ASHWIN KUMAR A G
PGDSBA – NOV'20

TABLE OF CONTENTS

Problem 1:

Problem 2:

Problem 1:

You are hired by one of the leading news channels CNBE who wants to analyze 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. Describe the data briefly. Interpret the inferences for each. Initial steps like head() .info(), Data Types, etc . Null value check, Summary stats, Skewness must be discussed.

Dataset Background:

To create an exit poll on behalf of news channel CNBE that will help in predicting overall win and seats covered by a particular party: 'Conservative' or 'Labour' Party.

Here the classification will be done using different classification models and finally these models are to be compared.

Inferences from the Data Dictionary:

There are 9 variables out of which the 'vote' variable is the dependent variable and the rest are independent variables.

Among the independent variables, except for 'age' variable which is continuous, all the other seven variables are categorical in nature.

Among the categorical independent variables, except for 'gender' all the others are ordered categorical variables.

EXPLORATORY DATA ANALYSIS:

IMPORTING THE DATASET

	Unnamed: 0	vote	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	gender
0	1	Labour	43	3	3	4	1	2	2	female
1	2	Labour	36	4	4	4	4	5	2	male
2	3	Labour	35	4	4	5	2	3	2	male
3	4	Labour	24	4	2	2	1	4	0	female
4	5	Labour	41	2	2	1	1	6	2	male

SHAPE OF THE DATASET:

The Election Dataset has 1525 Rows & 10 Columns Initially

We have remove the Column (Unnamed:0) which is not required for the Classification Models.

Hence,the Shape of the Dataset after treatment is 1525 Rows & 9 Columns. `(1525, 9)`

INFORMATION OF THE DATASET:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1525 entries, 0 to 1524
Data columns (total 9 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   vote                    1525 non-null   object
 1   age                     1525 non-null   int64
 2   economic.cond.national  1525 non-null   int64
 3   economic.cond.household 1525 non-null   int64
 4   Blair                   1525 non-null   int64
 5   Hague                   1525 non-null   int64
 6   Europe                  1525 non-null   int64
 7   political.knowledge     1525 non-null   int64
 8   gender                  1525 non-null   object
dtypes: int64(7), object(2)
memory usage: 107.4+ KB
```

Out of the 9 columns: 7 are 'int64' and 2 are 'object' data type.

CHECKING FOR MISSING VALUES & DUPLICATE ROWS IN THE DATASET:

	vote	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	gender
67	Labour	35	4	4	5	2	3	2	male
626	Labour	39	3	4	4	2	5	2	male
870	Labour	38	2	4	2	2	4	3	male
983	Conservative	74	4	3	2	4	8	2	female
1154	Conservative	53	3	4	2	2	6	0	female
1236	Labour	36	3	3	2	2	6	2	female
1244	Labour	29	4	4	4	2	2	2	female
1438	Labour	40	4	3	4	2	2	2	male

INFERENCES:

There are no Null values present in the dataset.

There are 8 Duplicated instances.

These are voter attributes and two or more voters with the all the attributes does not seem to be logical and hence they are considered to be duplicates and are dropped.

```
df.isnull().sum().sum()
```

```
0
```

```
df.drop_duplicates(inplace=True)
df.duplicated().sum()
```

```
0
```

Descriptive Statistics for the Dataset:

	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge
count	1525.000000	1525.000000	1525.000000	1525.000000	1525.000000	1525.000000	1525.000000
mean	54.182295	3.245902	3.140328	3.334426	2.746885	6.728525	1.542295
std	15.711209	0.880969	0.929951	1.174824	1.230703	3.297538	1.083315
min	24.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000
25%	41.000000	3.000000	3.000000	2.000000	2.000000	4.000000	0.000000
50%	53.000000	3.000000	3.000000	4.000000	2.000000	6.000000	2.000000
75%	67.000000	4.000000	4.000000	4.000000	4.000000	10.000000	2.000000
max	93.000000	5.000000	5.000000	5.000000	5.000000	11.000000	3.000000

INFERENCES:

'age' variable is the only integer column ; the mean and median values are nearly same which is the only integer column in the dataset.

'vote' have two unique values Labour and Conservative, which is also a dependent variable.

'gender' has two unique values male and female.

```
Labour          1057
Conservative     460
Name: vote, dtype: int64
```

```
female    808
male      709
Name: gender, dtype: int64
```

SKEWNESS:

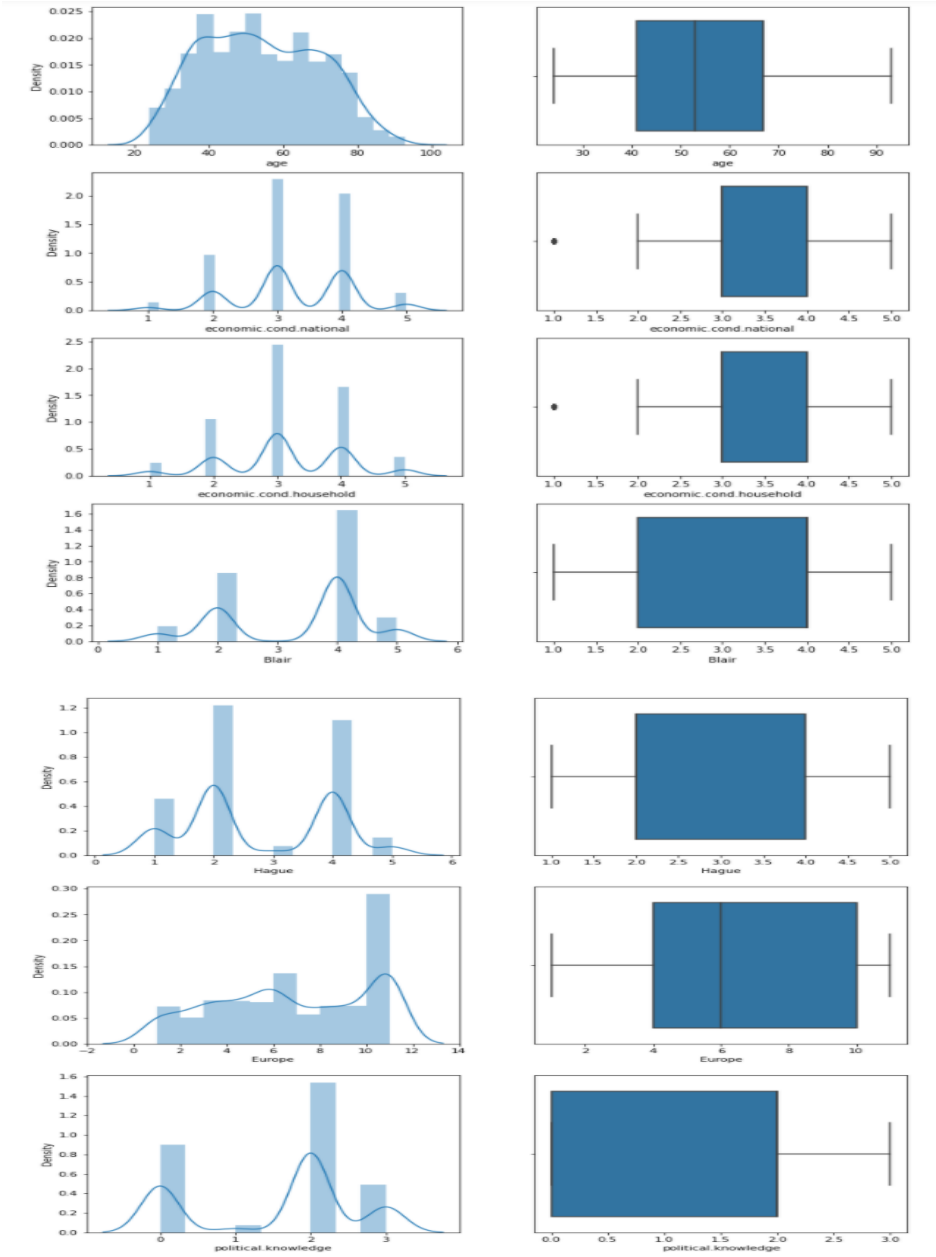
There is not much skewness observed in the variables.

