PROJECT REPORT - MACHINE LEARNING

By Prakash v Mahadole

Problem 1 :

Problem Statement:

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.

Data Dictionary:

1. vote: Party choice: Conservative or Labour
2. age: in years
3. economic.cond.national: Assessment of current national economic conditions, 1 to 5.
4. economic.cond.household: Assessment of current household economic conditions, 1 to 5.
5. Blair: Assessment of the Labour leader, 1 to 5.
6. Hague: Assessment of the Conservative leader, 1 to 5.
7. Europe: an 11-point scale that measures respondents' attitudes toward European integration. High scores represent 'Eurosceptic' sentiment.

Problem 2:

Problem Statement:

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:

1. President Franklin D. Roosevelt in 1941
2. President John F. Kennedy in 1961
3. President Richard Nixon in 1973 Loading all the necessary library for the model building. Now, reading the head and tail of the dataset to check whether data has been properly fed. ### Head of the data

Out[3]:

	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

Tail of the data

In [4]:
```
df.tail()
```

Out[4]:

	Unnamed: 0	vote	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	gender
1520	1521	Conservative	67	5	3	2	4	11	3	male
1521	1522	Conservative	73	2	2	4	4	8	2	male
1522	1523	Labour	37	3	3	5	4	2	2	male
1523	1524	Conservative	61	3	3	1	4	11	2	male
1524	1525	Conservative	74	2	3	2	4	11	0	female

Checking shape of the data

(1525, 9) 1525 rows and 9 Columns

Checking data info

```
In [8]: df.info()

<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
```


All the variables except vote and gender are int64 datatypes.

Checking null values

```
In [9]: df.isnull().sum()

Out[9]: vote                    0
        age                     0
        economic.cond.national  0
        economic.cond.household 0
        Blair                   0
        Hague                   0
        Europe                  0
        political.knowledge     0
        gender                  0
        dtype: int64
```

The dataset contains no null values.

Checking duplicate data

Out[10]:

	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

We have total 8 number of duplicate rows.

The dataset has few duplicates and removing them is the best choice as duplicates does not add any value.

After removing duplicate data we have

(1517, 9)

1517 rows and 9 columns

Checking data Discription

Out[13]:

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
vote	1517	2	Labour	1057	NaN	NaN	NaN	NaN	NaN	NaN	NaN
age	1517	NaN	NaN	NaN	54.2413	15.7017	24	41	53	67	93
economic.cond.national	1517	NaN	NaN	NaN	3.24522	0.881792	1	3	3	4	5
economic.cond.household	1517	NaN	NaN	NaN	3.13777	0.931069	1	3	3	4	5
Blair	1517	NaN	NaN	NaN	3.33553	1.17477	1	2	4	4	5
Hague	1517	NaN	NaN	NaN	2.74951	1.23248	1	2	2	4	5
Europe	1517	NaN	NaN	NaN	6.74028	3.29904	1	4	6	10	11
political.knowledge	1517	NaN	NaN	NaN	1.54054	1.08442	0	0	2	2	3
gender	1517	2	female	808	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Checking the skewness

```
Out[14]: Hague                       0.146191
         age                         0.139800
         Europe                     -0.141891
         economic.cond.household    -0.144148
         economic.cond.national     -0.238474
         political.knowledge        -0.422928
         Blair                      -0.539514
         dtype: float64
```

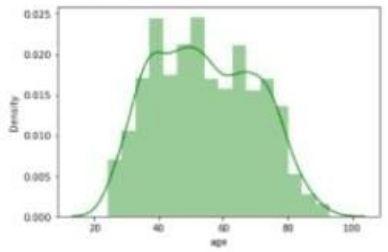
1.2 Perform Univariate and Bivariate Analysis. Do exploratory data analysis. Check for Outliers.

Exploratory Data Analysis

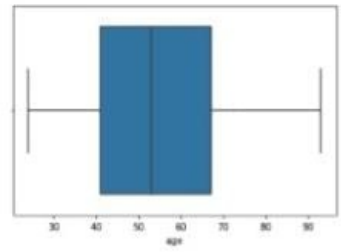
Univariate analysis / Bivariate Analysis

Data Description

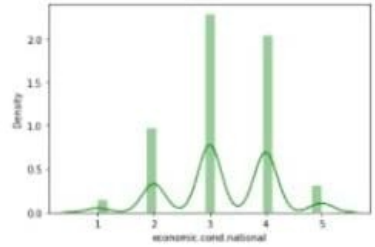
```
Description of age
--------------------------------------------------
count    1517.000000
mean       54.241266
std        15.701741
min        24.000000
25%        41.000000
50%        53.000000
75%        67.000000
max        93.000000
Name: age, dtype: float64
Distribution of age
--------------------------------------------------
```



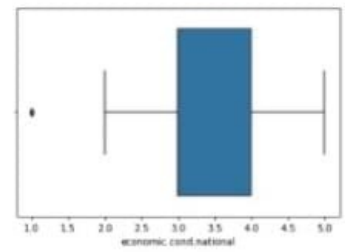

```
Boxplot of age
--------------------------------------------------
```



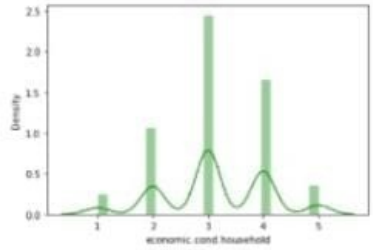

```
Description of economic.cond.national
--------------------------------------------------
count    1517.000000
mean        3.245221
std         0.881792
min         1.000000
25%         3.000000
50%         3.000000
75%         4.000000
max         5.000000
Name: economic.cond.national, dtype: float64
Distribution of economic.cond.national
--------------------------------------------------
```



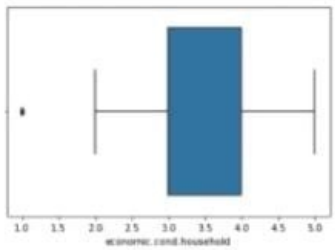

```
Boxplot of economic.cond.national
--------------------------------------------------
```



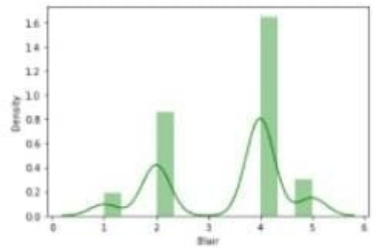
```
Description of economic.cond.household
----------------------------------------------------------
count    1517.000000
mean        3.137772
std         0.931869
min         1.000000
25%         3.000000
50%         3.000000
75%         4.000000
max         5.000000
Name: economic.cond.household, dtype: float64
Distribution of economic.cond.household
----------------------------------------------------------
```



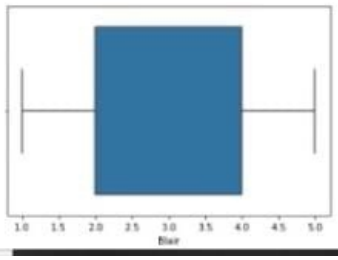

```
Boxplot of economic.cond.household
----------------------------------------------------------
```



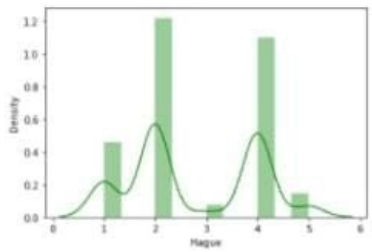

```
Description of Blair
----------------------------------------------------------
count    1517.000000
mean        3.335531
std         1.174772
min         1.000000
25%         2.000000
50%         4.000000
75%         4.000000
max         5.000000
Name: Blair, dtype: float64
Distribution of Blair
----------------------------------------------------------
```



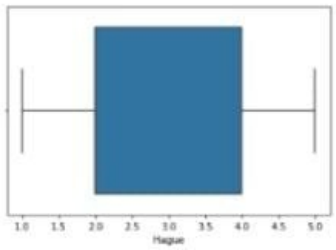

```
Boxplot of Blair
----------------------------------------------------------
```



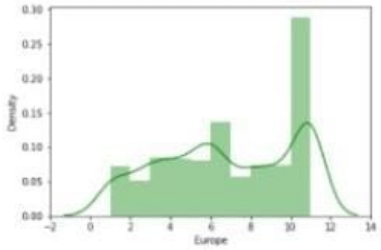

```
Description of Hague
----------------------------------------------------------
count    1517.000000
mean        2.749506
std         1.232479
min         1.000000
25%         2.000000
50%         2.000000
75%         4.000000
max         5.000000
Name: Hague, dtype: float64
Distribution of Hague
----------------------------------------------------------
```



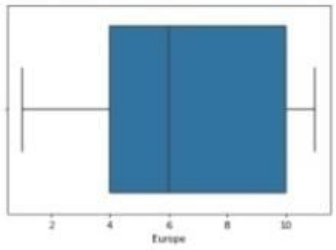
```
Boxplot of Hague
```



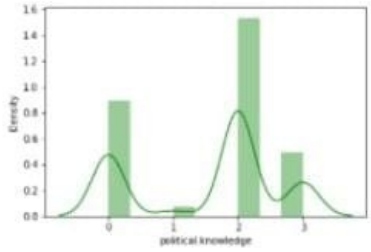

```
Description of Europe
count    1517.000000
mean        6.748277
std         3.299043
min         1.000000
25%         4.000000
50%         6.000000
75%        10.000000
max        11.000000
Name: Europe, dtype: float64
Distribution of Europe
```



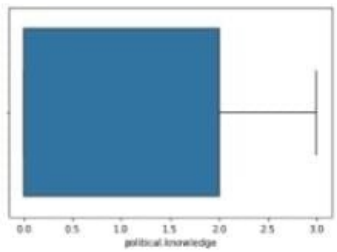

```
Boxplot of Europe
```




```
Description of political.knowledge
count    1517.000000
mean        1.540541
std         1.084417
min         0.000000
25%         0.000000
50%         2.000000
75%         2.000000
max         3.000000
Name: political.knowledge, dtype: float64
Distribution of political.knowledge
```




```
Boxplot of political.knowledge
```

