GREAT LEARNING

Project Report

Machine Learning

Monika Nanda

5/3/2020

Contents

Problem -1

You are hired by one of the leading news channel CNBE who wants to analyse recent elections. This survey was conducted on 1525 voters with 9 variables. You have to build a model, to predict which party a voter will vote for on the basis of the given information, to create an exit poll that will help in predicting overall win and seats covered by a particular party.

1.1 Read the dataset. Do the descriptive statistics and do null value condition check. Write an inference on it.

Exploratory Data Analysis

- Dataset contains 1525 rows and 9 columns
- **Columns Names** : 'vote', 'gender', 'age', 'economic.cond.national', 'economic.cond.household', 'Blair', 'Hague', 'Europe', 'political.knowledge'
- Numeric Variable : age
- Categorical Varibles : vote, gender, economic.cond.national, economic.cond.household, Blair, Hague, Europe, political.knowledge
- There are no Null Values
- 8 Duplicate Records found. Duplicates have been dropped for the analysis. Duplicate records are as follows

	vote	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	gender
Unnamed: 0									
68	Labour	35	4	4	5	2	3	2	male
627	Labour	39	3	4	4	2	5	2	male
871	Labour	38	2	4	2	2	4	3	male
984	Conservative	74	4	3	2	4	8	2	female
1155	Conservative	53	3	4	2	2	6	0	female
1237	Labour	36	3	3	2	2	6	2	female
1245	Labour	29	4	4	4	2	2	2	female
1439	Labour	40	4	3	4	2	2	2	male

Descriptive Statistics for the dataset

In [59]:
```
elections['age'].describe().T
```

Out[59]:
```
count    1525.000000
mean       54.182295
std        15.711209
min        24.000000
25%        41.000000
50%        53.000000
75%        67.000000
max        93.000000
Name: age, dtype: float64
```

In [62]:
```
elections[cat].describe().T
```

Out[62]:

	count	unique	top	freq
vote	1525	2	Labour	1063
gender	1525	2	female	812
economic.cond.national	1525	5	3	607
economic.cond.household	1525	5	3	648
Blair	1525	5	4	836
Hague	1525	5	2	624
Europe	1525	11	11	338
political.knowledge	1525	4	2	782

1.2 Perform Univariate and Bivariate Analysis. Do exploratory data analysis. Check for Outliers

Univariate Analysis

Values and Distribution of the Categorical Data

Vote

```
Conservative    0.30323
Labour          0.69677
Name: vote, dtype: float64
```

Gender

```
male      0.46737
female    0.53263
Name: gender, dtype: float64
```

Economic.cond.national

```
1    0.024390
5    0.054054
2    0.168754
4    0.354647
3    0.398154
Name: economic.cond.national, dtype: float64
```

Economic.cond.household

```
1    0.042848
5    0.060646
2    0.184575
4    0.286750
3    0.425181
Name: economic.cond.household, dtype: float64
```

Blair

```
3    0.000659
1    0.063942
5    0.100198
2    0.286091
4    0.549110
Name: Blair, dtype: float64
```

Hague

```
3    0.024390
5    0.048121
1    0.153593
4    0.367172
2    0.406724
Name: Hague, dtype: float64
```

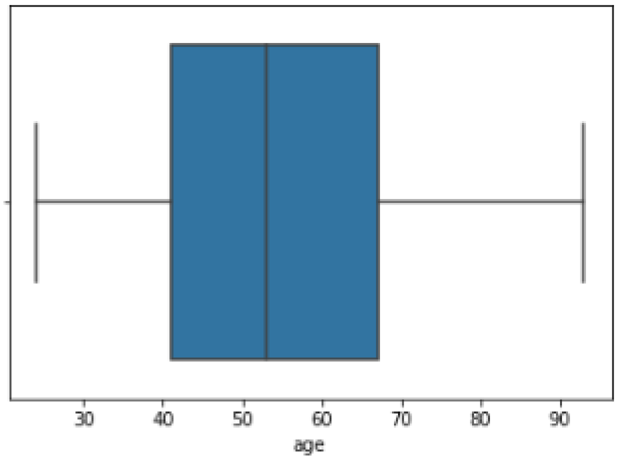
Europe

```
2     0.050758
7     0.056691
10    0.066579
1     0.071852
9     0.073171
8     0.073171
5     0.081081
4     0.083059
3     0.084377
6     0.136454
11    0.222808
Name: Europe, dtype: float64
```