```
Hague                        0.146191
age                          0.139800
gender                       0.130929
Europe                      -0.141891
economic.cond.household     -0.144148
economic.cond.national      -0.238474
political.knowledge         -0.422928
Blair                       -0.539514
vote                        -0.857014
dtype: float64
```

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

EDA:

Univariate Analysis:

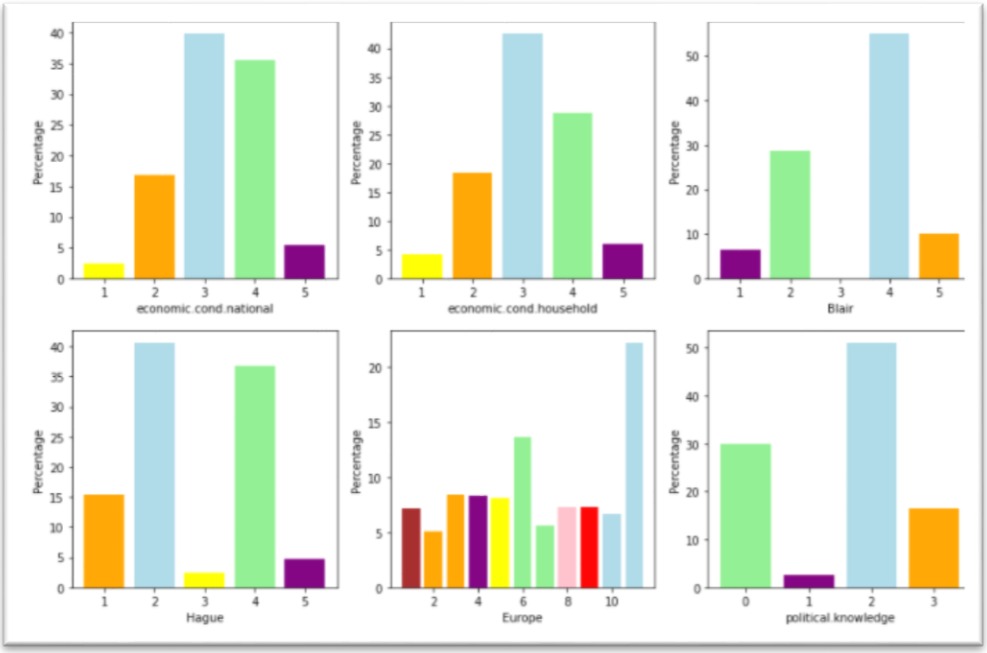


- ***In the 'age' variable it is observed that voters of all ages are taken in the survey with nearly equal representation of the different age groups. It may be noted that the number of higher and lower aged voters is comparatively less. There does not seem to be any skewness in the distribution.***
- ***For 'economic.cond.national' and 'economic.cond.household' variables it is observed that most surveyed voters have given an average ratings of 3 (or) 4.***
- ***'age' is the only integer variable and it is not having outliers. Also, the dist. plot shows that the variable is normally distributed.***

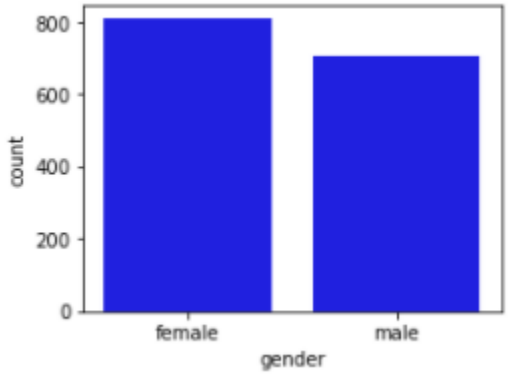
- *political.knowledge' variable shows that most surveyed voters have knowledge about the two parties' positions*
- *It is observed that all the ratings of 1 in 'economic.cond.national' and 'economic.cond.household' variables are considered as outliers . These values are not replaced and kept as it is.*

Categorical Variables:

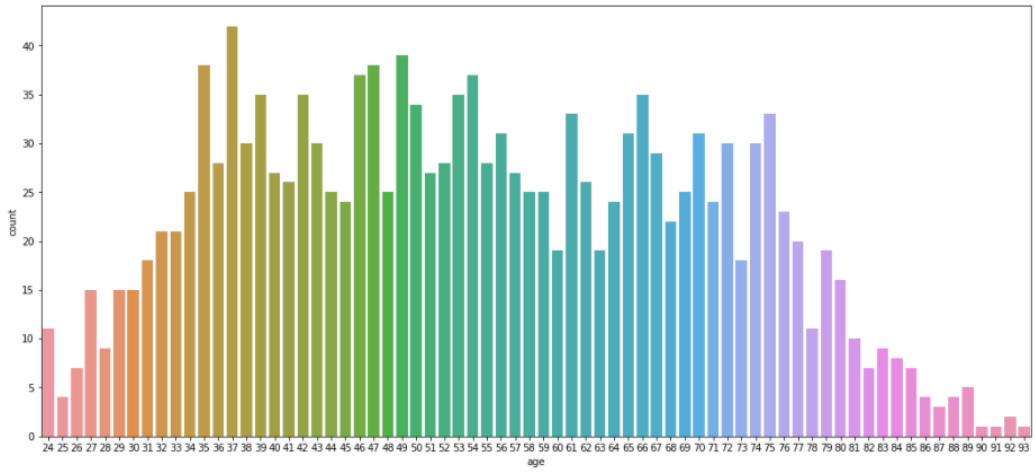
Below Plot demonstrates the % counts in each of the category



- *For 'Blair' of the Labour Party most surveyed voters gave a rating of 4 and above and for 'Hague' of the Conservative party most voters gave a rating of 2 or below.*
- *Most voters rated 3 or above for the national economic condition and economic condition of household.*
- *Most of the surveyed voters have some idea about the Labour and Conservative Party's positions*

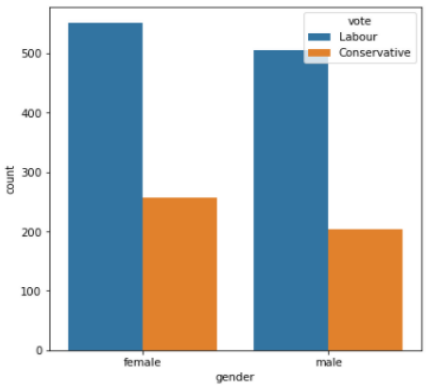


Number of Female Voters are large than Male

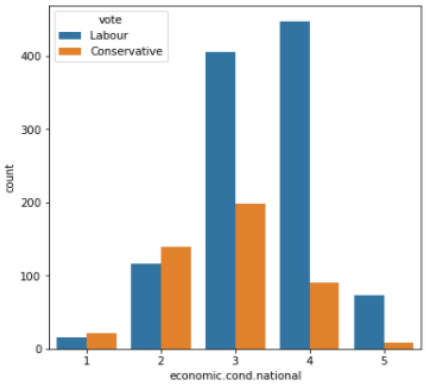


BIVARIATE ANALYSIS/MULTIVARIATE ANALYSIS:

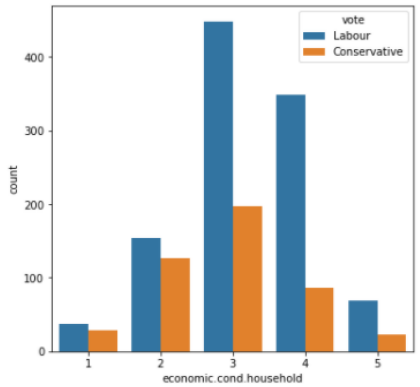
TARGET VARIABLE: 'vote'



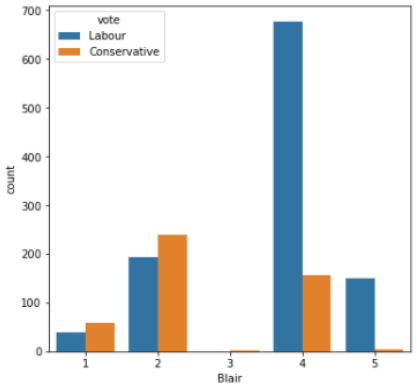
gender	female	male
vote		
Conservative	31.81	28.63
Labour	68.19	71.37



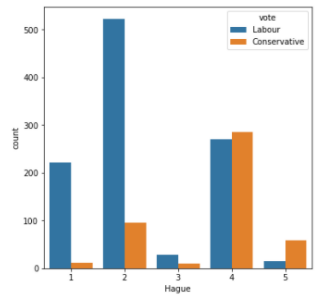
economic.cond.national	1	2	3	4	5
vote					
Conservative	56.76	54.69	32.95	16.91	10.98
Labour	43.24	45.31	67.05	83.09	89.02



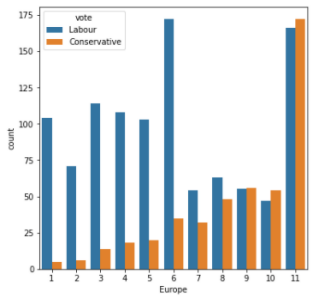
economic.cond.household	1	2	3	4	5
vote					
Conservative	43.08	45.0	30.54	19.77	25.0
Labour	56.92	55.0	69.46	80.23	75.0



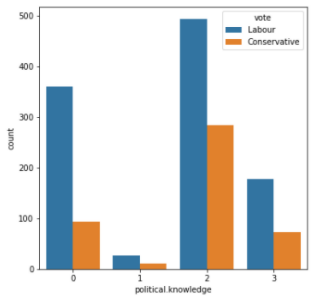
Blair	1	2	3	4	5
vote					
Conservative	60.82	55.3	100.0	18.85	1.97
Labour	39.18	44.7	0.0	81.15	98.03



Hague	1	2	3	4	5
vote					
Conservative	4.72	15.4	24.32	51.35	80.82
Labour	95.28	84.6	75.68	48.65	19.18



political.knowledge	0	1	2	3
vote				
Conservative	20.7	28.95	36.47	28.92
Labour	79.3	71.05	63.53	71.08

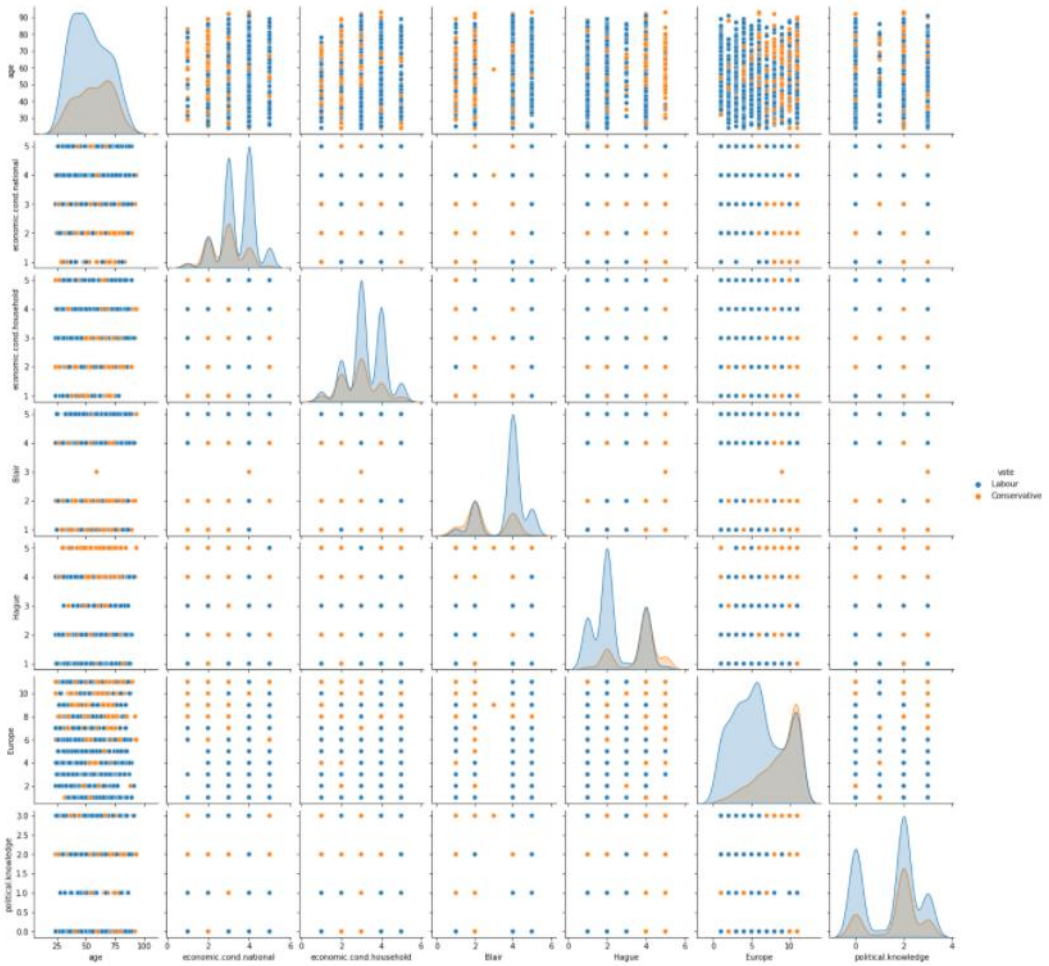


political.knowledge	0	1	2	3
vote				
Conservative	20.7	28.95	36.47	28.92
Labour	79.3	71.05	63.53	71.08

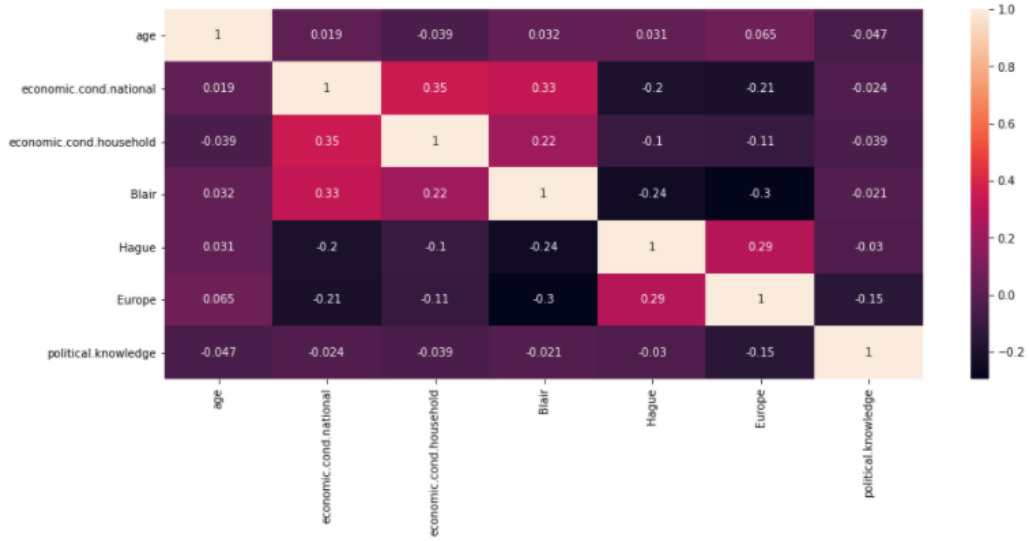
- ***Inferences:***
- ***No special observation can be made as the overall percentage of votes in favour of either Party remain same even when the gender of voters are considered.***
- ***Voters who have given higher ratings for current economic condition favour the Labour Party more.***

- ***Voters who have given higher ratings for current household economic condition favour the Labour Party more.***
- ***Voters who have given higher ratings for Blair favour the Labour Party more.***
- ***Voters who have given ratings of 3 or less for Hague are more in favour of the Labour party. Voters who have given a rating of 4 for Hague seem to be equally in favour of either parties, and voters who have given a rating of 5 for Hague seem to favour the Conservative party more.***
- ***For the lower ratings it is observed that the percentage of voters is much higher for Labour Party.Overall it seems to favour the Labour Party more.***

PAIRPLOT:



HEATMAP:



The pairplot and the correlation values between the numerical values show that there is no high correlation observed.

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 dataset : *The variables 'vote' and 'gender' have string values. Converting them into numeric values for modelling.*

```
feature: vote
['Labour', 'Conservative']
Categories (2, object): ['Conservative', 'Labour']
[1 0]


feature: gender
['female', 'male']
Categories (2, object): ['female', 'male']
[0 1]
```

	vote	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	gender
0	1	43	3	3	4	1	2	2	0
1	1	36	4	4	4	4	5	2	1
2	1	35	4	4	5	2	3	2	1
3	1	24	4	2	2	1	4	0	0
4	1	41	2	2	1	1	6	2	1

The object data types are converted to numerical. Now the data types of all the variables are in acceptable format for Modeling.

In the target variable 'vote', the category: 'Conservative' is coded as 0 and 'Labour' is coded as 1.

In the variable 'gender', the category: 'female' is coded as 0 and 'male' is coded as 1.

SCALING:

Scaling the data is not required for Logistic regression, LDA and Naive Baye's models as it is not necessary. But in case of KNN it is necessary to scale the data, It uses different distance metrics to find out the nearest neighbours and hence scaling is necessary for KNN. Scaling,gives the data similar weightage to all the variables.

TRAIN & TEST SPLIT:

```
X=df1.drop('labour_or_not',axis=1)
Y=df1['labour_or_not']
```

```
X_train,X_test, Y_train, Y_test=train_test_split(X,Y,train_size=0.70, random_state=1)
```

```
Y_train.value_counts(normalize=True)
```

```
1    0.71065
0    0.28935
Name: labour_or_not, dtype: float64
```

% of voters whose favour is Labour Party in training set 71.06

% of voters whose favour is Conservative Party in training set 28.93

1.4 Apply Logistic Regression and LDA

LOGISTIC REGRESSION:

1. Logistic Regression. Applying Logistic Regression and fitting the training data

```
Logistic_model = LogisticRegression(solver='newton-cg',max_iter=10000,penalty='none',verbose=True,n_jobs=2)
Logistic_model.fit(X_train, Y_train)
```

```
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   1 out of   1 | elapsed:    1.6s finished
```

```
LogisticRegression(max_iter=10000, n_jobs=2, penalty='none', solver='newton-cg',
                   verbose=True)
```

Predicting Train & Test Data:

```
y_train_prob=Logistic_model.predict_proba(X_train)
pd.DataFrame(y_train_prob).head()
```

	0	1
0	0.931825	0.068175
1	0.096984	0.903016
2	0.298416	0.701584
3	0.110210	0.889790
4	0.017223	0.982777

```
y_test_prob=Logistic_model.predict_proba(X_test)
pd.DataFrame(y_test_prob).head()
```

	0	1
0	0.424284	0.575716
1	0.148426	0.851574
2	0.007187	0.992813
3	0.836350	0.163650
4	0.068407	0.931593

Train & Test Accuracy:

```
Logistic_model.score(X_train,Y_train)
```

```
0.8312912346842601
```

```
Logistic_model.score(X_test,Y_test)
```

```
0.8355263157894737
```

Confusion matrix and Classification Report : Train Set

```
y_train_predict=Logistic_model.predict(X_train)
Logistic_model_score=Logistic_model.score(X_train,Y_train)
print(Logistic_model_score)

print(metrics.confusion_matrix(Y_train,y_train_predict))
print(metrics.classification_report(Y_train,y_train_predict))
```

```
0.8312912346842601
[[196 111]
 [ 68 686]]
              precision    recall  f1-score   support

           0       0.74      0.64      0.69       307
           1       0.86      0.91      0.88       754

    accuracy                           0.83      1061
   macro avg       0.80      0.77      0.79      1061
weighted avg       0.83      0.83      0.83      1061
```

Confusion matrix and Classification Report : Train Set

```
y_test_predict=Logistic_model.predict(X_test)
Logistic_model_score=Logistic_model.score(X_test,Y_test)
print(Logistic_model_score)

print(metrics.confusion_matrix(Y_test,y_test_predict))
print(metrics.classification_report(Y_test,y_test_predict))
```

```
0.8355263157894737
[[113  40]
 [ 35 268]]
              precision    recall  f1-score   support

           0       0.76      0.74      0.75       153
           1       0.87      0.88      0.88       303

    accuracy                           0.84       456
   macro avg       0.82      0.81      0.81       456
weighted avg       0.83      0.84      0.83       456
```

The model is not overfit nor underfit. Training and Testing results shows that the model is good with appropriate precision and recall values. The accuracy is nearly same for training and testing data.

LINEAR DISCRIMINANT ANALYSIS:

2. Linear Discriminant Analysis. Applying LDA and fitting the training data

```
LDA_model=LinearDiscriminantAnalysis()
LDA_model.fit(X_train,Y_train)
```

```
LinearDiscriminantAnalysis()
```

Predicting Train & Test Data:

```
y_train_prob=LDA_model.predict_proba(X_train)
pd.DataFrame(y_train_prob).head()
```

	0	1
0	0.949216	0.050784
1	0.078241	0.921759
2	0.307389	0.692611
3	0.078963	0.921037
4	0.012161	0.987839

```
y_test_prob=LDA_model.predict_proba(X_test)
pd.DataFrame(y_test_prob).head()
```

	0	1
0	0.462093	0.537907
1	0.133955	0.866045
2	0.006414	0.993586
3	0.861210	0.138790
4	0.056545	0.943455

Confusion matrix and Classification Report : Train Set

```
y_train_predict=LDA_model.predict(X_train)
LDA_model_score=LDA_model.score(X_train,Y_train)
print(LDA_model_score)

print(metrics.confusion_matrix(Y_train,y_train_predict))
print(metrics.classification_report(Y_train,y_train_predict))
```

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

           0       0.74      0.65      0.69       307
           1       0.86      0.91      0.89       754

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

Accuracy of the Train Set : 83.41

Confusion matrix and Classification Report : Test Set

```
y_test_predict=LDA_model.predict(X_test)
LDA_model_score=LDA_model.score(X_test,Y_test)
print(LDA_model_score)

print(metrics.confusion_matrix(Y_test,y_test_predict))
print(metrics.classification_report(Y_test,y_test_predict))
```

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

           0       0.77      0.73      0.74       153
           1       0.86      0.89      0.88       303

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

Accuracy of the Test Set : 83.33

- ***Training and Testing results shows that the model is good with appropriate precision and recall values. The accuracy is nearly same for training and testing data. No overfit performance metrics are obtained.***

1.5 Apply KNN Model and Naïve Bayes Model. Interpret the results.

KNN MODEL:

KNN it is necessary to scale the data, It uses different distance metrics as it is a distance based algorithm.

```
x=df1.drop("labour_or_not",axis=1)
y=df1["labour_or_not"]
x.head()
```

	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	male_or_not
0	43	3	3	4	1	2	2	0
1	36	4	4	4	4	5	2	1
2	35	4	4	5	2	3	2	1
3	24	4	2	2	1	4	0	0
4	41	2	2	1	1	6	2	1

SCALING THE DATA:

	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	male_or_not
0	-0.716161	-0.278185	-0.148020	0.565802	-1.419969	-1.437338	0.423832	-0.936736
1	-1.162118	0.856242	0.926367	0.565802	1.014951	-0.527684	0.423832	1.067536
2	-1.225827	0.856242	0.926367	1.417312	-0.608329	-1.134120	0.423832	1.067536
3	-1.926617	0.856242	-1.222408	-1.137217	-1.419969	-0.830902	-1.421084	-0.936736
4	-0.843577	-1.412613	-1.222408	-1.988727	-1.419969	-0.224465	0.423832	1.067536

APPLYING KNN & FITTING THE TRAINING DATA:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y, random_state=1)
```

```
from sklearn.neighbors import KNeighborsClassifier

KNN_model=KNeighborsClassifier()
KNN_model.fit(x_train,y_train)
```

```
KNeighborsClassifier()
```

Train & Test Accuracy:

```
y_train_predict=KNN_model.predict(x_train)
KNN_model_score=KNN_model.score(x_train,y_train)
```

```
print(KNN_model_score)
```

```
0.8566402814423922
```

```
y_test_predict=KNN_model.predict(x_test)

KNN_model_score=KNN_model.score(x_test, y_test)
```

```
print(KNN_model_score)
```

```
0.8263157894736842
```

Confusion matrix and Classification Report : Train Set

```
print(metrics.confusion_matrix(y_train,y_train_predict))
print(metrics.classification_report(y_train,y_train_predict))
```

```
[[232  95]
 [ 68 742]]
              precision    recall  f1-score   support

           0       0.77      0.71      0.74       327
           1       0.89      0.92      0.90       810

    accuracy                           0.86      1137
   macro avg       0.83      0.81      0.82      1137
weighted avg       0.85      0.86      0.85      1137
```

Accuracy of the Train Set : 85.66

Confusion matrix and Classification Report : Test Set

```
print(metrics.confusion_matrix(y_test,y_test_predict))
print(metrics.classification_report(y_test,y_test_predict))
```

```
[[ 91  42]
 [ 24 223]]
              precision    recall  f1-score   support

           0       0.79      0.68      0.73       133
           1       0.84      0.90      0.87       247

    accuracy                           0.83       380
   macro avg       0.82      0.79      0.80       380
weighted avg       0.82      0.83      0.82       380
```

Accuracy of the Test Set: 82.63

Training and Testing results demonstrates good precision and recall values. This KNN model has very good accuracy, precision & recall values.

Naive Bayes:

```
NB_model=GaussianNB()
NB_model.fit(X_train, Y_train)
```

```
GaussianNB()
```

Confusion matrix and Classification report : Test Set

```
Y_train_predict=NB_model.predict(X_train)
model_score=NB_model.score(X_train, Y_train)
print(model_score)
print(metrics.confusion_matrix(Y_train,Y_train_predict))

print(metrics.classification_report(Y_train,Y_train_predict))
```

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

           0       0.73      0.69      0.71       307
           1       0.88      0.90      0.89       754

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

Accuracy of the Train Set: 83.5

```
Y_test_predict=NB_model.predict(X_test)
model_score=NB_model.score(X_test, Y_test)
print(model_score)
print(metrics.confusion_matrix(Y_test,Y_test_predict))

print(metrics.classification_report(Y_test,Y_test_predict))
```

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

           0       0.74      0.73      0.73       153
           1       0.87      0.87      0.87       303

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

Accuracy of the Test Set: 82.23

The Naive Bayes model also performs well with better accuracy and recall values.

Considering the Overall performance metrics KNN classification report values has an upper edge over Naïve Bayes .

1.6 Model Tuning, Bagging (Random Forest should be applied for Bagging), and Boosting.

Using GridSearchCV and tuning the model which helps us in finding the best parameters for the model

Logistic Regression

The selected best hyperparameters are as follows:

```
grid_search_Log.fit(X_train,Y_train)
```

```
GridSearchCV(cv=10, estimator=LogisticRegression(random_state=1),
             param_grid={'class_weight': [{0: 0.5, 1: 0.5}, {0: 0.6, 1: 0.4},
                                          {0: 0.65, 1: 0.35}],
                         'max_iter': [1000, 5000, 10000], 'penalty': ['none'],
                         'solver': ['newton-cg', 'lbfgs', 'sag', 'saga'],
                         'tol': [1e-05, 0.0001, 0.001, 0.01, 0.1]},
             scoring='f1_macro')
```

```
grid_search_Log.best_params_
```

```
{'class_weight': {1: 0.35, 0: 0.65},
 'max_iter': 5000,
 'penalty': 'none',
 'solver': 'sag',
 'tol': 1e-05}
```

Performance of the Model based on Best Params:

```
-----------------------------------------------------TRAIN-------------------------------------------------------

              precision    recall  f1-score   support

           0       0.69      0.78      0.73       307
           1       0.90      0.86      0.88       754

    accuracy                           0.83      1061
   macro avg       0.79      0.82      0.80      1061
weighted avg       0.84      0.83      0.84      1061

AUC Score Training Data: 0.890

-----------------------------------------------------TEST--------------------------------------------------------

              precision    recall  f1-score   support

           0       0.70      0.82      0.76       153
           1       0.90      0.83      0.86       303

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

AUC Score Testing Data: 0.880
```

FEATURE IMPORTANCE:

	Coefficients
age	-0.013235
economic.cond.national	0.658302
economic.cond.household	0.099325
Blair	0.657199
Hague	-0.837269
Europe	-0.206362
political.knowledge	-0.291990
male_or_not	0.216195

Top 3 features of feature importance are : 'Hague', 'economic.cond.national' and 'Blair'.

LDA:

The selected best hyperparameters are as follows:

```
grid_search_LDA.fit(X_train,Y_train)
```

```
GridSearchCV(cv=10, estimator=LinearDiscriminantAnalysis(),
             param_grid={'shrinkage': ['auto'], 'solver': ['lsqr', 'eigen'],
                         'tol': [0.1, 0.001, 0.0001, 1e-05]},
             scoring='f1_macro')
```

```
grid_search_LDA.best_params_
```

```
{'shrinkage': 'auto', 'solver': 'lsqr', 'tol': 0.1}
```

```
--------------------------------------------------TRAIN-----------------------------------------------------

              precision    recall  f1-score   support

           0       0.74      0.65      0.69       307
           1       0.86      0.91      0.89       754

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

AUC Score Training Data: 0.890

--------------------------------------------------TEST------------------------------------------------------

              precision    recall  f1-score   support

           0       0.76      0.74      0.75       153
           1       0.87      0.88      0.88       303

    accuracy                           0.84       456
   macro avg       0.82      0.81      0.81       456
weighted avg       0.83      0.84      0.83       456

AUC Score Testing Data: 0.887
```

The Accuracy for training and testing is approximately 83% and 84% respectively and the AUC for training and testing is approximately 89% and 88.7%. The accuracy is nearly same for training and testing data.

FEATURE IMPORTANCE:

	Coefficients
age	-0.019361
economic.cond.national	0.632353
economic.cond.household	0.084974
Blair	0.767361
Hague	-0.939558
Europe	-0.238747
political.knowledge	-0.428492
male_or_not	0.119384

Top 3 features of feature importance are : 'Hague', 'Blair' and 'economic.cond.national'.

KNN:

The selected best hyperparameters are as follows:

```
grid_search_KNN.fit(X_train,Y_train)
```

```
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': [5, 6, 7, 8, 9, 10, 11, 12],
                         'p': [1, 2], 'weights': ['uniform', 'distance']},
             scoring='f1_macro')
```

```
grid_search_KNN.best_params_
```

```
{'n_neighbors': 10, 'p': 1, 'weights': 'uniform'}
```

```
------------------------------------------------------TRAIN-----------------------------------------------------

              precision    recall  f1-score   support

           0       0.76      0.71      0.73       307
           1       0.88      0.91      0.90       754

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

AUC Score Training Data: 0.906

------------------------------------------------------TEST------------------------------------------------------

              precision    recall  f1-score   support

           0       0.76      0.69      0.73       153
           1       0.85      0.89      0.87       303

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

AUC Score Testing Data: 0.860
```

The Accuracy for training and testing is approximately 85% and 82% respectively and the AUC for training and testing is approximately 90% and 86%. The accuracy is has few difference but almost the same for training and testing data. Feature Importance not applicable

Naïve Bayes:

The selected best hyperparameters are as follows :

```
grid_search_NB.fit(X_train,Y_train)
```

```
GridSearchCV(cv=10, estimator=GaussianNB(),
             param_grid={'var_smoothing': [0.01, 1e-05, 1e-09]},
             scoring='f1_macro')
```

```
grid_search_NB.best_params_
```

```
{'var_smoothing': 1e-05}
```

```
-------------------------------------------------TRAIN-------------------------------------------------------

              precision    recall  f1-score   support

           0       0.73      0.69      0.71       307
           1       0.88      0.90      0.89       754

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

AUC Score Training Data: 0.888

-------------------------------------------------TEST--------------------------------------------------------

              precision    recall  f1-score   support

           0       0.74      0.73      0.73       153
           1       0.87      0.87      0.87       303

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

AUC Score Testing Data: 0.876
```

The Accuracy for training and testing is approximately 84% and 82% respectively and the AUC for training and testing is approximately 88.8% and 87.6%. The accuracy is nearly same for training and testing data.Feature Importance not applicable

Random Forest:

The selected best hyperparameters are as follows:

```
grid_search.fit(X_train, Y_train)
```

```
GridSearchCV(cv=3, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [7, 8, 9, 10],
                         'max_features': [5, 6, 7, 8],
                         'min_samples_leaf': [9, 12, 15],
                         'min_samples_split': [50, 100],
                         'n_estimators': [50, 100], 'random_state': [1]})
```

```
grid_search.best_params_
```

```
{'max_depth': 7,
 'max_features': 5,
 'min_samples_leaf': 9,
 'min_samples_split': 50,
 'n_estimators': 50,
 'random_state': 1}
```

BAGGING (RANDOM FOREST CLASSIFIER):

```
from sklearn.ensemble import BaggingClassifier

RF_model= BaggingClassifier(base_estimator=grid_search,n_estimators=5,random_state=1)
RF_model=RF_model.fit(X_train,Y_train)
```

```
y_train_predict=RF_model.predict(X_train)
RF_model_score=RF_model.score(X_train,Y_train)
print(RF_model_score)

print(metrics.confusion_matrix(Y_train,y_train_predict))
print(metrics.classification_report(Y_train,y_train_predict)
```

```
0.8510838831291234
[[196 111]
 [ 47 707]]
              precision    recall  f1-score   support

           0       0.81      0.64      0.71       307
           1       0.86      0.94      0.90       754

    accuracy                           0.85      1061
   macro avg       0.84      0.79      0.81      1061
weighted avg       0.85      0.85      0.85      1061
```

```
y_test_predict = RF_model.predict(X_test)
model_score = RF_model.score(X_test, Y_test)
print(model_score)
print(metrics.confusion_matrix(Y_test, y_test_predict))
print(metrics.classification_report(Y_test, y_test_predict))
```

```
0.8135964912280702
[[ 97  56]
 [ 29 274]]
              precision    recall  f1-score   support

           0       0.77      0.63      0.70       153
           1       0.83      0.90      0.87       303

    accuracy                           0.81       456
   macro avg       0.80      0.77      0.78       456
weighted avg       0.81      0.81      0.81       456
```

The Accuracy for training and testing is approximately 85% and 81% respectively. The accuracy for training data is around 5% higher than testing data.

BOOSTING:

In Boosting two models are used:

1.Ada Boost Classifier

2.Gradient Boost Classifier

ADA BOOST CLASSIFIER

```
from sklearn.ensemble import AdaBoostClassifier
ADB_model=AdaBoostClassifier(n_estimators=100,random_state=1)
ADB_model.fit(X_train,Y_train)
```

```
AdaBoostClassifier(n_estimators=100, random_state=1)
```

CONFUSION MATRIX & CLASSIFICATION REPORT : TRAINING SET

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

           0       0.76      0.70      0.73       307
           1       0.88      0.91      0.90       754

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

CONFUSION MATRIX & CLASSIFICATION REPORT : TESTING SET

```
0.8135964912280702
[[112  41]
 [ 40 263]]
              precision    recall  f1-score   support

           0       0.74      0.73      0.73       153
           1       0.87      0.87      0.87       303

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

The Accuracy for training and testing is approximately 85% and 82% respectively . The majority class recall and f1-score is around 91% and 87% for training for testing data it is around 3%-4% less.

GRADIENT BOOST CLASSIFIER:

```
from sklearn.ensemble import GradientBoostingClassifier
gbc_model=GradientBoostingClassifier(random_state=1)
gbc_model.fit(X_train, Y_train)
```

```
GradientBoostingClassifier(random_state=1)
```

CONFUSION MATRIX & CLASSIFICATION REPORT : TRAINING SET

```
0.8925541941564562
[[211  96]
 [ 79 675]]
              precision    recall  f1-score   support

           0       0.73      0.69      0.71       307
           1       0.88      0.90      0.89       754

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

CONFUSION MATRIX & CLASSIFICATION REPORT : TESTING SET

```
0.8355263157894737
[[112  41]
 [ 40 263]]
              precision    recall  f1-score   support

           0       0.74      0.73      0.73       153
           1       0.87      0.87      0.87       303

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

The Accuracy for training and testing is approximately 84% and 82% respectively . The accuracy is nearly same for training and testing data.

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.

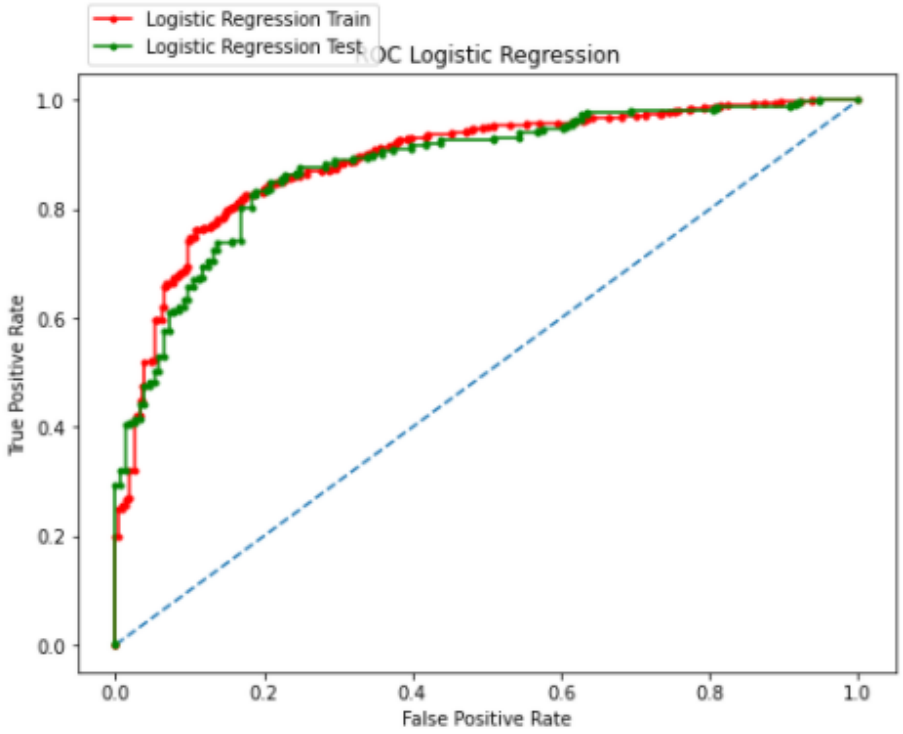
PERFORMANCE METRICS -LOGISTIC REGRESSION :

Further tuning the model we derive the below results:

AUC Score & Confusion Matrix for Train & Test Set:

```
AUC Score Training Data of Logistic Regression model: 0.890
AUC Score Testing Data of Logistic Regression model: 0.880
Train Set Confusion Matrix for Logistic Regression model :
[[238  69]
 [109 645]]
Test Set Confusion Matrix for Logistic Regression model :
[[125  28]
 [ 53 250]]
```

ROC Curve:



- ***The Accuracy for training and testing is approximately 83% and 82% respectively and the AUC for training and testing is approximately 89% and 88%. The accuracy is nearly same for training and testing data.***
- ***The majority class ('Labour' coded as 1) 'Precision', 'Recall' and 'F1-score' for training set is approximately 90%, 86% and 88% respectively.***
- ***The majority class ('Labour' coded as 1) 'Precision', 'Recall' and 'F1-score' for testing set is approximately 90%, 83% and 86% respectively.***
- ***The minority class ('Conservative' coded as 0) 'Precision', 'Recall' and 'F1-score' for training set is approximately 69%, 78% and 73% respectively.***
- ***The minority class ('Conservative' coded as 0) 'Precision', 'Recall' and 'F1-score' for testing set is approximately 70%, 82% and 76% respectively.***
- ***No overfit performance metrics are obtained.***

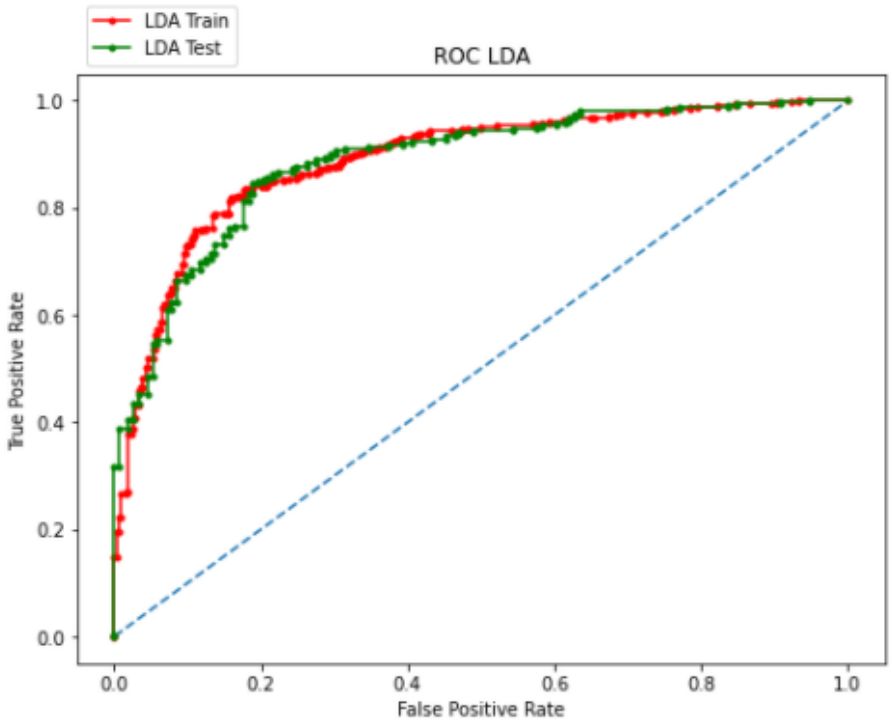
PERFORMANCE METRICS –LDA:

Further tuning the model we derive the below results:

AUC Score & Confusion Matrix for Train & Test Set:

```
AUC Score Training Data of LDA model: 0.890
AUC Score Testing Data of LDA model: 0.887
Train Set Confusion Matrix for LDA model :
[[200 107]
 [ 70 684]]
Test Set Confusion Matrix for LDA model :
[[113  40]
 [ 35 268]]
```

ROC Curve:



- ***The Accuracy for training and testing is approximately 83% and 84% respectively and the AUC for training and testing is approximately 89% and 88.7%. The accuracy is nearly same for training and testing data.***
- ***The majority class ('Labour' coded as 1) 'Precision', 'Recall' and 'F1-score' for training set is approximately 86%, 91% and 89% respectively.***

- *The majority class ('Labour' coded as 1) 'Precision', 'Recall' and 'F1-score' for testing set is approximately 87%, 88% and 88% respectively.*
- *The minority class ('Conservative' coded as 0) 'Precision', 'Recall' and 'F1-score' for training set is approximately 74%, 65% and 69% respectively.*
- *The minority class ('Conservative' coded as 0) 'Precision', 'Recall' and 'F1-score' for testing set is approximately 76%, 74% and 75% respectively.*
- *No overfit performance metrics are obtained.*

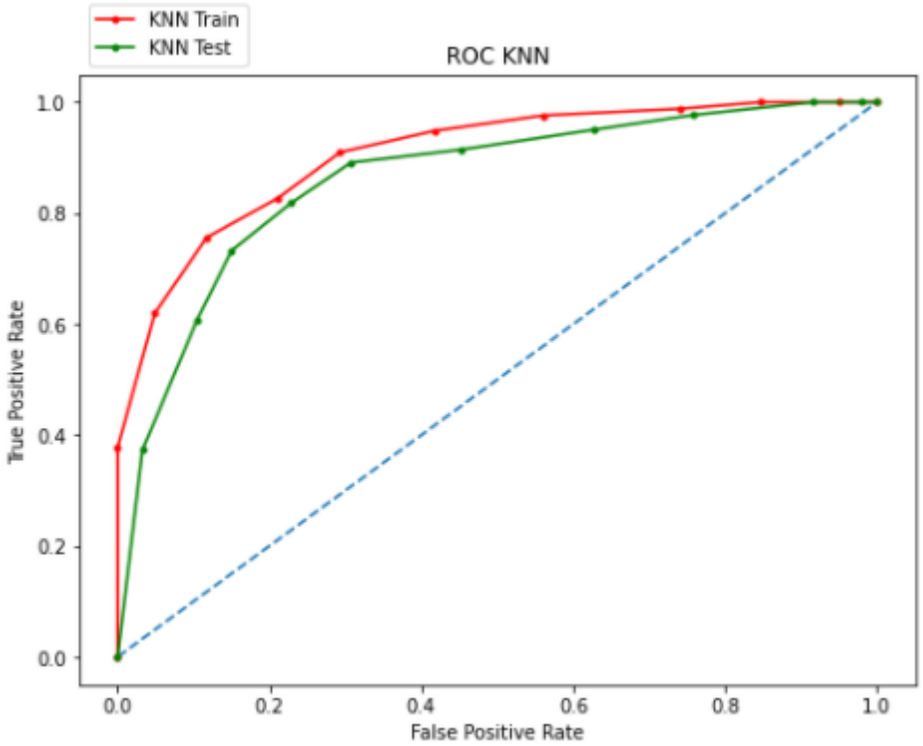
PERFORMANCE METRICS –KNN:

Further tuning the model we derive the below results:

AUC Score & Confusion Matrix for Train & Test Set:

```
AUC Score Training Data of KNN model: 0.906
AUC Score Testing Data of KNN model: 0.860
Train Confusion Matrix for KNN model :
[[217  90]
 [ 68 686]]
Test Confusion Matrix for KNN model :
[[106  47]
 [ 33 270]]
```

ROC Curve:



- ***The Accuracy for training and testing is approximately 85% and 82% respectively and the AUC for training and testing is approximately 90% and 86%. The accuracy is has few difference but almost the same for training and testing data.***
- ***The majority class ('Labour' coded as 1) 'Precision', 'Recall' and 'F1-score' for training set is approximately 88%, 91% and 90% respectively.***
- ***The majority class ('Labour' coded as 1) 'Precision', 'Recall' and 'F1-score' for testing set is approximately 85%, 89% and 87% respectively.***
- ***The minority class ('Conservative' coded as 0) 'Precision', 'Recall' and 'F1-score' for training set is approximately 76%, 71% and 73% respectively.***
- ***The minority class ('Conservative' coded as 0) 'Precision', 'Recall' and 'F1-score' for testing set is approximately 76%, 69% and 73% respectively.***
- ***The AUC score of Training data is just above 90% and also the performance metrics precision, recall and f1-score of the Labour class for Training Data is around 90%. Thus these values are on the overfit borderline .***

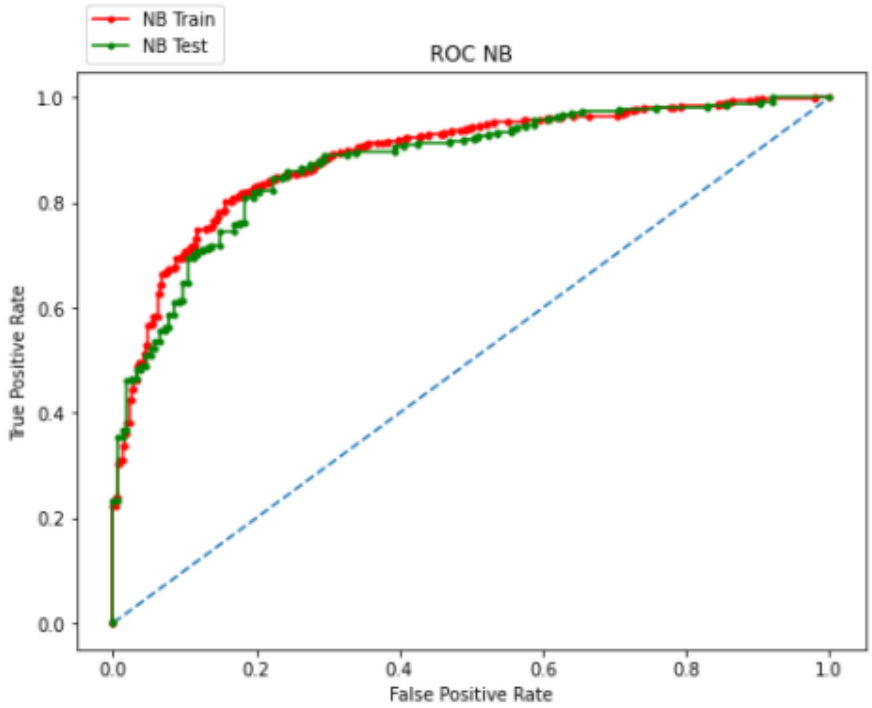
PERFORMANCE METRICS –Naïve Bayes:

Further tuning the model we derive the below results:

AUC Score & Confusion Matrix for Train & Test Set:

```
AUC Score Training Data of NB model: 0.888
AUC Score Testing Data of NB model: 0.876
Train Set Confusion Matrix for NB model :
[[211  96]
 [ 79 675]]
Test Set Confusion Matrix for NB model :
[[112  41]
 [ 40 263]]
```

ROC Curve:



- ***The Accuracy for training and testing is approximately 84% and 82% respectively and the AUC for training and testing is approximately 88.8% and 87.6%. The accuracy is nearly same for training and testing data.***
- ***The majority class ('Labour' coded as 1) 'Precision', 'Recall' and 'F1-score' for training set is approximately 88%, 90% and 89% respectively.***
- ***The majority class ('Labour' coded as 1) 'Precision', 'Recall' and 'F1-score' for testing set is approximately 87%, 87% and 87% respectively.***
- ***The minority class ('Conservative' coded as 0) 'Precision', 'Recall' and 'F1-score' for training set is approximately 73%, 69% and 71% respectively.***
- ***The minority class ('Conservative' coded as 0) 'Precision', 'Recall' and 'F1-score' for testing set is approximately 74%, 73% and 73% respectively.***

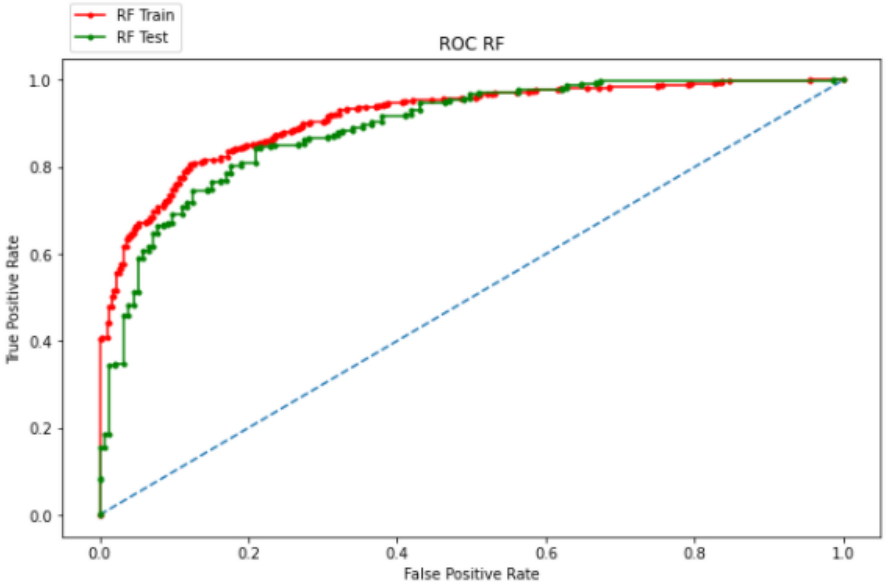
PERFORMANCE METRICS –Random Forest:

Further tuning the model we derive the below results:

AUC Score & Confusion Matrix for Train & Test Set:

```
AUC Score Training Data of RF model: 0.913
AUC Score Testing Data of RF model: 0.888
Train Set Confusion Matrix for RF model :
[[196 111]
 [ 47 707]]
Test Set Confusion Matrix for RF model :
[[ 97  56]
 [ 29 274]]
```

ROC:



- ***The majority class ('Labour' coded as 1) 'Precision', 'Recall' and 'F1-score' for training set is approximately 86%, 94% and 90% respectively.***
- ***The majority class ('Labour' coded as 1) 'Precision', 'Recall' and 'F1-score' for testing set is approximately 83%, 90% and 87% respectively.***
- ***The minority class ('Conservative' coded as 0) 'Precision', 'Recall' and 'F1-score' for training set is approximately 81%, 64% and 71% respectively.***
- ***The minority class ('Conservative' coded as 0) 'Precision', 'Recall' and 'F1-score' for testing set is approximately 77%, 63% and 70% respectively.***
- ***The majority class recall is around 94% for training and 90% for testing, also the AUC for training is 91.3%. All the other performance metrics are less than 90%.***

PERFORMANCE METRICS –ADA BOOSTING:

Further tuning the model we derive the below results:

AUC Score & Confusion Matrix for Train & Test Set:

TRAIN SET:

AUC: 0.915

TEST SET:

AUC: 0.877

TRAIN SET:

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

           0       0.76      0.70      0.73       307
           1       0.88      0.91      0.90       754

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

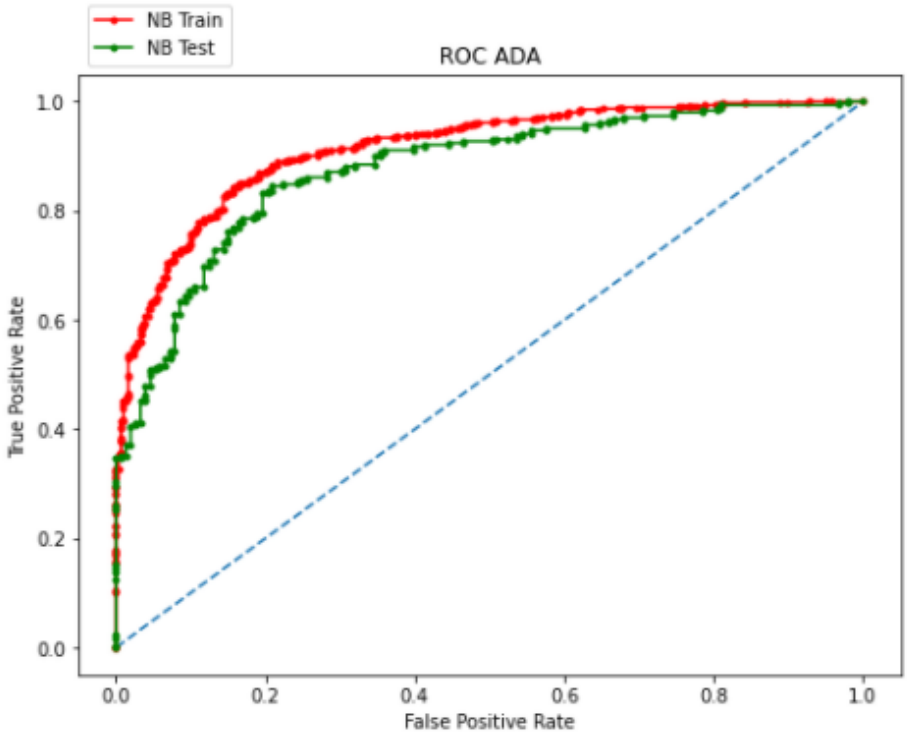
TEST SET:

```
0.8135964912280702
[[112  41]
 [ 40 263]]
              precision    recall  f1-score   support

           0       0.74      0.73      0.73       153
           1       0.87      0.87      0.87       303

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

ROC :



- *The Accuracy for training and testing is approximately 85% and 82% respectively and the AUC for training and testing is approximately 91.5% and 87.7%. The accuracy for training data and testing data are nearly same.*
- *The majority class ('Labour' coded as 1) 'Precision', 'Recall' and 'F1-score' for training set is approximately 88%, 91% and 90% respectively.*
- *The majority class ('Labour' coded as 1) 'Precision', 'Recall' and 'F1-score' for testing set is approximately 87%, 87% and 87% respectively.*
- *The minority class ('Conservative' coded as 0) 'Precision', 'Recall' and 'F1-score' for training set is approximately 76%, 70% and 73% respectively.*
- *The minority class ('Conservative' coded as 0) 'Precision', 'Recall' and 'F1-score' for testing set is approximately 74%, 73% and 73% respectively.*
- *The majority class recall and f1-score is around 91% and 90% for training for testing data it is around 3%-4% less. Also the AUC for training is 91.5% and for testing is nearly close at 87.7%. All the other performance metrics are less than 90%.*

PERFORMANCE METRICS –GRADIENT BOOSTING:

Further tuning the model we derive the below results:

AUC Score,Accuracy &Confusion Matrix for Train & Test Set:

TRAIN SET:

AUC: 0.951

TEST SET:

AUC: 0.899

TRAIN SET:

0.8925541941564562
[[211 96]
[79 675]]

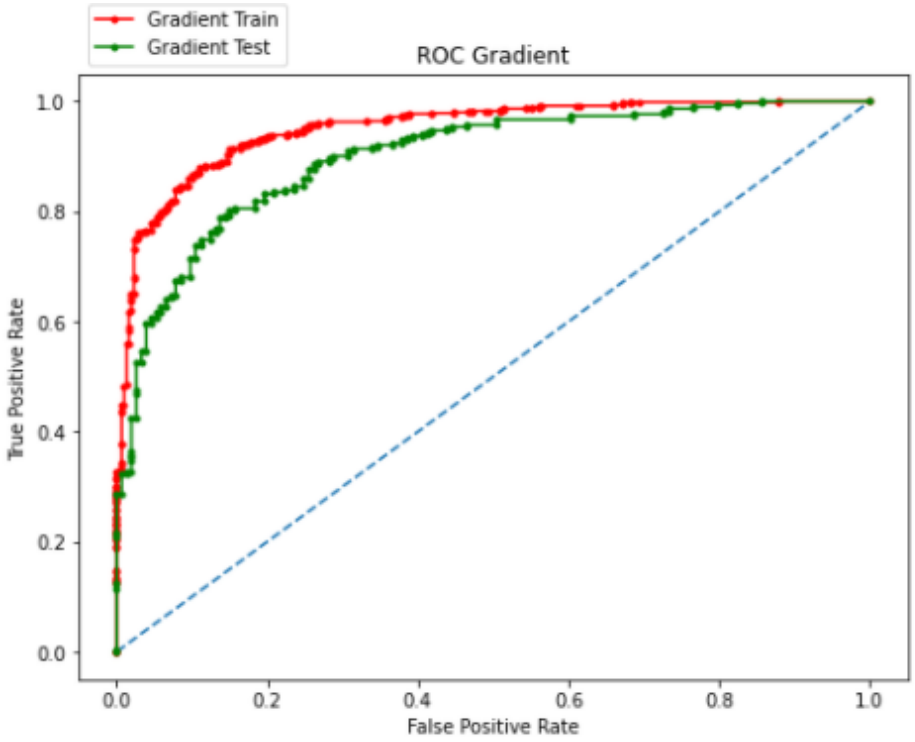
	precision	recall	f1-score	support
0	0.73	0.69	0.71	307
1	0.88	0.90	0.89	754
accuracy			0.84	1061
macro avg	0.80	0.79	0.80	1061
weighted avg	0.83	0.84	0.83	1061

TEST SET:

0.8355263157894737
[[112 41]
[40 263]]

	precision	recall	f1-score	support
0	0.74	0.73	0.73	153
1	0.87	0.87	0.87	303
accuracy			0.82	456
macro avg	0.80	0.80	0.80	456
weighted avg	0.82	0.82	0.82	456

ROC Curve:

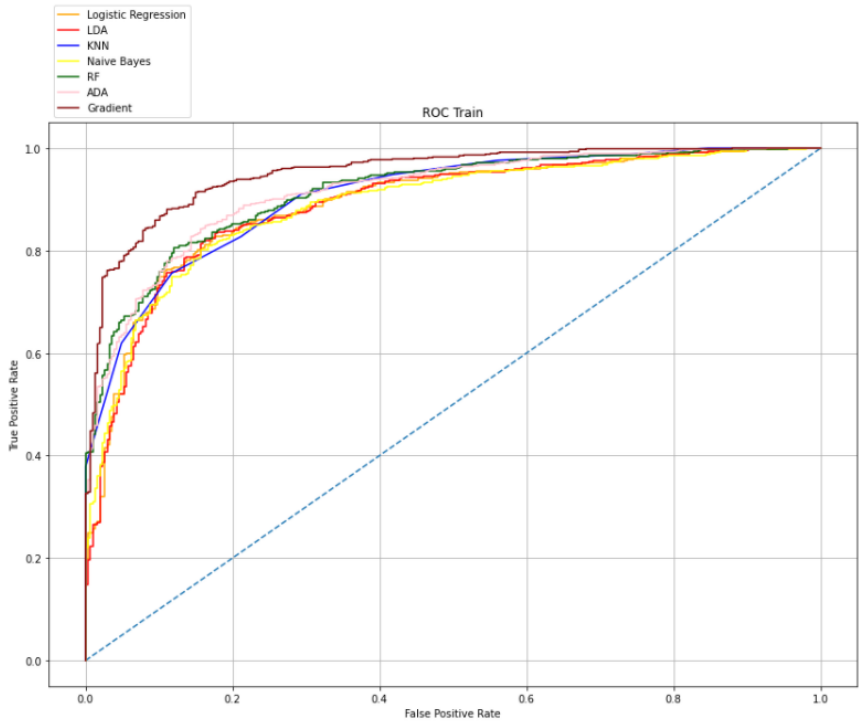


- ***The Accuracy for training and testing is approximately 84% and 82% respectively and the AUC for training and testing is approximately 95.1% and 89.9%. The accuracy for training data and testing data are nearly same.***
- ***The majority class ('Labour' coded as 1) 'Precision', 'Recall' and 'F1-score' for training set is approximately 88%, 90% and 89% respectively.***
- ***The majority class ('Labour' coded as 1) 'Precision', 'Recall' and 'F1-score' for testing set is approximately 87%, 87% and 87% respectively.***
- ***The minority class ('Conservative' coded as 0) 'Precision', 'Recall' and 'F1-score' for training set is approximately 73%, 69% and 71% respectively.***
- ***The minority class ('Conservative' coded as 0) 'Precision', 'Recall' and 'F1-score' for testing set is approximately 74%, 73% and 73% respectively.***
- ***The majority class recall is around 90% for training for testing data it is around 3% less. Also the AUC for training is 95.1% and for testing is nearly close at 90%. All the other performance metrics are less than 90%.***

IDENTIFYING THE BEST MODEL:

PERFORMANCE METRICS - TRAINING SET					
Model	Accuracy	AUC	Recall	Precision	f1 score
Logistic Regression	83	89	82	79	80
LDA	83	89	78	80	79
KNN	85	90	81	82	81
Naïve Bayes	84	88	79	80	80
Random Forest	85	91	79	84	81
Ada Boost	85	91	80	82	81
Gradient Boost	84	95	79	80	80

ROC Curve : Train

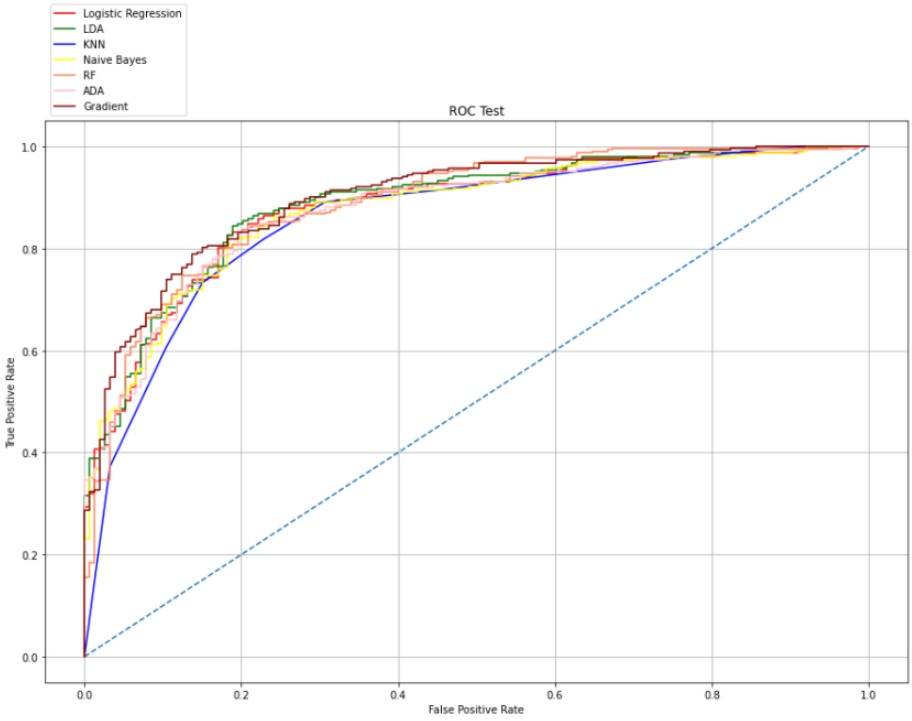


Inference:

The Gradient Boosting model seems to give the best ROC curve for the training data.

PERFORMANCE METRICS - TEST SET					
Model	**Accuracy**	**AUC**	**Recall**	**Precision**	**f1 score**
Logistic Regression	82	88	82	80	81
LDA	84	89	81	82	81
KNN	82	86	79	81	80
Naïve Bayes	82	87	80	80	80
Random Forest	81	88	77	80	78
Ada Boost	82	87	80	80	80
Gradient Boost	82	89	80	80	80

ROC Curve : Test



Inference:

There are few ROC curves overlapping with each other.

CONCLUSION:

Overfitting: Certain models have values of performance metrics above 90% but for these metrics the training and testing values are close to each other with a maximum difference of around 3%-4%.

Overall Performance Metrics:

- ***Accuracy: The highest accuracy for training is observed for Gradient Boost and for testing data it is observed for both KNN and Gradient Boost.***
- ***AUC Score: The highest AUC Score for training is observed for Gradient Boost and for testing data it is observed for Gradient Boost.***
- ***Recall macro: The highest Recall macro for training is observed for Gradient Boost and for testing data it is observed for Gradient Boost.***
- ***Precision macro: The highest Precision macro for training is observed for RF followed by Gradient Boost and for testing data it is observed for KNN followed by Gradient Boost.***
- ***f1-score macro: The highest Precision macro for training is observed for Gradient Boost and for testing data it is observed for Gradient Boost.***

Thus it observed that Gradient Boost model has performed very well in all the performance metrics both for training and testing data.

Henceforth, considering all the above points Gradient Boost Model is found to be the Optimized Model.

1.8 Based on these predictions, what are the insights?

The important insights of Target variable 'vote' with the independent variables are as follows:

- ***The Average age of voters who voted for Conservative Party seem to be a little higher than those who voted for Labour party.***
- ***Voters who have given higher ratings for current national economic conditions and current household economic conditions show favour the Labour Party highly.***
- ***Voters who have given 4 and above ratings for Blair favour the Labour Party highly.***
- ***Voters who have given ratings of 3 or less for Hague are more in favour of the Labour party. Voters who have given a rating of 4 for Hague seem to be equally in favour of either parties, and voters who have given a rating of 5 for Hague seem to favour the Conservative party more.***
- ***Voters with different levels of knowledge on the either Parties; seem to support Labour Party especially those who seem to have no political knowledge highly favour Labour Party.***
- ***For Gradient Boost Model, the top three features of feature importance are: 'Hague', 'Blair' and 'Europe'.***

	Importance
Hague	0.344333
Blair	0.187058
Europe	0.174572
age	0.097234
political.knowledge	0.086827
economic.cond.national	0.076627
economic.cond.household	0.030867
male_or_not	0.002482

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 1973

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

We are importing the nltk library to use the inaugural.fileds()

```
import nltk
import numpy as np
import pandas as pd
import re
nltk.download('inaugural')
from nltk.corpus import inaugural
inaugural.fileids()
```

```
[nltk_data] Downloading package inaugural to
[nltk_data]     C:\Users\dhine\AppData\Roaming\nltk_data...
[nltk_data]   Package inaugural is already up-to-date!
```

```
nltk.download('inaugural')
```

After importing the text file, we would first count the total number of characters in each file separately.

I) Number of Characters with Punctuations:

```
The number of characters in the speech by President Franklin D. Roosevelt in 1941 is: 6174
The number of characters in the speech by President John F. Kennedy in 1961 is: 6202
The number of characters in the speech by President Richard Nixon in 1973 is: 8122
```

Number of Characters without Punctuations:

```
The number of characters in the speech by President Roosevelt in 1941 is: 5962
The number of characters in the speech by President  Kennedy in 1961 is: 5998
The number of characters in the speech by President  Nixon in 1973 is: 7895
```

II) Number of Words with Punctuations:

```
The number of words in the speech by President Roosevelt in 1941 is: 1360
The number of words in the speech by President Kennedy in 1961 is: 1390
The number of words in the speech by President Nixon in 1973 is: 1819
```

Number of Words without Punctuations:

```
The number of words in the speech by President  Roosevelt in 1941 is: 1338
The number of words in the speech by President Kennedy in 1961 is: 1365
The number of words in the speech by President Nixon in 1973 is: 1802
```

III) Number of Sentences:

```
The Number of sentence in the speech by President Roosevelt in 1941 is: 68
The Number of sentence in the speech by Kennedy in 1961 is: 52
The Number of sentence in the speech by Nixon in 1973 is: 68
```

2.2 Remove all the stopwords from all three speeches.

```
from nltk.corpus import stopwords
nltk.download('stopwords')
```

Stopwords Feature is utilized by importing the above.

Number of StopWords Present in the Speech.

```
The number of stop words present in the speech by President Roosevelt in 1941 is: 694
The number of stop words present in the speech by President Kennedy in 1961 is: 660
The number of stop words present in the speech by President Nixon in 1973 is: 958
```

StopWords were removed from the each of the President's Speech.

```
def stopwords_remove(speech):
    speech = speech.lower()
    words = []
    for word in speech.split():
        if word not in set(stopwords.words('english')):
            words.append(word)
        speech = ' '.join(words)
    return(speech)
```

```
stopwords_roosevelt = stopwords_remove(roosevelt)
stopwords_kennedy = stopwords_remove(kennedy)
stopwords_nixon = stopwords_remove(nixon)
```

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)

For determining the words which occurred the most number of times in inaugural address for each president speech,Counter feature is imported.

```
from collections import Counter
```

I) From Roosevelt file we have the below words which are highly used in during the speech by the president.

```
counts_roosevelt = Counter(filtered_roosevelt.split())
counts_roosevelt.most_common(10)
```

```
[('nation', 11),
 ('know', 10),
 ('spirit', 9),
 ('democracy', 9),
 ('us', 8),
 ('life', 8),
 ('people', 7),
 ('america', 7),
 ('years', 6),
 ('freedom', 6)]
```

Top 3 Words are : nation , know , spirit

II) From Kennedy file we have the below words which are highly used in during the speech by the president.

```
counts_kennedy = Counter(filtered_kennedy.split())
counts_kennedy.most_common(10)
```

```
[('let', 16),
 ('us', 12),
 ('world', 8),
 ('sides', 8),
 ('new', 7),
 ('pledge', 7),
 ('citizens', 5),
 ('power', 5),
 ('shall', 5),
 ('free', 5)]
```

Top 3 Words are : let , us ,world

III) From Nixon file we have the below words which are highly used in during the speech by the president.

```
counts_nixon = Counter(filtered_nixon.split())
counts_nixon.most_common(10)
```

```
[('us', 26),
 ('let', 22),
 ('peace', 19),
 ('world', 16),
 ('new', 15),
 ('america', 13),
 ('responsibility', 11),
 ('government', 10),
 ('great', 9),
 ('home', 9)]
```

Top 3 Words are : us ,let , peace

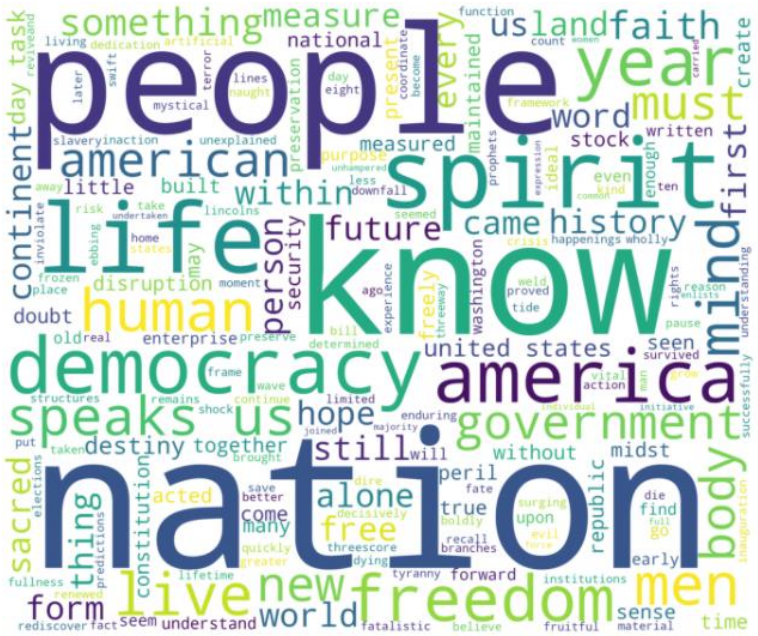
2.4 Plot the word cloud of each of the speeches of the variable. (after removing the stopwords)

StopWords have been removed from each of the president's speech document by importing `from nltk.corpus import stopwords` *into our dataset*

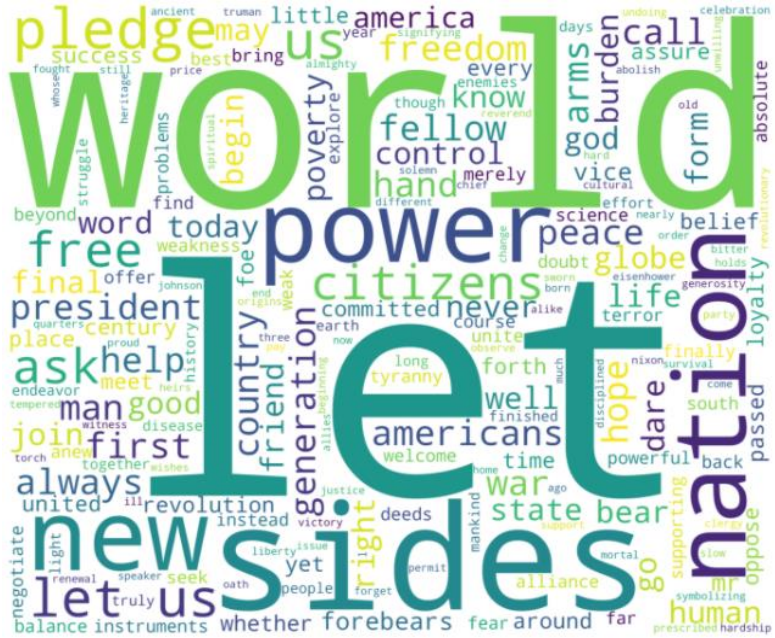
Word Cloud is a data visualization technique used for representing text data in which the size of each word indicates its frequency or importance.

Here we are creating the wordcloud for each of the President speech and we have imported the wordcloud by importing libraries `from wordcloud import WordCloud`

Wordcloud for Roosevelt speech:



Wordcloud for Kennedy speech:



Wordcloud for Nixon speech:

Great Learning