PROJECT ON MACHINE LEARNING

Prepared By: Barkha Agarwal

PROBLEM STATEMENT-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.

Dataset for Problem: Election_Data.xlsx

Data Ingestion:
1.1 Read the dataset. Do the descriptive statistics and do the null value condition check. Write an inference on it.
1.2 Perform Univariate and Bivariate Analysis. Do exploratory data analysis. Check for Outliers.

Data Preparation:
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).

Modeling:

1.4 Apply Logistic Regression and LDA (linear discriminant analysis).
1.5 Apply KNN Model and Naïve Bayes Model. Interpret the results.
1.6 Model Tuning, Bagging (Random Forest should be applied for Bagging), and Boosting.

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.

Inference:
1.8 Based on these predictions, what are the insights?

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

DATA INFORMATION:

```
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)
```

Observation:

- We have dropped the 'unnamed' column from the dataset as it is not useful for our study.

• Since the ‘vote’ variable is the target, we therefore have ‘vote’ as the dependent and rest 8 variables as the independent or predictor variables.

• The dataset had 8 duplicated values.

• The data set had 1525 rows and 9 columns.

• It has 7 numerical data types and 2 categorical data types.

• There is no null value in any column.

Checking for missing values:

```
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
```

There are no missing values present in the dataset.

Data description:

	count	mean	std	min	25%	50%	75%	max
age	1525.0	54.182295	15.711209	24.0	41.0	53.0	67.0	93.0
economic.cond.national	1525.0	3.245902	0.880969	1.0	3.0	3.0	4.0	5.0
economic.cond.household	1525.0	3.140328	0.929951	1.0	3.0	3.0	4.0	5.0
Blair	1525.0	3.334426	1.174824	1.0	2.0	4.0	4.0	5.0
Hague	1525.0	2.746885	1.230703	1.0	2.0	2.0	4.0	5.0
Europe	1525.0	6.728525	3.297538	1.0	4.0	6.0	10.0	11.0
political.knowledge	1525.0	1.542295	1.083315	0.0	0.0	2.0	2.0	3.0

Checking the skewness of the data:

```
vote                        0.858449
age                         0.144621
economic.cond.national     -0.240453
economic.cond.household    -0.149552
Blair                      -0.535419
Hague                       0.152100
Europe                     -0.135947
political.knowledge        -0.426838
gender                      0.130239
dtype: float64
```

The rule of thumb of skewness seems to be:

- If the skewness is between -0.5 and 0.5, the data are fairly symmetrical.
- If the skewness is between -1 and – 0.5 or between 0.5and 1, the data are moderately skewed.
- If the skewness is less than -1 or greater than 1, the data are highly skewed

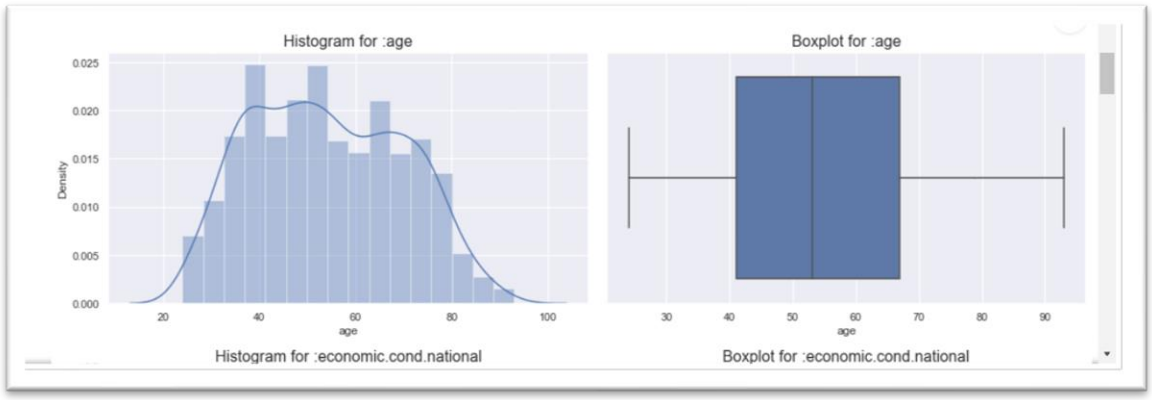
Insights:

- Here, we can see that there isn't much skewness in the data. All the values seem to be between -0.5 and 0.5.
- The value of 'Blair' is a little bit higher than -0.5.
- The data overall, is fairly symmetrical.

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

Univariate Analysis:

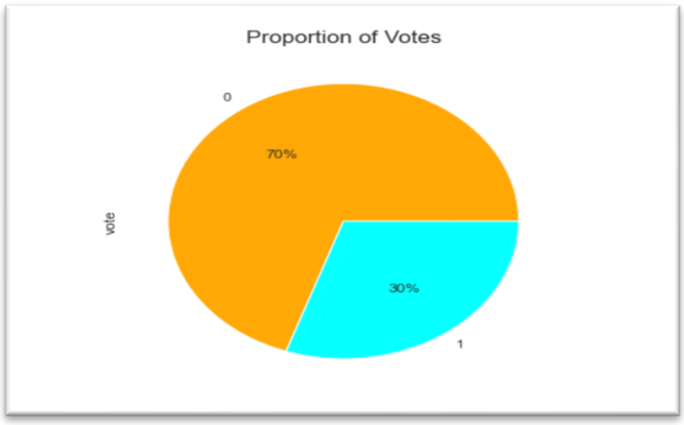
Histogram and box plot of 'age'



Observation:

- The data is normally distributed.
- Maximum number of people are aged between 40 and 70.
- Outliers are not present.

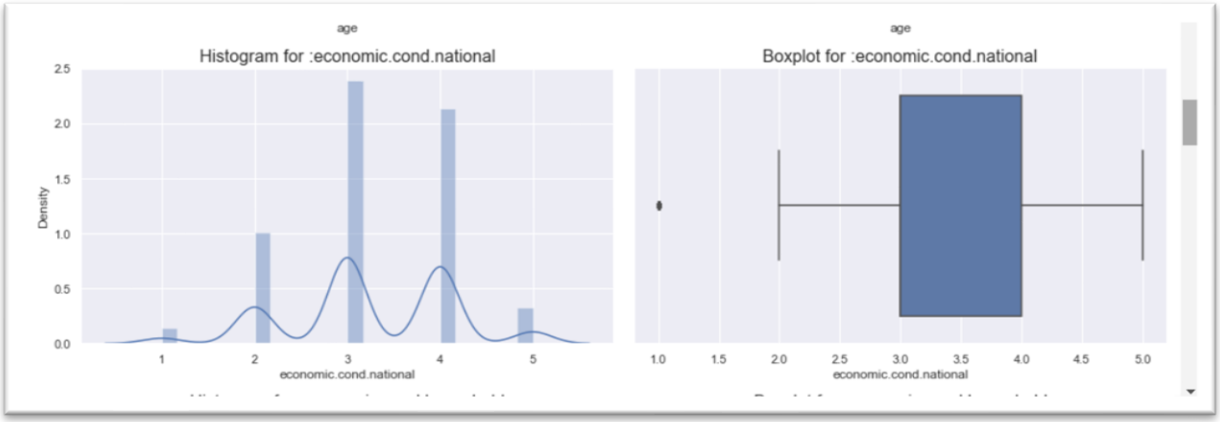
Pie plot of 'vote':



Observation:

- Labour party has higher number of votes. It has more than double the votes of conservative party.
- Labour party has 1057 votes which constitutes 70%.
- Conservative party has 460 votes which constitutes 30%.

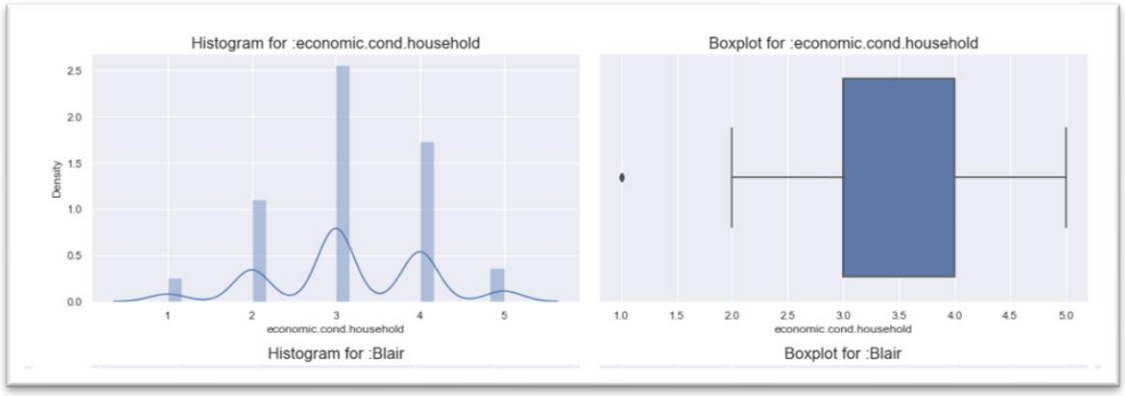
Histogram and box plot of ' economic.cond.national'



Observation:

- The top 2 variables are 3 and 4.
- 1 has the least value which is 37.
- 3 has the highest value which is 604. • 3 is slightly higher than the 2nd highest variable 4.

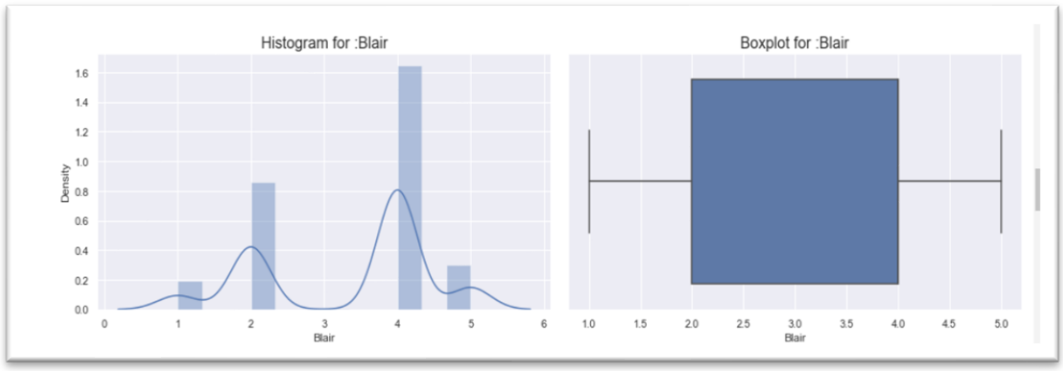
Histogram and box plot of ' economic.cond.household'



Observation:

- The top 2 variables are 3 and 4.
- 1 has the least value which is 65.
- 3 has the highest value which is 645.
- 3 is moderately higher than the 2nd highest variable 4 whose value is 435.

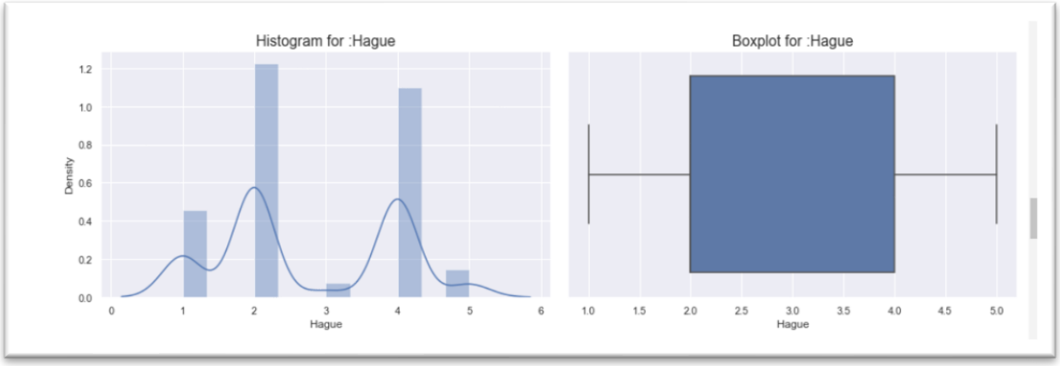
Histogram and box plot of ' Blair'



Observation:

- The top 2 variables are 2 and 4.
- 3 has the least value which is 1.
- 4 has the highest value
- 4 is much higher than the 2nd highest variable 2.

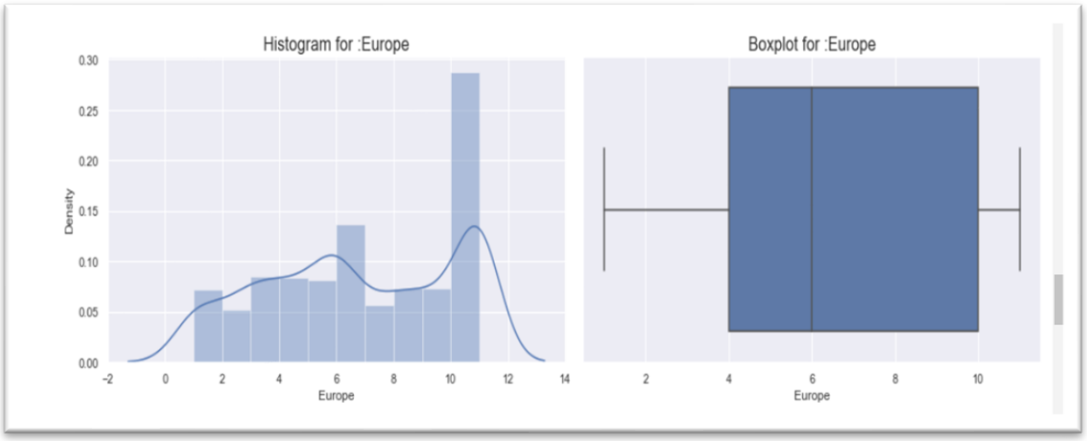
Histogram and box plot of ' Hague’



Observation:

- The top 2 variables are 2 and 4.
- 3 has the least value.
- 2 has the highest value.