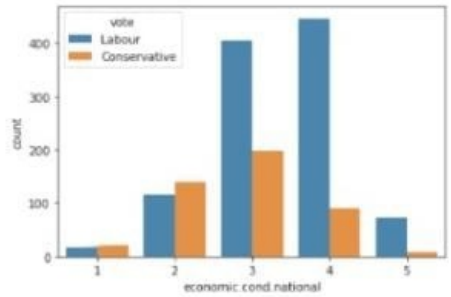


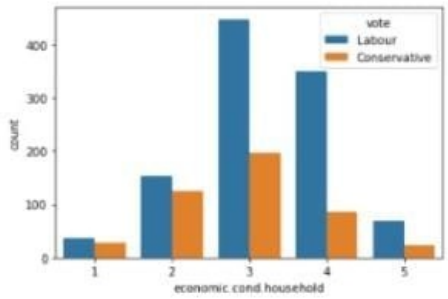
age is the only numaric variable,having no outlier and Also, the dist. plot shows that the variable is normally distributed

votes are large in number for ‘Labour’

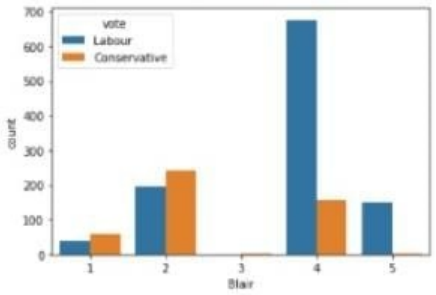
‘female’ voters large in number than ‘male’

```
Out[20]: <AxesSubplot:xlabel='economic.cond.national', ylabel='count'>
```

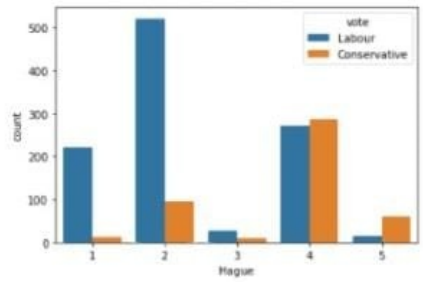


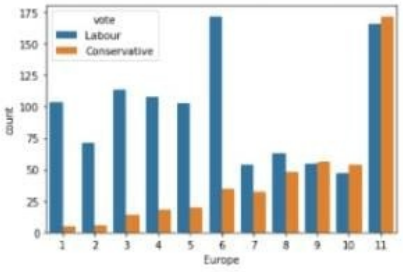



```
Out[22]: <AxesSubplot:xlabel='Blair', ylabel='count'>
```

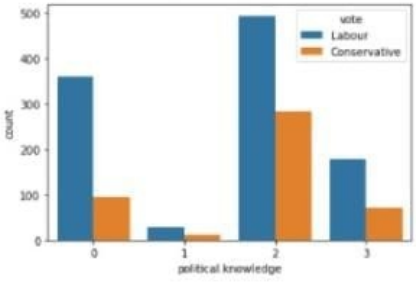


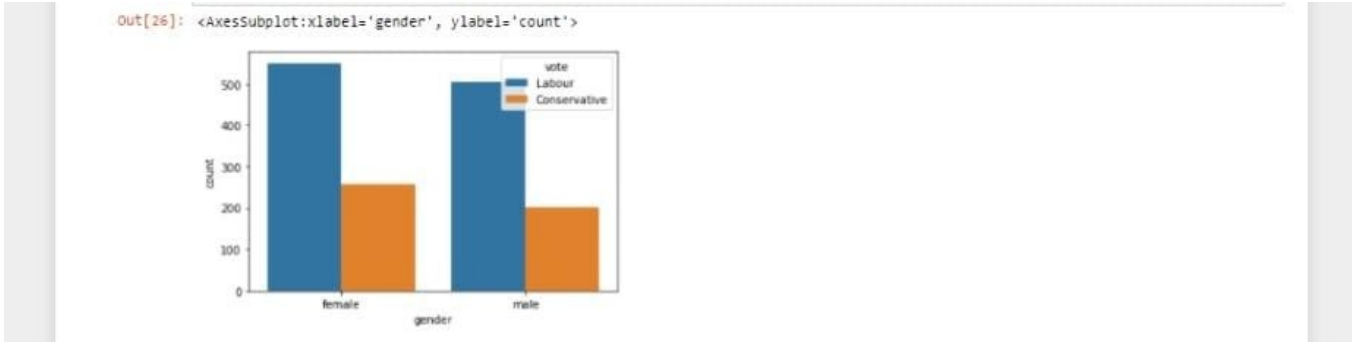

```
Out[23]: <AxesSubplot:xlabel='Hague', ylabel='count'>
```





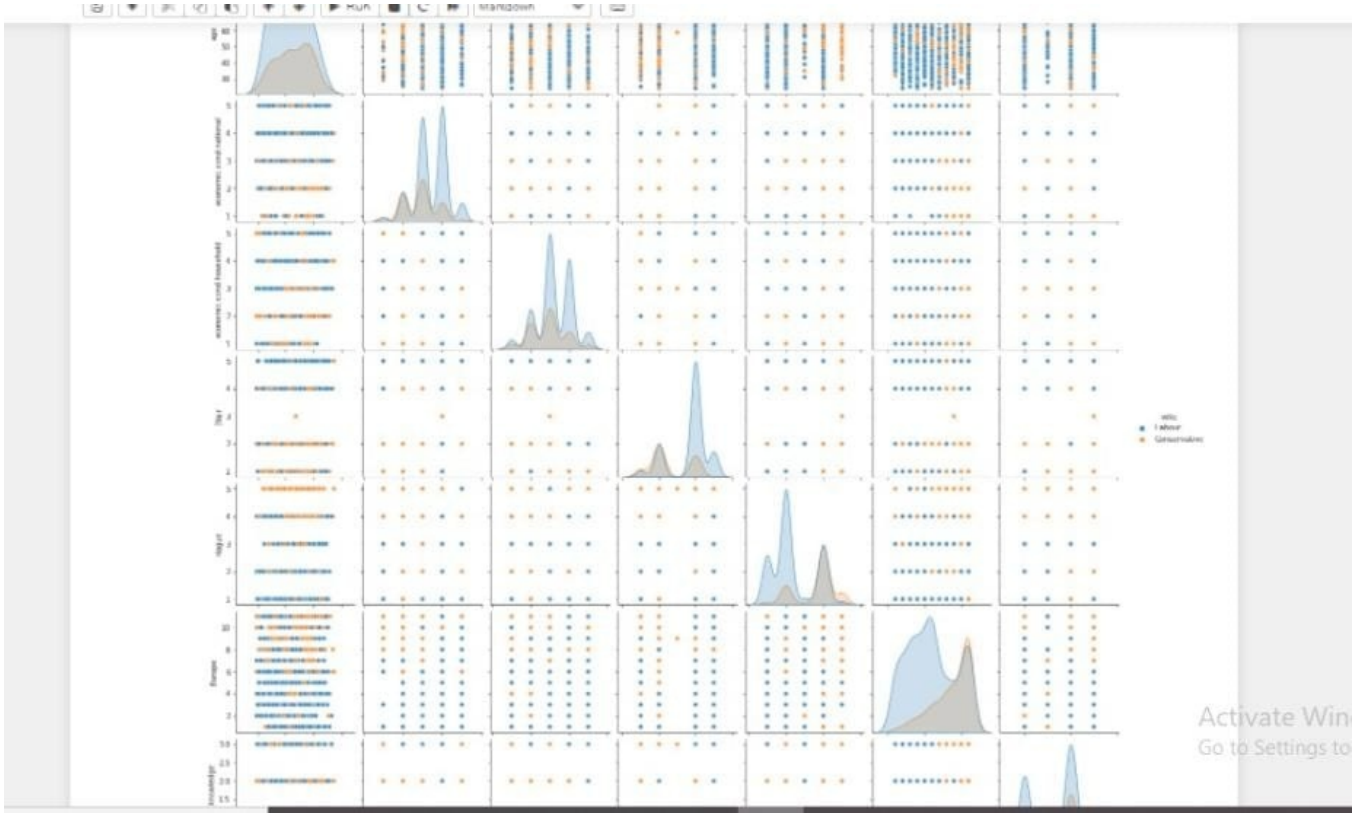

```
Out[25]: <AxesSubplot:xlabel='political.knowledge', ylabel='count'>
```



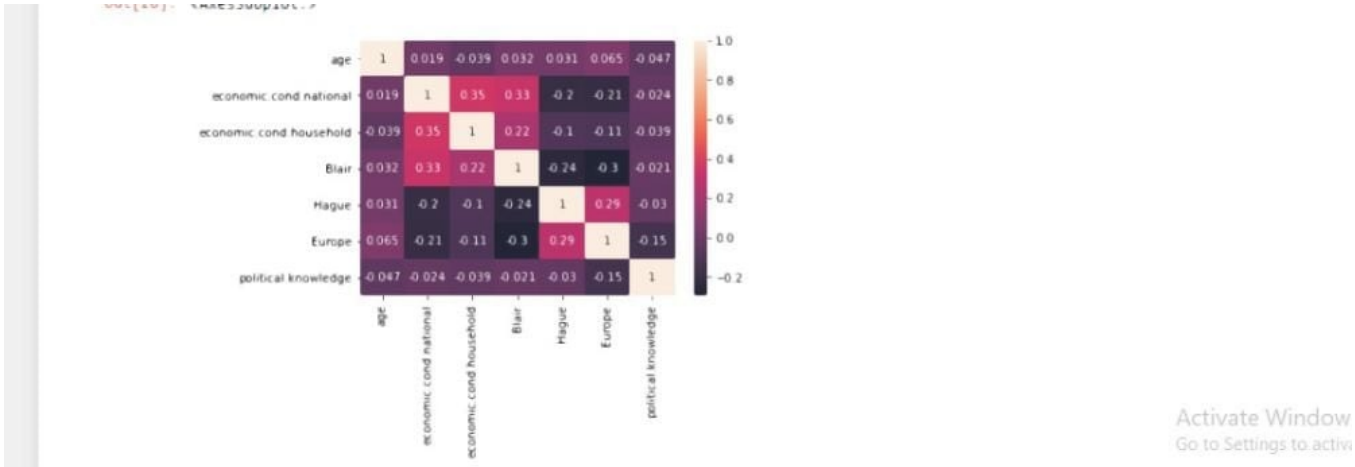


Labour gets the highest voting from both female and male voters. Almost in all the categories Labour is getting the maximum votes. Conservative gets a little bit high votes from Europe '11'. we could see people who vote Conservative are the people who are older. In variable Europe '1' are older people.

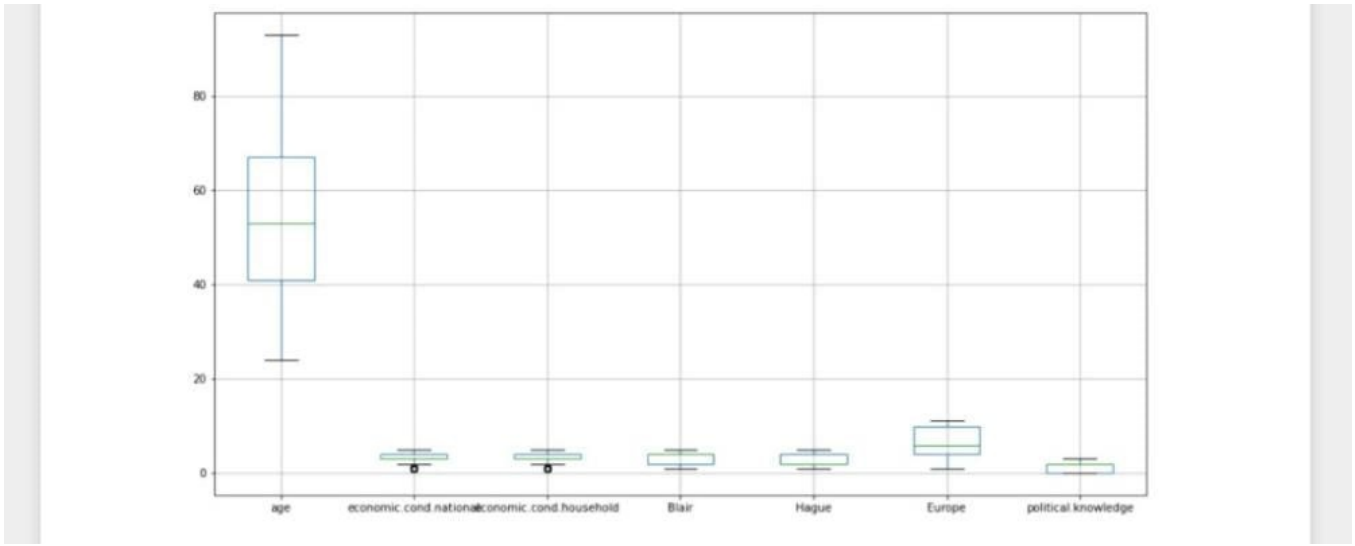
Pairplot



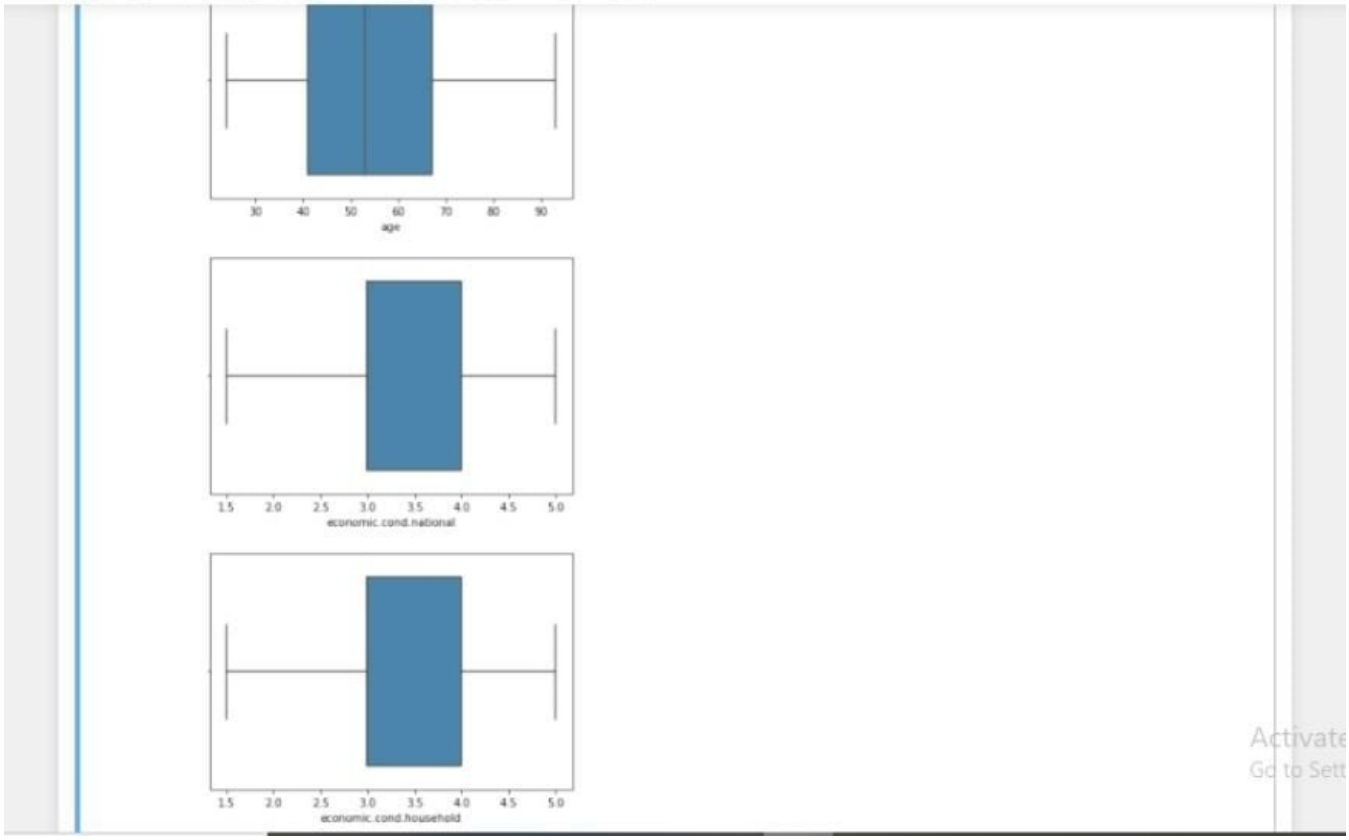
Heatmap

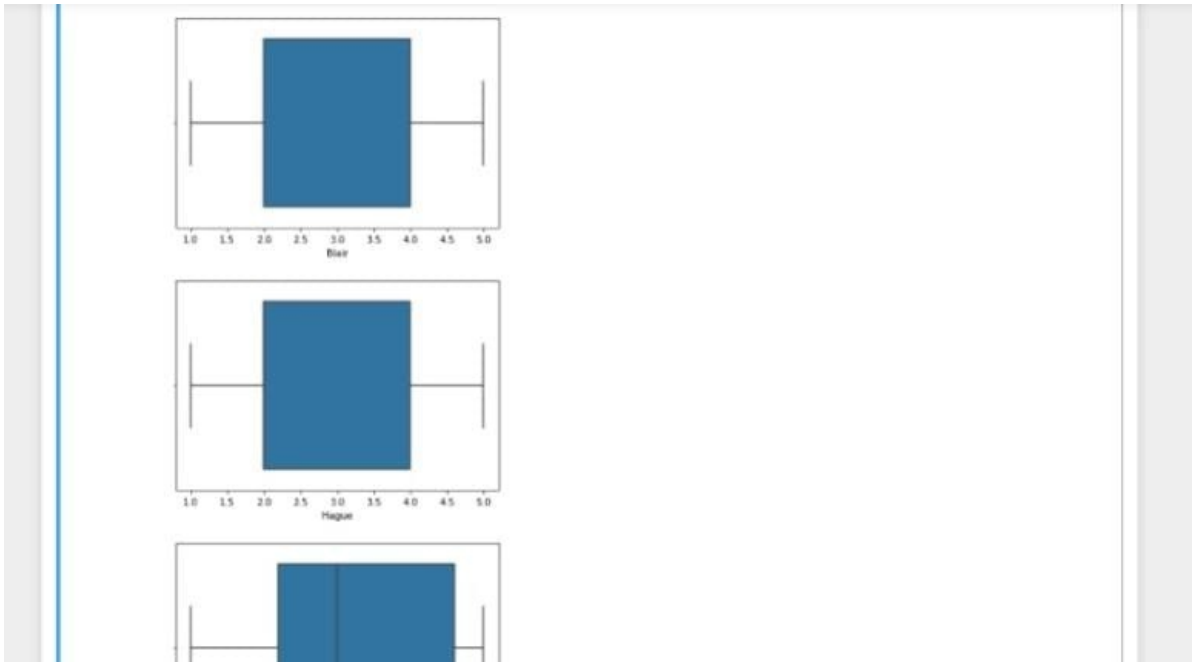


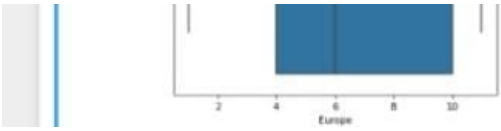
Boxplot before threating outliers

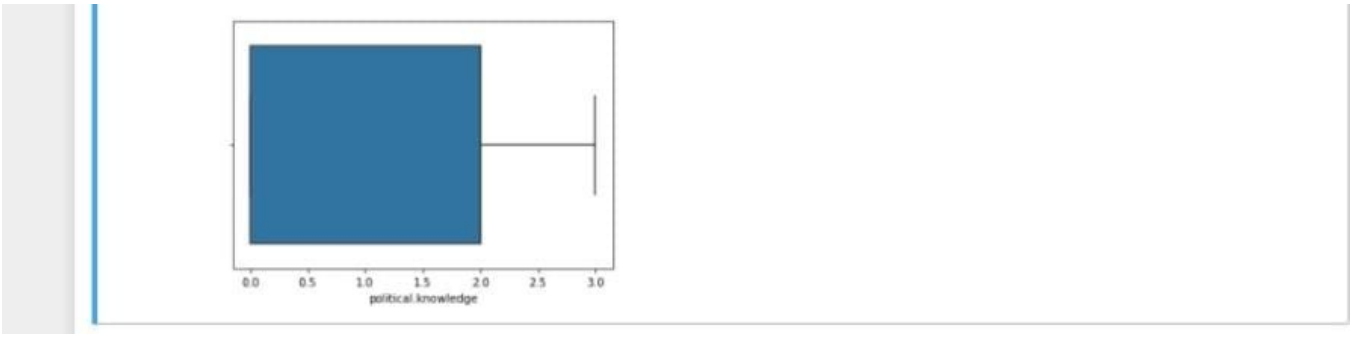


Boxplot of each variable after outlier treatment









Outliers has been treated succesfuly.

