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.

## 2. 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.

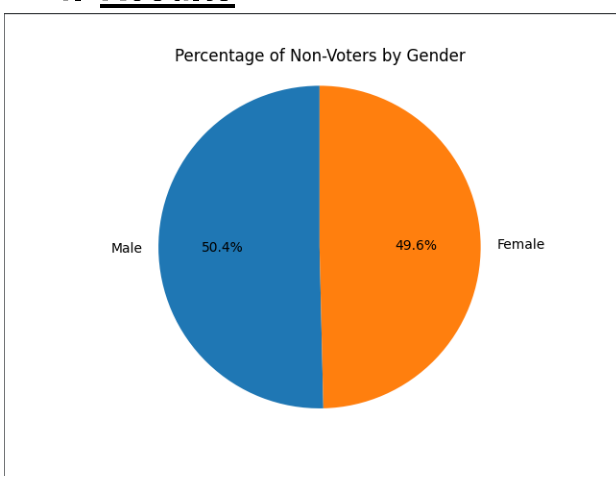
- **Powerpoint:**
    - Found the powerpoint template.
    - Formatted powerepoint and added initial information as draft 1.
        - Touched up powerpoint with other group members to finalize the presentation.
    - Added audio recording to my section of the presentation.

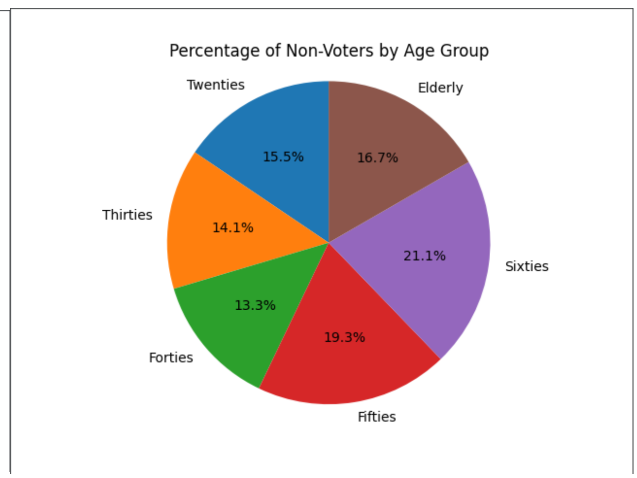
- **Group Final Report:**
    - Wrote first draft
        - Finalized rest of report with group members.

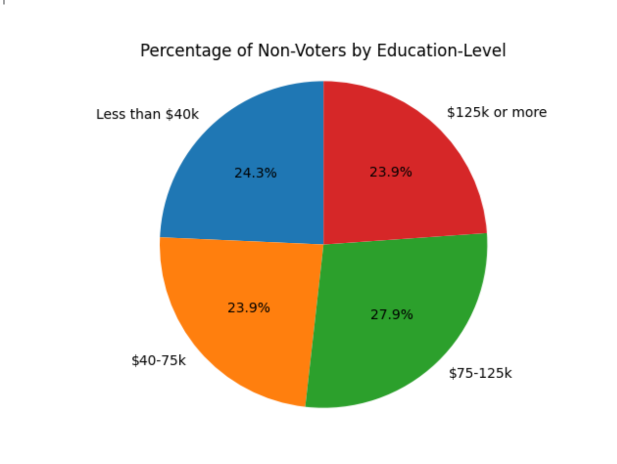
## 3. Personal Contribution in Detail

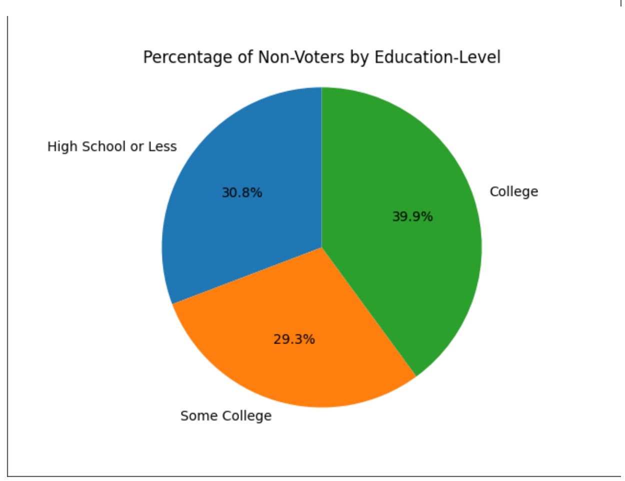
- Please see the diagrams in the results section along with the code appended at the end of the report.