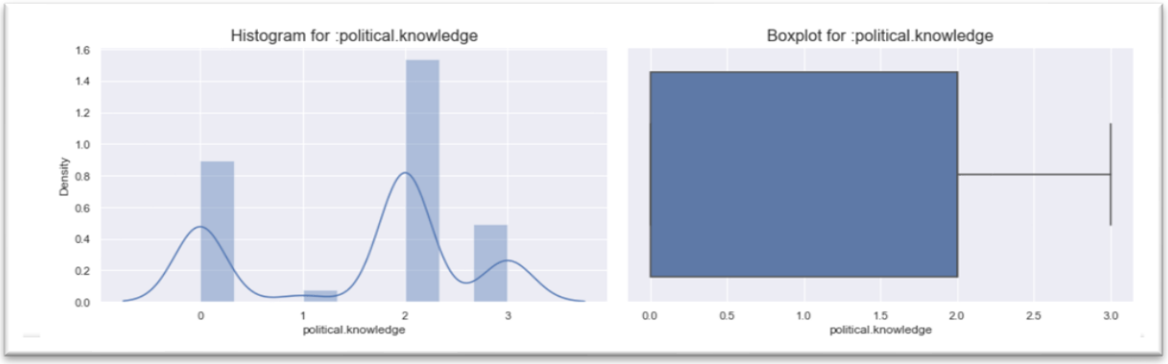
Histogram and box plot of ' Europe’



Observation:

- The top 2 variables are 11 and 6.
- 2 has the least value.
- 11 has the highest.
- 11 is moderately higher than the 2nd highest variable 6.

Histogram and box plot of ' political.knowledge



Observation:

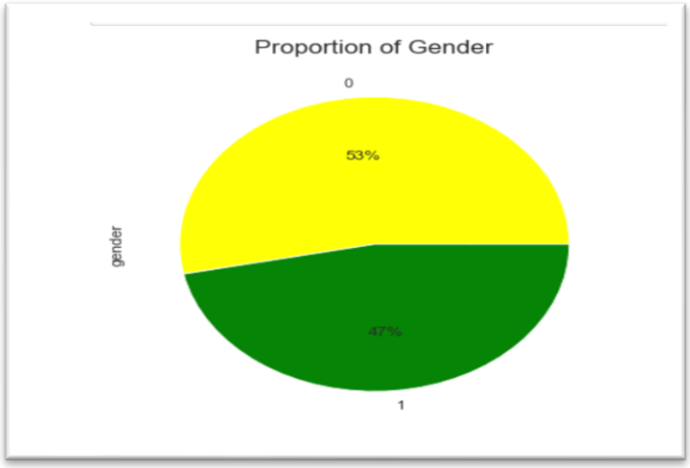
• The top 2 variables are 2 and 0.

• 1 has the least value which is 38. • 2 has the highest value.

• 2 is much higher than the 2nd highest variable 0.

• We can see that, 454 out of 1517 people do not have any knowledge of parties' positions on European integration which is 29.93% of the total population.

Pie plot of gender:



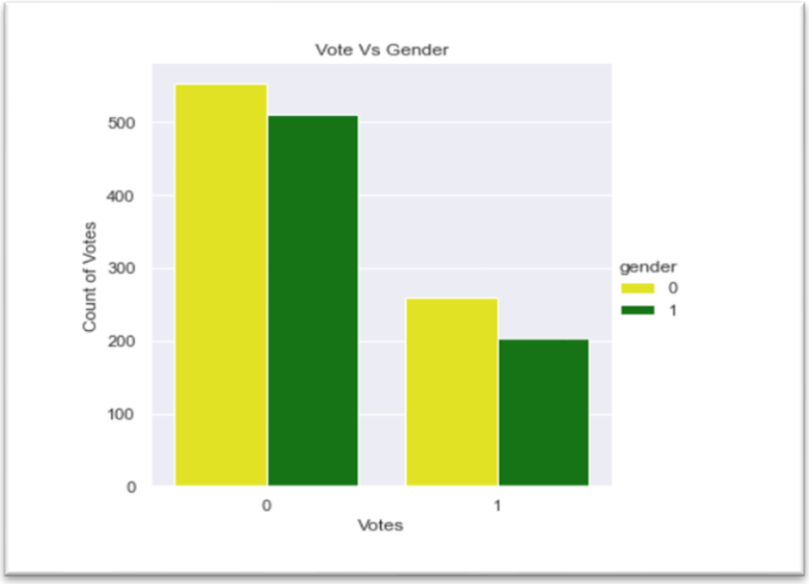
Observation:

• The proportion of female voters are more as compared to male voters.

• The percentage of female is 53% and that of male is 47%.

Bivariate Analysis:

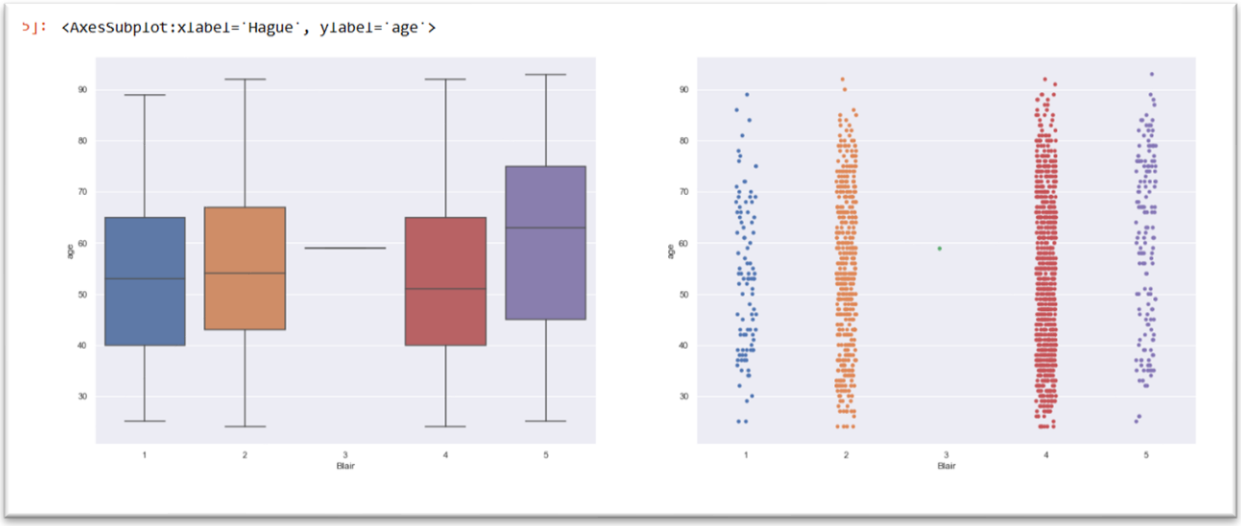
Viewing the exact values of the variables of 'vote' with respect to 'gender'



Observation:

- We can clearly see that, the labour party has got more votes than the conservative party.
- In every age group, the labour party has got more votes than the conservative party.
- Female votes are considerably higher than the male votes in both parties.
- In both genders, the labour party has got more votes than the conservative party.

Boxplot and Strip plot of Blair and 'age':



We have also drawn pair plot to check pair wise distribution of continuous variables.

Observation from pair plot:

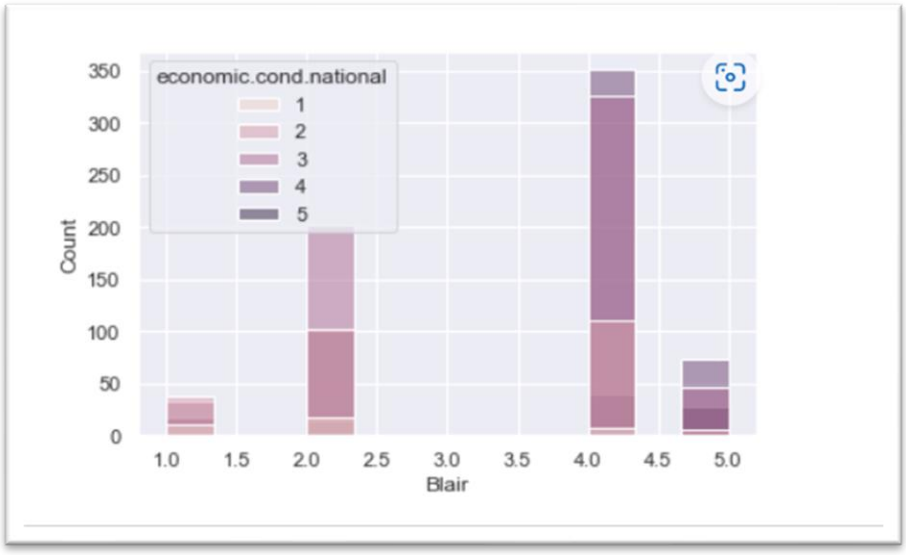
• Pair plot is a combination of histograms and scatter plots.

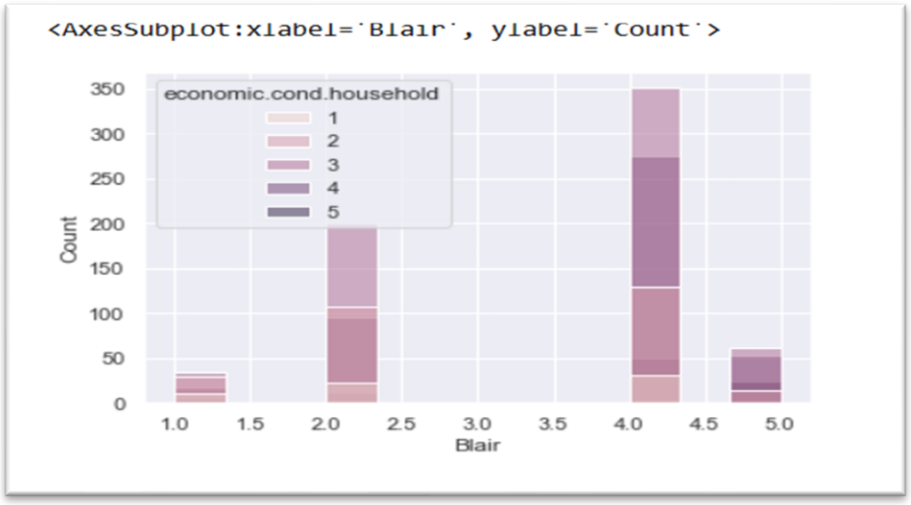
• From the histogram, we can see that, the 'Blair','Europe' and 'political.knowledge' variables are slightly left skewed.

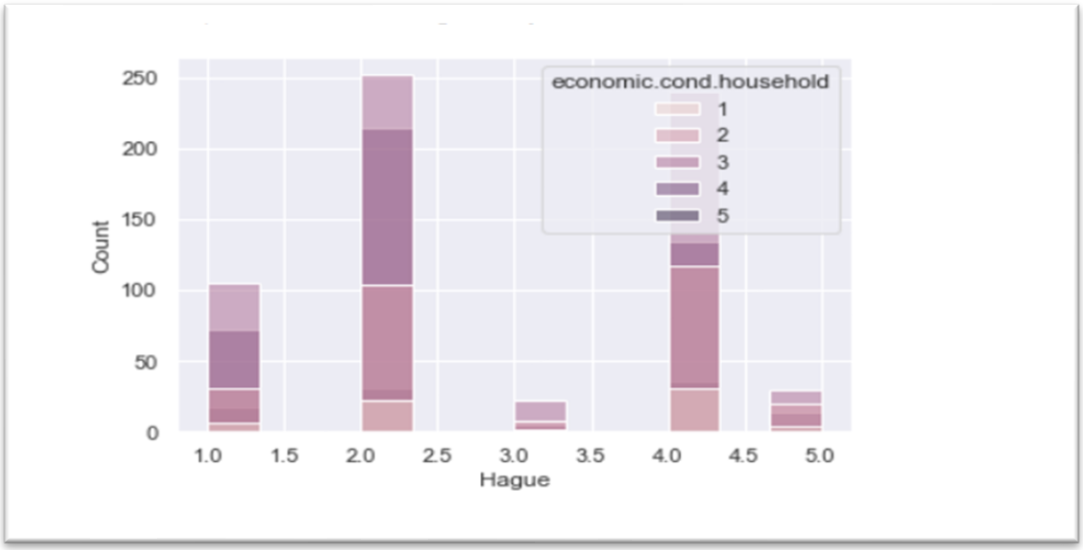
• All other variables seem to be normally distributed.

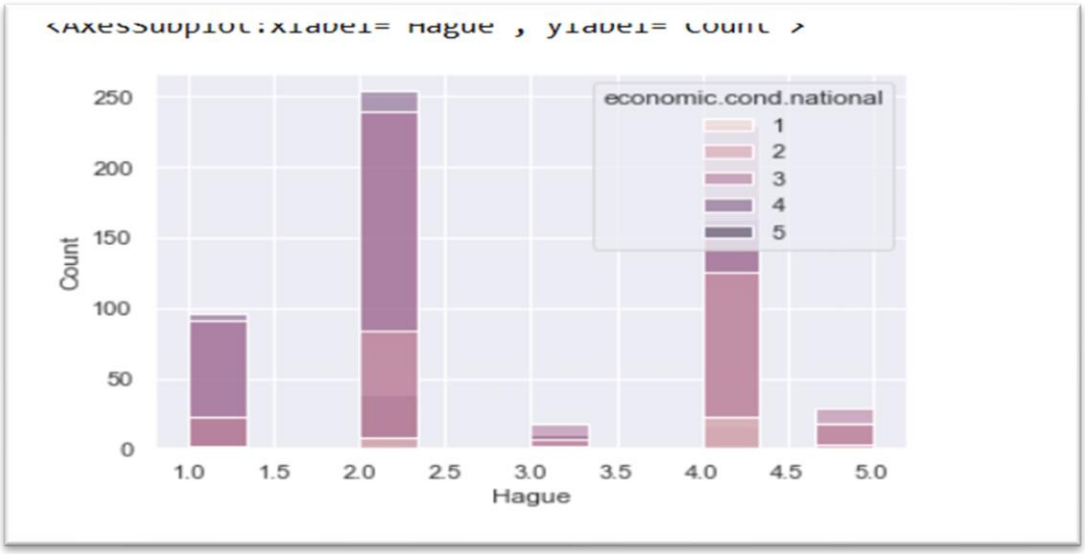
• From the scatter plots, we can see that, there is mostly no correlation between the variables.