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).

Data columns

Index(['age', 'economic.cond.national', 'economic.cond.household', 'Blair',

```
       'Hague', 'Europe', 'political.knowledge', 'vote_Labour', 'gender_male'],
      dtype='object')
```

Scaling

We are not going to scale the data 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, as it a distance-based algorithm (typically based on Euclidean distance). Scaling the data gives similar weightage to all the variables.

1.4 Apply Logistic Regression and LDA (linear discriminant analysis).

Logistic Regression

LogisticRegression(max_iter=10000, n_jobs=2, penalty='none', solver='newton-cg',

```
                   verbose=True)
```

Getting probabilities on training set.

```
    0           1
```

0 . 0.933264 0.066736

1 . 0.095272 0.904728

2 . 0.293630 0.706370

3 . 0.112030 0.887970

4 . 0.016233 0.983767

Getting probabilities on testing set.

```
    0           1
```

0 . 0.426549 0.573451

1 . 0.151457 0.848543

2 . 0.006491 0.993509

3 . 0.842674 0.157326

4 . 0.063533 0.936467

Confusion matrix and classification report on training data

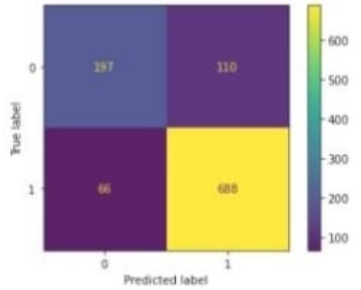
```
Confusion Matrix
[[197 110]
 [ 66 688]]

Classification Report
              precision    recall  f1-score   support

           0       0.75      0.64      0.69       307
           1       0.86      0.91      0.89       754

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

```
Out[45]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2010e007f10>
```



Confusion matrix and classification report on testing data

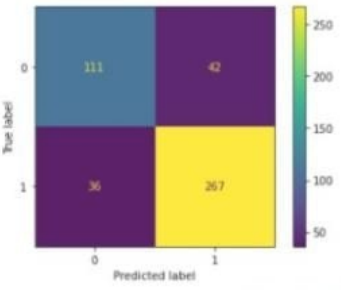
```
Confusion Matrix
[[111  42]
 [ 36 267]]

Classification report
              precision    recall  f1-score   support

           0       0.76      0.73      0.74       153
           1       0.86      0.88      0.87       303

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

```
Out[46]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2010e0367f0>
```



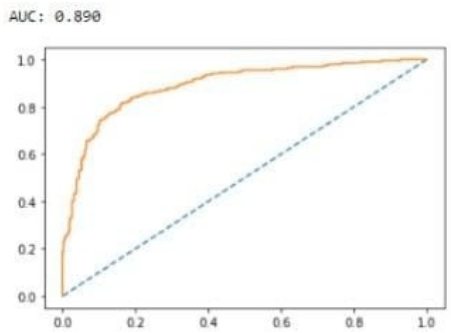
Checking train model score,

0.8341187558906692

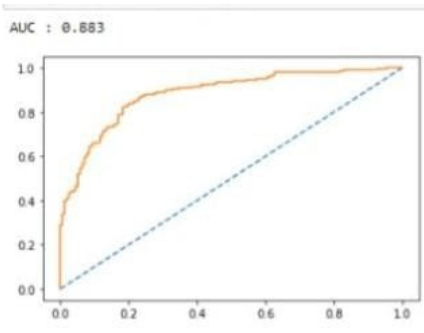
Checking test model score,

0.8289473684210527

AUC and ROC for training data



AUC and ROC for testing data



The model is not overfitting or underfitting. Training and Testing results shows that the model is excellent with good precision and recall values

Building LDA model

LinearDiscriminantAnalysis()

Getting probabilities on training set

0 1

0 . 0.950266 0.049734

1 . 0.077561 0.922439

2 . 0.305087 0.694913

3 . 0.080344 0.919656

4 . 0.011710 0.988290

Getting probabilities on testing set.

0 1

0 . 0.465970 0.534030

1 . 0.137501 0.862499

2 . 0.005997 0.994003

3 . 0.866101 0.133899

4 . 0.053663 0.946337

Confusion matrix and classification report on training data

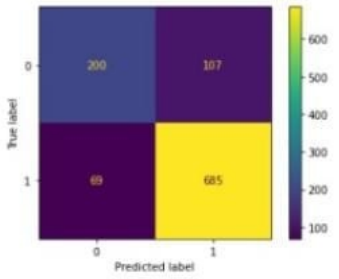
```
Confusion Matrix
[[200 107]
 [ 69 685]]

Classification Report
              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
```

Out[56]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2010e1ff940>



Confusion matrix and classification report on testing data

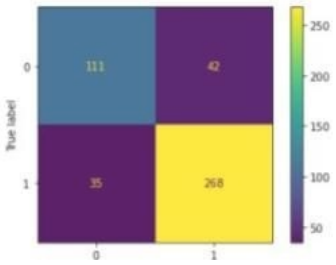
```
Confusion Matrix
[[111  42]
 [ 35 268]]

Classification report
              precision    recall  f1-score   support

           0       0.76      0.73      0.74       153
           1       0.86      0.88      0.87       303

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

Out[57]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2010e17af10>



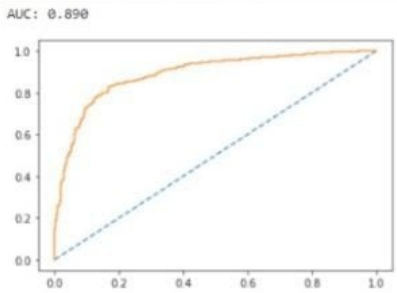
Checking train model score,

0.8341187558906692

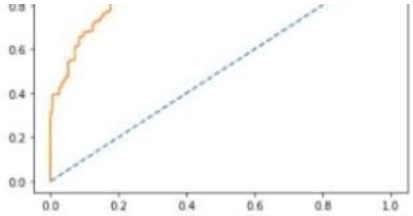
Checking test model score,

0.831140350877193

AUC and ROC for training data



AUC and ROC for testing data



Training and Testing results shows that the model is excellent with good precision and recall values. The LDA model is better than Logistic regression with better Test accuracy and recall values.

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

Gaussian Naive Bayes

GaussianNB()

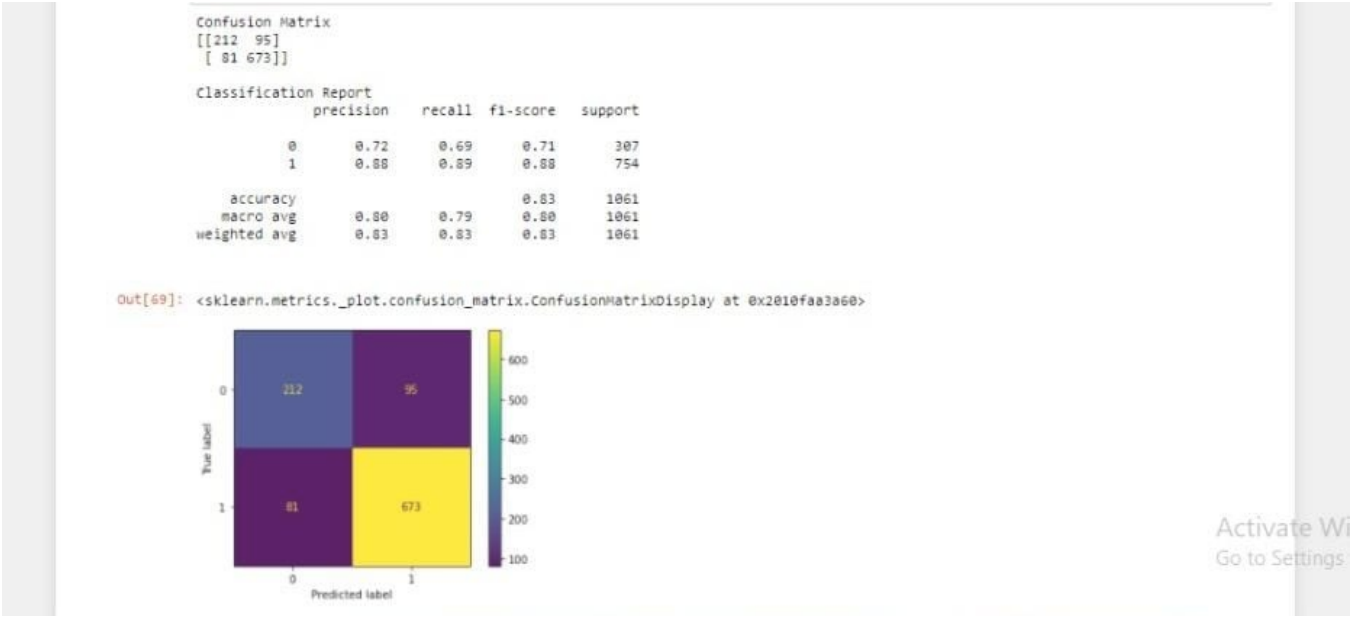
Checking train model score,

0.8341187558906692

Checking test model score,

0.8223684210526315

Confusion matrix and classification report on training data



```
Confusion Matrix
[[212  95]
 [ 81 673]]

Classification Report
              precision    recall  f1-score   support

           0       0.72      0.69      0.71       307
           1       0.88      0.89      0.88       754

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

```
Out[69]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2010faa3a60>
```


Confusion matrix and classification report on testing data

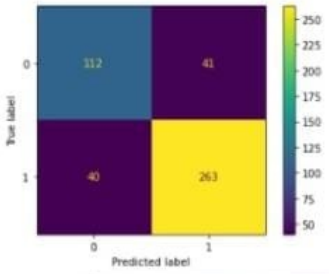
```
Confusion Matrix
[[112  41]
 [ 40 263]]

Classification report
              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
```

```
Out[70]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2010fafc550>
```



KNN Model

Data head after scaling

Out[80]:

	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	IsMale_or_not
0	-0.716161	-0.301648	-0.179882	0.565802	-1.419969	-1.437338	0.423832	-0.936736
1	-1.162118	0.870183	0.949003	0.565802	1.014951	-0.527684	0.423832	1.067536
2	-1.225827	0.870183	0.949003	1.417312	-0.608329	-1.134120	0.423832	1.067536
3	-1.926617	0.870183	-1.308386	-1.137217	-1.419969	-0.830902	-1.421084	-0.936736
4	-0.843577	-1.473479	-1.308386	-1.988727	-1.419969	-0.224465	0.423832	1.067536

Checking train model score,

0.8567389255419415

Checking test model score,

0.8267543859649122

Confusion matrix and classification report on training data

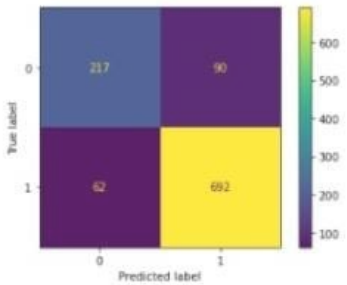
```
Confusion Matrix
[[217  90]
 [ 62 692]]

Classification Report
              precision    recall  f1-score   support

           0       0.78      0.71      0.74       307
           1       0.88      0.92      0.90       754

    accuracy                           0.86      1061
   macro avg       0.83      0.81      0.82      1061
weighted avg       0.85      0.86      0.85      1061
```

```
Out[87]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2010e185610>
```



Confusion matrix and classification report on testing data

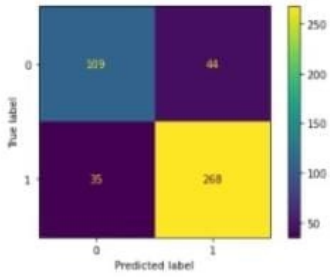
```
Confusion Matrix
[[109  44]
 [ 35 268]]

Classification report
              precision    recall  f1-score   support

           0       0.76      0.71      0.73       153
           1       0.86      0.88      0.87       303

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

```
Out[88]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2010fae38b0>
```



Training and Testing results shows that the model neither overfitting nor underfitting. The Naive Bayes model also performs well with better accuracy and recall values. Even though NB and KNN have same Train and Test accuracy. Based on their recall value in test dataset it is evident that KNN performs better than Naive 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

GridSearchCV(cv=5, estimator=LogisticRegression(),

```
             param_grid={'max_iter': [100, 200, 300, 400, 500],
                         'n_jobs': [1, 2, 3], 'penalty': ['l2', 'none'],
                         'solver': ['saga', 'lbfgs', 'newton-cg'],
                         'tol': [0.1, 0.01, 0.001]})
```

Grid search best params

{'max_iter': 400, 'n_jobs': 2, 'penalty': 'none', 'solver': 'saga', 'tol': 0.1}

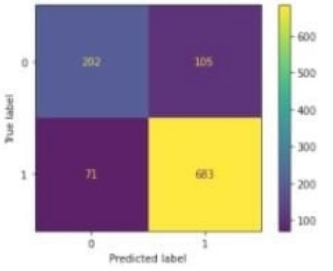
Confusion matrix and classification report on training data

```
0.8341187558906692
[[202 105]
 [ 71 683]]
              precision    recall  f1-score   support

           0       0.74      0.66      0.70       307
           1       0.87      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
```

```
Out[102]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2010d640430>
```



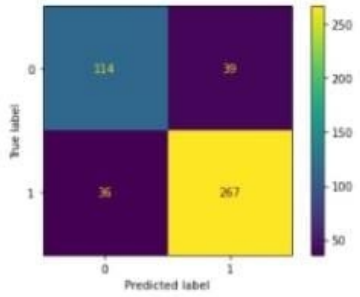
Confusion matrix and classification report on test data

```
0.8355263157894737
[[114  39]
 [ 36 267]]
              precision    recall  f1-score   support

           0       0.76      0.75      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      0.84       456
```

Out[103]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2010e2dd220>



Train model score

0.8341187558906692

Test model score

0.8355263157894737

KNN

Applying KNN model and using the hyperparameter Leaf size and n_neighbour to estimate the model parameters,

GridSearchCV(cv=5, estimator=KNeighborsClassifier(),

```
             param_grid={'leaf_size': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                                       13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
                                       23, 24, 25, 26, 27, 28, 29, 30, ...],
                         'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                                         13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
                                         23, 24, 25, 26, 27, 28, 29],
                         'p': [1, 2]})
```

Grid search best params

{'leaf_size': 1, 'n_neighbors': 28, 'p': 1}

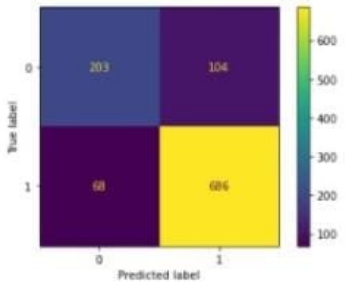
Confusion matrix and classification report on training data

```
0.8378887841658812
[[203 104]
 [ 68 686]]
              precision    recall  f1-score   support

           0       0.75      0.66      0.70       307
           1       0.87      0.91      0.89       754

    accuracy                           0.84      1061
   macro avg       0.81      0.79      0.80      1061
weighted avg       0.83      0.84      0.83      1061
```

```
Out[114]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x20111598f70>
```



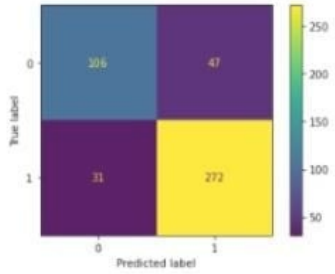
Confusion matrix and classification report on test data

```
0.8289473684210527
[[106  47]
 [ 31 272]]
              precision    recall  f1-score   support

           0       0.77      0.69      0.73       153
           1       0.85      0.90      0.87       303

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

```
Out[115]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x201114343a0>
```



KNN train accuracy acore

0.8378887841658812

KNN test accuracy acore

0.8289473684210527

Decision Tree Classifier (pruned)

Train accuracy score

0.8444863336475024

Test accuracy score

0.7894736842105263

Ensemble Technique - Bagging

(Decision tree used)

Using Bagging to improve the performance of the model.

Applying the model and predicting the train and test data

Applying Ada Boosting model and predicting the train and test

Doing Ada Boosting

AdaBoostClassifier(n_estimators=100, random_state=1)

Doing Gradient Boosting

GradientBoostingClassifier(random_state=1)

Random Forest

Applying Random forest, tuning the model to get the best parameters

GridSearchCV(cv=5, estimator=RandomForestClassifier(),

```
             param_grid={'max_depth': [8, 9, 10], 'max_features': [5, 6, 7],
                         'min_samples_leaf': [9, 12, 15],
                         'min_samples_split': [50, 70],
                         'n_estimators': [50, 100], 'random_state': [1]})
```

Grid search best params are

{'max_depth': 8,

'max_features': 5,

'min_samples_leaf': 15,

'min_samples_split': 50,

'n_estimators': 50,

'random_state': 1}

Train accuracy score for random forest

0.8510838831291234

Test accuracy score for random forest

0.8267543859649122

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.

Tuned Logistic Regression

Logistic regression model performs very good with 83% and 82% accuracy for train and test.

Train model score for Tuned Logistic Regression

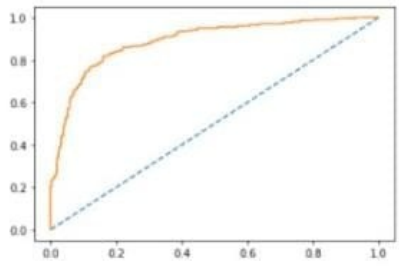
0.8341187558906692

Test model score for Tuned Logistic Regression

0.8355263157894737

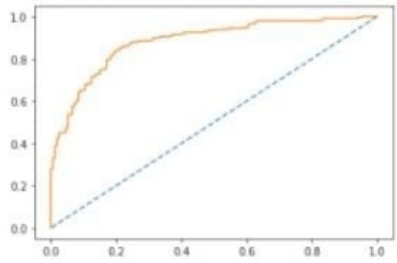
AUC and ROC for tunned logistic regression model train

```
AUC: 0.890
Out[152]: [<matplotlib.lines.Line2D at 0x2011163a880>]
```



AUC and ROC for tunned logistic regression model test

```
AUC: 0.883
```



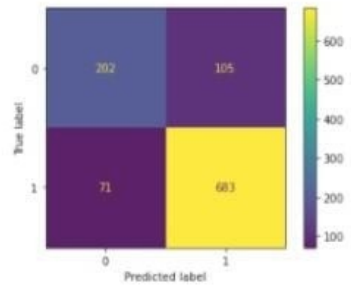
Performance Matrix on train data set

```
0.8341187558906692
[[202 105]
 [ 71 683]]
              precision    recall  f1-score   support

           0       0.74      0.66      0.70       307
           1       0.87      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

Out[154]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x201116a65b0>
```



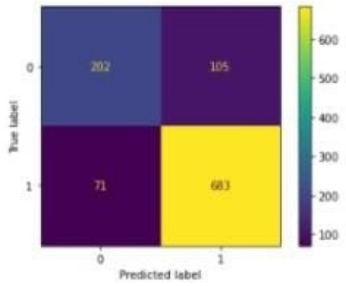
Confusion matrix and classification report for test

```
0.8341187558906692
[[202 105]
 [ 71 683]]
              precision    recall  f1-score   support

           0       0.74      0.66      0.70       307
           1       0.87      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
```

Out[154]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x201116a65b0>



Tuned KNN

KNN is also performed very good with 83% and 82% train test accuracy score,

KNN train accuracy score

0.8378887841658812

KNN test accuracy score

0.8289473684210527

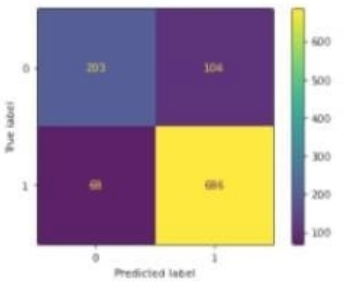
Confusion matrix and classification report for train

```
0.8378887841658812
[[203 104]
 [ 68 686]]
              precision    recall  f1-score   support

           0       0.75      0.66      0.70       307
           1       0.87      0.91      0.89       754

    accuracy                           0.84      1061
   macro avg       0.81      0.79      0.80      1061
weighted avg       0.83      0.84      0.83      1061
```

Out[157]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x201117c1e80>



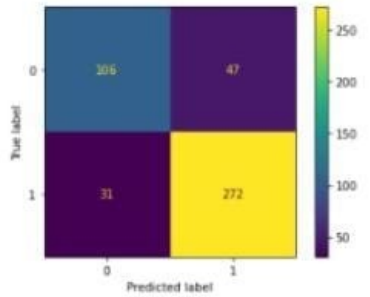
Confusion matrix and classification report for test

```
0.8289473684210527
[[106  47]
 [ 31 272]]
              precision    recall  f1-score   support

           0       0.77      0.69      0.73       153
           1       0.85      0.90      0.87       303

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

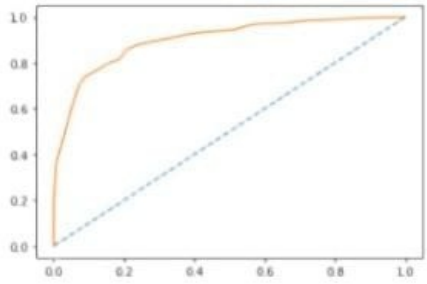
```
Out[158]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2011173e970>
```



AUC and ROC for tuned KNN model for train

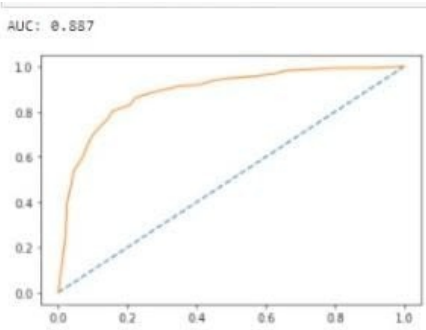
```
AUC: 0.901
```

```
Out[159]: [<matplotlib.lines.Line2D at 0x2011136c0d0>]
```



AUC and ROC for tuned KNN model for test

```
AUC: 0.887
```



Decision Tree pruned

Pruning/tuning DT with gini index and max depth = 4, and the model performance is better And not overfitting with 84% train accuracy and 79% test accuracy.

Decision Tree pruned train accuracy score

0.8444863336475024

Decision Tree pruned test accuracy score

0.7894736842105263

ENsemble Technique - Bagging (Decision tree used)

Decision tree is used as base estimator for Bagging.

Accuracy for train data

0.8623939679547596

Accuracy for test data

0.8245614035087719

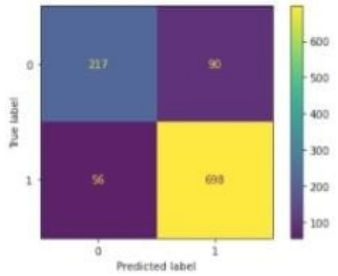
Confusion matrix and classification report for train

```
0.8623939679547596
[[217  90]
 [ 56 698]]
              precision    recall  f1-score   support

           0       0.79      0.71      0.75       307
           1       0.89      0.93      0.91       754

    accuracy                           0.86      1061
   macro avg       0.84      0.82      0.83      1061
weighted avg       0.86      0.86      0.86      1061
```

```
Out[164]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x20110ebe370>
```



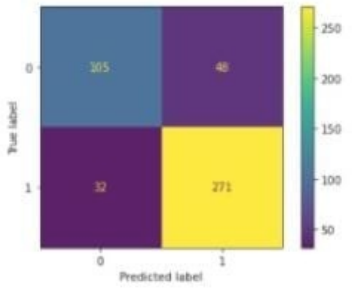
Confusion matrix and classification report for test

```
0.8245614035087719
[[105  48]
 [ 32 271]]
              precision    recall  f1-score   support

           0       0.77      0.69      0.72       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
```

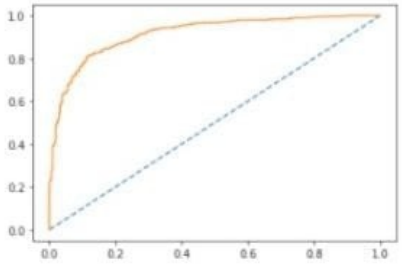
```
Out[165]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x201114a8cd0>
```



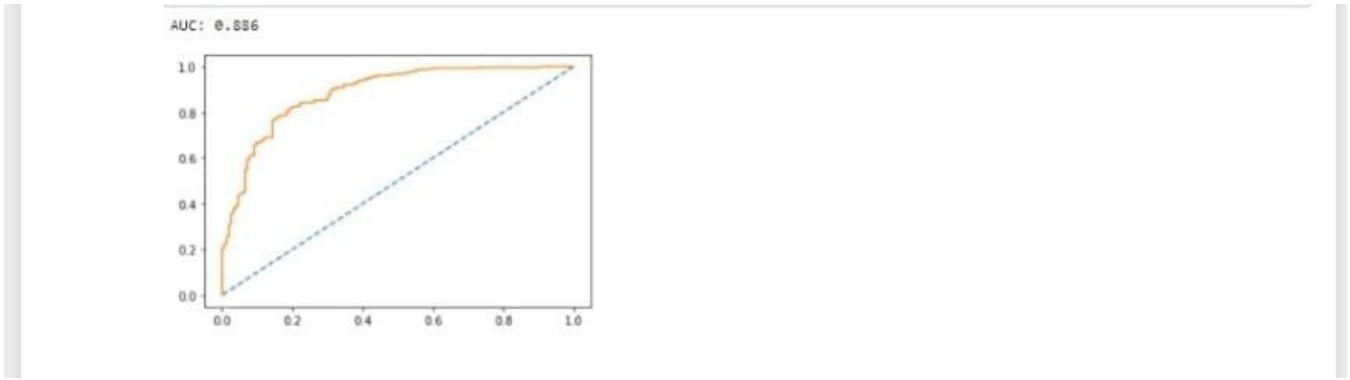
AUC and ROC for train

```
AUC: 0.915
```

```
Out[166]: [<matplotlib.lines.Line2D at 0x20111943430>]
```



AUC and ROC for test



Ada Boosting

Applying Ada Boosting model and predicting the train and test. The train and test accuracy are 85% and 81% respecting. We have seen models that performs better than this.

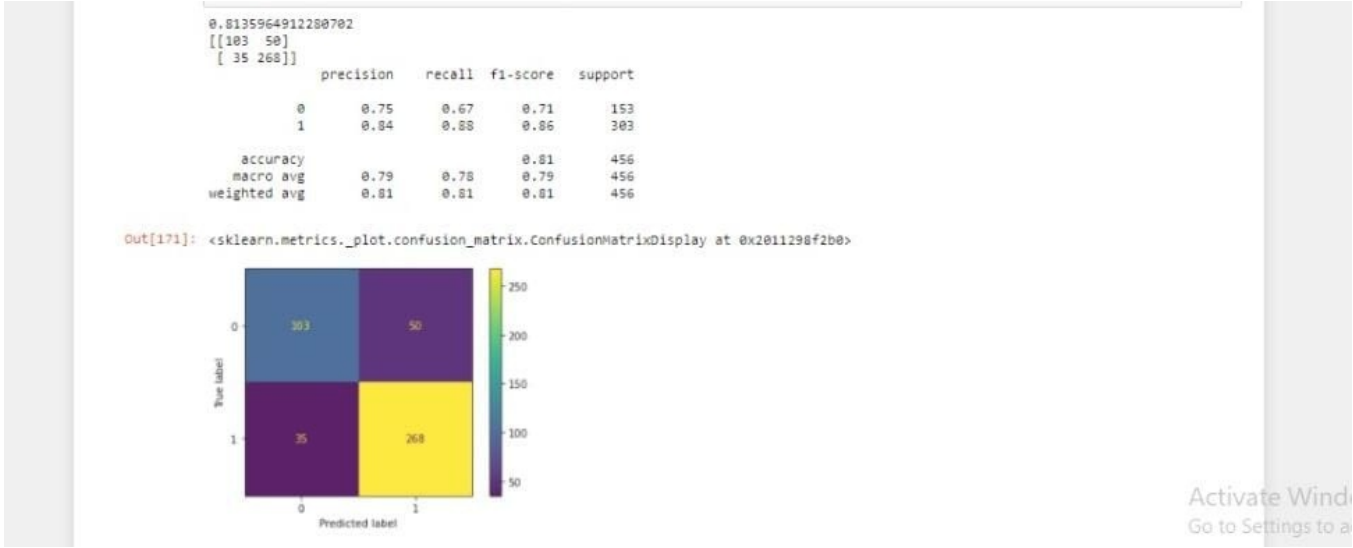
Model score for train

0.8501413760603205

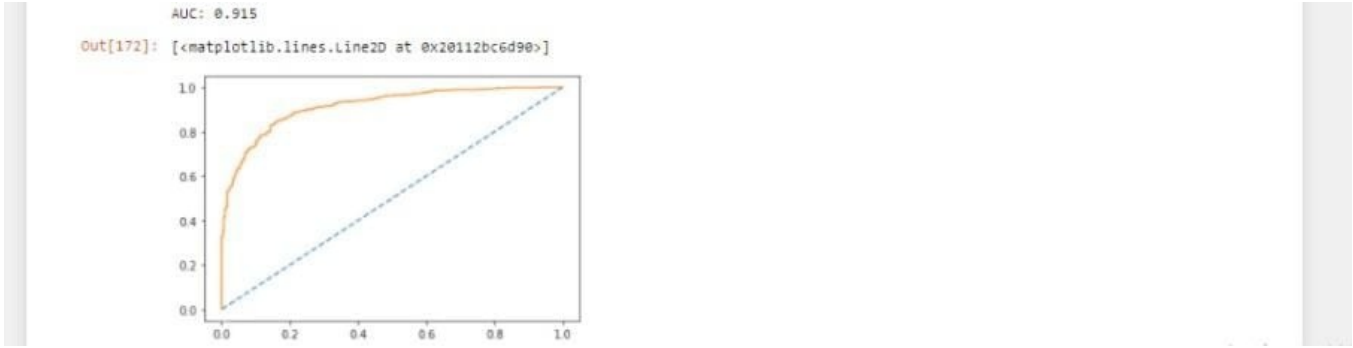
Model score for test

0.8135964912280702

Confusion matrix and cllassification report on test

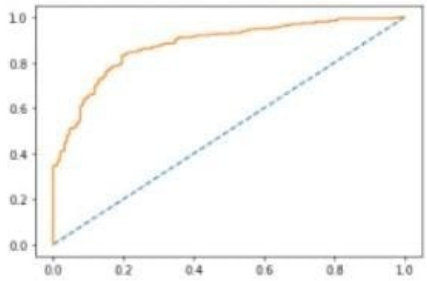


AUC and ROC for train



AUC and ROC for test

```
AUC: 0.877
Out[173]: [<matplotlib.lines.Line2D at 0x20112c1dfd0>]
```



Gradient Boosting

Train accuracy for gradient boosting

0.8925541941564562

test accuracy for gradient boosting

0.8355263157894737

Gradient Boosting model performs the best with 89% train accuracy and with 83% test accuracy. The precision, recall and f1 score is also good

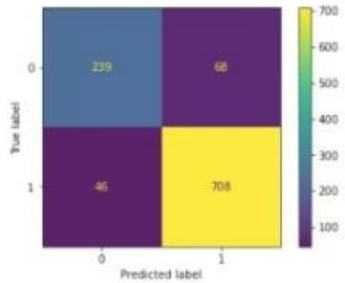
Confusion matrix and cllassification report on train

```
0.8925541941564562
[[239  68]
 [ 46 708]]
              precision    recall  f1-score   support

           0       0.84      0.78      0.81       307
           1       0.91      0.94      0.93       754

    accuracy                           0.89      1061
   macro avg       0.88      0.86      0.87      1061
weighted avg       0.89      0.89      0.89      1061

Out[175]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2011181d00>
```



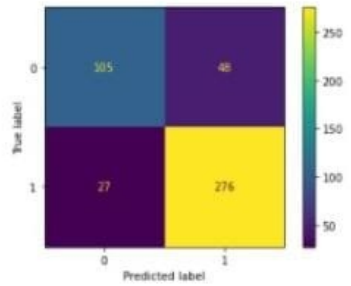
Confusion matrix and cllassification report on test

```
0.8355263157894737
[[105  48]
 [ 27 276]]
              precision    recall  f1-score   support

           0       0.80      0.69      0.74       153
           1       0.85      0.91      0.88       303

    accuracy                           0.84       456
   macro avg       0.82      0.80      0.81       456
weighted avg       0.83      0.84      0.83       456
```

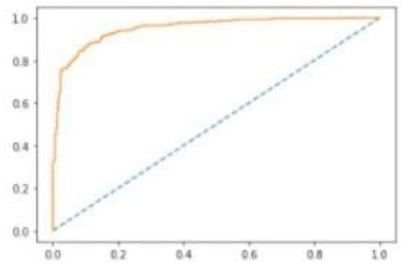
Out[176]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x20112b08580>



AUC and ROC for train and test¶

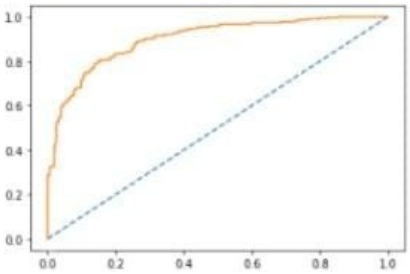
```
AUC: 0.951
```

Out[177]: [<matplotlib.lines.Line2D at 0x20112d8eaf0>]



```
AUC: 0.899
```

Out[178]: [<matplotlib.lines.Line2D at 0x20112de2cd0>]



Random Forest

Random Forest train accuracy score

0.8510838831291234

Random Forest test accuracy score

0.8267543859649122

Random Forest - Bagging

Random Forest (Bagging) train accuracy score

0.8520263901979265

Random Forest (Bagging) test accuracy score

0.8135964912280702 RF model with bagging applied, performs similar to the normal RF as they are not different. The model has good recall and precision also

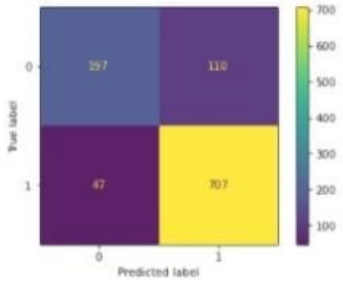
Confusion matrix and classification report for train data

```
0.8623939679547596
[[197 110]
 [ 47 707]]
              precision    recall  f1-score   support

           0       0.81      0.64      0.72       307
           1       0.87      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
```

```
Out[181]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x20112da50a0>
```



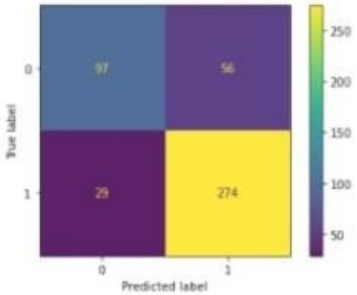
Confusion matrix and classification report for test data

```
0.8245614035087719
[[ 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
```

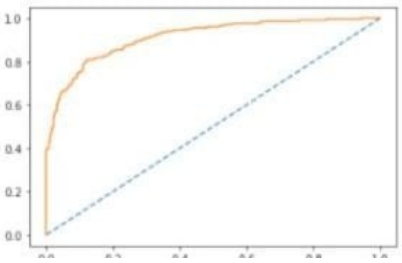
```
Out[182]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x20112dd6e50>
```



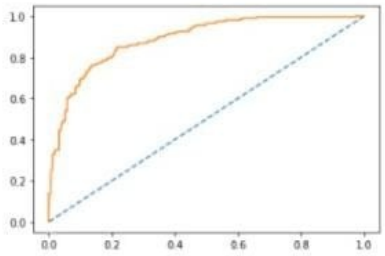
AUC and ROC for train and test data

```
AUC: 0.912
```

```
Out[183]: [<matplotlib.lines.Line2D at 0x20112f5c730>]
```



```
AUC: 0.888
```



Model Comparison and Best Model

Gradient Boosting model performs the best with 89% train accuracy. And also have 91% precision and 94% recall which is better than any other models that we have performed in here with the Election dataset.

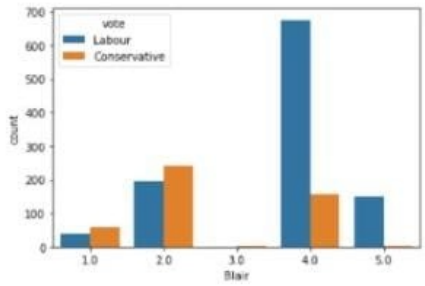
Rest all the models are more or less have same accuracy of 84%

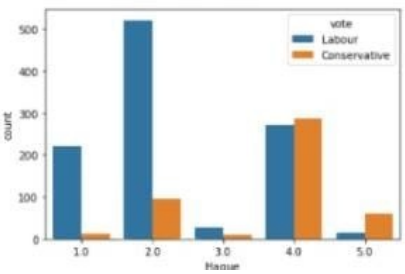
1.8 Based on these predictions, what are the insights?

The important variable in predicting the dependent variables are ‘Blair’ and ‘Hague’.

	Imp
age	0.042957
economic.cond.national	0.063725
economic.cond.household	0.013549
Blair	0.246707
Hague	0.435266
Europe	0.136791
political.knowledge	0.061086
IsMale_or_not	0.000000

These are the ratings that the people gave to the Leaders of the ‘Labour’ and ‘Conservative’ party





As the frequency distribution suggests most of the people gave 4 stars to ‘Blair’ and there are larger number of people gave 2 stars to ‘Hague’ which made an impact in the dependent variable ‘vote’

Problem 2

Problem Statement:

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:

1. President Franklin D. Roosevelt in 1941
2. President John F. Kennedy in 1961
3. President Richard Nixon in 1973 ### 2.1 Find the number of characters, words, and sentences for the mentioned documents. ### Importing the necessary libraries along with the standard import

Number of characters

Number of character in Roosevelt file: 7571

Number of character in Kennedy file: 7618

Number of character in Nixon file: 9991

Number of words in each test file

Number of words in Kennedy file: 1390

Number of words in Nixon file: 1819

Number of words in Roosevelt file: 1360

Number of sentences

Number of sentences in Nixon

	Text	sentences
0	Mr. Vice President, Mr. Speaker, Mr. Chief Jus...	68

Number of sentences in Kennedy

	Text	sentences
0	Vice President Johnson, Mr. Speaker, Mr. Chief...	52

Number of sentences in Roosevelt

	Text	sentences
0	On each national day of inauguration since 178...	67

President Franklin D. Roosevelt’s speech have 7571 Characters (including spaces) and 1360 words.

President John F. Kennedy’s Speech have 7618 Characters (including spaces) and 1390 words.

President Richard Nixon’s Speech have 9991 Characters (including spaces) and 1819 words.

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)

Top three words for Roosevelt

The top three words are Nation , know and Spirit.

Top three words for Kennedy

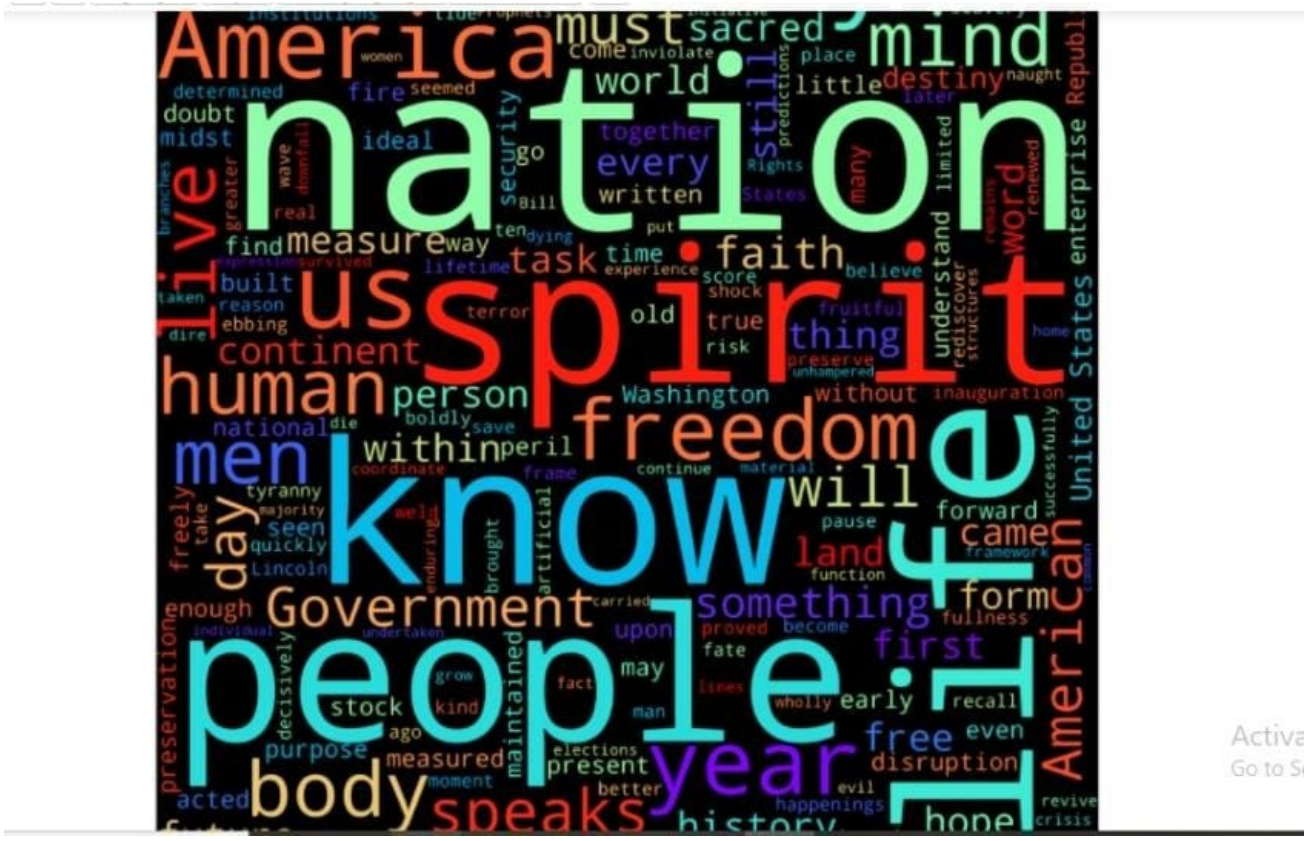
The top three words are Let, Us and World.

Top three words for Nixon

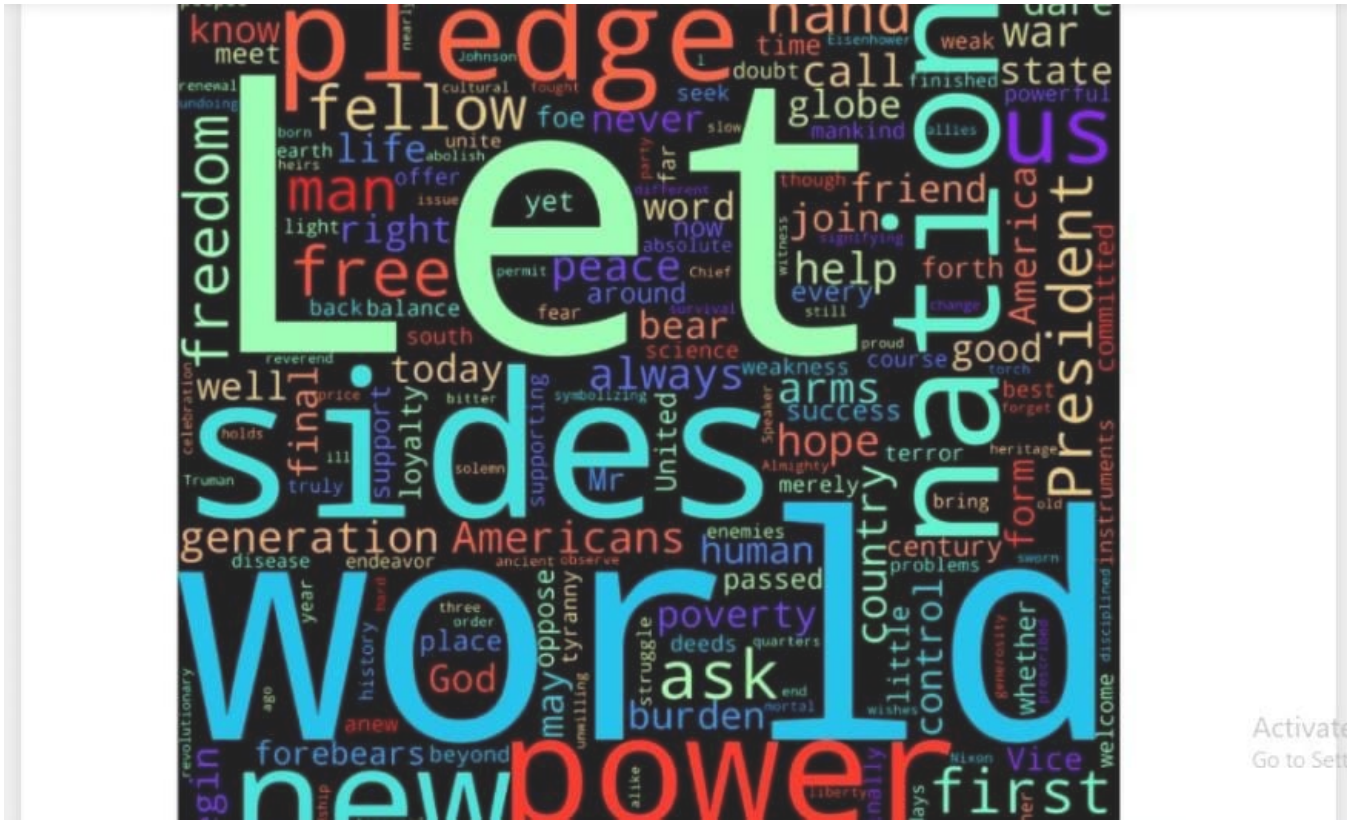
The top three words are Us, Let and America.

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

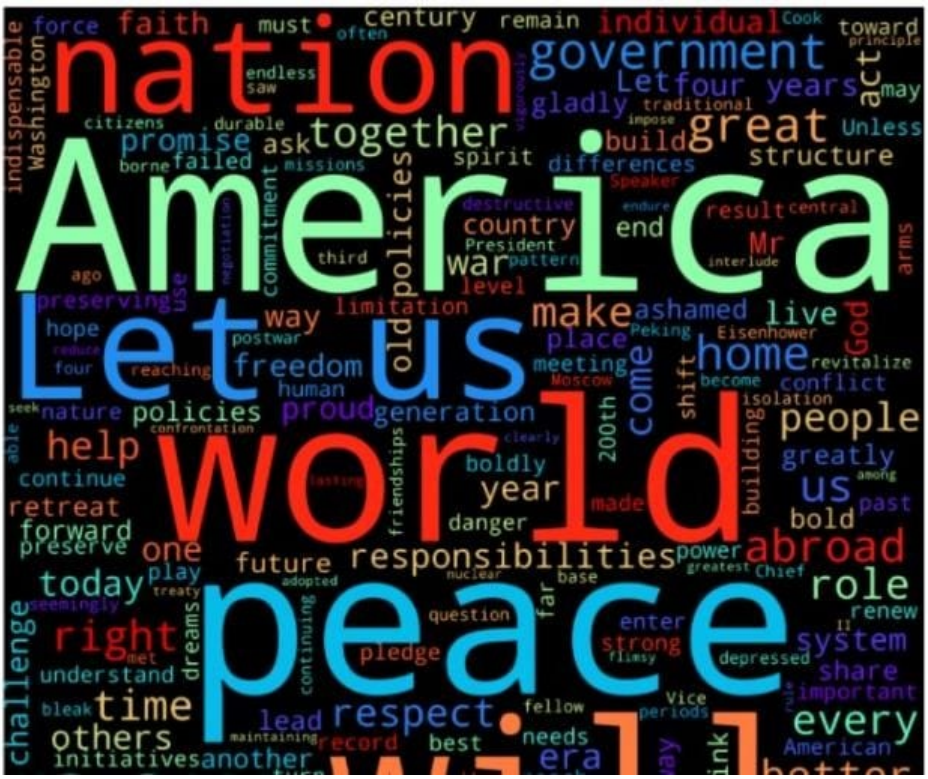
Plotting wordcloud for Roosevelt



Wordcloud for Kennedy



Wordcloud for Nixon



In [ ]: