees were the best model due to having the 2nd lowest mean error rate and highest F1 score, but no model stood out significantly.

Lessons We Have Learned

1. When in doubt, go granular - One sacrifice our team had to make in the interest of time for our project was using national socio-economic indicators vs state specific indicators. With more time and resources, we would prefer to go more granular and have state specific indicators (i.e. population breakdowns, education levels) to try and see if the level of granularity can have an impact on predictive power).

2. Socioeconomic data may not be predictive of voting outcomes - FiveThirtyEight specifically uses a variety of polling information which obviously directly relates to voting and is likely more predictive than socioeconomic factors. It may be hard to tell the way a person will vote based on generalized socio-economic data, due to the uniqueness of individuals.

3. Optimize the model - We found one of the strongest performing model by using GridSearch & iterating through 38,220 combinations of KNN models. Ultimately this showed us the power of GridSearch and its ability to find an optimal model in a way no human could.

Bibliography and Credits

Mei, Yajun. “Module 1” ISYE 7406: Data Mining and Statistical Learning.

Mei, Yajun. “Module 5” ISYE 7406: Data Mining and Statistical Learning.

Dowsett, Ben, et al. *FiveThirtyEight*, 22 Nov. 2022, https://fivethirtyeight.com/.

Appendix

ISYE 7406 2022 Midterm Election Prediction Project

Authors: Miguel Asse & Martin Huerta

This dataset will be looking at several sources of data to predict the 2022 Midterm Elections. There will be several sources of data cleaned and used:

Introduction

1. **Midterm Results** - Historical Midterm results sourced from the Open Intro Midterms Dataset
2. **Senate Results 1976-2020** - Historical Senate results sourced from MIT Election Data and Science Lab
3. **Unemployment Statistics 1951-2021 from the Current Population Survey** - Sourced from the U.S. Bureau of Labor Statistics
4. **Unemployment Statistics for 2022** - Sourced from the U.S. Bureau of Labor Statistics
5. **Historical Educational Achievement from 1940 - Present** - Sourced from United States Census Bureau
6. **2022 Senate Candidates by State and Party** - Sourced from Ballotpedia with an additional column indicating FiveThirtyEight's predicted winner's from the FiveThirtyEight Senate Forecast
7. **US Census Bureau's Region and Sub-Region breakdown of the States in the United States** - Sourced from the US Census Bureau website here
8. **Historical data for all races and for Hispanic origin (1610–2020) sourced from Wikipedia** here)

Data will be cleaned and the years from 1976 - 2020 will be used to predict the 2022 Midterm Election outcomes using a variety of classification methodds.

In [ ]:
```python
# Import necessary packages

import pandas as pd
import pandas_profiling
from pandas_profiling import ProfileReport
import chart_studio
from collections import defaultdict
import numpy as np
import altair as alt
import plotly.express as px
import plotly.graph_objects as go
import chart_studio.plotly as py
```

```
from plotly.subplots import make_subplots
import pprint
import re
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from statistics import mean
from sklearn.metrics import f1_score

%matplotlib inline
import chart_studio
import chart_studio.plotly as py

# Set Parameters for plot and pandas outputs
sns.set_style('whitegrid')
plt.rcParams['figure.figsize'] = [10, 5]
pd.set_option('display.max_columns', 30)
```

In [ ]:
```
from IPython.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

In [ ]:
```
# Read in datasets (note this assumes they are in the same folder as the Jupyter Noteb

midterms_data = pd.read_csv("midterms_raw.csv")
senate_data = pd.read_csv("senate_data_1976_2020_raw.csv", encoding='utf-8')
unemployment_historical = pd.read_csv("historical_unemployment_averages.csv")
current_unemployment = pd.read_csv("current_year_2022_unemployment_numbers.csv")
senate_candidates_2022_test_set = pd.read_csv("senate_candidates_2022_midterms_test_se
us_presidents_by_year = pd.read_csv('us_president_by_year.csv')
state_regional_breakdown = pd.read_csv("census_state_regional_breakdown.csv")
census_demographic_breakdown = pd.read_csv("census_demographic_breakdown_1970_to_today
education_achievement_by_year = pd.read_csv("proportion_of_educational_achievement_by_
plotly_api_key = pd.read_csv('plotly_api_key.csv')

# Create 2022 winners due to results being confirmed
senate_winners_2022 = senate_candidates_2022_test_set.copy()
senate_winners_2022['is_actual_winner'] = senate_winners_2022['is_actual_winner'].asty
senate_candidates_2022_test_set.drop('is_actual_winner', axis='columns', inplace=True)
```

In [ ]:
```
senate_data.head()
```

In [ ]:
```
senate_data.info()
```

In [ ]:
```
senate_data.isnull().sum()
```

Data Cleaning

Below, we'll do several iterations of data cleaning and merging to create our training and testing data

In [ ]:
```
# Finalize test Set

# Add regional data to the test set
```

```
senate_candidates_2022_test_set = senate_candidates_2022_test_set.merge(state_regional

senate_candidates_2022_test_set.drop(['sub_region'], axis='columns', inplace=True)

senate_candidates_2022_test_set = pd.get_dummies(senate_candidates_2022_test_set, pref

# Add educational data to the dataset. Since 2022 data does not currently exist from t
# we will use 2021 as a proxy.

education_achievement_2022_proxy = education_achievement_by_year.copy()
education_achievement_2022_proxy = education_achievement_2022_proxy[education_achieven
education_achievement_2022_proxy['year'] = 2022

senate_candidates_2022_test_set = senate_candidates_2022_test_set.merge(education_achi

senate_candidates_2022_test_set.drop(['total'], axis='columns', inplace=True)

senate_candidates_2022_test_set.head()
```

In [ ]:

```
# Remove unnecessary data from various dataframes

# Reduce Historical data to years on or after 1976
midterms_data_clean = midterms_data[midterms_data['year']>=1976]

# Only include general election results for senate data
senate_data_clean = senate_data[senate_data['stage']=='gen']
```

In [ ]:

```
senate_data_clean['party_simplified'].unique()
```

In [ ]:

```
# Clean up senate data:

# 1. Create percentage_of_vote column and senate_race_key based on year & state
senate_data['percentage_of_vote'] = senate_data['candidatevotes'] / senate_data['total

# Fill in missing values
senate_data['candidate'].fillna('No Candidate Listed', inplace=True)
senate_data['party_detailed'].fillna('No Party Detail Available', inplace=True)

senate_data['senate_race_key'] = senate_data['year'].astype(str) + senate_data['state'

senate_data_clean = senate_data.copy()
```

In [ ]:

```
# Remove duplicate winners due to Senate data being by unique by party_detailed column

cols = ['senate_race_key', 'year', 'state', 'state_po', 'candidate', 'candidatevotes',

senate_data_clean_limited = senate_data_clean[cols]
senate_dict = senate_data_clean_limited.to_dict('records')

final_dict = defaultdict(dict)

for item in senate_dict:
    key = item['senate_race_key']
    year = item['year']
    state = item['state']
    state_po = item['state_po']
    candidate = item['candidate']
    votes_won = item['candidatevotes']
```

```
    total_votes = item['totalvotes']
    party_detailed = item['party_detailed']
    party_simplified = item['party_simplified']
    if key in final_dict:
        final_dict[key]['year'] = year
        final_dict[key]['state'] = state
        final_dict[key]['candidate'] = candidate
        final_dict[key]['votes_won'] += votes_won
        final_dict[key]['total_votes'] = total_votes
        final_dict[key]['party_detailed'] += "|" + party_detailed
        final_dict[key]['total_votes'] = total_votes
        if final_dict[key]['party_simplified'] != party_simplified and final_dict[key]
            final_dict[key]['party_simplified'] += "|" + party_simplified
        else: pass

    else:
        final_dict[key]['year'] = year
        final_dict[key]['state'] = state
        final_dict[key]['state_po'] = state_po
        final_dict[key]['candidate'] = candidate
        final_dict[key]['votes_won'] = votes_won
        final_dict[key]['total_votes'] = total_votes
        final_dict[key]['party_detailed'] = party_detailed
        final_dict[key]['party_simplified'] = party_simplified
        final_dict[key]['total_votes'] = total_votes


# Create final senate dataframe

senate_data_final = pd.DataFrame.from_dict(data=final_dict, orient="index").reset_inde
senate_data_final.rename(mapper={"index":"senate_race_key"},axis="columns", inplace=Tr
senate_data_final["percentage_of_vote"] = senate_data_final["votes_won"] / senate_data
```

In [ ]:
```
# Determine winners of each race for each year

winners = senate_data_final.loc[senate_data_final.groupby(by=['year','state'])['percen
columns = ['senate_race_key', 'candidate']
winners_final = winners[columns]
winners_final

# Add winner outcome column to senate_data_final

senate_data_merged = senate_data_final.merge(winners_final, on='senate_race_key', how=
senate_data_final['winner'] = senate_data_final['candidate'] == senate_data_merged['ca
senate_data_final['number_of_supporting_parties'] = senate_data_final['party_simplifie
```

In [ ]:
```
## Add is_candidate_democrat, is_candidate_republican, is_candidate_other fields to se

senate_data_final["is_candidate_democrat"] = senate_data_final['party_simplified'].str
senate_data_final["is_candidate_republican"] = senate_data_final['party_simplified'].s
senate_data_final["is_candidate_other"] = (senate_data_final['party_simplified'].str.c



senate_data_final.head()
```

In [ ]:
```
senate_data_final.merge(midterms_data, how="left", on="year")

senate_data_final.head()
```

In [ ]:
```
# Merge Unemployment data & Senate data

unemployment_by_year = unemployment_historical[['year', 'total_unemployed_percentage_o

senate_and_unemployment = senate_data_final.merge(unemployment_by_year, how="inner", o

# Change unemployemnt rate to a decimal based percentage

senate_and_unemployment['total_unemployed_percentage_of_labor_force'] = senate_and_une

# Add presidental party and boolean columns

senate_unemployment_and_president = senate_and_unemployment.merge(us_presidents_by_yea

senate_unemployment_and_president["is_current_president_democrat"] = senate_unemployme
senate_unemployment_and_president["is_current_president_republican"] = senate_unemploy

# Merge in state level region and sub region data

senate_unemployment_and_president = senate_unemployment_and_president.merge(state_regi
```

In [ ]:
```
# Create dummy variables for region and drop sub_region due to collinearity issues

senate_unemployment_and_president = pd.get_dummies(senate_unemployment_and_president,

senate_unemployment_and_president.drop(['sub_region'],axis="columns", inplace=True)

senate_unemployment_and_president.head()
```

In [ ]:
```
# Bring in white population and non-white population statistics by year into dataset

white_vs_non_white_census = census_demographic_breakdown[['year', 'white_population_pr

senate_unemployment_and_president = senate_unemployment_and_president.merge(white_vs_n

senate_unemployment_and_president.head()
```

In [ ]:
```
# Add educational achievement data by year for adults aged 25+

senate_unemployment_and_president = senate_unemployment_and_president.merge(education_

senate_unemployment_and_president.drop('total', axis="columns", inplace=True)

senate_unemployment_and_president.head()
```

In [ ]:
```
# Create training dataset

training_data = senate_unemployment_and_president.copy()

training_data['is_candidate_democrat'] = training_data['is_candidate_democrat'].apply(
training_data['is_candidate_republican'] = training_data['is_candidate_republican'].ap
training_data['is_candidate_other'] = training_data['is_candidate_other'].apply(lambda
training_data['is_current_president_democrat'] = training_data['is_current_president_d
training_data['is_current_president_republican'] = training_data['is_current_president

training_data['IsCandidateWinner'] = np.where(training_data['winner']==True, 1, 0)

# Drop unnecessary values from training dataset
```

```
training_data.drop(['winner', 'senate_race_key', 'state_po', 'year', 'state', 'candida
                    'votes_won', 'total_votes', 'party_detailed', 'party_simplified',
                    'president_name', 'presidential_party'], axis=1, inplace=True)

training_data.head()
```

In [ ]:
```
# Add white population and non-white population statistics by year into testing datase

candidates_2022 = senate_candidates_2022_test_set['candidate']
state_2022 = senate_candidates_2022_test_set['state']
predicted_winner_2022 = senate_candidates_2022_test_set['is_predicted_winner_538']

senate_candidates_2022_test_set = senate_candidates_2022_test_set.merge(white_vs_non_w

senate_candidates_2022_test_set.drop(['party_simplified',
 'year',
 'party_detailed',
 'candidate',
 'is_predicted_winner_538',
 'state'], axis=1, inplace=True)

X = training_data.drop('IsCandidateWinner', axis = 1)
train_cols = X.columns
senate_candidates_2022_test_set = senate_candidates_2022_test_set[train_cols]
senate_candidates_2022_test_set.head()
```

Exploratory Data Analysis

As part of the exploratory data analysis of the project we're going to:

1. Visualize the Senate Map visually usig Plotly Expres

In [ ]:
```
training_data.info()
```

In [ ]:
```
training_data.describe()
```

In [ ]:
```
training_data.isnull().sum()
```

In [ ]:
```
us_senate_winners_map = px.choropleth(
    data_frame=winners,
    locations='state_po',
    locationmode='USA-states',
    color='party_simplified',
    animation_frame='year',
    hover_name='state_po',
    color_discrete_map={'DEMOCRAT': 'BLUE', 'REPUBLICAN': 'RED', 'OTHER': 'GREEN'},
    scope='usa',
    labels={'party_simplified': 'Political Party'},
    title="Senate Winners Over Time"
)

# us_senate_winners_map
```

In [ ]:
```
# Export Plotly chart to Chart Studio
```

```
chart_studio.tools.set_credentials_file(username='gatormig', api_key=plotly_api_key.il
py.plot(us_senate_winners_map, filename = 'us_senate_winners_map_over_time', auto_oper
```

In [ ]:
```
# See of winning party members over time
f,ax_line = plt.subplots(1,1,figsize=(6,3))

winners = senate_unemployment_and_president[(senate_unemployment_and_president['winner
winning_parties = winners[['year', 'is_candidate_democrat', 'is_candidate_republican',
winners_over_time = winning_parties.groupby('year', as_index=False).mean()

# Remove last row as that was a special election with only 2 seats in January
winners_over_time.drop(winners_over_time.tail(1).index,inplace=True)

# Create winners over time chart
winners_chart = sns.lineplot(x='year', y='value', hue='variable',
                            data=pd.melt(winners_over_time, ['year']),
                            palette=dict(is_candidate_democrat="#0000FF", is_candidate
                            ax=ax_line)
plt.legend(title='Party', bbox_to_anchor=(1.02, 1), loc='upper left', borderaxespad=0)
winners_chart.set_title('Proportion of Winning Candidates By Party Over Time')
winners_chart.set_xlabel('Year')
winners_chart.set_ylabel('Proportion of Winners')
plt.grid(None)
plt.show()
```

In [ ]:
```
# See unemployment rate over time
f,ax_line = plt.subplots(1,1,figsize=(6,3))

unique_unemployment_by_year = winners[['year', 'total_unemployed_percentage_of_labor_f
unemployment_by_year = unique_unemployment_by_year.groupby('year', as_index=False).mea

# Create winners over time chart
unemployment_chart = sns.lineplot(
    x='year', y='value',
    data=pd.melt(unemployment_by_year,['year']),
    ax=ax_line)
unemployment_chart.set_title('National Unemployment Rate (as %) Over Time')
unemployment_chart.set_xlabel('Year')
unemployment_chart.set_ylabel('Unemployment Rate')
unemployment_chart.set_ylim(0, 0.10)
plt.grid(None)
plt.show()
```

In [ ]:
```
#sns.pairplot(senate_unemployment_and_president, hue = 'winner', palette = 'Set1')

#plt.show()
```

In [ ]:
```
f,ax_line = plt.subplots(1,1,figsize=(6,3))
training_correlation = training_data.corr()
corr_chart = sns.heatmap(
    data=training_correlation,
    vmin=-1,
    vmax=1,
    annot=False,
    cmap='RdBu',
    fmt='.1f',
    ax=ax_line)
```

```
corr_chart.set_title("Training Data Correlation Heatmap")
plt.show()
```

In [ ]:
```
# Determine the proportion of winners versus candidates over time to determine class b

rate_over_winners_over_time = senate_unemployment_and_president.groupby('year')['winne
    total_winners='sum',
    total_candidates='count').reset_index()

rate_over_winners_over_time['proportion_of_winners_vs_candidates'] = rate_over_winners

print("The average rate of winners vs candidates over time is: ", str(rate_over_winner
```

In [ ]:
```
f,ax_line = plt.subplots(1,1,figsize=(6,3))

winners_vs_losers_chart = sns.lineplot(
    data=rate_over_winners_over_time,
    x='year',
    y='proportion_of_winners_vs_candidates',
    ax=ax_line
)
winners_vs_losers_chart.axhline(
    rate_over_winners_over_time['proportion_of_winners_vs_candidates'].mean(),
    color='red',
    lw=1)

plt.title("Proportion of Winners vs Total Candidates By Year With Average (In Red)")
plt.xlabel("Year")
plt.ylabel("Proportion of Winners vs Total")
plt.ylim(0, 0.40)
plt.show()
```

Methodology

Logistic Regression

In [ ]:
```
#Train-Test Split using 20% Test
```

In [ ]:
```
X = training_data.drop('IsCandidateWinner', axis = 1)
y = training_data['IsCandidateWinner']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
                                                    random_state=101)
```

In [ ]:
```
#Import Logistic Regression
from sklearn.linear_model import LogisticRegression

#Fitting Model
logistic_model = LogisticRegression(class_weight='balanced', random_state=123)
logistic_model.fit(X_train, y_train)
```

In [ ]:
```
#Predictions
```

```
log_pred = logistic_model.predict(X_test)
```

In [ ]:
```
#Model Evaluation
from sklearn.metrics import classification_report, confusion_matrix
f,cm_ax = plt.subplots(1,1,figsize=(3,3))

print("Mean Error Rate is: ", str(mean(log_pred != y_test)), "\n")
print("The F1 Score is: ", f1_score(y_test, log_pred), "\n")
cm_display_lr = ConfusionMatrixDisplay.from_predictions(y_test, log_pred, display_labe
plt.grid(None)
```

Decision Trees

In [ ]:
```
#Fitting the model
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(class_weight='balanced', random_state=123)
dt.fit(X_train, y_train)
```

In [ ]:
```
#Model Evaluation
dt_pred = dt.predict(X_test)
```

In [ ]:
```
f,cm_ax = plt.subplots(1,1,figsize=(3,3))
print("Mean Error Rate is: ", str(mean(dt_pred != y_test)), "\n")
print("The F1 Score is: ", f1_score(y_test, dt_pred), "\n")
cm_display_dt = ConfusionMatrixDisplay.from_predictions(y_test, dt_pred, display_label
plt.grid(None)
```

In [ ]:
```
#Tree Visualization
from IPython.display import Image
from six import StringIO
from sklearn.tree import export_graphviz
import pydot

features = list(training_data.columns[1:])
features
```

In [ ]:
```
dot_data = StringIO()
export_graphviz(dt, out_file=dot_data,feature_names=features,filled=True,rounded=True)

graph = pydot.graph_from_dot_data(dot_data.getvalue())
dt_graph = Image(graph[0].create_png())

dt_graph
```

Random Forests

In [ ]:
```
#Fitting the model
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100, class_weight='balanced', random_state=12
rf.fit(X_train, y_train)
```

In [ ]:
```
#Model Evaluation
rf_pred = rf.predict(X_test)
```

In [ ]:
```
f,cm_ax = plt.subplots(1,1,figsize=(3,3))

print("Mean Error Rate is: ", str(mean(rf_pred != y_test)), "\n")
print("The F1 Score is: ", f1_score(y_test, rf_pred), "\n")
cm_display_rf = ConfusionMatrixDisplay.from_predictions(y_test, rf_pred, display_label
plt.grid(None)
```

In [ ]:
```
from sklearn.inspection import permutation_importance

feature_names = [f"feature {i}" for i in range(X_train.shape[1])]
importances = rf.feature_importances_
std = np.std([tree.feature_importances_ for tree in rf.estimators_], axis=0)

forest_importances = pd.Series(importances, index=X_train.columns)

fig, fi_ax = plt.subplots(1,1,figsize=(6,3))

forest_importances.plot.bar(yerr=std, ax=fi_ax)
ax.set_title("Feature importances using MDI")
ax.set_ylabel("Mean decrease in impurity")
plt.show()
```

KNN

Standardize the Variables

Because the KNN classifier predicts the class of a given test observation by identifying the observations that are nearest to it, the scale of the variables matters. Any variables that are on a large scale will have a much larger effect on the distance between the observations, and hence on the KNN classifier, than variables that are on a small scale.

In [ ]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(training_data.drop('IsCandidateWinner',axis=1))
```

In [ ]:
```
scaled_features = scaler.transform(training_data.drop('IsCandidateWinner',axis=1))
```

In [ ]:
```
df_feat = pd.DataFrame(scaled_features,columns=training_data.columns[:-1])
df_feat.head()
```

In [ ]:
```
from sklearn.model_selection import train_test_split
```

In [ ]:
```
X_traink, X_testk, y_traink, y_testk = train_test_split(scaled_features, training_data
                                                    test_size=0.20, random_state = 12
```

In [ ]:
```
#Using KNN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_traink,y_traink)
```

In [ ]:
```
predk = knn.predict(X_testk)
```

In [ ]:
```
#Evaluation
f,cm_ax = plt.subplots(1,1,figsize=(3,3))
from sklearn.metrics import classification_report,confusion_matrix

print("Mean Error Rate is: ", str(mean(predk != y_test)), "\n")
print("The F1 Score is: ", f1_score(y_test, predk), "\n")
cm_display_knn1 = ConfusionMatrixDisplay.from_predictions(y_testk, predk, display_labe
plt.grid(None)
```

In [ ]:
```
#Choosing a K value
n_neighbors = []
error_rate = []

# Will take some time
for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_traink,y_traink)
    pred_i = knn.predict(X_testk)
    n_neighbors.append(i)
    error_rate.append(np.mean(pred_i != y_testk))

knn_error_rates = list(zip(n_neighbors, error_rate))

knn_error_df = pd.DataFrame(knn_error_rates, columns = ['Neighbors', 'Error Rate'])

knn_error_df.head()
```

KNN Grid Search

Below we'll use KNN Grid Search to find the best algorithm for the KNN model for our data based on a variety of parameters

Note - Code commented out below for time-running purposes both the algorithm found the following was optimal:

- **algorithm** - 'kd_tree'
- **leaf_size** - 3
- **metric** - 'euclidean'
- **n_neighbors** - 9

In [ ]:
```
#knn_grid = KNeighborsClassifier()
#k_range = list(range(2, 40))
#leaf_size = list(range(2, 50))

#parameter_grid = {
#    'n_neighbors': k_range,
#    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
#    'leaf_size': leaf_size,
#    'metric': ['cityblock', 'euclidean', 'l1', 'l2', 'manhattan'],
#}
```

```
#grid_search = GridSearchCV(estimator = knn_grid, param_grid = parameter_grid, cv = 10

#grid_search.fit(X_traink, y_traink)

# Display the best parameters based on the GridSearchCV
#print(grid_search.best_params_)
```

In [ ]:
```
plt.figure(figsize=(6,3))
sns.lineplot(data= knn_error_df, x='Neighbors', y='Error Rate', color='blue', linestyl
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
plt.show()
```

In [ ]:
```
# Find minimum K-Value where Neighbors > 1
f,cm_ax = plt.subplots(1,1,figsize=(3,3))

minimum_k = int(knn_error_df.iloc[knn_error_df[knn_error_df['Neighbors'] > 1]['Error R

print("The Minimum K-Value is: ", minimum_k, "\n")

knn = KNeighborsClassifier(n_neighbors=minimum_k)

knn.fit(X_traink,y_traink)
pred = knn.predict(X_testk)

print("Mean Error Rate is: ", str(mean(pred != y_testk)), "\n")
print("The F1 Score is: ", f1_score(y_testk, pred), "\n")
cm_display_knn = ConfusionMatrixDisplay.from_predictions(y_testk, pred, display_labels
plt.grid(None)
```

KNN Grid Search Output

We'll run the KNN Code as well with our GridSearch Values to evaluate performance against the test set

In [ ]:
```
f,cm_ax = plt.subplots(1,1,figsize=(3,3))

knn_grid = KNeighborsClassifier(n_neighbors=9, algorithm='kd_tree', leaf_size=3, metri

knn_grid.fit(X_traink,y_traink)
pred_grid = knn_grid.predict(X_testk)

print("The Grid Search KNN output is below: \n")

print("Mean Error Rate is: ", str(mean(pred_grid != y_testk)), "\n")
print("The F1 Score is: ", f1_score(y_testk, pred_grid), "\n")
cm_display_knn_grid = ConfusionMatrixDisplay.from_predictions(y_testk, pred_grid, disp
plt.grid(None)
```

Boosting - AdaBoostClassifier

In [ ]:
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
f,cm_ax = plt.subplots(1,1,figsize=(3,3))

ada = AdaBoostClassifier(random_state=123)
ada.fit(X_train, y_train)
ada_preds = ada.predict(X_test)
print("Mean Error Rate is: ", str(mean(ada_preds != y_test)), "\n")
print("The F1 Score is: ", f1_score(y_test, ada_preds), "\n")
cm_display_ada = ConfusionMatrixDisplay.from_predictions(y_test, ada_preds, display_la
plt.grid(None)
```

Results

As part of our predictions, we'll have to scale our test set for KNN

In [ ]:
```
# Scale the prediction data for KNN
scaler_pred = StandardScaler()
scaler_pred.fit(senate_candidates_2022_test_set)
scaled_pred_features = scaler_pred.transform(senate_candidates_2022_test_set)
knn_df_pred_feat = pd.DataFrame(scaled_pred_features,columns=senate_candidates_2022_te

knn_df_pred_feat.head()
```

In [ ]:
```
logistic_midterms_pred = logistic_model.predict(senate_candidates_2022_test_set)
dt_midterms_pred = dt.predict(senate_candidates_2022_test_set)
rf_midterms_pred = rf.predict(senate_candidates_2022_test_set)
knn_midterms_pred = knn.predict(knn_df_pred_feat.values)
knn_grid_midterms_pred = knn_grid.predict(knn_df_pred_feat.values)
boosting_midterms_pred = ada.predict(senate_candidates_2022_test_set)

predicted_results = pd.DataFrame({'Candidate': candidates_2022, 'State': state_2022, '
predicted_results['Sum of Predictions'] = predicted_results['Logistic'] + predicted_re
predicted_results.sort_values(by=['Sum of Predictions'], inplace=True, ascending=False
pd.set_option('display.max_rows', None)

predicted_results.head()
```

In [ ]:
```
logistic_missclassified = (len(predicted_winner_2022) - sum(predicted_winner_2022 == l
dt_misclassified = (len(predicted_winner_2022) - sum(predicted_winner_2022 == dt_midte
rf_misclassified = (len(predicted_winner_2022) - sum(predicted_winner_2022 == rf_midte
knn_misclassified = (len(predicted_winner_2022) - sum(predicted_winner_2022 == knn_mic
knn_grid_misclassified = (len(predicted_winner_2022) - sum(predicted_winner_2022 == kn
boosting_misclassified = (len(predicted_winner_2022) - sum(predicted_winner_2022 == bo
```

In [ ]:
```
model_names = ['Logistic', 'Decision Trees', 'Random Forest', 'KNN', 'KNN - Grid Searc
mean_error_rates = [logistic_missclassified, dt_misclassified, rf_misclassified, knn_m

model_error_rates = list(zip(model_names, mean_error_rates))

mean_error_rates_df = pd.DataFrame(model_error_rates, columns = ['Model Name', 'Mean E

mean_error_rates_df = mean_error_rates_df.sort_values('Mean Error Rate', ascending=Tru

mean_error_rates_df
```

Evaluation against 538 and Actual Results

Below is the code that evaluates our best results (KNN Grid Search) against the 538 Predictions and the actual midterm election results

In [ ]:
```
senate_winners_2022_results = senate_winners_2022[['candidate', 'state', 'is_predictec

best_predictions = predicted_results[['Candidate', 'State', 'KNN - Grid Search']]

best_predictions = best_predictions.rename(mapper={'Candidate':'candidate', 'State':'s

evaluation = senate_winners_2022_results.merge(best_predictions, on='candidate', how='

evaluation.rename(mapper={'state_x':'state'}, axis='columns', inplace=True)

evaluation.drop('state_y', axis='columns', inplace=True)

evaluation.head()
```

In [ ]:
```
evaluation['predicted_correctly_vs_538'] = evaluation['is_predicted_winner_538'] == ev
evaluation['actual_vs_538'] = evaluation['is_predicted_winner_538'] == evaluation['is_
evaluation['predicted_correctly_vs_actual_winner'] = evaluation['is_actual_winner'] ==

evaluation.head()
```

In [ ]:
```
f,cm_ax = plt.subplots(1,1,figsize=(3,3))

evaluation['is_actual_winner'] = np.where(evaluation['is_actual_winner']==1, True, Fal
evaluation['predicted_winner'] = np.where(evaluation['predicted_winner']==1, True, Fal

print("Mean Error Rate is: ", str(mean(evaluation['predicted_winner'] != evaluation['i
print("The F1 Score is: ", f1_score(evaluation['predicted_winner'], evaluation['is_act

cm_evaluation_actual = ConfusionMatrixDisplay.from_predictions(
    evaluation['is_actual_winner'],
    evaluation['predicted_winner'],
    display_labels=['Not Winner', 'Winner'],
    cmap='Blues',
    ax=cm_ax)
plt.grid(None)
```

In [ ]:
```
f,cm_ax = plt.subplots(1,1,figsize=(3,3))
evaluation['is_predicted_winner_538'] = np.where(evaluation['is_predicted_winner_538']
evaluation['predicted_winner'] = np.where(evaluation['predicted_winner']==1, True, Fal

print("Mean Error Rate is: ", str(mean(evaluation['predicted_winner'] != evaluation['i
print("The F1 Score is: ", f1_score(evaluation['predicted_winner'], evaluation['is_pre

cm_evaluation_actual = ConfusionMatrixDisplay.from_predictions(
    evaluation['is_predicted_winner_538'],
    evaluation['predicted_winner'],
    display_labels=['Not Winner', 'Winner'],
    cmap='Blues',
    ax=cm_ax)
plt.grid(None)
```

In [ ]:
```
f,cm_ax = plt.subplots(1,1,figsize=(3,3))
print("Mean Error Rate is: ", str(mean(evaluation['is_actual_winner'] != evaluation['i
print("The F1 Score is: ", f1_score(evaluation['is_actual_winner'], evaluation['is_pre

cm_evaluation_actual = ConfusionMatrixDisplay.from_predictions(
    evaluation['is_predicted_winner_538'],
    evaluation['is_actual_winner'],
    display_labels=['Not Winner', 'Winner'],
    cmap='Blues',
    ax=cm_ax)
plt.grid(None)
```

Appendix

In [ ]:
```
profile = ProfileReport(training_data, title="Pandas Profiling Report")
```

In [ ]:
```
# profile.to_notebook_iframe()
```