DATS 6103

Individual Final Report

May 1, 2021

Bradley Reardon

1. **Introduction**

* The 2020 US Presidential Election caused a great deal of speculation surrounding voter-turnout, largely driven by the high-profile candidates and new voting methods enacted to account for Covid-19 safety precautions. Furthermore, there is a long-standing interest in what drives eligible voters to either vote or not vote, and who to vote for if a vote is cast. We decided to dive deeper into the latter and explore which characteristics and features of a voter drove them to choose between Donald Trump and Joe Biden in the 2020 US Presidential Election.
* In this project, we trained and tested variations of both random forest and gradient boosting classifiers using survey data compiled by Ipsos, a multinational market research and consulting firm, and FiveThirtyEight, an American website that focuses on opinion poll analysis, politics, economics, and sports blogging. Our goal was to accurately predict the answer a survey taker might choose for question 23, “Which presidential candidate are you planning to support?”
* The shared work consisted of deciding on a dataset and project idea, cleaning of the dataset, exploratory data analysis, preprocessing, modeling, model comparison, GUI development, creating a powerpoint presentation, writing the group report, and creating a demo of the GUI.

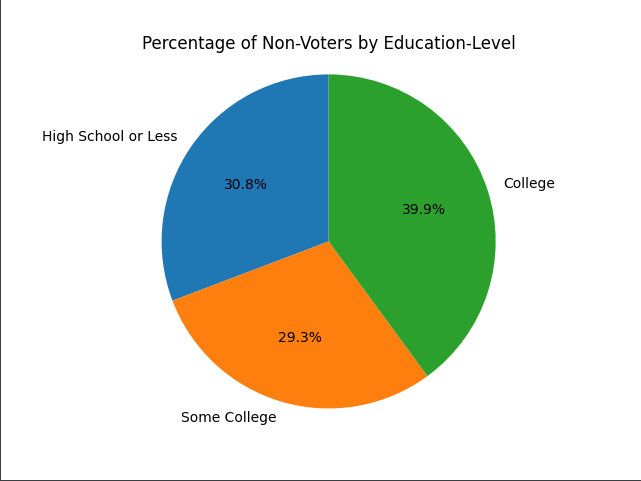
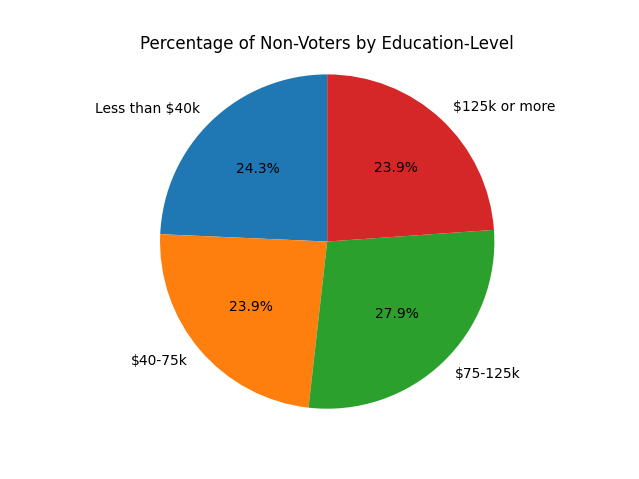
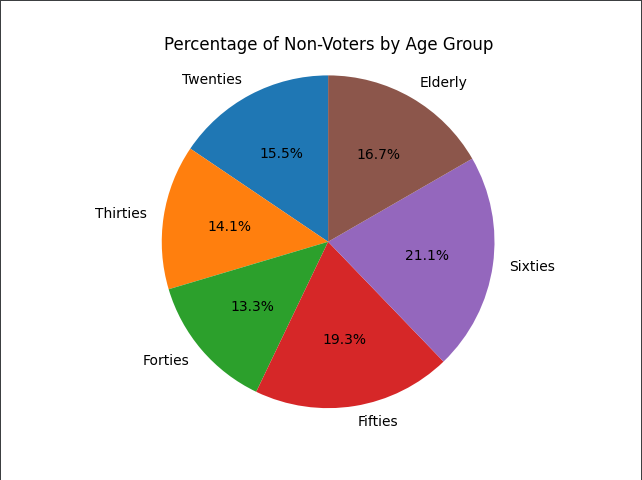
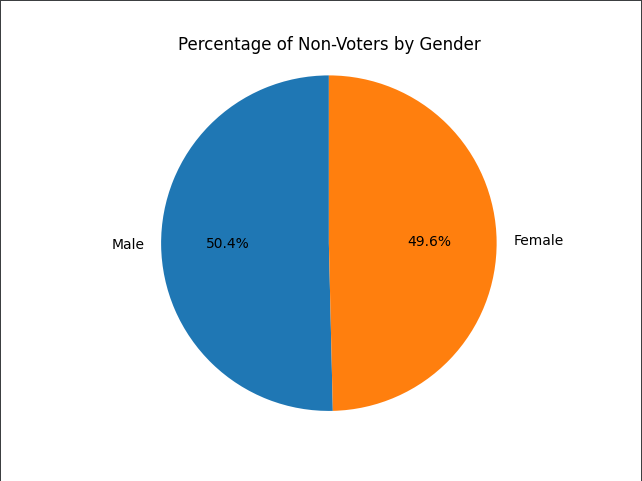
1. **Personal Contribution**

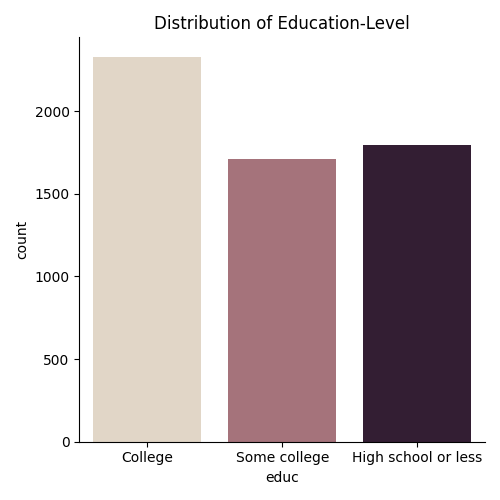
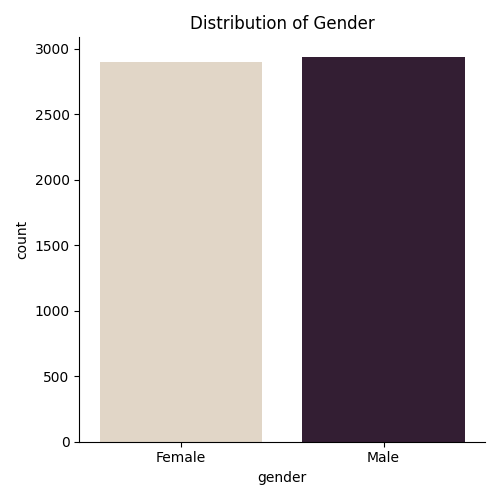
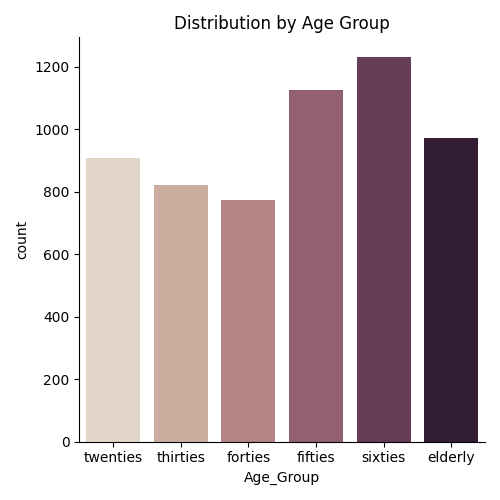
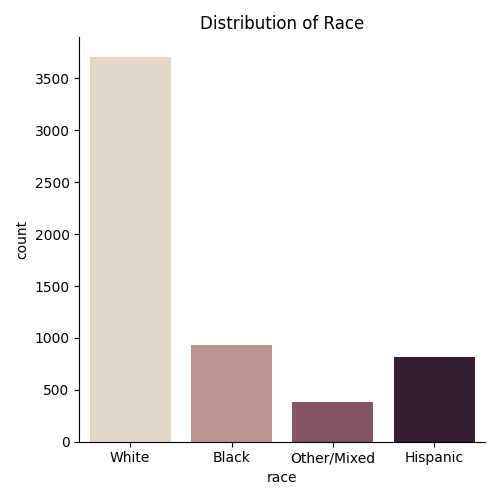
* **Code:**
  + Preprocessing
    - Added new column, “Age\_Group” in order to create a pie chart showing the distribution by age group.
    - Split data into train and test sets.
  + EDA
    - Created pie charts and histograms to check for normality in our demographic features.
    - Calculated and plotted feature importance
  + Modeling
    - Trained and tested the full models for both the random forest and gradient boosting classifiers as well as the gradient boosting slim model.
    - Tested and compared the models to see which model performed the best.
  + Merging of Code
    - Provided the initial merge of our code into a single final python file. Divya and I then both touched up the file until it was finalized.
* **Powerpoint:**
  + Found the powerpoint template.
  + Formatted powerepoint and added initial information as draft 1.
    - Touched up powerpoint with other group members to finalize the presentation.
  + Added audio recording to my section of the presentation.
* **Group Final Report:**
  + Wrote first draft
    - Finalized rest of report with group members.

1. **Personal Contribution in Detail**

* Please see the diagrams in the results section along with the code appended at the end of the report.

1. **Results**





|  |  |  |  |
| --- | --- | --- | --- |
| **Model 1**  **Random Forest - Full Model** | **Model 2**  **Random Forest - Slim Model** | **Model 3**  **Gradient Boosting - Full Model** | **Model 4**  **Gradient Boosting - Slim Model** |
| F1-score: 0.98  Accuracy score: 0.97 | F1-score: 0.93  Accuracy score: 0.93 | F1-score: 0.97  Accuracy score: 0.97 | F1-score: 0.50  Accuracy score: 0.51 |

1. **Summary**
   * Using FiveThirtyEight survey data, we decided to predict who voters would vote for president based on their survey answers
   * We conducted exploratory data analysis to better understand the group of voters and make sure the classes were balanced
   * We preprocessed our data - label encoding, dropping columns and observations
   * We fit both random forest and gradient boosting models, and we ran on “full” and “slim” feature sets
   * We saw extremely high accuracy and f1 scores
   * Random forest did better than gradient boosting, and “full” feature models did better than “slim” feature models
2. **Percent of Code**
   * 0%. No code was copied from any external sources. Syntax and functions were referenced from official python package sites, but we wrote all code on our own.
3. **References**

* <https://scikit-learn.org>
* <https://numpy.org/>
* <https://pandas.pydata.org/>
* <https://pypi.org/project/PyQt5/>
* <https://medium.com/analytics-vidhya/evaluating-a-random-forest-model-9d165595ad56>
* <https://www.datasciencecentral.com/profiles/blogs/decision-tree-vs-random-forest-vs-boosted-trees-explained#:~:text=Like%20random%20forests%2C%20gradient%20boosting,one%20tree%20at%20a%20time>
* <https://morningconsult.com/opinions/to-persuade-or-to-turn-out-voters-is-that-the-question/>
* https://www.bloomberg.com/graphics/2020-us-election-results/methodology

**Appended Code – My Personal Code**

#------------------------------------------------------  
# Import necessary packages  
#------------------------------------------------------  
  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.model\_selection import train\_test\_split  
from sklearn.model\_selection import cross\_validate  
from sklearn.preprocessing import LabelEncoder  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.feature\_selection import SelectFromModel  
import seaborn as sns  
from sklearn.metrics import confusion\_matrix  
from sklearn.metrics import accuracy\_score  
from sklearn.metrics import classification\_report  
from sklearn.metrics import roc\_auc\_score  
from sklearn.ensemble import GradientBoostingClassifier  
  
#------------------------------------------------------  
# Read in csv file  
#------------------------------------------------------  
  
nv\_df = pd.read\_csv('nonvoters\_data.csv')  
# print(nv\_df.shape)  
# print(nv\_df.columns)  
print(nv\_df.shape)  
#------------------------------------------------------  
# Preprocessing  
#------------------------------------------------------  
  
nv\_df.columns = ['RespId', 'weight',  
 'q1\_uscitizen',  
 'q2\_important\_voting','q2\_important\_jury','q2\_important\_following','q2\_important\_displaying','q2\_important\_census',  
 'q2\_important\_pledge','q2\_important\_military','q2\_important\_respect','q2\_important\_god','q2\_important\_protesting',  
 'q3\_statement\_racism1`','q3\_statement\_racism2','q3\_statement\_feminine',  
 'q3\_statement\_msm','q3\_statement\_politiciansdontcare','q3\_statement\_besensitive',  
 'q4\_impact\_officialsfed','q4\_impact\_officialsstate','q4\_impact\_officialslocal',  
 'q4\_impact\_news','q4\_impact\_wallstreet','q4\_impact\_lawenforcement',  
 'q5\_electionmatters',  
 'q6\_officialsarelikeyou',  
 'q7\_governmentdesign',  
 'q8\_trust\_presidency','q8\_trust\_congress','q8\_trust\_supremecourt','q8\_trust\_cdc','q8\_trust\_electedofficials',  
 'q8\_trust\_fbicia','q8\_trust\_newsmedia','q8\_trust\_police','q8\_trust\_postalservice',  
 'q9\_politicalsystems\_democracy','q9\_politicalsystems\_experts','q9\_politicalsystems\_strongleader','q9\_politicalsystems\_army',  
 'q10\_disability','q10\_chronic\_illness','q10\_unemployed','q10\_evicted',  
 'q11\_lostjob','q11\_gotcovid','q11\_familycovid',  
 'q11\_coviddeath','q11\_worriedmoney','q11\_quitjob',  
 'q14\_view\_of\_republicans',  
 'q15\_view\_of\_democrats',  
 'q16\_how\_easy\_vote',  
 'q17\_secure\_votingmachines','q17\_secure\_paperballotsinperson','q17\_secure\_paperballotsmail','q17\_secure\_electronicvotesonline',  
 'q18\_votingsituations1','q18\_votingsituations2','q18\_votingsituations3','q18\_votingsituations4','q18\_votingsituations5',  
 'q18\_votingsituations6','q18\_votingsituations7','q18\_votingsituations8','q18\_votingsituations9','q18\_votingsituations10',  
 'q19\_get\_more\_voting1','q19\_get\_more\_voting2','q19\_get\_more\_voting3','q19\_get\_more\_voting4','q19\_get\_more\_voting5',  
 'q19\_get\_more\_voting6','q19\_get\_more\_voting7','q19\_get\_more\_voting8','q19\_get\_more\_voting9','q19\_get\_more\_voting10',  
 'q20\_currentlyregistered',  
 'q21\_plan\_to\_vote',  
 'q22\_whynotvoting\_2020',  
 'q23\_which\_candidate\_supporting',  
 'q24\_preferred\_voting\_method',  
 'q25\_howcloselyfollowing\_election',  
 'q26\_which\_voting\_category',  
 'q27\_didyouvotein18','q27\_didyouvotein16','q27\_didyouvotein14',  
 'q27\_didyouvotein12','q27\_didyouvotein10','q27\_didyouvotein08',  
 'q28\_whydidyouvote\_past1','q28\_whydidyouvote\_past2','q28\_whydidyouvote\_past3','q28\_whydidyouvote\_past4',  
 'q28\_whydidyouvote\_past5','q28\_whydidyouvote\_past6','q28\_whydidyouvote\_past7','q28\_whydidyouvote\_past8',  
 'q29\_whydidyounotvote\_past1','q29\_whydidyounotvote\_past2','q29\_whydidyounotvote\_past3','q29\_whydidyounotvote\_past4','q29\_whydidyounotvote\_past5',  
 'q29\_whydidyounotvote\_past6','q29\_whydidyounotvote\_past7','q29\_whydidyounotvote\_past8','q29\_whydidyounotvote\_past9','q29\_whydidyounotvote\_past10',  
 'q30\_partyidentification',  
 'q31\_republicantype',  
 'q32\_democratictype',  
 'q33\_closertowhichparty',  
 'ppage', 'educ', 'race', 'gender', 'income\_cat', 'voter\_category'  
 ]  
  
nv\_df.drop(['q1\_uscitizen','q22\_whynotvoting\_2020',  
 'q28\_whydidyouvote\_past1','q28\_whydidyouvote\_past2','q28\_whydidyouvote\_past3','q28\_whydidyouvote\_past4',  
 'q28\_whydidyouvote\_past5','q28\_whydidyouvote\_past6','q28\_whydidyouvote\_past7','q28\_whydidyouvote\_past8',  
 'q29\_whydidyounotvote\_past1','q29\_whydidyounotvote\_past2','q29\_whydidyounotvote\_past3','q29\_whydidyounotvote\_past4','q29\_whydidyounotvote\_past5',  
 'q29\_whydidyounotvote\_past6','q29\_whydidyounotvote\_past7','q29\_whydidyounotvote\_past8','q29\_whydidyounotvote\_past9','q29\_whydidyounotvote\_past10',  
 'q31\_republicantype',  
 'q32\_democratictype',  
 'q33\_closertowhichparty',  
 'RespId',  
 'weight'  
 ], axis=1, inplace=True)  
  
age\_labels\_cut = ['twenties', 'thirties', 'forties', 'fifties', 'sixties', 'elderly']  
age\_bins= [20, 30, 40, 50, 60, 70, 200]  
nv\_df['Age\_Group'] = pd.cut(nv\_df['ppage'], bins = age\_bins, labels = age\_labels\_cut, right = False)  
  
  
# %%-----------------------------------------------------------------------  
# Race Pie Chart & Histogram  
  
  
distinct\_races = set(nv\_df['race'])  
total\_race = nv\_df['race'].count()  
hispanic\_percentage = nv\_df[nv\_df['race'] == 'Hispanic']['race'].count()/total\_race  
other\_mixed\_percentage = nv\_df[nv\_df['race'] == 'Other/Mixed']['race'].count()/total\_race  
white\_percentage = nv\_df[nv\_df['race'] == 'White']['race'].count()/total\_race  
black\_percentage = nv\_df[nv\_df['race'] == 'Black']['race'].count()/total\_race  
race\_percentages = [white\_percentage, black\_percentage, hispanic\_percentage, other\_mixed\_percentage]  
race\_labels = ['White', 'Black', 'Hispanic', 'Other/Mixed']  
  
race\_pie, ax1 = plt.subplots()  
ax1.pie(race\_percentages, labels=race\_labels, autopct='%1.1f%%', startangle=90)  
ax1.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.  
plt.title(label = 'Percentage of Non-Voters by Race')  
plt.show()  
  
sns.catplot(x='race', kind='count', palette = "ch:.25", data = nv\_df)  
  
# %%-----------------------------------------------------------------------  
# Gender Pie Chart & Histogram  
  
  
distinct\_genders = set(nv\_df['gender'])  
total\_gender = nv\_df['gender'].count()  
male\_percentage = nv\_df[nv\_df['gender'] == 'Male']['gender'].count()/total\_gender  
female\_percentage = nv\_df[nv\_df['gender'] == 'Female']['gender'].count()/total\_gender  
gender\_percentages = [male\_percentage, female\_percentage]  
gender\_labels = ['Male', 'Female']  
gender\_pie, ax1 = plt.subplots()  
ax1.pie(gender\_percentages, labels=gender\_labels, autopct='%1.1f%%', startangle=90)  
ax1.axis('equal')  
plt.title(label = 'Percentage of Non-Voters by Gender')  
plt.show()  
  
sns.catplot(x='gender', kind='count', palette = "ch:.25", data = nv\_df)  
  
# %%-----------------------------------------------------------------------  
# Age Pie Chart & Histogram  
  
  
total\_age = nv\_df['Age\_Group'].count()  
twenties = nv\_df[nv\_df['Age\_Group'] == 'twenties']['Age\_Group'].count()/total\_age  
thirties = nv\_df[nv\_df['Age\_Group'] == 'thirties']['Age\_Group'].count()/total\_age  
forties = nv\_df[nv\_df['Age\_Group'] == 'forties']['Age\_Group'].count()/total\_age  
fifties = nv\_df[nv\_df['Age\_Group'] == 'fifties']['Age\_Group'].count()/total\_age  
sixties = nv\_df[nv\_df['Age\_Group'] == 'sixties']['Age\_Group'].count()/total\_age  
elderly = nv\_df[nv\_df['Age\_Group'] == 'elderly']['Age\_Group'].count()/total\_age  
age\_percentages = [twenties, thirties, forties, fifties, sixties, elderly]  
age\_labels = ['Twenties', 'Thirties', 'Forties', 'Fifties', 'Sixties', 'Elderly']  
age\_pie, ax1 = plt.subplots()  
ax1.pie(age\_percentages, labels=age\_labels, autopct='%1.1f%%', startangle=90)  
ax1.axis('equal')  
plt.title(label = 'Percentage of Non-Voters by Age Group')  
plt.show()  
  
sns.catplot(x='Age\_Group', kind='count', palette = "ch:.25", data = nv\_df)  
plt.title(label = 'Distribution by Age Group')  
  
# %%-----------------------------------------------------------------------  
# Education-Level Pie Chart & Histogram  
  
  
distinct\_educ = set(nv\_df['educ'])  
total\_educ = nv\_df['educ'].count()  
hs\_percentage = nv\_df[nv\_df['educ'] == 'High school or less']['educ'].count()/total\_educ  
some\_college\_percentage = nv\_df[nv\_df['educ'] == 'Some college']['educ'].count()/total\_educ  
college\_percentage = nv\_df[nv\_df['educ'] == 'College']['educ'].count()/total\_educ  
educ\_percentages = [hs\_percentage, some\_college\_percentage, college\_percentage]  
educ\_labels = ['High School or Less', 'Some College', 'College']  
  
educ\_pie, ax1 = plt.subplots()  
ax1.pie(educ\_percentages, labels=educ\_labels, autopct='%1.1f%%', startangle=90)  
ax1.axis('equal')  
plt.title(label = 'Percentage of Non-Voters by Education-Level')  
plt.show()  
  
sns.catplot(x='educ', kind='count', palette = "ch:.25", data = nv\_df)  
  
# %%-----------------------------------------------------------------------  
# Income Category Pie Chart & Histogram  
  
  
distinct\_income = set(nv\_df['income\_cat'])  
total\_income= nv\_df['income\_cat'].count()  
  
income1\_percentage = nv\_df[nv\_df['income\_cat'] == 'Less than $40k']['income\_cat'].count()/total\_income  
income2\_percentage = nv\_df[nv\_df['income\_cat'] == '$40-75k']['income\_cat'].count()/total\_income  
income3\_percentage = nv\_df[nv\_df['income\_cat'] == '$75-125k']['income\_cat'].count()/total\_income  
income4\_percentage = nv\_df[nv\_df['income\_cat'] == '$125k or more']['income\_cat'].count()/total\_income  
educ\_percentages = [income1\_percentage, income2\_percentage, income3\_percentage, income4\_percentage]  
educ\_labels = ['Less than $40k', '$40-75k', '$75-125k', '$125k or more']  
  
income\_pie, ax1 = plt.subplots()  
ax1.pie(educ\_percentages, labels=educ\_labels, autopct='%1.1f%%', startangle=90)  
ax1.axis('equal')  
plt.title(label = 'Percentage of Non-Voters by Education-Level')  
plt.show()  
  
sns.catplot(x='income\_cat', kind='count', palette = "ch:.25", data = nv\_df)  
  
#------------------------------------------------------  
# Race vs. Candidate  
#------------------------------------------------------  
  
race\_gender\_candidate = nv\_df.pivot\_table(index = 'race', columns = 'gender', values = 'q23\_which\_candidate\_supporting')  
sns.heatmap(race\_gender\_candidate)  
plt.show()  
  
#------------------------------------------------------  
# Modeling  
#------------------------------------------------------  
  
# %%-----------------------------------------------------------------------  
# Apply label encoder to features where necessary  
  
le = LabelEncoder()  
nv\_df['educ'] = le.fit\_transform(nv\_df['educ'])  
nv\_df['race'] = le.fit\_transform(nv\_df['race'])  
nv\_df['gender'] = le.fit\_transform(nv\_df['gender'])  
nv\_df['income\_cat'] = le.fit\_transform(nv\_df['income\_cat'])  
nv\_df['voter\_category'] = le.fit\_transform(nv\_df['voter\_category'])  
nv\_df['Age\_Group'] = le.fit\_transform(nv\_df['Age\_Group'])  
  
# %%-----------------------------------------------------------------------  
# Create train and test sets  
  
nv\_df\_mod = nv\_df[(nv\_df['q23\_which\_candidate\_supporting'] == 1) | (nv\_df['q23\_which\_candidate\_supporting'] == 2)]  
X = nv\_df\_mod.drop('q23\_which\_candidate\_supporting', axis=1)  
y = nv\_df\_mod['q23\_which\_candidate\_supporting']  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 100)  
clf = RandomForestClassifier(n\_estimators=100)  
clf.fit(X\_train, y\_train)  
sel = SelectFromModel(clf)  
  
# %%-----------------------------------------------------------------------  
# Feature Importance  
  
importances = clf.feature\_importances\_  
f\_importances = pd.Series(importances, nv\_df\_mod.iloc[:, :-1].columns)  
f\_importances.sort\_values(ascending=False, inplace=True)  
f\_importances.plot(x='Features', y='Importance', kind='bar', figsize=(16, 9), rot=90, fontsize=15)  
plt.tight\_layout()  
plt.title('Feature Importance')  
plt.show()  
  
# %%-----------------------------------------------------------------------  
# Dropped Questions 14 and 15; trained model  
  
X\_Train\_dropped\_14\_15 = X\_train.drop(['q14\_view\_of\_republicans',  
 'q15\_view\_of\_democrats', ], axis = 1)  
clf\_dropped\_14\_15 = RandomForestClassifier(n\_estimators=100)  
clf\_dropped\_14\_15.fit(X\_Train\_dropped\_14\_15, y\_train)  
  
X\_Test\_dropped\_14\_15 = X\_test.drop(['q14\_view\_of\_republicans',  
 'q15\_view\_of\_democrats'], axis = 1)  
# print(X\_Train\_dropped\_14\_15.columns.get\_loc('q23\_which\_candidate\_supporting'))  
# %%-----------------------------------------------------------------------  
# Predictions  
  
y\_pred = clf.predict(X\_test)  
y\_pred\_score = clf.predict\_proba(X\_test)  
  
y\_pred\_dropped\_14\_15 = clf\_dropped\_14\_15.predict(X\_Test\_dropped\_14\_15)  
y\_pred\_score\_dropped\_14\_15 = clf\_dropped\_14\_15.predict\_proba(X\_Test\_dropped\_14\_15)  
  
# %%-----------------------------------------------------------------------  
# calculate metrics for base model  
  
print("\n")  
print("Results Using All Features: \n")  
  
print("Classification Report: ")  
print(classification\_report(y\_test,y\_pred))  
print("\n")  
  
print("Accuracy : ", accuracy\_score(y\_test, y\_pred) \* 100)  
print("\n")  
  
print("ROC\_AUC : ", roc\_auc\_score(y\_test,y\_pred\_score[:,1]) \* 100)  
  
# calculate metrics for new model  
print("\n")  
print("Results Without Q14 and Q15 features: \n")  
print("Classification Report: ")  
print(classification\_report(y\_test,y\_pred\_dropped\_14\_15))  
print("\n")  
print("Accuracy : ", accuracy\_score(y\_test, y\_pred\_dropped\_14\_15) \* 100)  
print("\n")  
print("ROC\_AUC : ", roc\_auc\_score(y\_test,y\_pred\_score\_dropped\_14\_15[:,1]) \* 100)  
  
# %%-----------------------------------------------------------------------  
# confusion matrix for base model  
conf\_matrix = confusion\_matrix(y\_test, y\_pred)  
class\_names = nv\_df\_mod['q23\_which\_candidate\_supporting'].unique()  
  
  
df\_cm = pd.DataFrame(conf\_matrix, index=class\_names, columns=class\_names )  
  
plt.figure(figsize=(5,5))  
  
hm = sns.heatmap(df\_cm, cbar=False, annot=True, square=True, fmt='d', annot\_kws={'size': 20}, yticklabels=df\_cm.columns, xticklabels=df\_cm.columns)  
  
hm.yaxis.set\_ticklabels(hm.yaxis.get\_ticklabels(), rotation=0, ha='right', fontsize=20)  
hm.xaxis.set\_ticklabels(hm.xaxis.get\_ticklabels(), rotation=0, ha='right', fontsize=20)  
plt.ylabel('True label',fontsize=20)  
plt.xlabel('Predicted label',fontsize=20)  
# Show heat map  
plt.tight\_layout()  
plt.show()  
  
# %%-----------------------------------------------------------------------  
  
# Confusion matrix for new model  
  
conf\_matrix = confusion\_matrix(y\_test, y\_pred\_dropped\_14\_15)  
class\_names = nv\_df\_mod['q23\_which\_candidate\_supporting'].unique()  
  
  
df\_cm = pd.DataFrame(conf\_matrix, index=class\_names, columns=class\_names )  
  
plt.figure(figsize=(5,5))  
  
hm = sns.heatmap(df\_cm, cbar=False, annot=True, square=True, fmt='d', annot\_kws={'size': 20}, yticklabels=df\_cm.columns, xticklabels=df\_cm.columns)  
  
hm.yaxis.set\_ticklabels(hm.yaxis.get\_ticklabels(), rotation=0, ha='right', fontsize=20)  
hm.xaxis.set\_ticklabels(hm.xaxis.get\_ticklabels(), rotation=0, ha='right', fontsize=20)  
plt.ylabel('True label',fontsize=20)  
plt.xlabel('Predicted label',fontsize=20)  
# Show heat map  
plt.tight\_layout()  
plt.show()  
  
# %%-----------------------------------------------------------------------  
  
# Gradient Boosting Classifier  
  
gb\_clf = GradientBoostingClassifier(n\_estimators=100, learning\_rate=0.05)  
gb\_clf.fit(X\_train, y\_train)  
gb\_pred = gb\_clf.predict(X\_test)  
gb\_score = gb\_clf.predict\_proba(X\_test)  
  
# Calculate metrics for boosting model  
  
print("\n")  
print("Results Using Gradient Boosting & All Features: \n")  
  
print("Classification Report: ")  
print(classification\_report(y\_test,gb\_pred))  
print("\n")  
  
print("Accuracy : ", accuracy\_score(y\_test, gb\_pred) \* 100)  
print("\n")  
  
print("ROC\_AUC : ", roc\_auc\_score(y\_test,gb\_score[:,1]) \* 100)