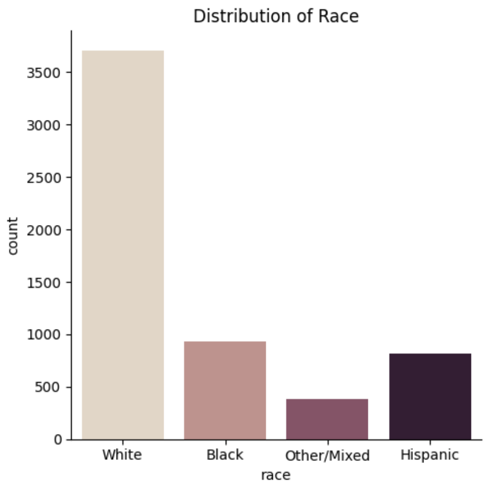
## 4. Results

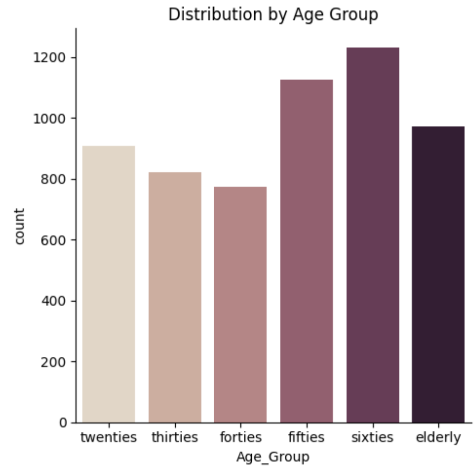


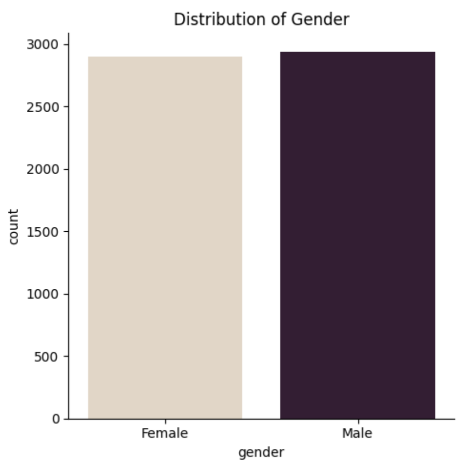



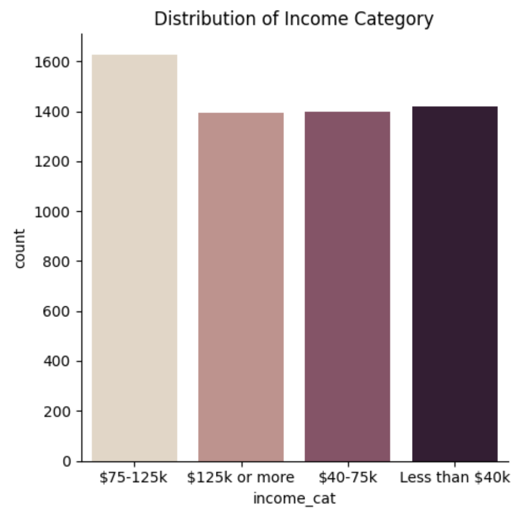



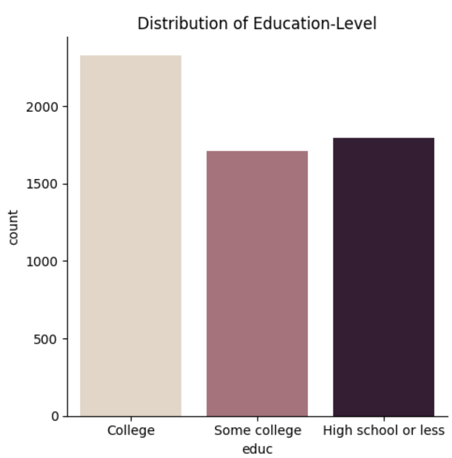




Distribution of Race
count
0
500
1000
1500
2000
2500
3000
3500
White
Black
Other/Mixed
Hispanic
race


Distribution by Age Group
count
0
200
400
600
800
1000
1200
twenties
thirties
forties
fifties
sixties
elderly
Age_Group


Distribution of Gender
count
0
500
1000
1500
2000
2500
3000
Female
Male
gender


Distribution of Income Category
count
0
200
400
600
800
1000
1200
1400
1600
$75-125k
$125k or more
$40-75k
Less than $40k
income_cat


Distribution of Education-Level
count
0
500
1000
1500
2000
College
Some college
High school or less
educ

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

## 5. 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

## 6. 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.

## 7. 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_c
dc','q8_trust_electedofficials',

'q8_trust_fbicia','q8_trust_newsmedia','q8_trust_police','q8_trust_postalserv
ice',

'q9_politicalsystems_democracy','q9_politicalsystems_experts','q9_politicalsy
stems_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_pap
erballotsmail','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_whydidyounotvo
te_past3','q29_whydidyounotvote_past4','q29_whydidyounotvote_past5',

'q29_whydidyounotvote_past6','q29_whydidyounotvote_past7','q29_whydidyounotvo
te_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_whydidyounotvo
te_past3','q29_whydidyounotvote_past4','q29_whydidyounotvote_past5',

'q29_whydidyounotvote_past6','q29_whydidyounotvote_past7','q29_whydidyounotvo
te_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()

# %%-----------------------------------------------------------------------
```

```python
# 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)
```