Political.Knowledge

```
1    0.025049
3    0.164140
0    0.299275
2    0.511536
Name: political.knowledge, dtype: float64
```

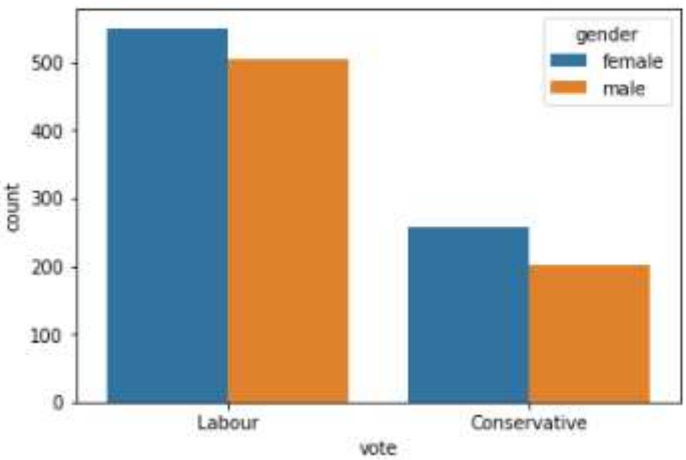
Age – Outlier Check



There are no outliers in variable age.

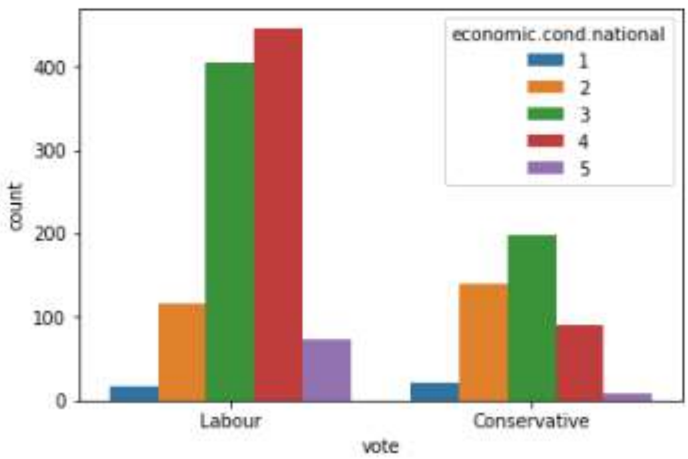
Bi-Variate Analysis

- Vote and Gender

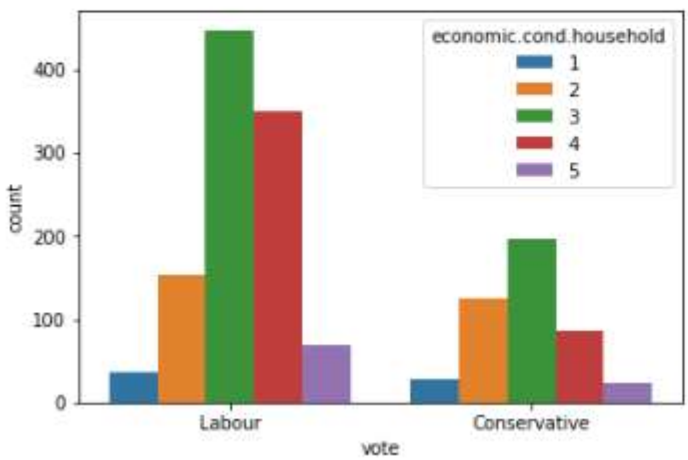


Gender is not very different for different Vote Categories

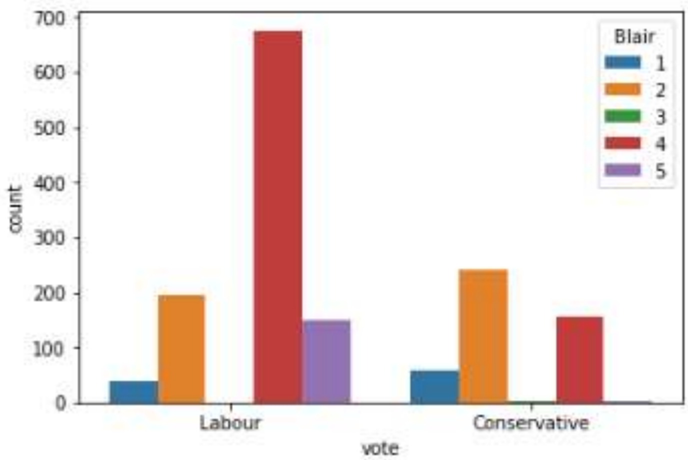
- Vote and Economic.cond.national



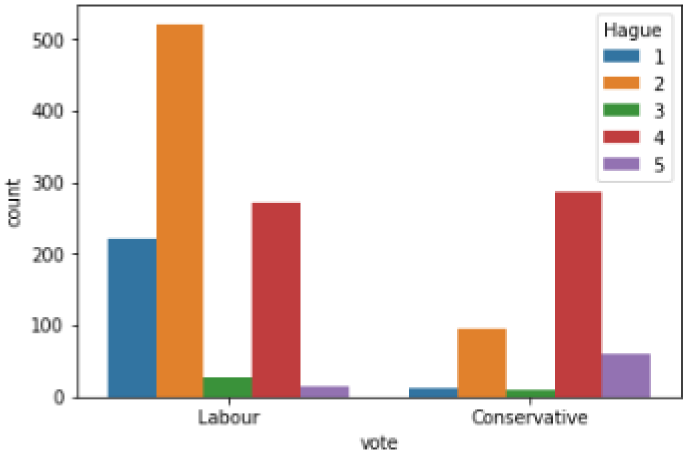
- Vote and Economic.cond.household



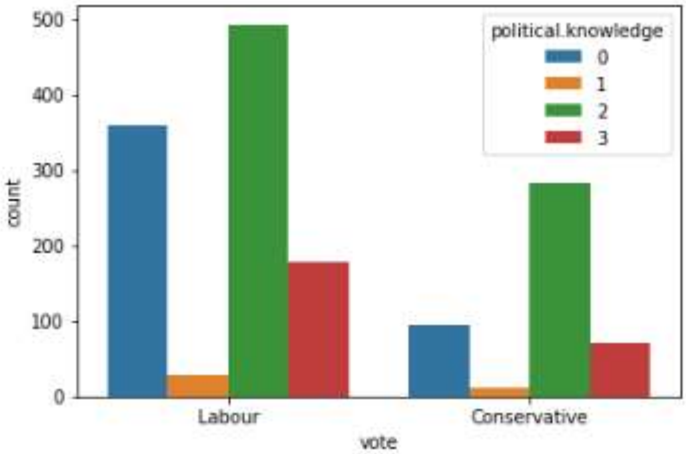
- Vote and Blair



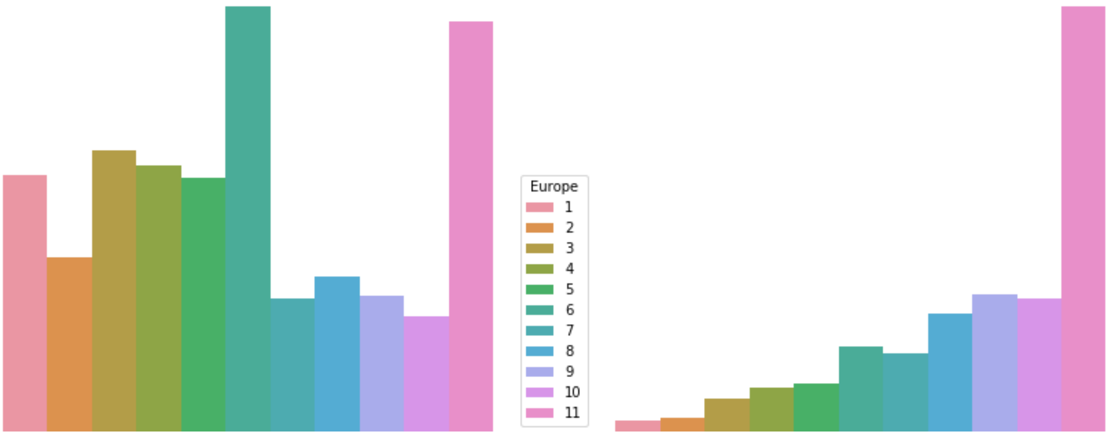
- Vote and Hague



- Vote and Political.knowledge



- Vote and Europe



1.3 Encode the data (having string values) for Modelling. Is Scaling necessary here or not? Data Split: Split the data into train and test (70:30)

Encoding the data

Vote:

Labour -> 0

Conservative -> 1

Gender:

Male -> 1

Female -> 0

```
#Target Encoding

cleanup_nums = {"vote":     {"Labour": 0, "Conservative": 1},
                "gender" : {"male":1,"female":0}}

df.replace(cleanup_nums, inplace=True)
df.vote.unique()
```

```
array([0, 1], dtype=int64)
```

```
df.gender.value_counts(normalize=True)
```

```
0    0.53263
1    0.46737
Name: gender, dtype: float64
```

Scaling is not necessary as most variables are categorical in nature and only one is numeric

Splitting the data into Train and Test (70:30)

```
# Copy all the predictor variables into X dataframe
X = df.drop('vote', axis=1)

# Copy target into the y dataframe.
y = df['vote']
```

```
# Split X and y into training and test set in 75:25 ratio
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30 , random_state=1)
```

1.4 Apply Logistic Regression and LDA (Linear Discriminant Analysis).

Logistic Regression

Performance Matrix on Train Data Set

```
0.8350612629594723
[[687  67]
 [108 199]]
              precision    recall  f1-score   support

           0       0.86      0.91      0.89       754
           1       0.75      0.65      0.69       307

    accuracy                           0.84      1061
   macro avg       0.81      0.78      0.79      1061
weighted avg       0.83      0.84      0.83      1061
```

Performance Matrix on Test Data Set

```
0.8245614035087719
[[266  37]
 [ 43 110]]
              precision    recall  f1-score   support

           0       0.86      0.88      0.87       303
           1       0.75      0.72      0.73       153

    accuracy                           0.82       456
   macro avg       0.80      0.80      0.80       456
weighted avg       0.82      0.82      0.82       456
```

Linear Discriminant Analysis

Performance Matrix on Train Data Set

```
0.8341187558906692
[[685  69]
 [107 200]]
              precision    recall  f1-score   support

           0       0.86      0.91      0.89       754
           1       0.74      0.65      0.69       307

    accuracy                           0.83      1061
   macro avg       0.80      0.78      0.79      1061
weighted avg       0.83      0.83      0.83      1061
```

Performance Matrix on Test Data Set

```
0.8333333333333334
[[269  34]
 [ 42 111]]
              precision    recall  f1-score   support

           0       0.86      0.89      0.88       303
           1       0.77      0.73      0.74       153

    accuracy                           0.83       456
   macro avg       0.82      0.81      0.81       456
weighted avg       0.83      0.83      0.83       456
```

1.5 Apply KNN Model, Naïve Bayes Model and Support Vector Machine (SVM) model.

KNN Model

```
from sklearn.neighbors import KNeighborsClassifier
KNN_model = KNeighborsClassifier()
KNN_model.fit(X_train,y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

Performance Matrix on Train Data Set

```
0.8529688972667295
[[701  53]
 [103 204]]
              precision    recall  f1-score   support

           0       0.87      0.93      0.90       754
           1       0.79      0.66      0.72       307

    accuracy                           0.85      1061
   macro avg       0.83      0.80      0.81      1061
weighted avg       0.85      0.85      0.85      1061
```

Performance Matrix on Test Data Set

```
0.8157894736842105
[[269  34]
 [ 42 111]]
              precision    recall  f1-score   support

           0       0.86      0.89      0.88       303
           1       0.77      0.73      0.74       153

    accuracy                           0.83       456
   macro avg       0.82      0.81      0.81       456
weighted avg       0.83      0.83      0.83       456
```

Naïve Bayes Model

```
from sklearn.naive_bayes import GaussianNB
NB_model = GaussianNB()
NB_model.fit(X_train, y_train)
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

Performance Matrix on Train Data Set

```
0.8350612629594723
[[675  79]
 [ 96 211]]
              precision    recall  f1-score   support

           0       0.88      0.90      0.89       754
           1       0.73      0.69      0.71       307

    accuracy                           0.84      1061
   macro avg       0.80      0.79      0.80      1061
weighted avg       0.83      0.84      0.83      1061
```

Performance Matrix on Test Data Set

```
0.8223684210526315
[[263  40]
 [ 41 112]]
              precision    recall  f1-score   support

           0       0.87      0.87      0.87       303
           1       0.74      0.73      0.73       153

    accuracy                           0.82       456
   macro avg       0.80      0.80      0.80       456
weighted avg       0.82      0.82      0.82       456
```

Support Vector Machine (SVM) Model

```
from sklearn import svm

clfSVM = svm.SVC(random_state=1)
clfSVM.fit(X_train, y_train)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=1, shrinking=True, tol=0.001,
    verbose=False)
```

Performance Matrix on Train Data Set

```
0.7323279924599434
[[751   3]
 [281  26]]
              precision    recall  f1-score   support

           0       0.73      1.00      0.84       754
           1       0.90      0.08      0.15       307

    accuracy                           0.73      1061
   macro avg       0.81      0.54      0.50      1061
weighted avg       0.78      0.73      0.64      1061
```

Performance Matrix on Test Data Set

```
0.6864035087719298
[[301   2]
 [141  12]]
              precision    recall  f1-score   support

           0       0.68      0.99      0.81       303
           1       0.86      0.08      0.14       153

    accuracy                           0.69       456
   macro avg       0.77      0.54      0.48       456
weighted avg       0.74      0.69      0.59       456
```

1.6 Model Tuning, Bagging and Boosting.

Using Random Forest Classifier for Bagging

Random Forest Classifier (Before Tuning)

Performance Matrix on Train Data Set

```
1.0
[[754   0]
 [  0 307]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       754
           1       1.00      1.00      1.00       307

    accuracy                           1.00      1061
   macro avg       1.00      1.00      1.00      1061
weighted avg       1.00      1.00      1.00      1061
```

Performance Matrix on Test Data Set

```
0.8289473684210527
[[276  27]
 [ 51 102]]
              precision    recall  f1-score   support

           0       0.84      0.91      0.88       303
           1       0.79      0.67      0.72       153

    accuracy                           0.83       456
   macro avg       0.82      0.79      0.80       456
weighted avg       0.83      0.83      0.82       456
```

This is an overfit model. Hence, tuning the model for better performance using GridSearchCV.

Performance Matrix on Train Data Set After Tuning the Model

```
0.8680490103675778
[[703  51]
 [ 89 218]]
              precision    recall  f1-score   support

           0       0.89      0.93      0.91       754
           1       0.81      0.71      0.76       307

    accuracy                           0.87      1061
   macro avg       0.85      0.82      0.83      1061
weighted avg       0.87      0.87      0.87      1061
```

Performance Matrix on Test Data Set After Tuning the Model

```
0.8245614035087719
[[274  29]
 [ 51 102]]
              precision    recall  f1-score   support

           0       0.84      0.90      0.87       303
           1       0.78      0.67      0.72       153

    accuracy                           0.82       456
   macro avg       0.81      0.79      0.80       456
weighted avg       0.82      0.82      0.82       456
```

Boosting – XGBoost

Performance Matrix on Train Data Set

```
0.8878416588124411
[[703  51]
 [ 68 239]]
              precision    recall  f1-score   support

           0       0.91      0.93      0.92       754
           1       0.82      0.78      0.80       307

    accuracy                           0.89      1061
   macro avg       0.87      0.86      0.86      1061
weighted avg       0.89      0.89      0.89      1061
```

Performance Matrix on Test Data Set

```
0.8267543859649122
[[270  33]
 [ 46 107]]
              precision    recall  f1-score   support

           0       0.85      0.89      0.87       303
           1       0.76      0.70      0.73       153

    accuracy                           0.83       456
   macro avg       0.81      0.80      0.80       456
weighted avg       0.82      0.83      0.82       456
```

Boosting – AdaBoost

Performance Matrix on Train Data Set

```
0.8501413760603205
[[688  66]
 [ 93 214]]
              precision    recall  f1-score   support

           0       0.88      0.91      0.90       754
           1       0.76      0.70      0.73       307

    accuracy                           0.85      1061
   macro avg       0.82      0.80      0.81      1061
weighted avg       0.85      0.85      0.85      1061
```

Performance Matrix on Test Data Set

```
0.8135964912280702
[[268  35]
 [ 50 103]]
              precision    recall  f1-score   support

           0       0.84      0.88      0.86       303
           1       0.75      0.67      0.71       153

    accuracy                           0.81       456
   macro avg       0.79      0.78      0.79       456
weighted avg       0.81      0.81      0.81       456
```

1.7 Performance Metrics: Check the performance of Predictions on Train and Test sets using Accuracy, Confusion Matrix, Plot ROC curve and get ROC_AUC score for each model. Final Model: Compare the models and write inference which model is best/optimized.

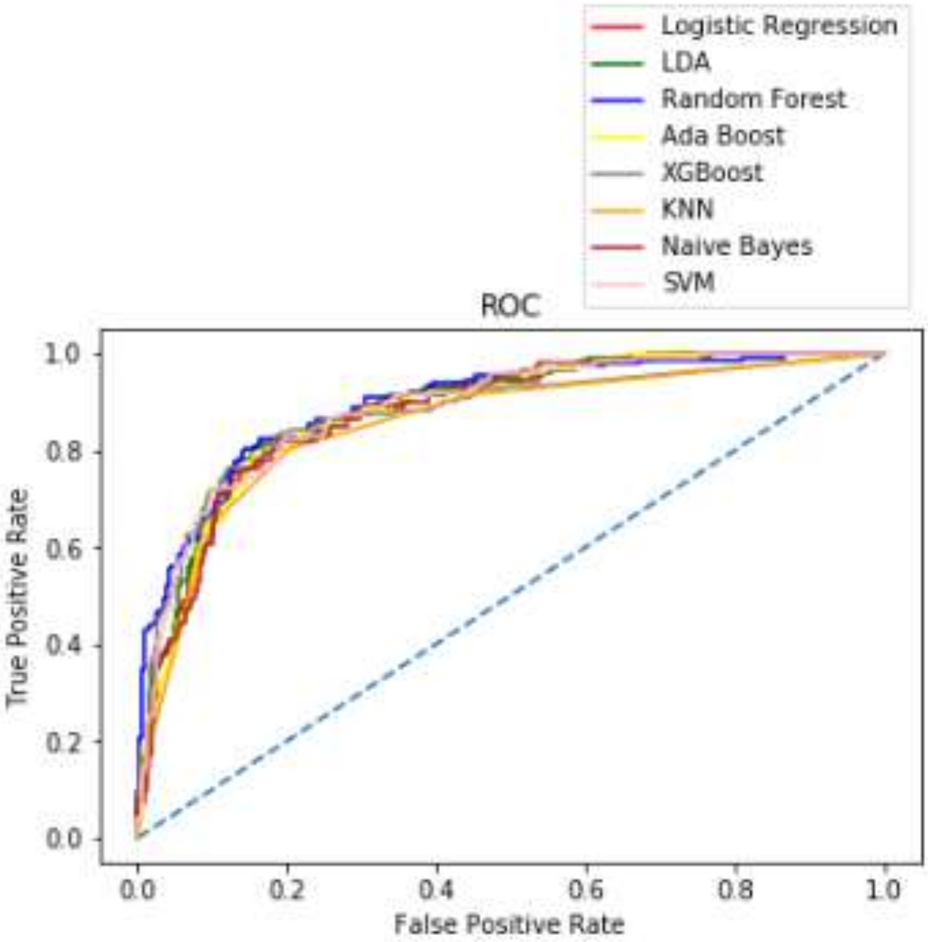
Performance Evaluation - Train Set

Model	Accuracy	Precision	Recall	F1-Score	AUC
Logistic Regression	84	83	84	83	89
LDA	83	83	83	83	88.9
KNN	85	85	85	85	92.3
Naïve Bayes	84	83	84	83	88.8
SVM	73	78	73	64	87.8
Random Forest	87	87	87	87	93.5
XGBoost	89	89	89	89	94.0
AdaBoost	85	85	85	85	91.5

Performance Evaluation - Test Set

Model	Accuracy	Precision	Recall	F1-Score	AUC
Logistic Regression	82	82	82	82	87.9
LDA	83	83	83	83	88.8
KNN	82	81	82	81	85.2
Naïve Bayes	82	82	82	82	87.6
SVM	69	74	69	59	88.6
Random Forest	82	82	82	82	89.5
XGBoost	83	82	83	82	87.3
AdaBoost	81	81	81	81	87.7

From the above comparison, we can say that except for SVM, all other models perform well for our problem. Since, the accuracy, recall and precision is more than 80% for all models except SVM, we can say that these are good models and can be used for predictions.



Cross validation Scores for LDA Model:

```
array([0.79439252, 0.77358491, 0.83962264, 0.85849057, 0.85849057,
       0.8490566 , 0.80188679, 0.8490566 , 0.81132075, 0.82075472])
```

Cross Validation Scores for Naïve Bayes

```
array([0.80373832, 0.78301887, 0.8490566 , 0.83962264, 0.90566038,
       0.8490566 , 0.78301887, 0.83962264, 0.81132075, 0.82075472])
```

Cross validation Scores for Random Forest Classifier

```
array([0.8411215 , 0.82075472, 0.83962264, 0.83962264, 0.90566038,
       0.83962264, 0.81132075, 0.86792453, 0.80188679, 0.82075472])
```

Cross validation Scores for AdaBoost

```
array([0.80373832, 0.79245283, 0.83018868, 0.85849057, 0.89622642,
       0.81132075, 0.78301887, 0.83018868, 0.78301887, 0.8490566 ])
```

Cross validation Scores for XGGBoost

```
array([0.8411215 , 0.82075472, 0.8490566 , 0.83962264, 0.86792453,
       0.83962264, 0.85849057, 0.81132075, 0.75471698, 0.78301887])
```

Cross validation Scores for KNN Model

```
array([0.80373832, 0.78301887, 0.76415094, 0.80188679, 0.82075472,
       0.81132075, 0.79245283, 0.82075472, 0.77358491, 0.77358491])
```

Final Model: Linear Discriminant Analysis

Conclusion: Final Model Chosen for this problem is Linear Discriminant Analysis as this is the most consistent model and the scores between train and test dataset do not vary much.

1.8 Based on these predictions, what are the insights?

- The coefficient for age is 0.020037048856610378
- The coefficient for economic.cond.national is -0.6049204499917713
- The coefficient for economic.cond.household is -0.050069046956978766
- The coefficient for Blair is -0.7424003897819802
- The coefficient for Hague is 0.9266343785776768
- The coefficient for Europe is 0.22361192469849644
- The coefficient for political.knowledge is 0.430334842433205
- The coefficient for gender is -0.14907997566596093

INFERENCE:

1. Voters with high value of economic.cond.national are more likely to vote for “Labour” than “Conservative”
2. Voters with high value of Blair are more likely to vote for “Labour” than “Conservative”
3. Voters with high value of Hague are more likely to vote for “Conservative”
4. Political Knowledge is a good predictor. Voters with no political knowledge are more likely to vote for “Labour”
5. Voters with ‘Eurosceptic’ sentiment (i.e. high value of variable Europe) are more likely to vote for “Conservative”
6. Age and economic.cond.household are not very good predictors of vote.

Problem 2: In this particular project, we are going to work on the inaugural corpora from the nltk in Python. We will be looking at the following speeches of the Presidents of the United States of America:

President Franklin D. Roosevelt in 1941

President John F. Kennedy in 1961

President Richard Nixon in 1963

2.1 Find the number of characters, words and sentences for the mentioned documents

Number of Characters:

- Number of Characters in **'1941-Roosevelt.txt'**: 7571
- Number of Characters in **'1961-Kennedy.txt'**: 7618
- Number of Characters in **'1973-Nixon.txt'**: 9991

Number of Words:

- Number of Words in **'1941-Roosevelt.txt'**: 1345
- Number of Words in **'1961-Kennedy.txt'**: 1368
- Number of Words in **'1973-Nixon.txt'** : 1805

Number of Sentences:

- No of Sentences in **'1941-Roosevelt.txt'**: 68
- No of Sentences in **'1961-Kennedy.txt'**: 52
- No of Sentences in **'1973-Nixon.txt'**: 68

2.2 Remove all the stopwords from the three speeches.

Removing Stopwords using nltk.corpus library

Stopwords included:

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

```
Roosevelt_token = regexp_tokenize(Roosvelt.lower(), "[\w']+")
```

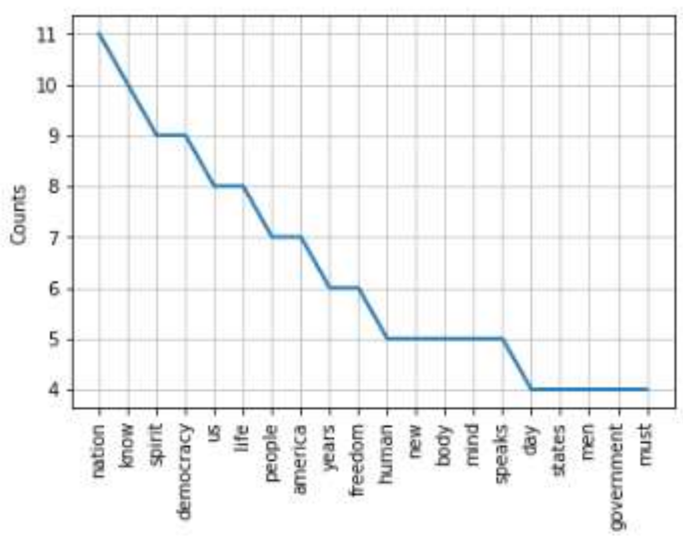
```
filtered_Roosevelt = [word for word in Roosevelt_token if not word in stop]
```

```
Kennedy_token = regexp_tokenize(Kennedy.lower(), "[\w']+")
filtered_Kennedy = [word for word in Kennedy_token if not word in stop]
```

```
Nixon_token = regexp_tokenize(Nixon.lower(), "[\w']+")
filtered_Nixon = [word for word in Nixon_token if not word in stop]
```

2.3 Which word occurs the most number of times in his inaugural address for each president? Mention the top three words. (after removing the stopwords)

Frequency Distribution of words in '1941-Roosevelt.txt' after removing stop words:



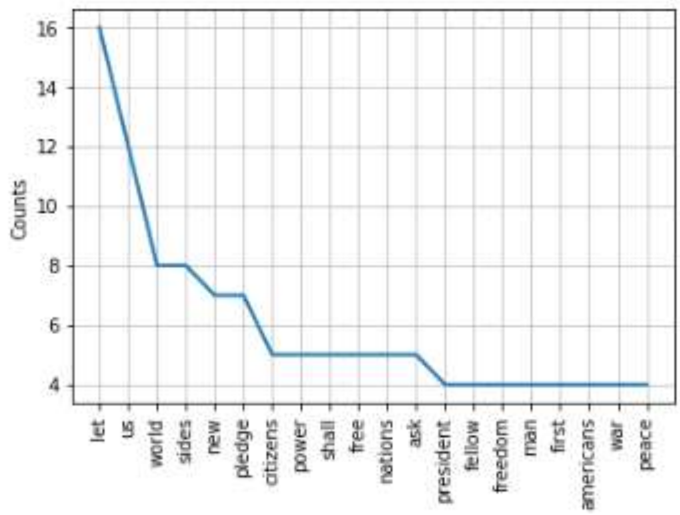
Most Common Words:

```
[('nation', 11), ('know', 10), ('spirit', 9), ('democracy', 9), ('us', 8)]
```

Top Three Words:

1. Nation
2. Know
3. Spirit

Frequency Distribution of words in '1961-Kennedy.txt' after removing stop words:



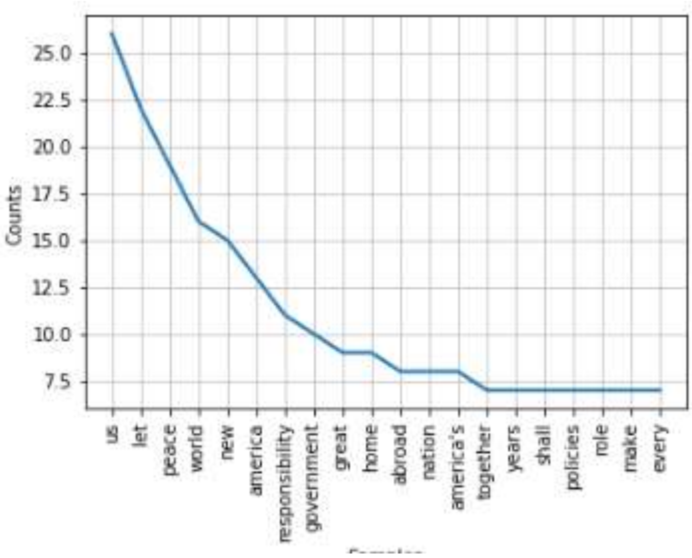
Most Common Words:

```
[('let', 16), ('us', 12), ('world', 8), ('sides', 8), ('new', 7)]
```

Top Three Words:

1 Let
2 Us
3 World

Frequency Distribution of words in '1973-Nixon.txt' after removing stop words:



Most Common Words:

```
[('us', 26), ('let', 22), ('peace', 19), ('world', 16), ('new', 15)]
```

Top Three Words:

1 Us
2 Let
3 Peace

2.4 Plot the word cloud of each of the three speeches. (after removing the stopwords)

Word Cloud - '1941-Roosevelt.txt'

Word Cloud - ‘1961-Kennedy.txt’

Word Cloud - ‘1973-Nixon.txt’