```
def confusion_matrix(mat, title):
  '''
  mat - confusion matrix for a given model
  title - the title of the confusion matrix visualization
  '''
  class_names=[0,1] # name  of classes
  fig, ax = plt.subplots()
  tick_marks = np.arange(len(class_names))
  plt.xticks(tick_marks, class_names)
  plt.yticks(tick_marks, class_names)
  sns.heatmap(pd.DataFrame(mat), annot=True, cmap="YlGnBu" ,fmt='g')
  ax.xaxis.set_label_position("top")
  plt.tight_layout()
  plt.title(title, y=1.1)
  plt.ylabel('Actual label')
  plt.xlabel('Predicted label');


def display(results):
    print(f'Best parameters are: {results.best_params_}')
    print("\n")
    mean_score = results.cv_results_['mean_test_score']
    std_score = results.cv_results_['std_test_score']
    params = results.cv_results_['params']
    for mean,std,params in zip(mean_score,std_score,params):
        print(f'{round(mean,3)} + or -{round(std,3)} for the {params}')


def pr_curve(model, x_test, y_test, title, weight=None):
  '''
  model - the trained model
  x_test - testing data for independent variables
  y_test - testing data for depednent varaible
  title - title of the precision-recall curve plot
  weight - sample weights for the model
  '''
  y_pred_proba = model.predict_proba(x_test)[::,1]
  ap = metrics.average_precision_score(y_test, y_pred_proba, sample_weight= weight)
  precision, recall, threshold = metrics.precision_recall_curve(y_test,  y_pred_proba, sample
  plt.plot(recall,precision, label="AP="+ str(ap))
  plt.legend(loc=3)
  plt.xlabel("Recall")
  plt.ylabel("Precision")
  plt.title(title);


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn import metrics
from sklearn.preprocessing import OneHotEncoder
from sklearn.utils.class_weight import compute_sample_weight
from sklearn.ensemble import RandomForestClassifier
from sklearn.utils import resample
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn import svm
from sklearn.ensemble import BaggingClassifier
from sklearn.impute import KNNImputer
```

Question Formulation

Using historical data to train a classifer to predict turnout for each individual in the 2014 General Election.

Data Cleaning

```
voter_file = pd.read_csv("voterfile.csv")
```

```
voter_file.head()
```

	optimus_id	age	party	ethnicity	maritalstatus	dwellingtype	income	educa
0	861681	60.0	Republican	European	Married	Single Family	75k-125k	Deg
1	1084850	20.0	Independent	European	NaN	NaN	Unknown	
2	644435	28.0	Non-Partisan	European	NaN	NaN	Unknown	

```
print(f"There are {voter_file.shape[0]} rows and {voter_file.shape[1]} columns in the voter d
```

```
There are 50000 rows and 39 columns in the voter dataframe
```

```
# Remove duplicate votes
voter_file = voter_file.drop_duplicates(subset="optimus_id")
```

```
# Finding how the percentage of missing values per column
voter_file.isnull().sum().sort_values(ascending=False)/voter_file.shape[0]
```

```
donates_to_liberal_causes          0.998404
donates_to_conservative_causes     0.997114
intrst_musical_instruments_in_hh   0.991342
intrst_nascar_in_hh                0.954727
petowner_dog                       0.912319
occupationindustry                 0.836182
maritalstatus                      0.613132
dwellingtype                       0.521685
net_worth                          0.519618
home_owner_or_renter               0.476596
education                          0.448290
ethnicity                          0.103973
age                                0.000348
cd                                 0.000082
p10_precinct_turnout               0.000020
p12_precinct_turnout               0.000020
g08_precinct_turnout               0.000020
g10_precinct_turnout               0.000020
g12_precinct_turnout               0.000020
p08_precinct_turnout               0.000020
party                              0.000000
vh12g                              0.000000
income                             0.000000
dma                                0.000000
vh14p                              0.000000
vh06p                              0.000000
vh12p                              0.000000
vh10g                              0.000000
vh10p                              0.000000
vh08g                              0.000000
vh08p                              0.000000
vh06g                              0.000000
vh04g                              0.000000
vh04p                              0.000000
vh02g                              0.000000
vh02p                              0.000000
vh00g                              0.000000
vh00p                              0.000000
optimus_id                         0.000000
dtype: float64
```

There are multiple columns with more than 30% of the data missing, including donates_to_liberal_causes, donates_to_conservative_causes, intrst_musical_instruments_in_hh, intrst_nascar_in_hh... etc. In the typical context, these columns would be dropped immediately given the limited size. With those sizes, imputation methods will probably bias the sample with a small subset of data available.

```
voter_file = voter_file.drop(columns=['donates_to_liberal_causes', 'donates_to_conservative_c
                                    'intrst_musical_instruments_in_hh','intrst_nascar_in_hh',
```

```
# Mean Imputation for Age
voter_file['age'] = voter_file['age'].fillna(np.mean(voter_file['age']))
```

```
voter_file['ethnicity'] = voter_file['ethnicity'].fillna(voter_file['ethnicity'].value_counts
```

```
voter_raw = pd.get_dummies(voter_file, columns=['ethnicity', 'party', 'dma', "income"])
```

```
voter_raw
```

	optimus_id	age	maritalstatus	dwellingtype	education	cd	vh14p	vh12g	vh12
0	861681	69.0	Married	Single Family Dwelling Unit	Bach Degree - Extremely Likely	4.0	0	0	
1	1084850	20.0	NaN	NaN	NaN	2.0	0	0	
2	644435	28.0	NaN	NaN	NaN	3.0	0	0	
49999	878074	69.0	NaN		College -	2.0	0	0	

```
# Categories for party
voter_file['party'].value_counts()
```

```
Democratic              19437
Republican              16914
Non-Partisan             9404
American Independent     2282
Other                     414
Libertarian               373
Green                      33
Natural Law                 2
Name: party, dtype: int64
```

```
# Categories for dma
voter_file['dma'].value_counts()
```

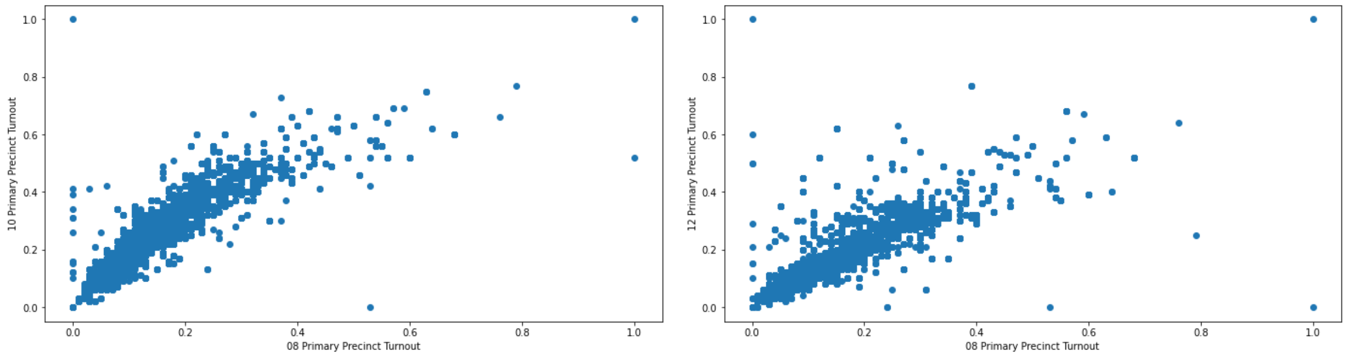
```
LAS VEGAS DMA (EST.)         34641
RENO DMA (EST.)              13253
SALT LAKE CITY DMA (EST.)      939
LOS ANGELES DMA (EST.)          26
Name: dma, dtype: int64
```

Exploratory Analysis

Let's visualize some of the relationships for precinct turnouts from 2010 and 2012 against 2008 turn out.

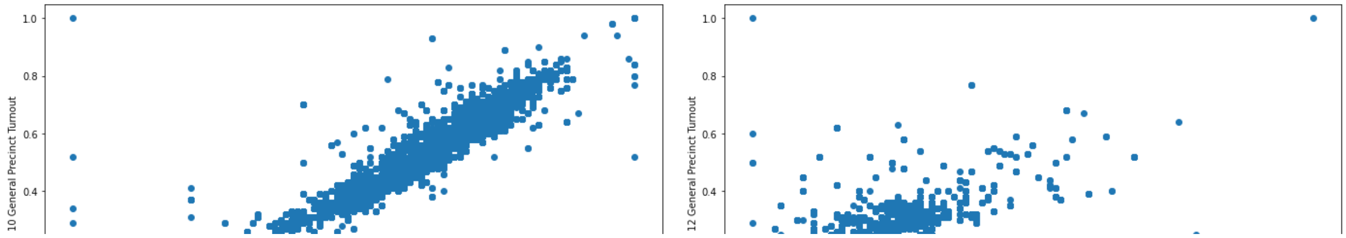
```
plt.subplots(figsize=(24,6))

plt.subplot(1, 2, 1)
plt.scatter(voter_raw['p08_precinct_turnout'], voter_raw['p10_precinct_turnout'])
plt.xlabel("08 Primary Precinct Turnout")
plt.ylabel("10 Primary Precinct Turnout")
```

```
plt.subplot(1,2, 2)
plt.scatter(voter_raw['p08_precinct_turnout'], voter_raw['p12_precinct_turnout'])
plt.xlabel("08 Primary Precinct Turnout")
plt.ylabel("12 Primary Precinct Turnout")

plt.subplots_adjust(wspace=0.1)
```



It seems like the precinct turnouts for primaries are relatively stable over the years, hence a strong positive linear relationship for both 2008 and 2010 with 2012 primaries. The trend seems to hold for general elections in those years as well. See the visualizations below:

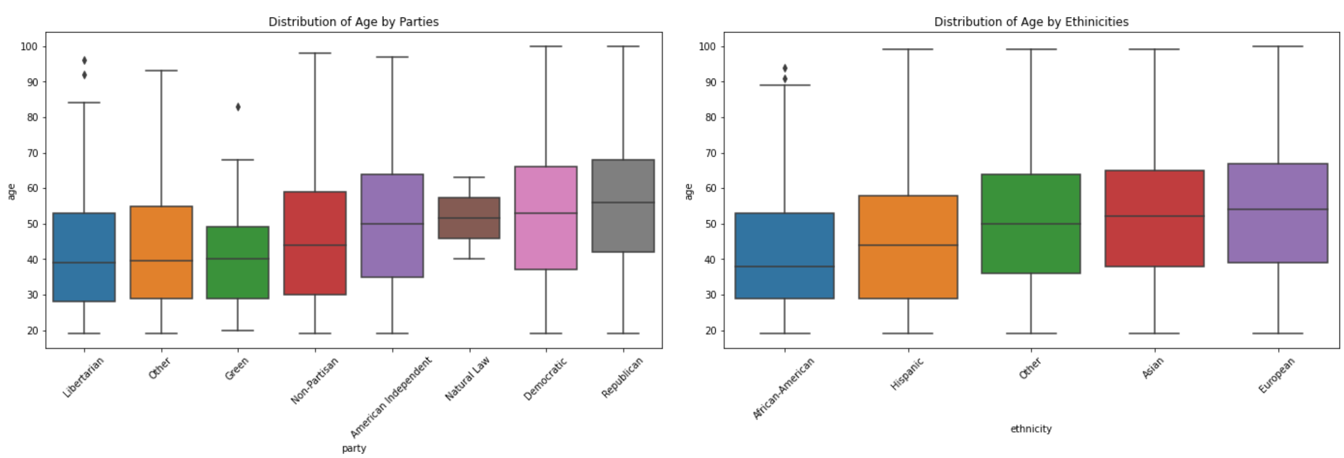
```
plt.subplots(figsize=(24,6))

plt.subplot(1, 2, 1)
plt.scatter(voter_raw['g08_precinct_turnout'], voter_raw['g10_precinct_turnout'])
plt.xlabel("08 General Precinct Turnout")
plt.ylabel("10 General Precinct Turnout")

plt.subplot(1, 2, 2)
plt.scatter(voter_raw['p08_precinct_turnout'], voter_raw['p12_precinct_turnout'])
plt.xlabel("08 General Precinct Turnout")
plt.ylabel("12 General Precinct Turnout")

plt.subplots_adjust(wspace=0.1);
```



Similarly for general elections, there is strong linear relationships between 08, 10 precinct turnout with 12's precinct turnout. It appears historical data of precinct turnout is a relatively stable factor in the election turn out.

Let's turn out attention to age group

```
plt.subplots(figsize=(24,6))

plt.subplot(1, 2, 1)
age_groups = voter_file.loc[:,['party', 'age']].groupby(['party']).median().sort_values(by='a
sns.boxplot(x=voter_file["party"], y=voter_file["age"], order=age_groups.index)
plt.title("Distribution of Age by Parties")
plt.xticks(rotation=45)

plt.subplot(1, 2, 2)
ethnic_groups = voter_file.loc[:,['ethnicity', 'age']].groupby(['ethnicity']).median().sort_v
sns.boxplot(x=voter_file['ethnicity'], y=voter_file["age"], order=ethnic_groups.index)
plt.title("Distribution of Age by Ethinicities")
plt.xticks(rotation=45)


plt.subplots_adjust(wspace=0.1);
```

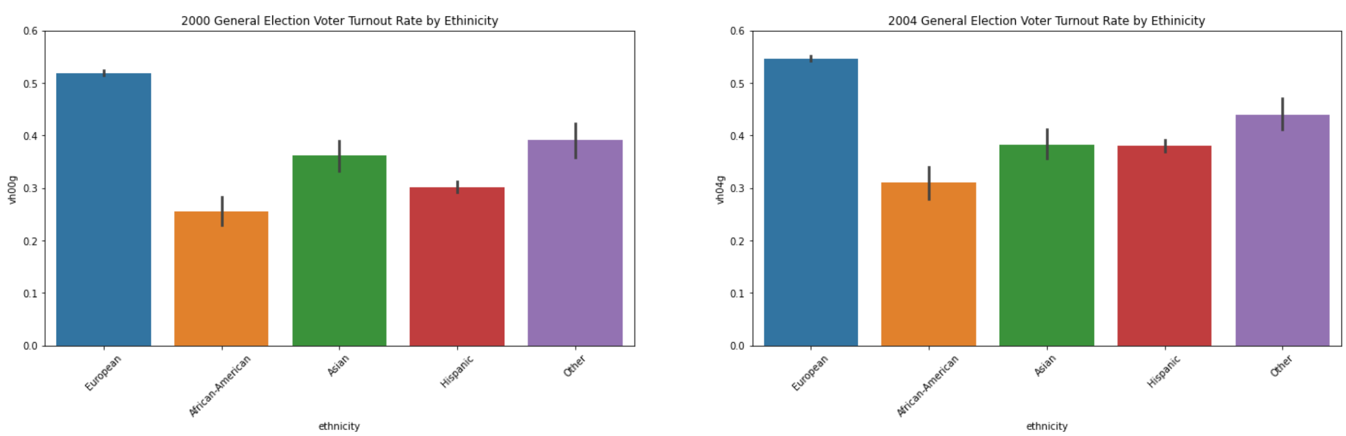


The distribution of age is relatively similar for Libertarian, Other, and Green party. Overall, the Democratic Party has a similar age distribution as Republican party, with the median age of the latter party being slightly higher.

For ethnicies by age, African-American and Hispanics have a lower distribution of age, suggesting their voting democraphic is generally younger. Whilst European, Asian. and Other Ethnic groups generally have a higher age distribution, with median age around 50.

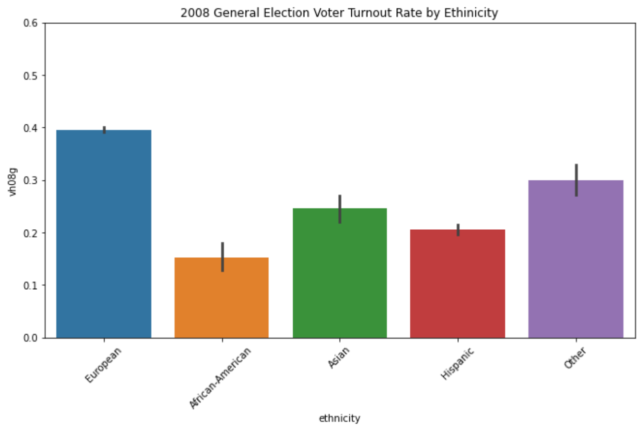
What about their general election turnout by Ethnicies?

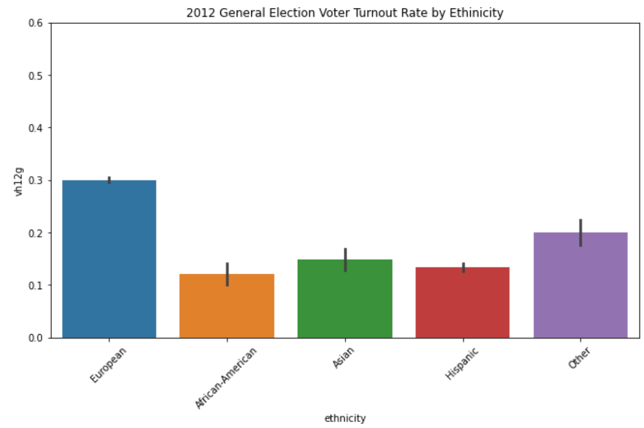
```
plt.subplots(figsize=(24,6))

plt.subplot(1, 2, 1)
sns.barplot(x=voter_file['ethnicity'], y=voter_file['vh00g'])
plt.title("2000 General Election Voter Turnout Rate by Ethinicity")
plt.xticks(rotation=45)
plt.ylim(0, 0.6)

plt.subplot(1, 2, 2)
sns.barplot(x=voter_file['ethnicity'], y=voter_file['vh04g']);
plt.title("2004 General Election Voter Turnout Rate by Ethinicity")
plt.xticks(rotation=45)
plt.ylim(0, 0.6);
```




```
plt.subplots(figsize=(24,6))
plt.subplot(1, 2, 1)
sns.barplot(x=voter_file['ethnicity'], y=voter_file['vh08g'])
plt.title("2008 General Election Voter Turnout Rate by Ethinicity")
plt.xticks(rotation=45)
```

```
plt.ylim(0, 0.6);

plt.subplot(1, 2, 2)
sns.barplot(x=voter_file['ethnicity'], y=voter_file['vh12g'])
plt.title("2012 General Election Voter Turnout Rate by Ethinicity")
plt.xticks(rotation=45)
plt.ylim(0, 0.6);
```





Across all 4 general elections, voter turnout rate by ethnicities rank as follows:

- Europeans (highest)
- Other
- Asians
- Hispanics
- Afircan-American (lowest)

What about actual turnouts? Our goal is to predict general election in 14. Let's look at the election turnouts that has the strongest correlation with the general elections in 04, 08, and 12.

```
voter_raw_numeric = voter_raw.select_dtypes(['float', 'int'])
```

```
voter_raw_numeric.corr()['vh04g'].sort_values(ascending=False)
```

```
vh04g           1.000000
vh06g           0.631368
vh08g           0.563683
vh10g           0.518312
vh04p           0.497850
vh02g           0.489522
vh00g           0.462723
vh12g           0.452752
```

```
vh08p                     0.393214
age                       0.388486
vh02p                     0.348232
vh10p                     0.337537
vh06p                     0.335827
vh00p                     0.323119
vh12p                     0.308196
vh14p                     0.266674
g10_precinct_turnout      0.258251
g08_precinct_turnout      0.246518
g12_precinct_turnout      0.235602
p10_precinct_turnout      0.235551
p08_precinct_turnout      0.209012
p12_precinct_turnout      0.205271
optimus_id                0.086996
cd                        0.008242
Name: vh04g, dtype: float64
```

Aside from the turnout beyond 04 general election, the variabales with the highest correlations are 04 primaries, 02 general, and 00 general. In this particular election, age has stronger correlations with primaries in 02 and 00.

Unfortunately precinct turnout also has a lower correlation relative to election in recent years.

```
voter_raw_numeric.corr()['vh08g'].sort_values(ascending=False)
```

```
vh08g                     1.000000
vh10g                     0.682869
vh12g                     0.618007
vh08p                     0.582530
vh06g                     0.566373
vh04g                     0.563683
vh04p                     0.490645
vh10p                     0.471273
vh12p                     0.419840
vh06p                     0.418494
vh00g                     0.413220
age                       0.400413
vh02p                     0.360621
vh14p                     0.357843
vh02g                     0.353627
vh00p                     0.346732
g10_precinct_turnout      0.255104
p10_precinct_turnout      0.246402
g08_precinct_turnout      0.244679
p08_precinct_turnout      0.226702
g12_precinct_turnout      0.220937
p12_precinct_turnout      0.213181
optimus_id                0.104074
cd                       -0.008934
Name: vh08g, dtype: float64
```

Likewise for the 08 general election. The variables with highest correlations are 08 primaries, 06 general and 04 general. This time round, 04 & 06 primary alongside 00 general have higher correlations compared to age.

```
voter_raw_numeric.corr()['vh12g'].sort_values(ascending=False)
```

```
vh12g                    1.000000
vh10g                    0.631462
vh08g                    0.618007
vh12p                    0.574946
vh08p                    0.508695
vh10p                    0.505352
vh14p                    0.461424
vh04g                    0.452752
vh04p                    0.438827
vh06g                    0.438434
vh06p                    0.387223
age                      0.371408
vh00g                    0.352246
vh00p                    0.330534
vh02p                    0.330250
vh02g                    0.279756
g10_precinct_turnout     0.227240
g08_precinct_turnout     0.221749
p10_precinct_turnout     0.219799
p08_precinct_turnout     0.208440
g12_precinct_turnout     0.190325
p12_precinct_turnout     0.185689
optimus_id               0.091274
cd                      -0.021153
Name: vh12g, dtype: float64
```

Though the order of correlation changes, the top 3 variables that correlates with 12 general is still 12 primary, 10 general and 08 general.

It seems like the closer to the target year, the higher the correlation. This makes sense since voting preferences should be similar in recent years when events that dictate voting behavior is still fresh in mind.

It appears that the closer to the recent years, the more weight is given to historic voting records.

None of the ethnicities have voter turnout rate above 0.6, but it appears that the 2000 and 2004 elections have higher turnout overall.

Modeling

Assumptions:

- Modeling to predict 2012 General Election can generalize to 2014 General Election
- Predicting consecutive general elections are more accurate than predicting primary elections in the same year
- Simpler models are better (Occam's Razor)

```
X_g12 = voter_raw.loc[:, ["vh10g", "vh08g", "vh12p", "age"]]
y_g12  = voter_raw.loc[:, "vh12g"]


X_train_g12, X_test_g12, y_train_g12, y_test_g12 = train_test_split(X_g12, y_g12, test_size=0
```

Logistic Regression

Given that data is avalable for training and testing the 2012 election general election, let's train data to predict 2012 general election. After optimizing the model, we could see how well the model generalize with selected features.

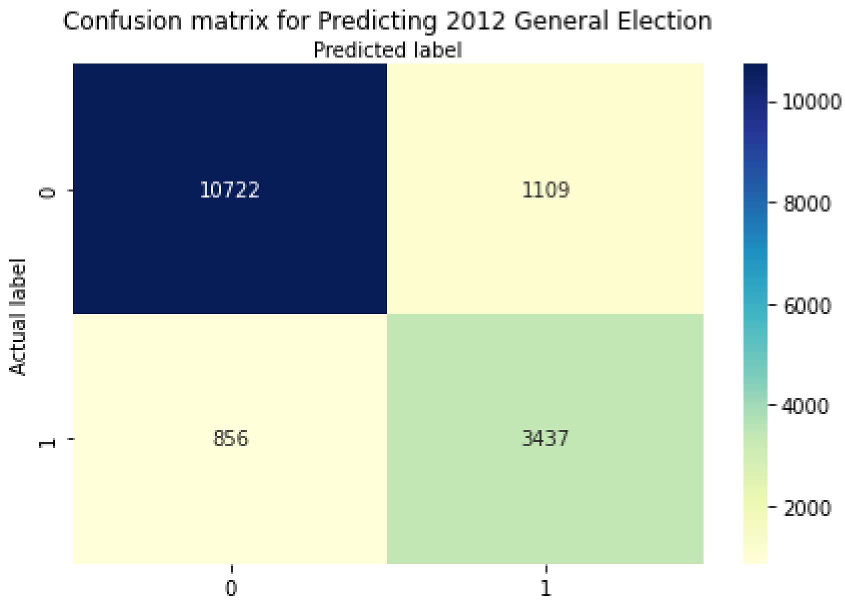
```
# Create an instance of Logistic Regression Classifier and fit the data.
logreg_g12 = LogisticRegression(C=1e5)
logreg_g12.fit(X_train_g12, y_train_g12)

# Predicting with test dataset
y_pred_g12_logreg = logreg_g12.predict(X_test_g12)
cnf_matrix_g12_logreg = metrics.confusion_matrix(y_test_g12, y_pred_g12_logreg)
print("accuracy:" + str(metrics.accuracy_score(y_test_g12, y_pred_g12_logreg)))
```

```
accuracy:0.8781319771768792
```

```
# Reports Classification Results
print(metrics.classification_report(y_test_g12, y_pred_g12_logreg,  labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.93      0.91      0.92     11831
           1       0.76      0.80      0.78      4293

    accuracy                           0.88     16124
   macro avg       0.84      0.85      0.85     16124
weighted avg       0.88      0.88      0.88     16124
```

```
# Visualizing the Confusion Matrix
confusion_matrix(cnf_matrix_g12_logreg, 'Confusion matrix for Predicting 2012 General Electio
```



There appears to be class imbalance with class 0 of the actual labels. Let's proceed with metrics such as precision, recall, f1, as well as the Precision-Recall curve.

To accomodate for class imbalance, we will undersample from the majority class (voter turnout=0) and use sample weights argument in the regression).

The following models are the final decisions.

```
# Undersampling Majority Class
voter_raw_downsample = resample(voter_raw[voter_raw['vh12g'] == 0],
                                 replace=True,
                                 n_samples=voter_raw[voter_raw['vh12g'] == 1].shape[0],
                                 random_state=123)
voter_raw_train = pd.concat([voter_raw_downsample, voter_raw[voter_raw['vh12g'] == 1]])

# Attmped Oversampling Minority Class - had worst performance
# voter_raw_upsample = resample(voter_raw[voter_raw['vh12g'] == 1],
#                                  replace=True,
#                                  n_samples=voter_raw[voter_raw['vh12g'] == 0].shape[0],
#                                  random_state=123)
# voter_raw_train = pd.concat([voter_raw_upsample, voter_raw[voter_raw['vh12g'] == 0]])

# Display new class counts
print (voter_raw_train['vh12g'].value_counts())

X_g12 = voter_raw_train.loc[:, ["vh10g", "vh08g", "vh12p", "age"]]
y_g12  = voter_raw_train.loc[:, "vh12g"]

X_train_g12, X_test_g12, y_train_g12, y_test_g12 = train_test_split(X_g12, y_g12, test_size=0
```

```
1    12900
0    12900
Name: vh12g, dtype: int64
```

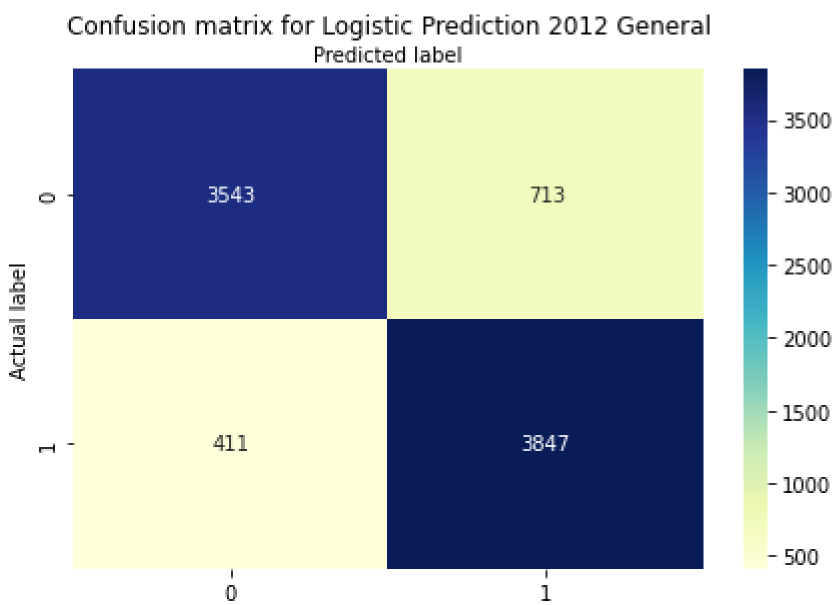
```
# Create an instance of Logistic Regression Classifier and fit the data.
logreg_g12_weighted = LogisticRegression(C=1e5,  class_weight = "balanced")
logreg_g12_weighted.fit(X_train_g12, y_train_g12)

# Predicting with test dataset directly - 5-Fold Cross Validation showed similar metrics
y_pred_g12_weighted = logreg_g12_weighted.predict(X_test_g12)
print("Logistics Regression Accuracy: " + str(metrics.accuracy_score(y_test_g12, y_pred_g12_w
```

```
     Logistics Regression Accuracy: 0.8679821470519145
```

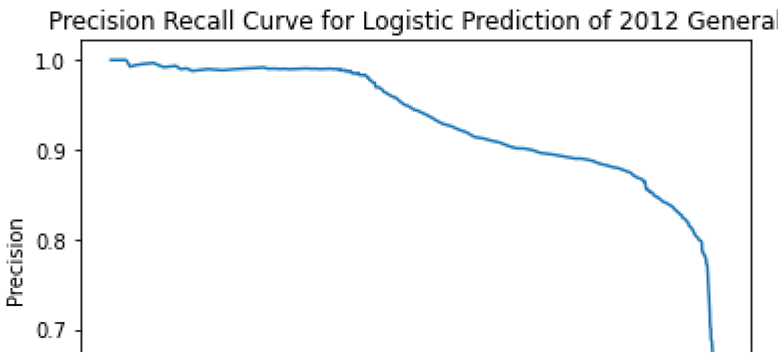
```
print(metrics.classification_report(y_test_g12, y_pred_g12_weighted,  labels=[0,1]))
```

```
                   precision    recall  f1-score   support

                0       0.90      0.83      0.86      4256
                1       0.84      0.90      0.87      4258

         accuracy                           0.87      8514
        macro avg       0.87      0.87      0.87      8514
     weighted avg       0.87      0.87      0.87      8514
```

```
confusion_matrix(metrics.confusion_matrix(y_test_g12, y_pred_g12_weighted), 'Confusion matrix
```




```
pr_curve(logreg_g12_weighted, X_test_g12, y_test_g12, "Precision Recall Curve for Logistic Pr
```



Although accuracy is slight lower, precision, recall, and F1-score for class 1 have significantly improve compared to not setting the class weight. However, one should also attempt different classification techniques to ensure the best algorithm is employed.

Random Forest Classifier

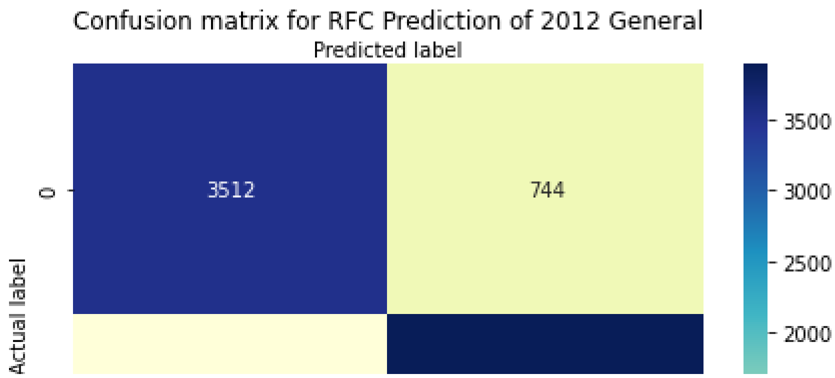
```
rfc_g12 = RandomForestClassifier(max_depth = 10, class_weight="balanced")
rfc_g12.fit(X_train_g12, y_train_g12)

# Predicting with test dataset
y_pred_g12_rfc = rfc_g12.predict(X_test_g12)

print("Random Forest Classification Accuracy: " + str(metrics.accuracy_score(y_test_g12, y_pr
```

```
     Random Forest Classification Accuracy: 0.8690392295043458
```

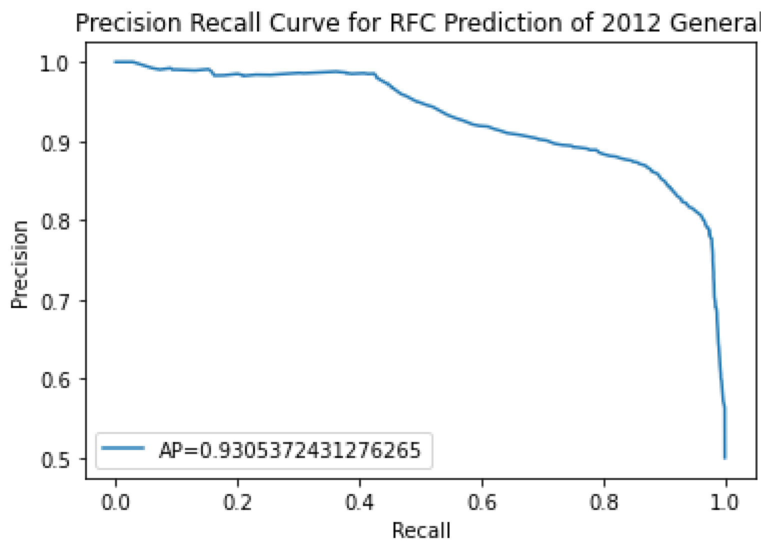
```
print(metrics.classification_report(y_test_g12, y_pred_g12_rfc,  labels=[0,1]))
```

```
                  precision    recall  f1-score   support

               0       0.90      0.83      0.86      4256
               1       0.84      0.91      0.87      4258

        accuracy                           0.87      8514
       macro avg       0.87      0.87      0.87      8514
    weighted avg       0.87      0.87      0.87      8514
```

```
confusion_matrix(metrics.confusion_matrix(y_test_g12, y_pred_g12_rfc),
                 'Confusion matrix for RFC Prediction of 2012 General')
```




```
pr_curve(rfc_g12, X_test_g12, y_test_g12, "Precision Recall Curve for RFC Prediction of 2012
```



Despite a higher accuracy, metrics such as precision, recall, and f1-score for class 1 are all inferior to the logistic regression model.

Modifications to some of the hyperparameters for the vaious models were attempted using GridSearchCV, unfortunately performance hasn't improve.

To utilize a less computational expensive tool, RandomSearchCV was also attempted but to no avail.

```
parameters = {
    "n_estimators":[5,10,50,100,250],
    "max_depth":[2,4,8,16,32,None]

}


cv = GridSearchCV(rfc_g12,parameters,cv=5)
cv.fit(X_train_g12,y_train_g12)

     GridSearchCV(cv=5, error_score=nan,
                  estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
```

```
                                                  class_weight='balanced',
                                                  criterion='gini', max_depth=10,
                                                  max_features='auto',
                                                  max_leaf_nodes=None,
                                                  max_samples=None,
                                                  min_impurity_decrease=0.0,
                                                  min_impurity_split=None,
                                                  min_samples_leaf=1,
                                                  min_samples_split=2,
                                                  min_weight_fraction_leaf=0.0,
                                                  n_estimators=100, n_jobs=None,
                                                  oob_score=False,
                                                  random_state=None, verbose=0,
                                                  warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [2, 4, 8, 16, 32, None],
                         'n_estimators': [5, 10, 50, 100, 250]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
display(cv)
```

```
Best parameters are: {'max_depth': 4, 'n_estimators': 100}


0.857 + or -0.012 for the {'max_depth': 2, 'n_estimators': 5}
0.858 + or -0.008 for the {'max_depth': 2, 'n_estimators': 10}
0.866 + or -0.006 for the {'max_depth': 2, 'n_estimators': 50}
0.864 + or -0.01 for the {'max_depth': 2, 'n_estimators': 100}
0.866 + or -0.007 for the {'max_depth': 2, 'n_estimators': 250}
0.868 + or -0.005 for the {'max_depth': 4, 'n_estimators': 5}
0.869 + or -0.005 for the {'max_depth': 4, 'n_estimators': 10}
0.869 + or -0.005 for the {'max_depth': 4, 'n_estimators': 50}
0.87 + or -0.004 for the {'max_depth': 4, 'n_estimators': 100}
0.87 + or -0.006 for the {'max_depth': 4, 'n_estimators': 250}
0.867 + or -0.005 for the {'max_depth': 8, 'n_estimators': 5}
0.867 + or -0.004 for the {'max_depth': 8, 'n_estimators': 10}
0.868 + or -0.005 for the {'max_depth': 8, 'n_estimators': 50}
0.868 + or -0.004 for the {'max_depth': 8, 'n_estimators': 100}
0.868 + or -0.004 for the {'max_depth': 8, 'n_estimators': 250}
0.865 + or -0.005 for the {'max_depth': 16, 'n_estimators': 5}
0.865 + or -0.004 for the {'max_depth': 16, 'n_estimators': 10}
0.865 + or -0.004 for the {'max_depth': 16, 'n_estimators': 50}
0.865 + or -0.004 for the {'max_depth': 16, 'n_estimators': 100}
0.866 + or -0.004 for the {'max_depth': 16, 'n_estimators': 250}
0.865 + or -0.006 for the {'max_depth': 32, 'n_estimators': 5}
0.867 + or -0.004 for the {'max_depth': 32, 'n_estimators': 10}
0.866 + or -0.004 for the {'max_depth': 32, 'n_estimators': 50}
0.866 + or -0.004 for the {'max_depth': 32, 'n_estimators': 100}
0.866 + or -0.004 for the {'max_depth': 32, 'n_estimators': 250}
0.864 + or -0.005 for the {'max_depth': None, 'n_estimators': 5}
0.866 + or -0.005 for the {'max_depth': None, 'n_estimators': 10}
0.866 + or -0.005 for the {'max_depth': None, 'n_estimators': 50}
0.865 + or -0.004 for the {'max_depth': None, 'n_estimators': 100}
0.865 + or -0.004 for the {'max_depth': None, 'n_estimators': 250}
```

Decision Tree Classifier

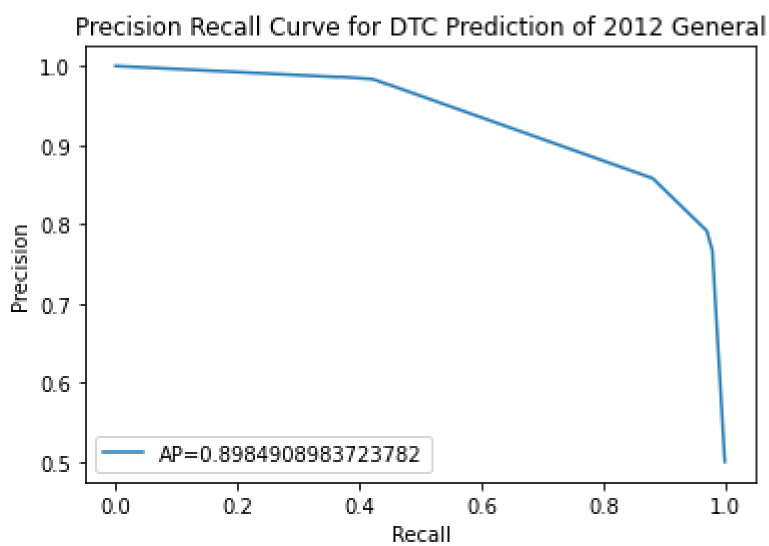
```
clf_g12 = DecisionTreeClassifier(max_depth =3, random_state = 42, class_weight="balanced")
clf_g12.fit(X_train_g12, y_train_g12)
y_pred_g12_dtc = clf_g12.predict(X_test_g12)
print("Decision Tree Classification Accuracy: " + str(metrics.accuracy_score(y_test_g12, y_pr
```

```
    Decision Tree Classification Accuracy: 0.8677472398402631
```

```
print(metrics.classification_report(y_test_g12,y_pred_g12_dtc,  labels=[0,1]))
```

```
                  precision    recall  f1-score   support

               0       0.88      0.85      0.87      4256
               1       0.86      0.88      0.87      4258

        accuracy                           0.87      8514
       macro avg       0.87      0.87      0.87      8514
    weighted avg       0.87      0.87      0.87      8514
```

```
pr_curve(clf_g12, X_test_g12, y_test_g12, "Precision Recall Curve for DTC Prediction of 2012
```



Support Vector Machines

```
svmc = svm.SVC(class_weight="balanced")
svmc.fit(X_train_g12, y_train_g12)
y_pred_g12_svmc = svmc.predict(X_test_g12)
print("SVM Classification Accuracy: " + str(metrics.accuracy_score(y_test_g12, y_pred_g12_svm
```

```
    SVM Classification Accuracy: 0.8639887244538408
```

```
print(metrics.classification_report(y_test_g12,y_pred_g12_svmc,  labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.93      0.79      0.85      4256
           1       0.82      0.94      0.87      4258

    accuracy                           0.86      8514
   macro avg       0.87      0.86      0.86      8514
weighted avg       0.87      0.86      0.86      8514
```

Bagging Classifier

```
bagc = BaggingClassifier(base_estimator=svm.SVC(),n_estimators=10, random_state=0).fit(X_trai
y_pred_g12_bagc = bagc.predict(X_test_g12)
print("Bagging Classification Accuracy: " + str(metrics.accuracy_score(y_test_g12, y_pred_g12
```

```
Bagging Classification Accuracy: 0.8639887244538408
```

```
print(metrics.classification_report(y_test_g12,y_pred_g12_bagc,  labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.93      0.79      0.85      4256
           1       0.82      0.94      0.87      4258

    accuracy                           0.86      8514
   macro avg       0.87      0.86      0.86      8514
weighted avg       0.87      0.86      0.86      8514
```

Results

Given the metrics are vastly similar across different algorithms, it is simpler to select one model.

It appears logistic regression with undersampling and sample weights has the most consistent results so far. I will train the model with training data and generate the csv file below.

```
# Create an instance of Logistic Regression Classifier and fit the data.
logreg_g12_weighted_actual = LogisticRegression(C=1e5, class_weight="balanced")
logreg_g12_weighted_actual.fit(X_g12, y_g12)
```

```
LogisticRegression(C=100000.0, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
```

```
                       max_iter=100, multi_class='auto', n_jobs=None, penalty='l2',
                       random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                       warm_start=False)
```

```
X_g12_actual = voter_raw.loc[:, ["optimus_id", "vh10g", "vh08g", "vh12p", "age"]]
```

```
X_g12_drop = X_g12_actual.drop('optimus_id', axis=1)
vh12g_prob = logreg_g12_weighted.predict_proba(X_g12_drop)[::,1]
y_pred_g12_logreg = logreg_g12_weighted.predict(X_g12_drop)
```

```
X_g12_actual['vote'] = y_pred_g12_logreg
X_g12_actual['vote_prob'] = vh12g_prob
```

```
X_g12_actual.head()
```

	optimus_id	vh10g	vh08g	vh12p	age	vote	vote_prob
0	861681	1	1	0	69.0	1	0.812878
1	1084850	0	0	0	20.0	0	0.021749
2	644435	0	0	0	28.0	0	0.025820
3	57683	0	0	0	78.0	0	0.073639
4	167371	1	1	0	68.0	1	0.809514

```
X_g12_actual.to_csv("VoteClassification_JoshuaWu.csv")
```

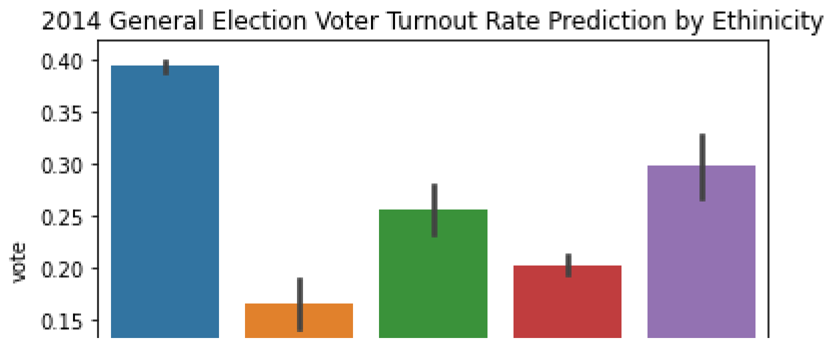
Analysis

```
final_df = X_g12_actual.merge(voter_file, left_on="optimus_id", right_on="optimus_id", how="i
```

```
final_df
```

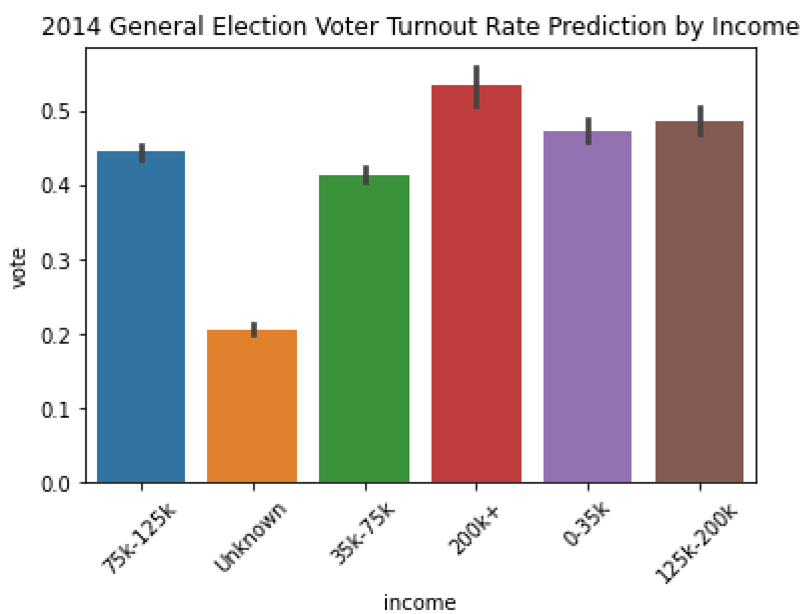
	optimus_id	vh10g_x	vh08g_x	vh12p_x	age_x	vote	vote_prob	age_y	party
0	861681	1	1	0	69.0	1	0.812878	69.0	Republican
1	1084850	0	0	0	20.0	0	0.021749	20.0	American Independent
2	644435	0	0	0	28.0	0	0.025820	28.0	Non-Partisan
3	57683	0	0	0	78.0	0	0.073639	78.0	American Independent
4	167371	1	1	0	68.0	1	0.809514	68.0	Democratic
...	...	...	...	...	...	...	...	...	...
48854	251398	0	0	0	23.0	0	0.023196	23.0	American Independent
48855	684299	0	0	0	24.0	0	0.023699	24.0	Democratic
48856	369815	0	0	0	28.0	0	0.025820	28.0	Non-Partisan

```
sns.barplot(x=final_df['ethnicity'], y=final_df['vote'])
plt.title("2014 General Election Voter Turnout Rate Prediction by Ethinicity")
plt.xticks(rotation=45);
```



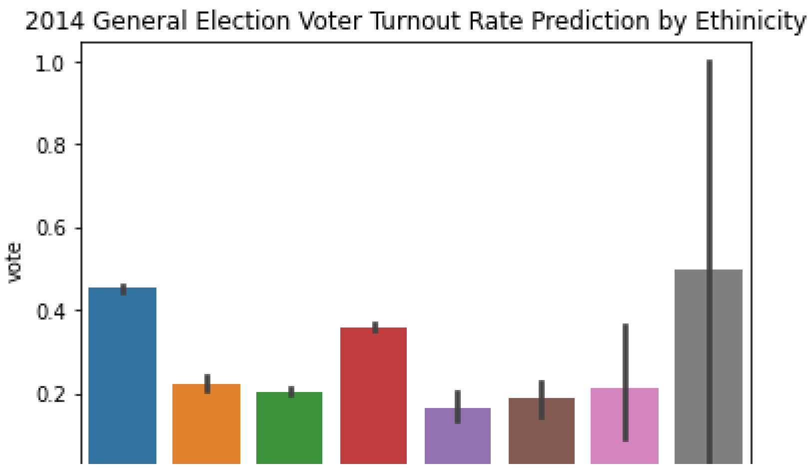
Consistent with previous years, the demographics with highest preidcted turnout rates are Europeans, Asians, and Other ethnicities.


```
sns.barplot(x=final_df['income'], y=final_df['vote'])
plt.title("2014 General Election Voter Turnout Rate Prediction by Income")
plt.xticks(rotation=45);
```



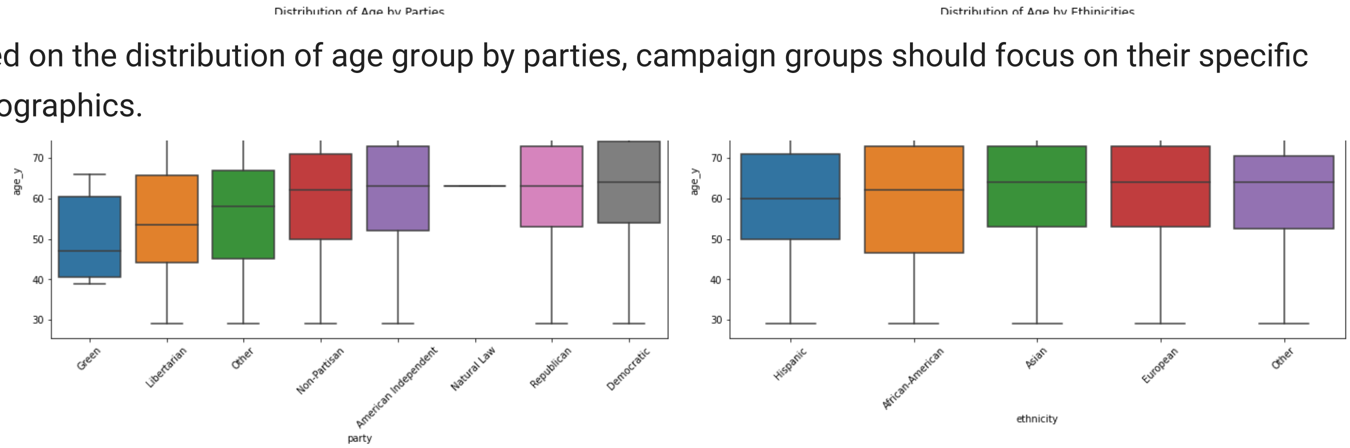
High income group (200k+) have the highest voter turnout rate, closely followed by middle-income groups (125k-200k). These should be demographics of focus in campaigns.

```
sns.barplot(x=final_df['party'], y=final_df['vote'])
plt.title("2014 General Election Voter Turnout Rate Prediction by Ethinicity")
plt.xticks(rotation=45);
```




```
will_vote = final_df[final_df['vote']==1]

plt.subplots(figsize=(24,6))

plt.subplot(1, 2, 1)
age_groups = will_vote.loc[:,['party', 'age_y']].groupby(['party']).median().sort_values(by='
sns.boxplot(x=will_vote["party"], y=will_vote["age_y"], order=age_groups.index)
plt.title("Distribution of Age by Parties")
plt.xticks(rotation=45)

plt.subplot(1, 2, 2)
ethnic_groups = will_vote.loc[:,['ethnicity', 'age_y']].groupby(['ethnicity']).median().sort_
sns.boxplot(x=will_vote['ethnicity'], y=will_vote["age_y"], order=ethnic_groups.index)
plt.title("Distribution of Age by Ethinicities")
plt.xticks(rotation=45)


plt.subplots_adjust(wspace=0.1);
```

Based on the distribution of age group by parties, campaign groups should focus on their specific demographics.



✓ 0s completed at 11:53 AM