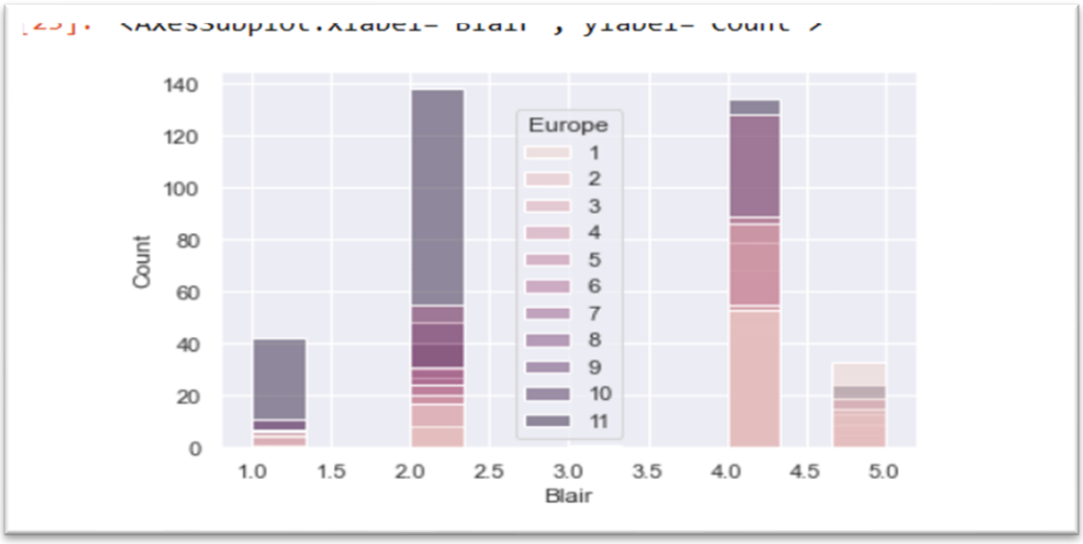
Plotting various histplots with different variables.

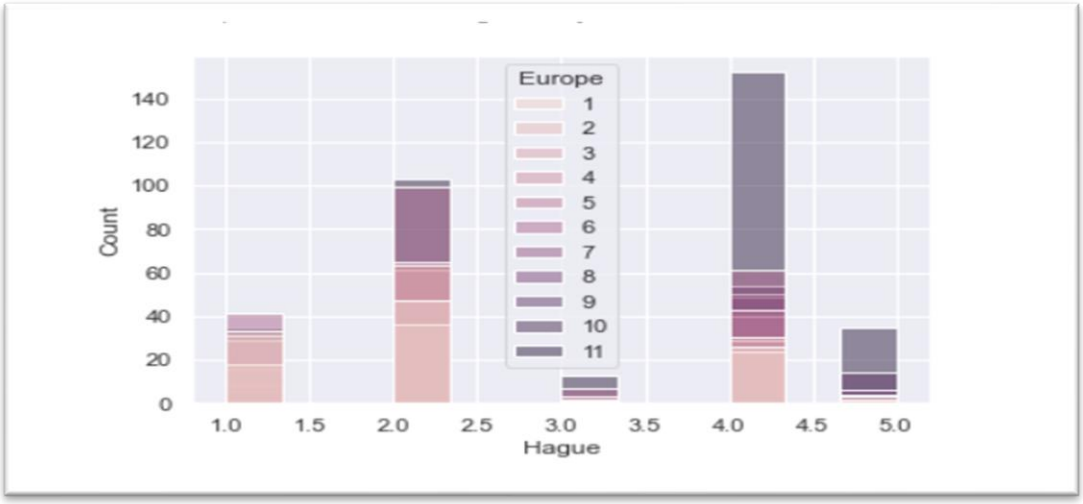


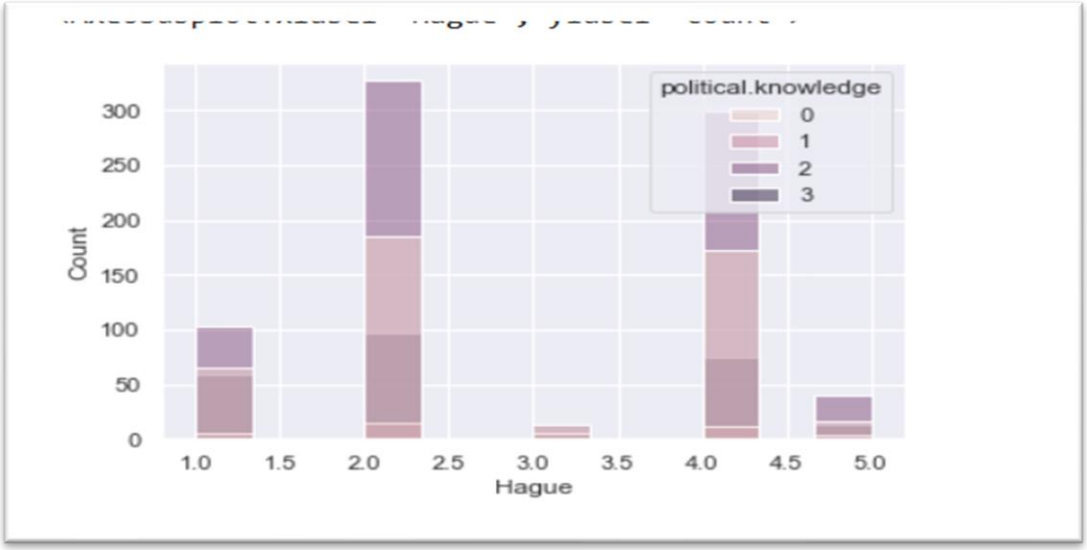



economic.cond.household
1
2
3
4
5
Count
250
200
150
100
50
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Hague


<AxesSubplot:xlabel= Hague , ylabel= Count >
economic.cond.national
1
2
3
4
5
Count
250
200
150
100
50
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Hague


Europe
1
2
3
4
5
6
7
8
9
10
11
Count
Blair
0
20
40
60
80
100
120
140
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

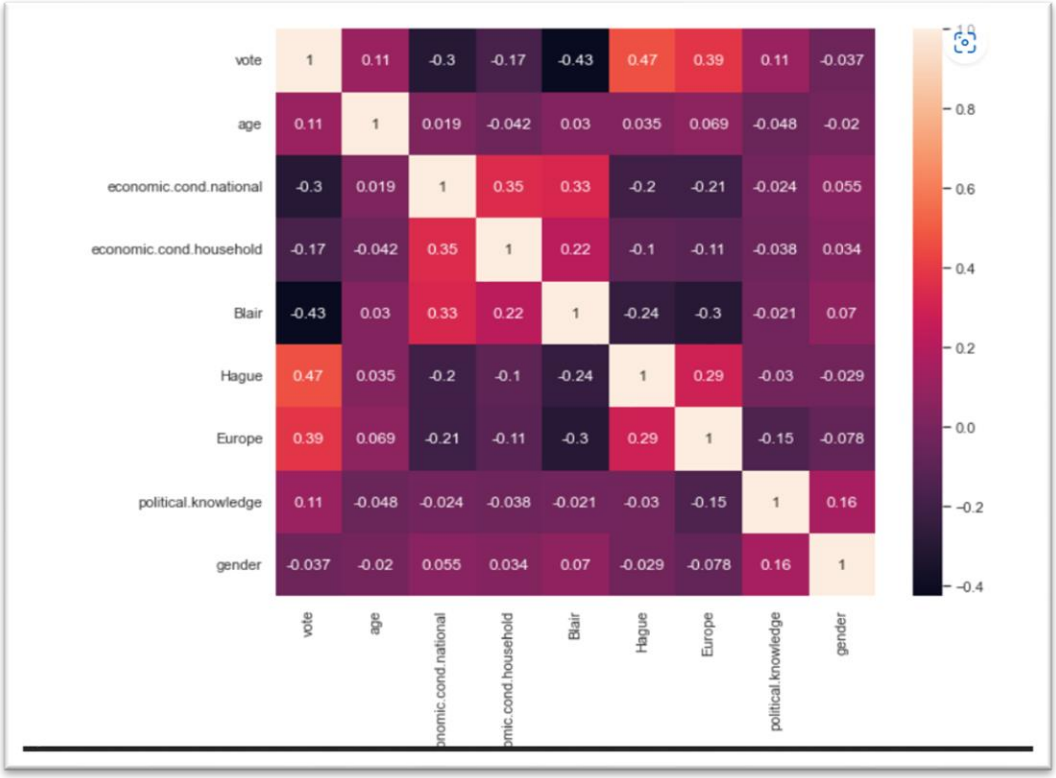

Europe
1
2
3
4
5
6
7
8
9
10
11
Count
Hague
0
20
40
60
80
100
120
140
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


political.knowledge
0
1
2
3
Count
Hague
0
50
100
150
200
250
300
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

We can use the correlation matrix to view them more clearly.

Correlation matrix is a table which shows the correlation coefficient between variables. Correlation values range from -1 to +1. For values closer to zero, it means that, there is no linear trend between two variables. Values close to 1 means that the correlation is positive.

The correlation heat map helps us to visualize the correlation between two variables.



Observation:

• We can see that, mostly there is no correlation in the dataset through this matrix. There are some variables that are moderately positively correlated and some that are slightly negatively correlated.

• ‘economic.cond.national’ with ‘economic.cond.household’ have moderate positive correlation.

• ‘Blair' with 'economic.cond.national' and ‘economic.cond.household’ have moderate positive correlation.

• ‘Europe’ with ‘Hague’ have moderate positive correlation.

• 'Hague' with 'economic.cond.national' and 'Blair' have moderate negative correlation.

• 'Europe' with 'economic.cond.national' and 'Blair' have moderate negative correlation.

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).

Viewing the data before encoding:

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

Viewing the data after encoding:

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

Why scaling ?

• The dataset contains features highly varying in magnitudes, units and range between the 'age' column and other columns.

• But since, most of the machine learning algorithms use Eucledian distance between two data points in their computations, this is a problem.

• If left alone, these algorithms only take in the magnitude of features neglecting the units. • The results would vary greatly between different units.

• The features with high magnitudes will weigh in a lot more in the distance calculations than features with low magnitudes.

• To supress this effect, we need to bring all features to the same level of magnitudes. This can be achieved by scaling.

• In this case, we have a lot of encoded, ordinal, categorical and continuous variables. So, we use the minmax scaler technique to scale the data

Viewing the data after scaling:

]:

	vote	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	gender
1	0.0	0.275362	0.50	0.50	0.75	0.00	0.1	0.666667	0.0
2	0.0	0.173913	0.75	0.75	0.75	0.75	0.4	0.666667	1.0
3	0.0	0.159420	0.75	0.75	1.00	0.25	0.2	0.666667	1.0
4	0.0	0.000000	0.75	0.25	0.25	0.00	0.3	0.000000	0.0
5	0.0	0.246377	0.25	0.25	0.00	0.00	0.5	0.666667	1.0

Train-test-split:

Our model will use all the variables and 'vote' is the target variable. The train-test split is a technique for evaluating the performance of a machine learning algorithm. The procedure involves taking a dataset and dividing it into two subsets.

• Train Dataset: Used to fit the machine learning model.

• Test Dataset: Used to evaluate the fit machine learning model

The data is divided into 2 subsets, training and testing set. Earlier, we have extracted the target variable ‘vote’ in a separate vector for subsets. Random state chosen as 1.

• Training Set: 70percent of data.

• Testing Set: 30 percent of the data.

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

Logistic Regression Model:

There are no outliers present in the continuous variable 'age'. Our model will use all the variables and 'vote' is the target variable.

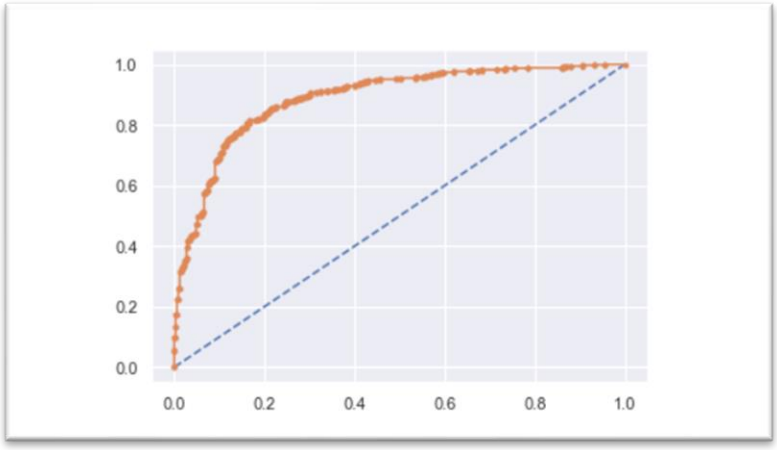
Accuracy - Train data: 0.84

Accuracy - Test data: 0.82

Classification report - Train data:

	precision	recall	f1-score	support
0	0.77	0.68	0.72	332
1	0.86	0.91	0.89	735
accuracy			0.84	1067
macro avg	0.82	0.79	0.80	1067
weighted avg	0.83	0.84	0.83	1067

AUC and ROC for the training data:

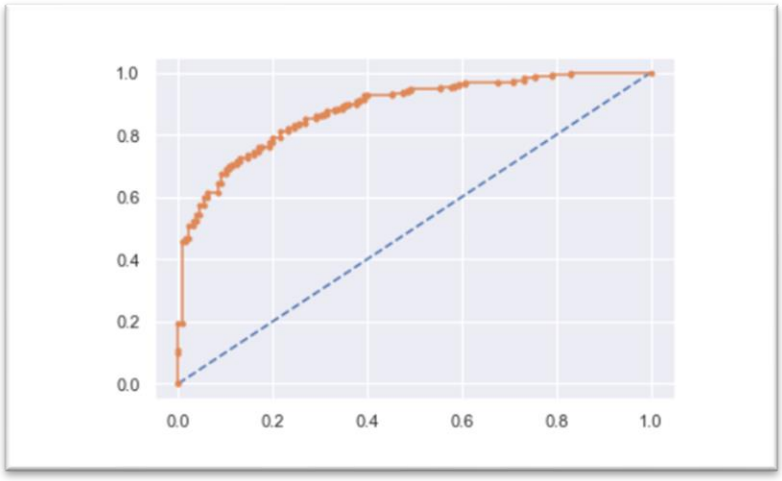


Classification report - Test data:

```
[[ 84  46]
 [ 36 292]]
```

	precision	recall	f1-score	support
0	0.70	0.65	0.67	130
1	0.86	0.89	0.88	328
accuracy			0.82	458
macro avg	0.78	0.77	0.77	458
weighted avg	0.82	0.82	0.82	458

AUC and ROC for the test data:



We can see from this model that neither it is over fitted or under fitted.

Linear Discriminant Analysis Model:

Accuracy - Train data: 0.84

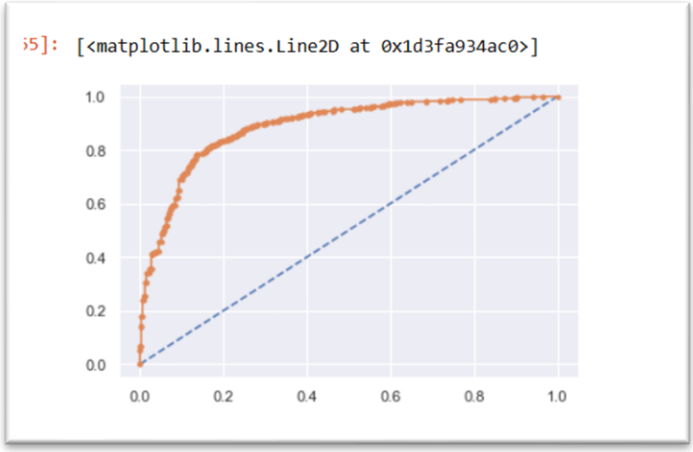
Accuracy - Test data: 0.82

Classification report - Train data:

```
[[233  75]
 [ 99 660]]
              precision    recall  f1-score   support

           0       0.70      0.76      0.73       308
           1       0.90      0.87      0.88       759

    accuracy                           0.84      1067
   macro avg       0.80      0.81      0.81      1067
weighted avg       0.84      0.84      0.84      1067
```

AUC and ROC for the training data:

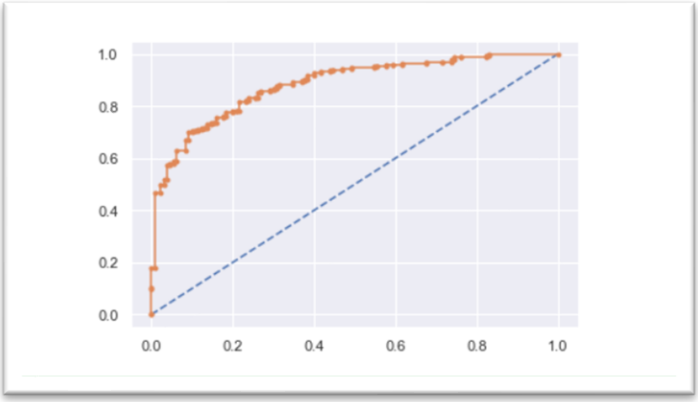
```
[<matplotlib.lines.Line2D at 0x1d3fa934ac0>]
```



Classification report - Test data:

```
[[ 86  39]
 [ 44 289]]
              precision    recall  f1-score   support

           0       0.66      0.69      0.67       125
           1       0.88      0.87      0.87       333

    accuracy                           0.82       458
   macro avg       0.77      0.78      0.77       458
weighted avg       0.82      0.82      0.82       458
```

AUC and ROC for the test data:



The model is not over-fitted or under-fitted.

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

K-Nearest Neighbor Model:

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point.

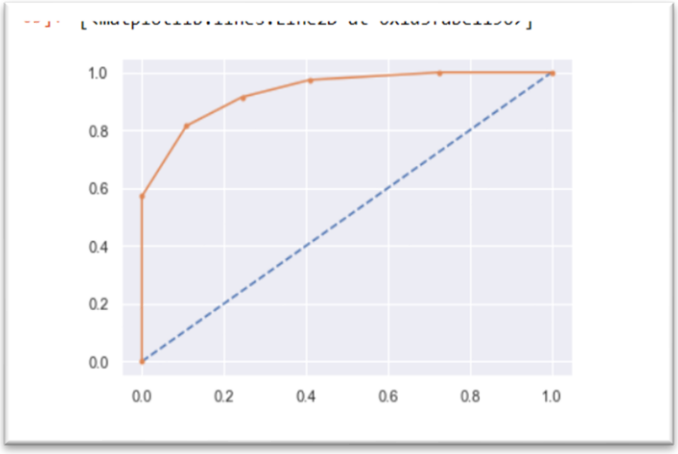
Accuracy - Train data: 0.84

Accuracy - Test data: 0.82

Classification report - Train data:

```
0.8369259606373008
[[233  99]
 [ 75 660]]
              precision    recall  f1-score   support

           0       0.76      0.70      0.73       332
           1       0.87      0.90      0.88       735

    accuracy                           0.84      1067
   macro avg       0.81      0.80      0.81      1067
weighted avg       0.83      0.84      0.84      1067
```

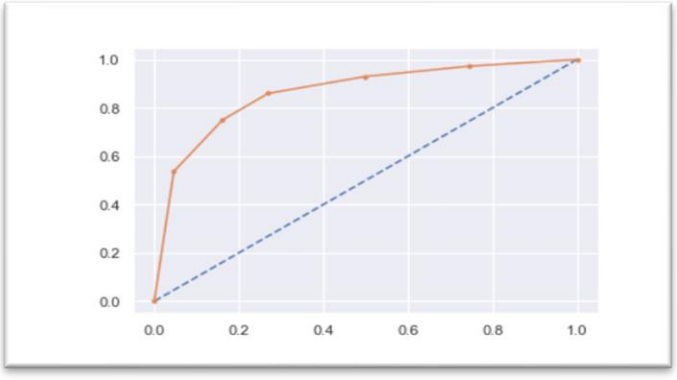
AUC and ROC for the training data:



Classification report - Test data:

```
0.8187772925764192
[[ 86  44]
 [ 39 289]]
              precision    recall  f1-score   support

           0       0.69      0.66      0.67       130
           1       0.87      0.88      0.87       328

    accuracy                           0.82       458
   macro avg       0.78      0.77      0.77       458
weighted avg       0.82      0.82      0.82       458
```

AUC and ROC for the training data:



The model is neither over fitted or under fitted as the difference is less than 10%.

Naïve Bayes Model:

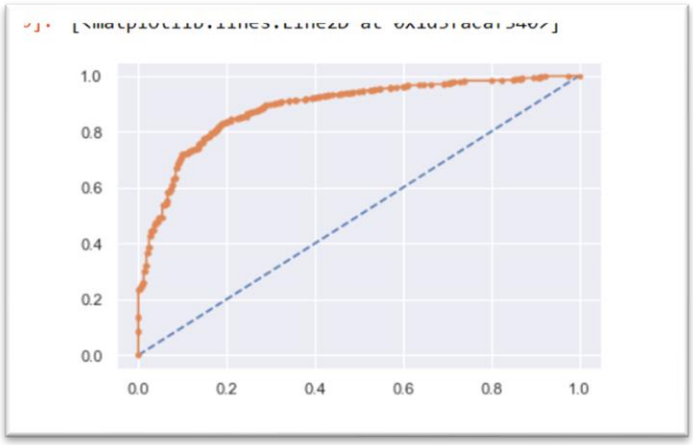
Accuracy - Train data: 0.83

Accuracy - Test data: 0.83

Classification report - Train data:

```
[[240  86]
 [ 92 649]]
              precision    recall  f1-score   support

           0       0.72      0.74      0.73       326
           1       0.88      0.88      0.88       741

    accuracy                           0.83      1067
   macro avg       0.80      0.81      0.80      1067
weighted avg       0.83      0.83      0.83      1067
```

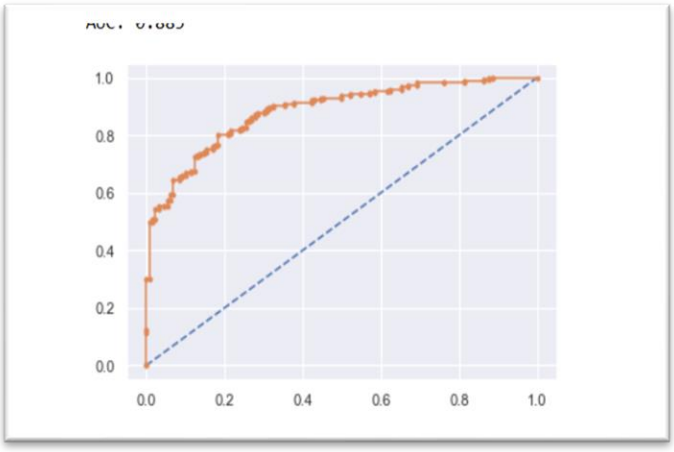
AUC and ROC for the training data:



Classification report - Test data:

```
[[ 94  44]
 [ 36 284]]
              precision    recall  f1-score   support

           0       0.72      0.68      0.70       138
           1       0.87      0.89      0.88       320

    accuracy                           0.83       458
   macro avg       0.79      0.78      0.79       458
weighted avg       0.82      0.83      0.82       458
```

AUC and ROC for the test data:



The model is neither over fitted or under fitted as the difference is less than 10%.

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

We will perform SMOTE here using the Naive Bayes with SMOTE and KNN with SMOTE. Synthetic Minority Oversampling Technique (SMOTE) is a statistical technique for increasing the number of cases in your dataset in a balanced way.

Naive Bayes with SMOTE:

Accuracy - Train data: 0.82

Accuracy - Test data: 0.79

Classification report - Train data:

```
0.8217687074829932
[[596 139]
 [123 612]]
              precision    recall  f1-score   support

           0       0.83      0.81      0.82       735
           1       0.81      0.83      0.82       735

    accuracy                           0.82      1470
   macro avg       0.82      0.82      0.82      1470
weighted avg       0.82      0.82      0.82      1470
```

Classification report - Test data:

```
0.7882096069868996
[[103  27]
 [ 70 258]]
              precision    recall  f1-score   support

           0       0.60      0.79      0.68       130
           1       0.91      0.79      0.84       328

    accuracy                           0.79       458
   macro avg       0.75      0.79      0.76       458
weighted avg       0.82      0.79      0.80       458
```

KNN with SMOTE:

Accuracy - Train data: 0.89

Accuracy - Test data: 0.77

Classification report - Train data:

```
0.889795918367347
[[690  45]
 [117 618]]
              precision    recall  f1-score   support

           0       0.86      0.94      0.89       735
           1       0.93      0.84      0.88       735

    accuracy                           0.89      1470
   macro avg       0.89      0.89      0.89      1470
weighted avg       0.89      0.89      0.89      1470
```

Classification report - Test data:

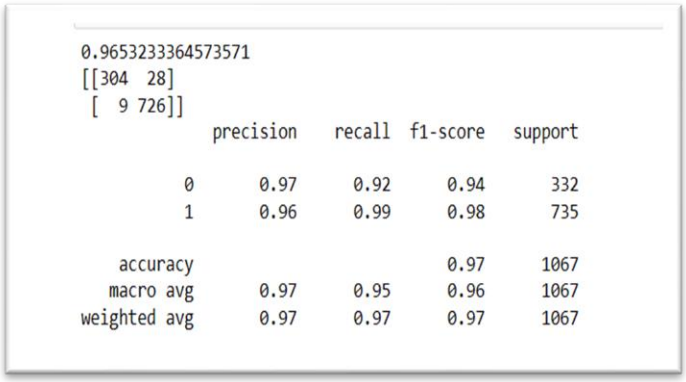
```
0.7663755458515283
[[106  24]
 [ 83 245]]
              precision    recall  f1-score   support

           0       0.56      0.82      0.66       130
           1       0.91      0.75      0.82       328

    accuracy                           0.77       458
   macro avg       0.74      0.78      0.74       458
weighted avg       0.81      0.77      0.78       458
```

Ensemble technique – Bagging(Random Forest):

Accuracy - Train data: 0.97

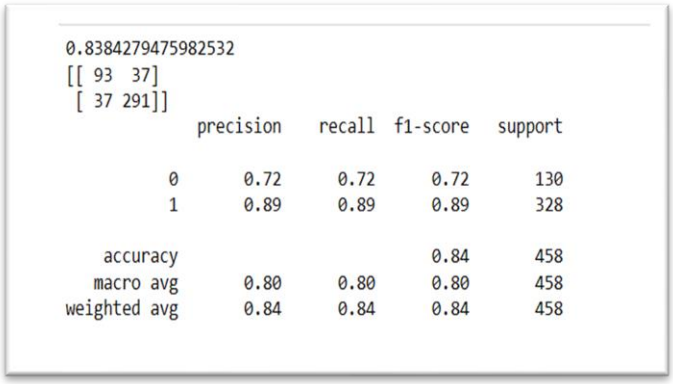
Accuracy - Test data: 0.83

Classification report - Train data:



```
0.9653233364573571
[[304  28]
 [  9 726]]
              precision    recall  f1-score   support

           0       0.97      0.92      0.94       332
           1       0.96      0.99      0.98       735

    accuracy                           0.97      1067
   macro avg       0.97      0.95      0.96      1067
weighted avg       0.97      0.97      0.97      1067
```

Classification report - Test data:



```
0.8384279475982532
[[ 93  37]
 [ 37 291]]
              precision    recall  f1-score   support

           0       0.72      0.72      0.72       130
           1       0.89      0.89      0.89       328

    accuracy                           0.84       458
   macro avg       0.80      0.80      0.80       458
weighted avg       0.84      0.84      0.84       458
```

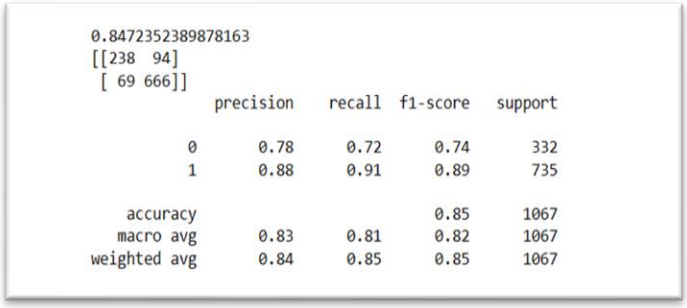
The Random Forest model even after using bagging technique, is still over-fitted.

Ensemble technique – Ada Boosting:

Accuracy - Train data: 0.85

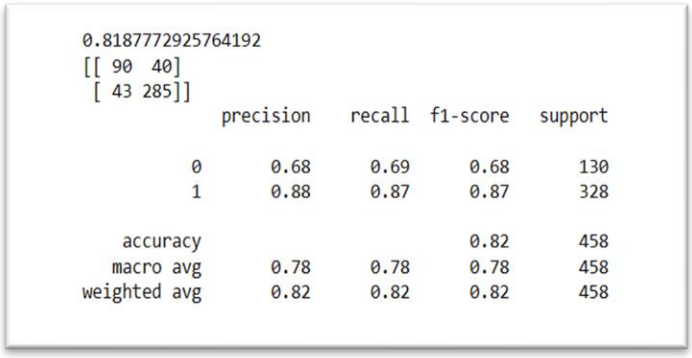
Accuracy - Test data: 0.82

Classification report - Train data:



```
0.8472352389878163
[[238  94]
 [ 69 666]]
              precision    recall  f1-score   support

           0       0.78      0.72      0.74       332
           1       0.88      0.91      0.89       735

    accuracy                           0.85      1067
   macro avg       0.83      0.81      0.82      1067
weighted avg       0.84      0.85      0.85      1067
```

Classification report - Test data:



```
0.8187772925764192
[[ 90  40]
 [ 43 285]]
              precision    recall  f1-score   support

           0       0.68      0.69      0.68       130
           1       0.88      0.87      0.87       328

    accuracy                           0.82       458
   macro avg       0.78      0.78      0.78       458
weighted avg       0.82      0.82      0.82       458
```

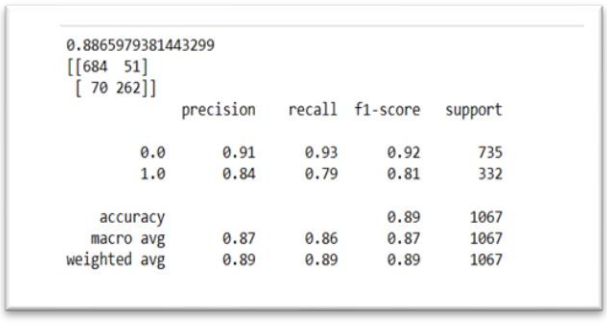
The model is not over-fitted. The values are good. Therefore, the model is a good model.

<u>Ensemble technique - Gradient Boosting:</u>

Accuracy - Train data: 0.89

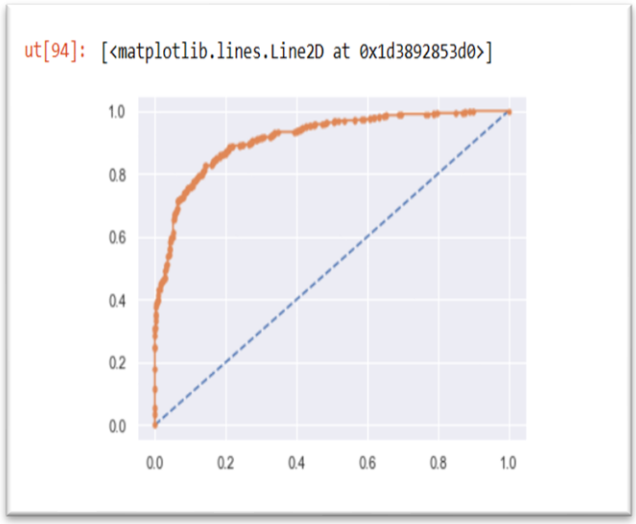
Accuracy - Test data: 0.83

Classification report - Train data:

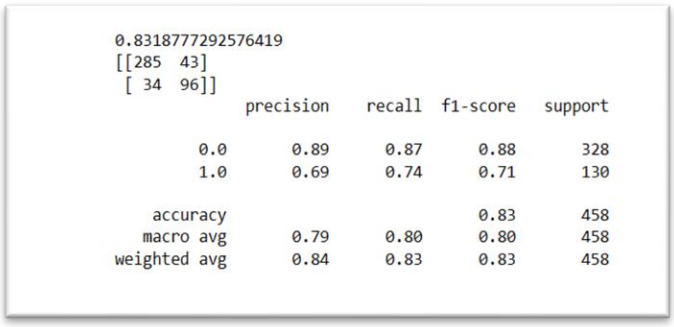


```
0.8865979381443299
[[684  51]
 [ 70 262]]
              precision    recall  f1-score   support

         0.0       0.91      0.93      0.92       735
         1.0       0.84      0.79      0.81       332

    accuracy                           0.89      1067
   macro avg       0.87      0.86      0.87      1067
weighted avg       0.89      0.89      0.89      1067
```

AUC and ROC for the training data:



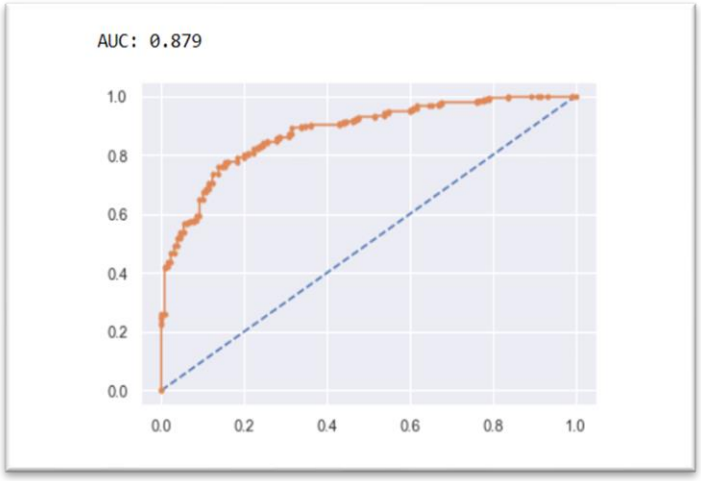
Classification report - Test data:



0.8318777292576419

```
[[285  43]
 [ 34  96]]
              precision    recall  f1-score   support

         0.0       0.89      0.87      0.88       328
         1.0       0.69      0.74      0.71       130

    accuracy                           0.83       458
   macro avg       0.79      0.80      0.80       458
weighted avg       0.84      0.83      0.83       458
```

AUC and ROC for the test data:



The model is not over-fitted. The values are all most same as Ada Boosting model. The model is a good model.

Cross Validation on Naive Bayes Model

Cross-validation is a technique for evaluating ML models by training respective models on subsets of the available input data and evaluating them on the complementary subset of the data. Use cross-validation to detect overfitting, ie, failing to generalize a pattern.

Training set:

```
]: array([0.80952381, 0.79591837, 0.83673469, 0.82312925, 0.82993197,
          0.84353741, 0.76870748, 0.79591837, 0.82312925, 0.82993197])
```

Testing set:

```
105]: array([0.82608696, 0.84782609, 0.82608696, 0.80434783, 0.76086957,
             0.80434783, 0.84782609, 0.91304348, 0.88888889, 0.82222222])
```

After 10 fold cross validation, scores both on train and test data set respectively for all 10 folds are almost same. Hence our model is valid.

Model Tuning using hyper parameters

Model tuning is the experimental process of finding the optimal values of hyperparameters to maximize model performance. Hyperparameters are the set of variables whose values cannot be estimated by the model from the training data. These values control the training process.

Here, we will use Gris search for model tuning. One method is to try out different values and then pick the value that gives the best score. This technique is known as a grid search. If we had to select the values for two or more parameters, we would evaluate all combinations of the sets of values thus forming a grid of values.

LOGISTIC REGRESSION USING GRID SEARCH:

Best parameters:

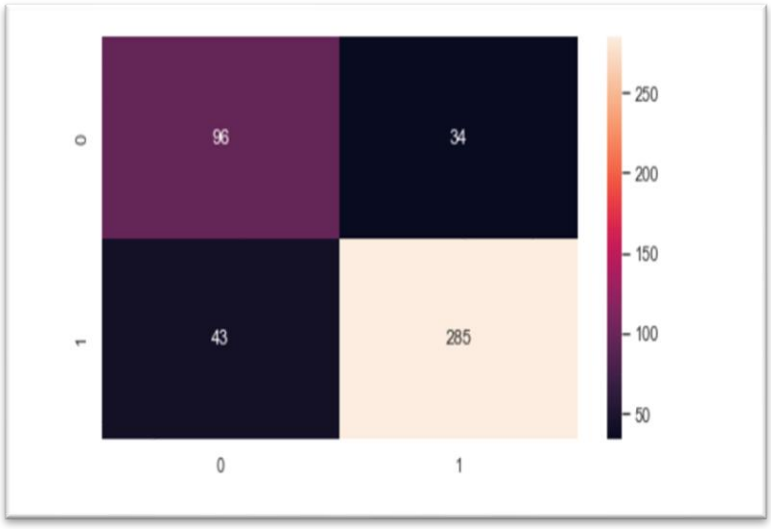
```
107]: GridSearchCV(estimator=LogisticRegression(max_iter=2000, random_state=1),
                   param_grid={'C': [0.001, 0.009, 0.01, 0.09, 1, 5, 10, 25],
                               'penalty': ['l1', 'l2']},
                   scoring='recall')
```

Accuracy - Test data: 0.83

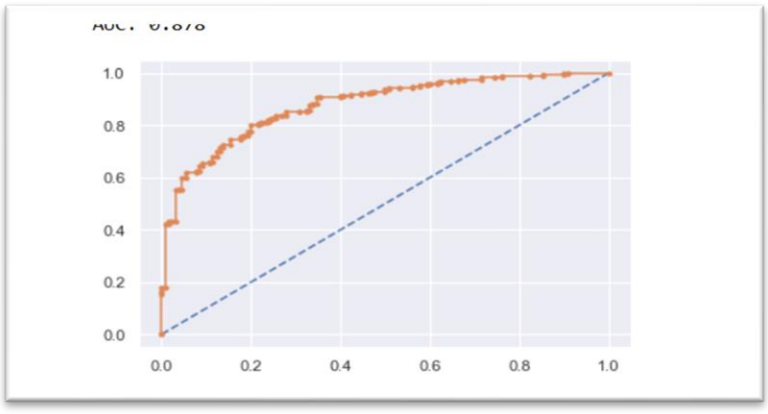
Classification report - Test data:

```
0.7161572052401747
[[ 96  34]
 [ 43 285]]
              precision    recall  f1-score   support

           0       0.69      0.74      0.71       130
           1       0.89      0.87      0.88       328

    accuracy                           0.83       458
   macro avg       0.79      0.80      0.80       458
weighted avg       0.84      0.83      0.83       458
```

Confusion Matrix:



AUC and ROC for the test data:



Comparison on performance of both regular and tuned logistic regression models:

TEST DATA	REGULAR MODEL(%)	TUNED MODEL(%)
ACCURACY	0.82	0.83
PRECISION	0.86	0.89
RECALL	0.89	0.87
F1-SCORE	0.88	0.88

As we can see from the above tabular comparison, there is not much difference between the performance regular LR model and tuned LR model.

• The values are high overall and there is no over-fitting or under-fitting. Therefore both models are equally good models.

LINEAR DISCRIMINANT ANALYSIS USING GRID SEARCH:

Best parameters:

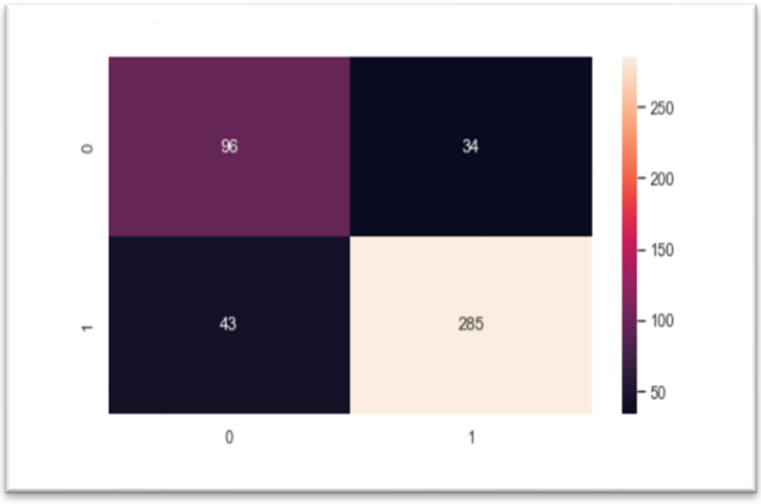
```
GridSearchCV(estimator=LinearDiscriminantAnalysis(),
             param_grid={'solver': ['svd', 'lsqr', 'eigen']}, scoring='recall')
```

Accuracy - Test data: 0.83

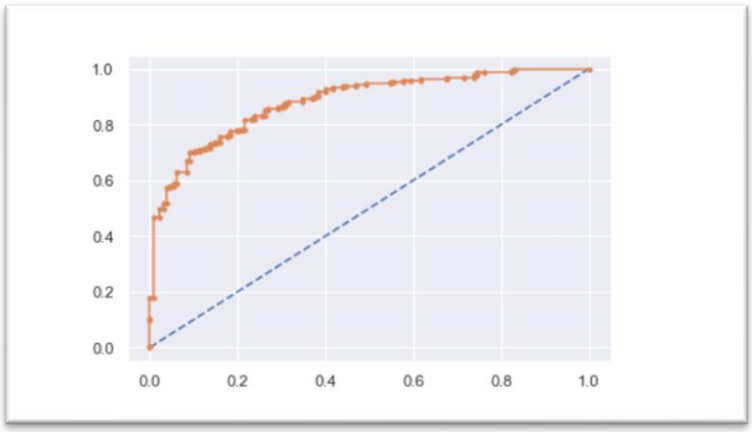
Classification report - Test data:

```
0.8187772925764192
[[ 96  34]
 [ 43 285]]
              precision    recall  f1-score   support

           0       0.69      0.74      0.71       130
           1       0.89      0.87      0.88       328

    accuracy                           0.83       458
   macro avg       0.79      0.80      0.80       458
weighted avg       0.84      0.83      0.83       458
```

Confusion Matrix:



AUC and ROC for the test data:



Comparison on performance of both regular and tuned linear discriminant analysis models:

TEST DATA	**REGULAR MODEL(%)**	**TUNED MODEL(%)**
ACCURACY	0.82	0.83
PRECISION	0.88	0.89
RECALL	0.87	0.87
F1-SCORE	0.87	0.88

As we can see from the above tabular comparison, there is not much difference between the performance of regular LDA model and tuned LDA model.

• The values are high overall and there is no over-fitting or under-fitting. Therefore, both models are equally good models.

KNN USING GRID SEARCH:

Best parameters:

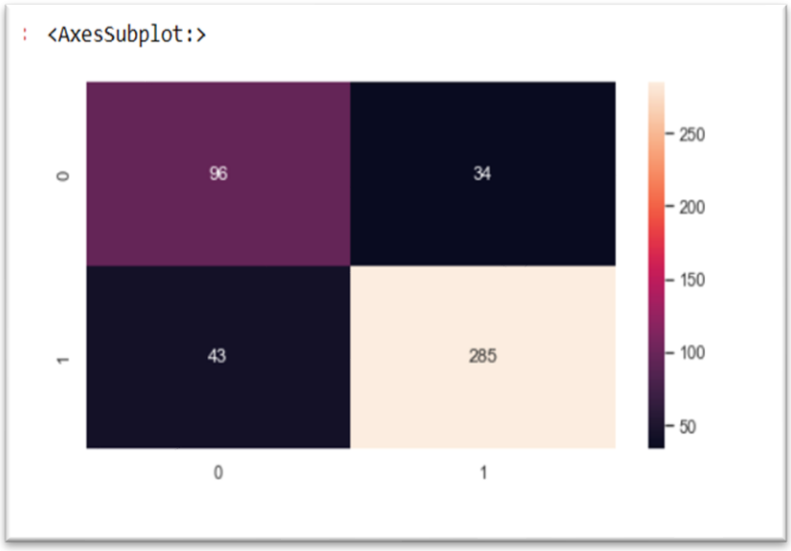
```
Out[133]: GridSearchCV(estimator=KNeighborsClassifier(),
                       param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree',
                                                 'brute']},
                       scoring='recall')
```

Accuracy - Test data: 0.83

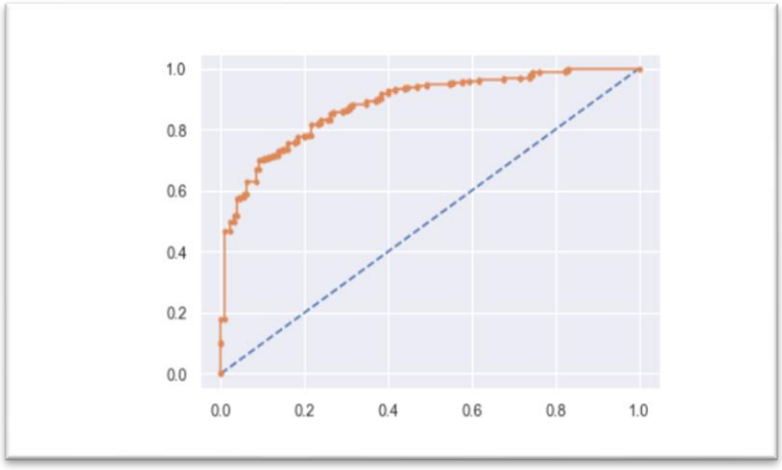
Classification report - Test data

```
0.8231441048034934
[[ 96  34]
 [ 43 285]]
              precision    recall  f1-score   support

           0       0.69      0.74      0.71       130
           1       0.89      0.87      0.88       328

    accuracy                           0.83       458
   macro avg       0.79      0.80      0.80       458
weighted avg       0.84      0.83      0.83       458
```

Confusion Matrix:



AUC and ROC for the test data:



Comparison on performance of both regular and tuned KNN models:

TEST DATA	REGULAR MODEL(%)	TUNED MODEL(%)
ACCURACY	0.82	0.83
PRECISION	0.87	0.89
RECALL	0.88	0.87
F1-SCORE	0.87	0.88

• There is no over-fitting or under-fitting in the tuned KNN model. Overall, it is a good model.

ADA BOOST USING GRID SEARCH:

Best parameters:

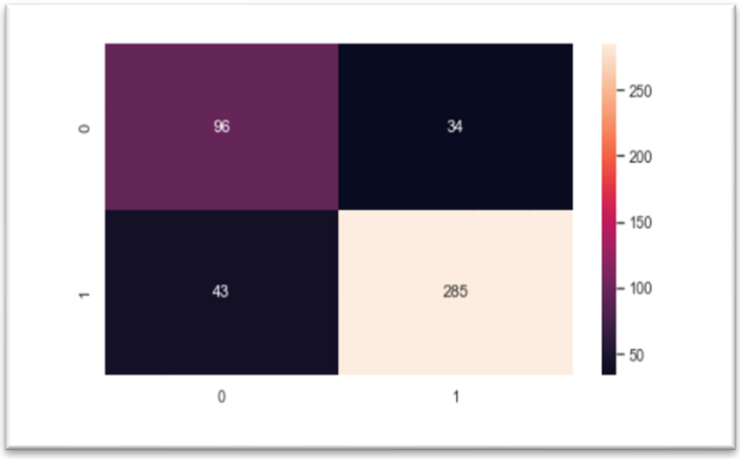
```
GridSearchCV(estimator=AdaBoostClassifier(n_estimators=100, random_state=1),
             param_grid={'algorithm': ['SAMME', 'SAMME.R'],
                         'learning_rate': [0.1, 0.01, 0.001],
                         'n_estimators': [100, 500, 1000]},
             scoring='recall')
```

Accuracy - Test data: 0.83

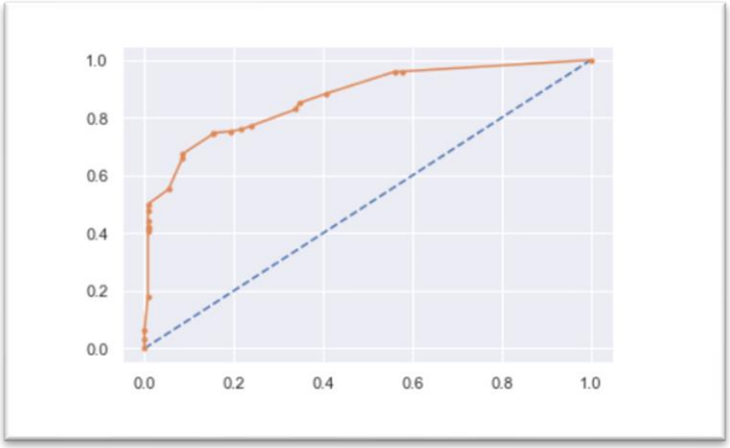
Classification report - Test data

```
0.8122270742358079
[[ 96  34]
 [ 43 285]]
              precision    recall  f1-score   support

           0       0.69      0.74      0.71       130
           1       0.89      0.87      0.88       328

    accuracy                           0.83       458
   macro avg       0.79      0.80      0.80       458
weighted avg       0.84      0.83      0.83       458
```

Confusion Matrix:



AUC and ROC for the test data:



Comparison on performance of both regular and tuned Ada Boost models:

TEST DATA	REGULAR MODEL(%)	TUNED MODEL(%)
ACCURACY	0.82	0.83
PRECISION	0.88	0.89
RECALL	0.87	0.87
F1-SCORE	0.87	0.88

• There is no over-fitting or under-fitting in the tuned Ada Boost model. Overall, it is a good model.

NAÏVE BAYES USING GRID SEARCH:

Best parameters:

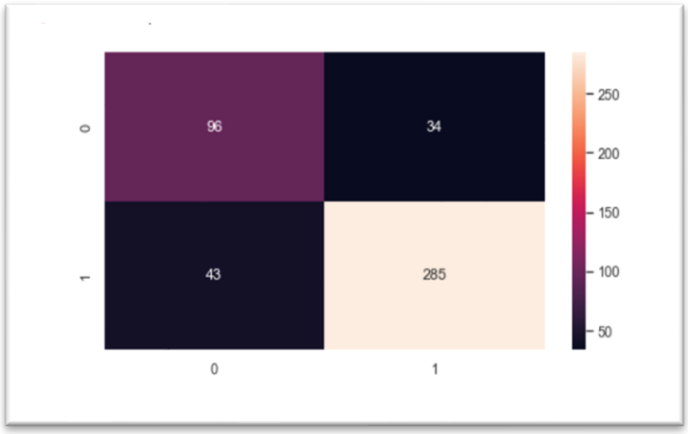
```
[177]: GridSearchCV(estimator=KNeighborsClassifier(),
                    param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                                'leaf_size': [30, 45, 50], 'p': [2, 1],
                                'weights': ['uniform', 'distance']},
                    scoring='recall')
```

Accuracy - Test data: 0.83

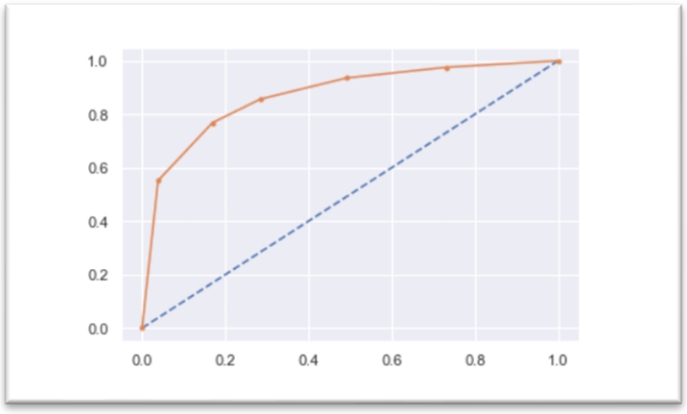
Classification report - Test data

```
0.8165938864628821
[[ 96  34]
 [ 43 285]]
              precision    recall  f1-score   support

           0       0.69      0.74      0.71       130
           1       0.89      0.87      0.88       328

    accuracy                           0.83       458
   macro avg       0.79      0.80      0.80       458
weighted avg       0.84      0.83      0.83       458
```

Confusion Matrix:



AUC and ROC for the test data:



Comparison on performance of both regular and tuned Naïve Bayes models:

TEST DATA	**REGULAR MODEL(%)**	**TUNED MODEL(%)**
ACCURACY	0.82	0.83
PRECISION	0.87	0.89
RECALL	0.89	0.87
F1-SCORE	0.88	0.88

• There is no over-fitting or under-fitting in the tuned Naïve Bayes model. Overall, it is a good model.

DECISION TREE USING GRID SEARCH:

Best parameters:

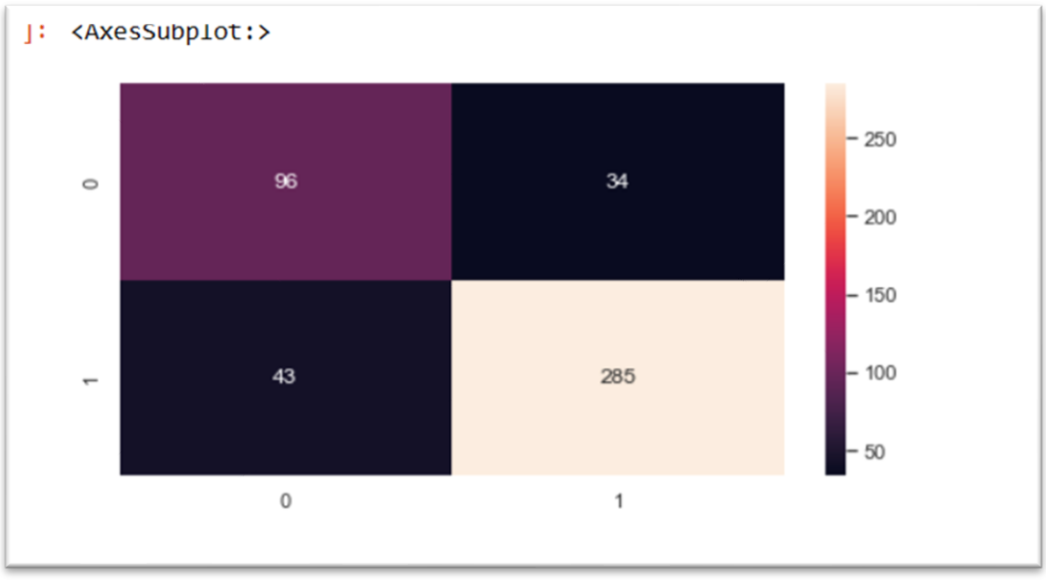
```
]: GridSearchCV(estimator=DecisionTreeClassifier(),
                param_grid={'max_depth': [5, 10, 15, 20],
                            'min_samples_leaf': [15, 25, 35, 50],
                            'min_samples_split': [30, 50, 70, 100],
                            'random_state': [0]},
                scoring='recall')
```

Accuracy - Test data: 0.83

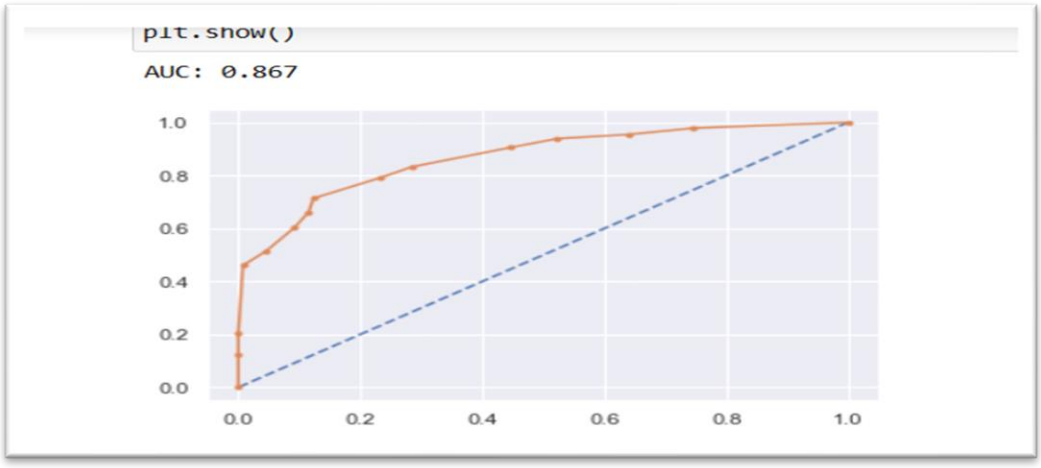
Classification report - Test data

```
0.8056768558951966
[[ 96  34]
 [ 43 285]]
              precision    recall  f1-score   support

           0       0.69      0.74      0.71       130
           1       0.89      0.87      0.88       328

    accuracy                           0.83       458
   macro avg       0.79      0.80      0.80       458
weighted avg       0.84      0.83      0.83       458
```

Confusion Matrix:



AUC and ROC for the test data:



Comparison on performance of both regular and tuned Decision Tree models:

TEST DATA	REGULAR MODEL(%)	TUNED MODEL(%)
ACCURACY	0.74	0.83
PRECISION	0.83	0.89
RECALL	0.80	0.87
F1-SCORE	0.81	0.88

• The tuned model is performing better as compared to regular model.

RANDOM FOREST USING GRID SEARCH:

Best parameters:

```
ıt[189]: {'max_depth': 5,
          'min_samples_leaf': 50,
          'min_samples_split': 30,
          'random_state': 0}
```

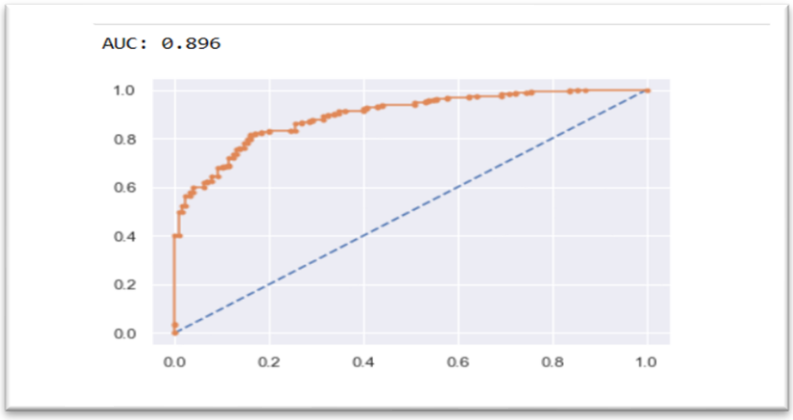
Accuracy - Test data: 0.83

Classification report - Test data

```
0.8231441048034934
[[ 96  34]
 [ 43 285]]
              precision    recall  f1-score   support

           0       0.69      0.74      0.71       130
           1       0.89      0.87      0.88       328

    accuracy                           0.83       458
   macro avg       0.79      0.80      0.80       458
weighted avg       0.84      0.83      0.83       458
```

Confusion Matrix:

AUC and ROC for the test data:



Comparison on performance of both regular and tuned Random Forest models:

TEST DATA	REGULAR MODEL(%)	TUNED MODEL(%)
ACCURACY	0.83	0.83
PRECISION	0.88	0.89
RECALL	0.88	0.87
F1-SCORE	0.88	0.88

• Both the regular and the tuned model are performing the same way.

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.

Comparison of train data of all models in a structured tabular manner:

MODELS	ACCURACY	PRECISION	RECALL	F1-SCORE	AUC
LR-REGULAR	0.84	0.86	0.91	0.89	0.89
LDA-REGULAR	0.84	0.90	0.87	0.88	0.89
KNN – Regular	0.84	0.87	0.90	0.88	0.93
KNN – SMOTE	0.89	0.93	0.84	0.88	-
Naïve Bayes - Regular	0.83	0.88	0.88	0.88	0.89
Naïve Bayes - SMOTE	0.82	0.81	0.83	0.82	-
Random Forest - Regular	1.0	1.0	1.0	1.0	1.0
Bagging - Regular	0.97	0.96	0.99	0.98	-
Ada Boosting - Regular	0.85	0.88	0.91	0.89	0.91
Gradient Boosting - Regular	0.89	0.91	0.93	0.92	-
Decision Tree - Regular	1.0	1.0	1.0	1.0	1.0

Comparison of test data of all models in a structured tabular manner:

MODELS	ACCURACY	PRECISION	RECALL	F1-SCORE	AUC
LR-REGULAR	0.82	0.86	0.88	0.89	0.88
LR-TUNED	0.83	0.89	0.87	0.88	0.88
LDA-REGULAR	0.82	0.88	0.87	0.87	0.88
LDA-TUNED	0.83	0.89	0.87	0.88	0.88
KNN – Regular	0.82	0.87	0.88	0.87	0.86
KNN – Tuned	0.83	0.89	0.87	0.88	0.88
KNN – SMOTE	0.77	0.91	0.85	0.72	-
Naïve Bayes - Regular	0.83	0.87	0.89	0.88	0.89
Naïve Bayes - Tuned	0.83	0.89	0.87	0.88	0.87
Naïve Bayes - SMOTE	0.79	0.91	0.79	0.84	-
Random Forest - Regular	0.83	0.88	0.88	0.88	0.89
Random Forest - Tuned	0.83	0.89	0.87	0.88	0.90
Bagging - Regular	0.84	0.89	0.89	0.89	-
Ada Boosting - Regular	0.82	0.88	0.87	0.87	0.88
Ada Boosting - Tuned	0.83	0.89	0.87	0.88	0.87
Gradient Boosting - Regular	0.83	0.89	0.87	0.88	-
Decision Tree - Regular	0.76	0.85	0.80	0.83	0.72
Decision Tree - Tuned	0.83	0.89	0.87	0.88	0.87

1.8 Based on these predictions, what are the insights?

INSIGHTS:

Labour party has more than double the votes of conservative party.

• Most number of people have given a score of 3 and 4 for the national economic condition and the average score is 3.245221

• Most number of people have given a score of 3 and 4 for the household economic condition and the average score is 3.137772

• Blair has higher number of votes than Hague and the scores are much better for Blair than for Hague.
• The average score of Blair is 3.335531 and the average score of Hague is 2.749506. So, here we can see that, Blair has a better score.

• On a scale of 0 to 3, about 30% of the total population has zero knowledge about politics/parties.

• People who gave a low score of 1 to a certain party, still decided to vote for the same party instead of voting for the other party. This can be because of lack of political knowledge among the people.

• People who have higher Eurosceptic sentiment, has voted for the conservative party and lower the Eurosceptic sentiment, higher the votes for Labour party.

• Out of 454 people who gave a score of 0 for political knowledge, 360 people have voted for the labour party and 94 people have voted for the conservative party.

• All models performed well on training data set as well as test dataset. The tuned models have performed better than the regular models.

• There is no over-fitting in any model except Random Forest and Bagging regular models and Decision Tree.

Business recommendations:

• Hyper-parameters tuning is an important aspect of modelbuilding. There are limitations to this as to process these combinations, huge amount of processing power is required. But if tuning can be done with many sets of parameters, we might get even better results.

• Gathering more data will also help in training the models and thus improving the predictive powers.
• We can also create a function in which all the models predict the outcome in sequence. This will help in better understanding and the probability of what the outcome will be.

PROBLEM STATEMENT-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

(Hint: use .words(), .raw(), .sent() for extracting counts)

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

2.2 Remove all the stopwords from all three speeches.

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)

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

Code Snippet to extract the three speeches:

```
"
import nltk
nltk.download('inaugural')
from nltk.corpus import inaugural
inaugural.fileids()
inaugural.raw('1941-Roosevelt.txt')
inaugural.raw('1961-Kennedy.txt')
inaugural.raw('1973-Nixon.txt')
"
```

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

We have imported inaugural module of nltk.corpus library to get speeches of the Presidents of the United States of America. After that we have used inaugural modules raw method to extract the speech of President Franklin D. Roosevelt, President John F. Kennedy and President Richard Nixon.

To create a temporary function lambda can be used. These functions do not require a name like a def function, however the output is same as defining a permanent function. As these functions are temporary, memory consumption is less in comparison to permanent function. Also, there are multiple ways to get a similar output.

Number of words:

- There are 1323 words in President Franklin D. Roosevelt's speech.
- There are 1364 words in President John F. Kennedy's speech.
- There are 1769 words in President Richard Nixon's speech.

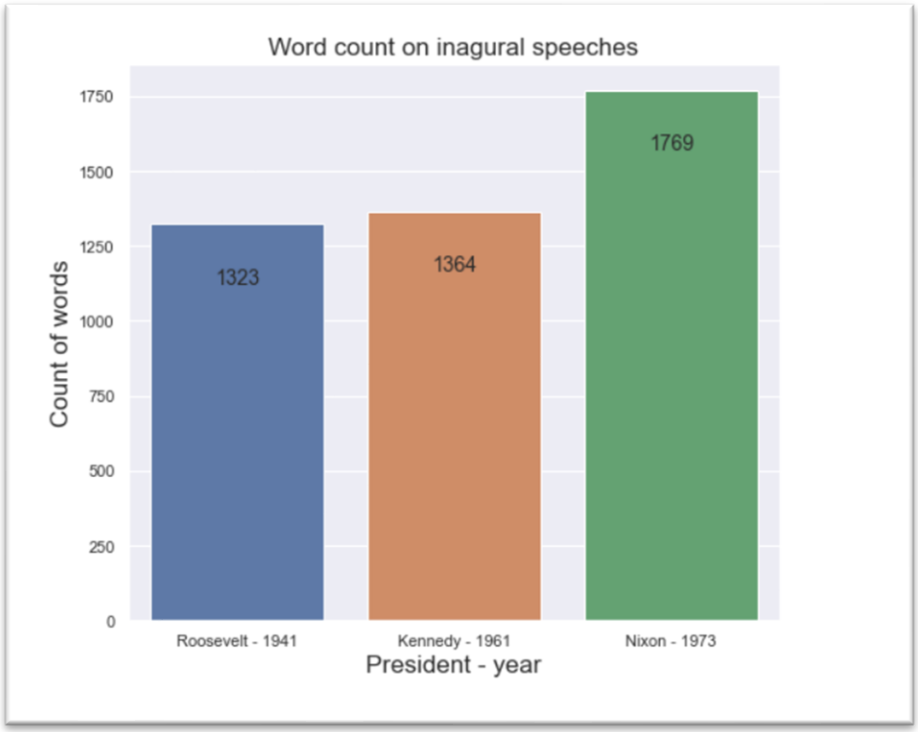
[10]:

	president	text	word_count
1941-Roosevelt	Roosevelt - 1941	On each national day of inauguration since 178...	1323
1961-Kennedy	Kennedy - 1961	Vice President Johnson, Mr. Speaker, Mr. Chief...	1364
1973-Nixon	Nixon - 1973	Mr. Vice President, Mr. Speaker, Mr. Chief Jus...	1769

There are 1323 words in President Franklin D. Roosevelt's speech.

• There are 1364 words in President John F. Kennedy's speech.

• There are 1769 words in President Richard Nixon's speech



Number of characters:

We have used python's len function on each row speech to identify Number of characters in speech.

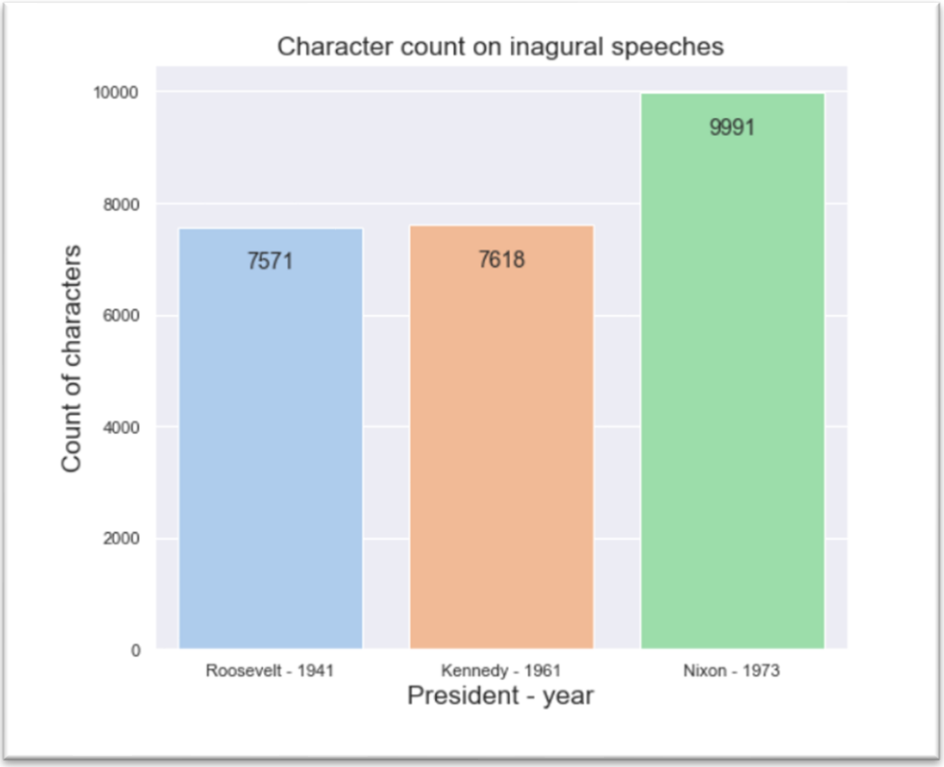
• President Franklin D. Roosevelt's speech have 7571 characters (including spaces).

• President John F. Kennedy's speech have 7618 characters (including spaces).

• President Richard Nixon's speech have 9991 characters (including spaces).

]:

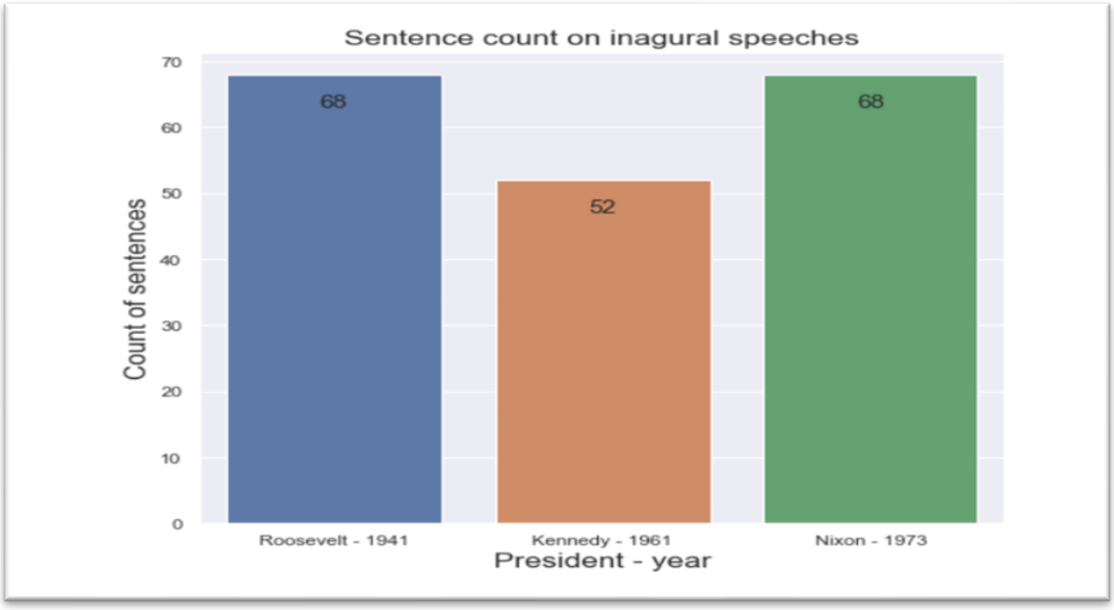
	president	text	word_count	char_count
1941-Roosevelt	Roosevelt - 1941	On each national day of inauguration since 178...	1323	7571
1961-Kennedy	Kennedy - 1961	Vice President Johnson, Mr. Speaker, Mr. Chief...	1364	7618
1973-Nixon	Nixon - 1973	Mr. Vice President, Mr. Speaker, Mr. Chief Jus...	1769	9991



Number of sentences:

- There are 68 sentences in President Franklin D. Roosevelt's speech.
- There are 52 sentences in President John F. Kennedy's speech.
- There are 68 sentences in President Richard Nixon's speech.

	president	text	word_count	char_count	sents_count
1941-Roosevelt	Roosevelt - 1941	On each national day of inauguration since 178...	1323	7571	68
1961-Kennedy	Kennedy - 1961	Vice President Johnson, Mr. Speaker, Mr. Chief...	1364	7618	52
1973-Nixon	Nixon - 1973	Mr. Vice President, Mr. Speaker, Mr. Chief Jus...	1769	9991	68



2.2 Remove all the stopwords from all three speeches.

Before, removing the stop-words, we have changed all the letters to lowercase and we have removed special characters and punctuation marks.

Data before the removal of stop-words:

4]:

	president	text	word_count	char_count	sents_count
1941-Roosevelt	Roosevelt - 1941	national day inauguration since 1789 people re...	1323	7571	68
1961-Kennedy	Kennedy - 1961	vice president johnson speaker chief justice p...	1364	7618	52
1973-Nixon	Nixon - 1973	vice president speaker chief justice senator c...	1769	9991	68

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).

The top three words in the speech are:

nation 11

know 10

spirit 9

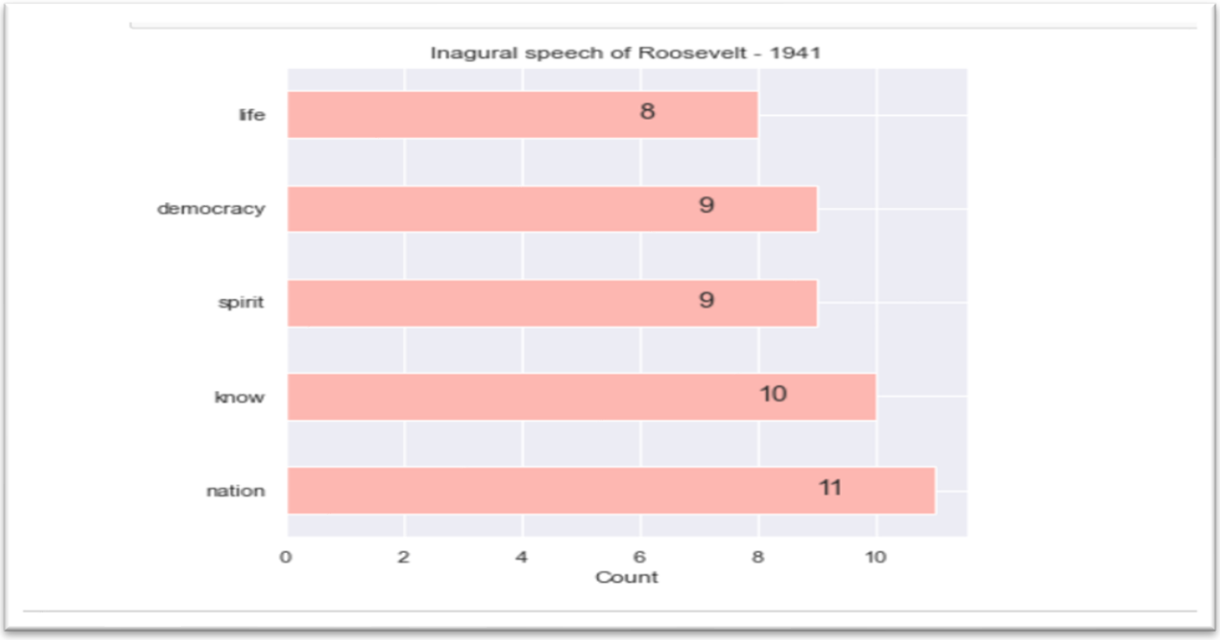
We will plot a frequency plot to check which three words occur most number of times in all the three Presidents of the United States of America.

Top 3 words in Roosevelt's speech:

nation 11

know 10

spirit 9

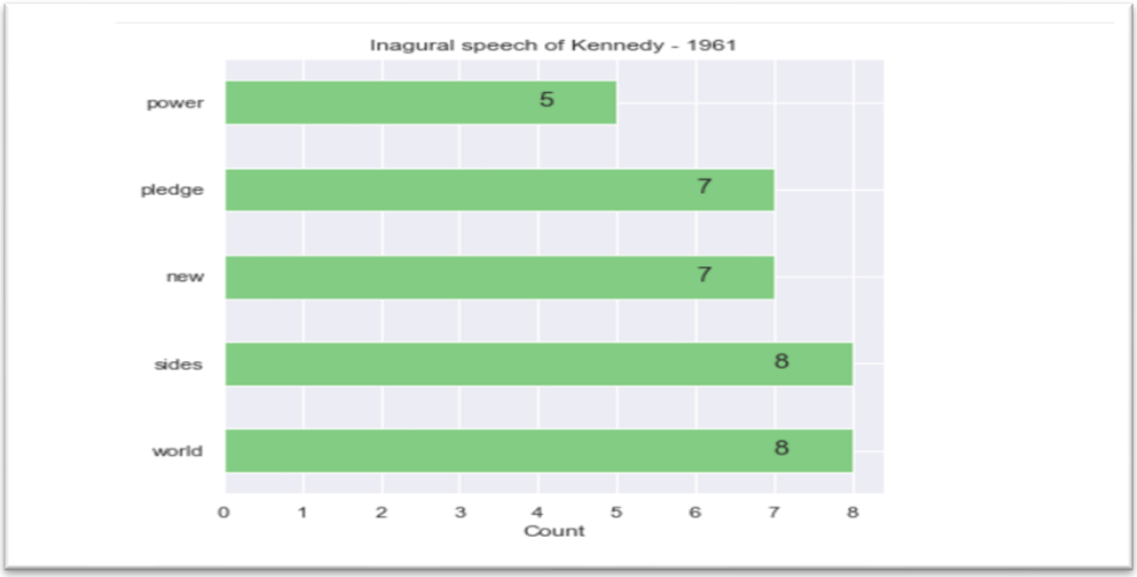


Top 3 words in Kennedy's speech:

world 8

sides 8

new 7

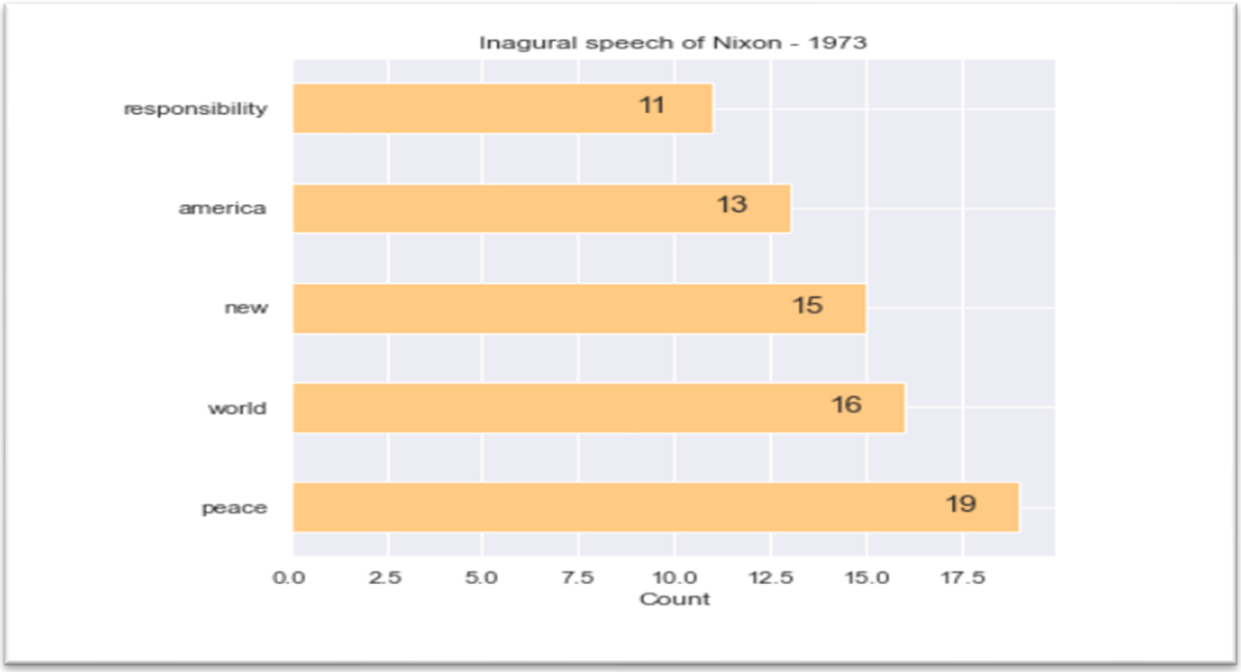


Top 3 words in Nixon's speech:

peace 19

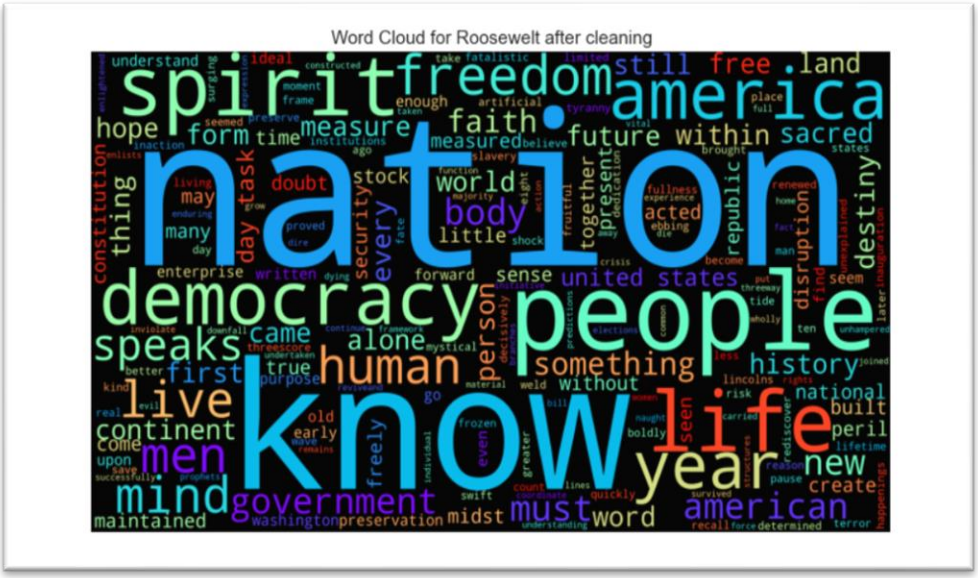
world 16

new 15

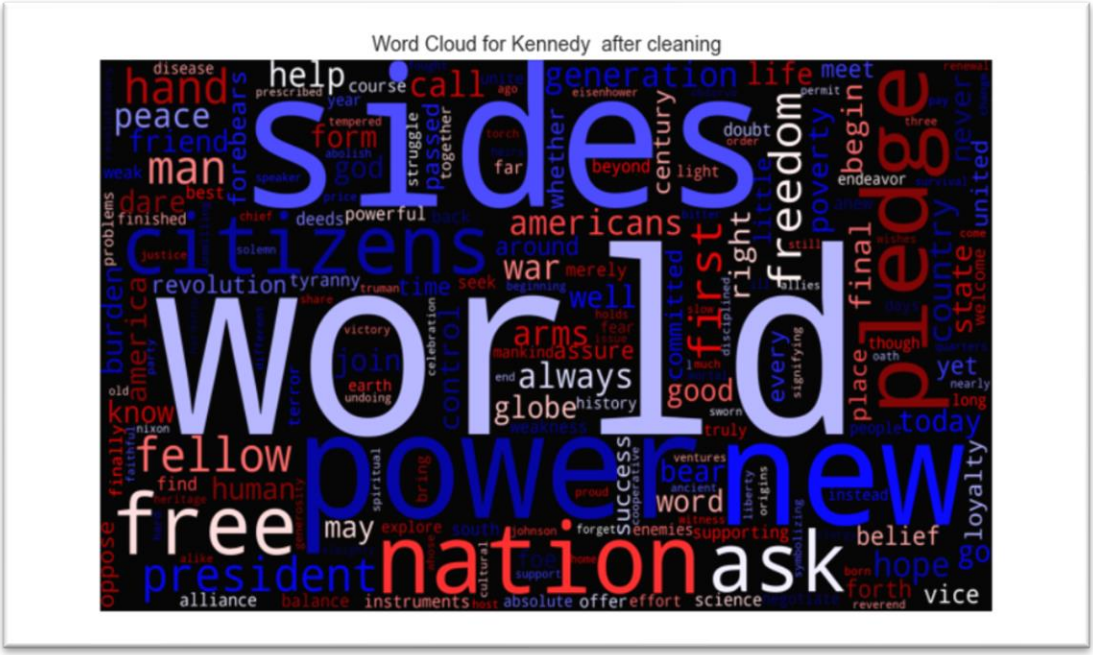


2.4 Plot the word cloud of each of the speeches of the variable. (after removing the stopwords).

Word cloud of Roosevelt's speech:



Word cloud of Kennedy's speech:



Word cloud of Nixon's speech: