greatlearning
Power Ahead

GREAT LAKES
INSTITUTE OF MANAGEMENT

MACHINE LEARNING

Group Assignment


Lavanya


Nikhil


Rekha


Rajiv

Submission Dt:

August 9th,2021

About Us

This section will cover brief introduction about us. We are seasoned professionals with diverse experience.

Rajiv

Am a CA and currently working as **Vice President & Group Head of Internal Audits** for Flipkart, Myntra, PhonePe, e-Kart and Walmart India. Prior to that I have worked in different audit and governance roles in PwC, Coke and Diageo.

Rekha

Overall **20+ years** of corporate and management experience. Currently (from 2018) working as **Strategy Consultant (Independently)** for firms and companies that range from hospitality, manufacturing, and retail industries. Before moving to an independent role was with **KPMG Global Services for 12 years**. Hands-on manager with expertise in accounting systems development, fiscal management. and financial reporting. Proven record of developing and implementing financial and operational controls that improve P&L scenario and competitively position firms.

Nikhil

Overall **14+ years of cross functional experience** in FMCG and Alcobev industry. Working with **Diageo PLC** since 2017, looking after **Data Analytics CoE for Global Audit & Risk department**. Before to that worked with **Coca-Cola for 10 years** and completed stint in operations, supply chain (direct & indirect), Master data maintenance (SAP cross functional role) and Data analytics (Internal Audit).

Lavanya

Working with **Novo Nordisk** as a Manager, having **13 years of techno-functional experience** in SAP Fiori, Embedded Analytics,SCP, ABAP on HANA and also having SAP Consulting experience on UX design strategy, RFP's/Pursuits and solutioning. During this tenure, worked in **various SAP ERP implementation projects spanning across Manufacturing, Healthcare, Retail and FMCG domains**. Prior to this, I have worked with **Accenture, Caterpillar and Wipro**.

Table of Contents

Problem Statement: 3

1. Data Ingestion 3

1.1 Read the dataset. Do the descriptive statistics and do the null value condition check. Write an inference on it 3

1.2 Perform Univariate and Bivariate Analysis. Do exploratory data analysis. Check for Outliers. 6

2. Data Preparation 15

2.1 Encode the data (having string values) for Modelling 15

2.2 Train – Test Split 15

2.3 Data Scaling 16

3. Modelling 16

3.1 Apply Logistic Regression and LDA (linear discriminant analysis). Interpret the results 16

3.2 Apply KNN Model and Naïve Bayes Model. Interpret the results 20

3.3 Bagging *(Random Forest should be applied for Bagging)* & Boosting models. Interpret the results. 22

3.4 Model tuning and Interpret the results 28

3.5 Performance Metrics 43

4. Inference & Insights 46

4.1 Inferences 46

4.2 Insights 46

Problem Statement:

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

Remarks: All coding performed in Jupyter notebook, named **"Machine Learning_GA_Grp1"**, has been duly attached.

Dataset for Problem: **Election_Data.xlsx**

Data Dictionary:

Variable Name	Description
Vote	Party Choice: Conservative Labour.
Age	In years.
economic.cond.national	Assessment of current national economic conditions, 1 to 5.
economic.cond.household	Assessment of current household economic conditions, 1 to 5.
Blair	Assessment of the Labour leader, 1 to 5.
Hague	Assessment of the Conservative leader, 1 to 5
Europe	an 11-point scale that measures respondents 'attitudes toward European integration. High scores represent "Eurosceptic" sentiment.
political knowledge	political knowledge: Knowledge of parties' positions on European integration, 0 to 3.
gender	female or male.

1. Data Ingestion

We imported all necessary libraries before we load the dataset.

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

Import Libraries

```
import pandas as pd
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')

import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier

import sklearn.metrics as metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score,roc_auc_score,confusion_matrix,roc_curve
```

Read the dataset & null values and duplicate check

i. Load and read the data set "Election_Data.xlsx"

ii. Understand the dataset by displaying the first few rows (code: df.head(2))

	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

iii. Analysing the dimension of the dataset and the data type of variables in the dataset (code: df.shape() & df.info())

```
The number of columns (variables) in the dataset is 9
The number of rows (observations per variable) in the dataset is 1525

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1525 entries, 1 to 1525
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: 119.1+ KB
```

Insights:

- Seven variables are numerical (i.e., age, economic.cond.national, economic.cond.household, Blair, Hague, Europe, political.knowledge)
- Two variables are categorical (i.e., vote & gender)

iv. Checking for null value in the dataset (code: df.isnull().sum())

- No missing values in the dataset.

```
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
```

v. Checking the existence of duplicate records in the dataset using duplicate function:

- The dataset has no unique identifier which could establish the fact that the records are duplicate and therefore retained in the dataset.

Descriptive Statistics

Steps for analysis:

Generating the descriptive statistics summary to display the results of both numerical and categorical variables. The variables that generate ordinal value is treated as categorical variables while generating the statistical results.

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
vote	1525	2	Labour	1063	NaN	NaN	NaN	NaN	NaN	NaN	NaN
age	1525.0	NaN	NaN	NaN	54.182295	15.711209	24.0	41.0	53.0	67.0	93.0
economic.cond.national	1525.0	NaN	NaN	NaN	3.245902	0.880969	1.0	3.0	3.0	4.0	5.0
economic.cond.household	1525.0	NaN	NaN	NaN	3.140328	0.929951	1.0	3.0	3.0	4.0	5.0
Blair	1525.0	NaN	NaN	NaN	3.334426	1.174824	1.0	2.0	4.0	4.0	5.0
Hague	1525.0	NaN	NaN	NaN	2.746885	1.230703	1.0	2.0	2.0	4.0	5.0
Europe	1525.0	NaN	NaN	NaN	6.728525	3.297538	1.0	4.0	6.0	10.0	11.0
political.knowledge	1525.0	NaN	NaN	NaN	1.542295	1.083315	0.0	0.0	2.0	2.0	3.0
gender	1525	2	female	812	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Insights:

Numerical Variables:

- There is a very insignificant gap between mean and median for numeric variables.
- Average & median "economic.cond.national" rating is 3, which shows that current national economic conditions is relatively good.
- Average & median "economic.cond.household" rating is also 3, which shows that current household economic conditions is relatively good.
- Average & median "Blair" rating is between 3 and 4, which shows that labour leader inclination is higher. This may be the key factor deciding while predicting the party vote.
- Average & median "Hague" rating is between 2 to 3, which shows that conservative leader inclination is lower than Blair. This may be the key factor deciding while predicting the party vote.
- Average & median "Europe" rating is between 6 to 7, which shows that voter has higher inclination towards ‘Eurosceptic’ sentiment.

Categorical Variables:

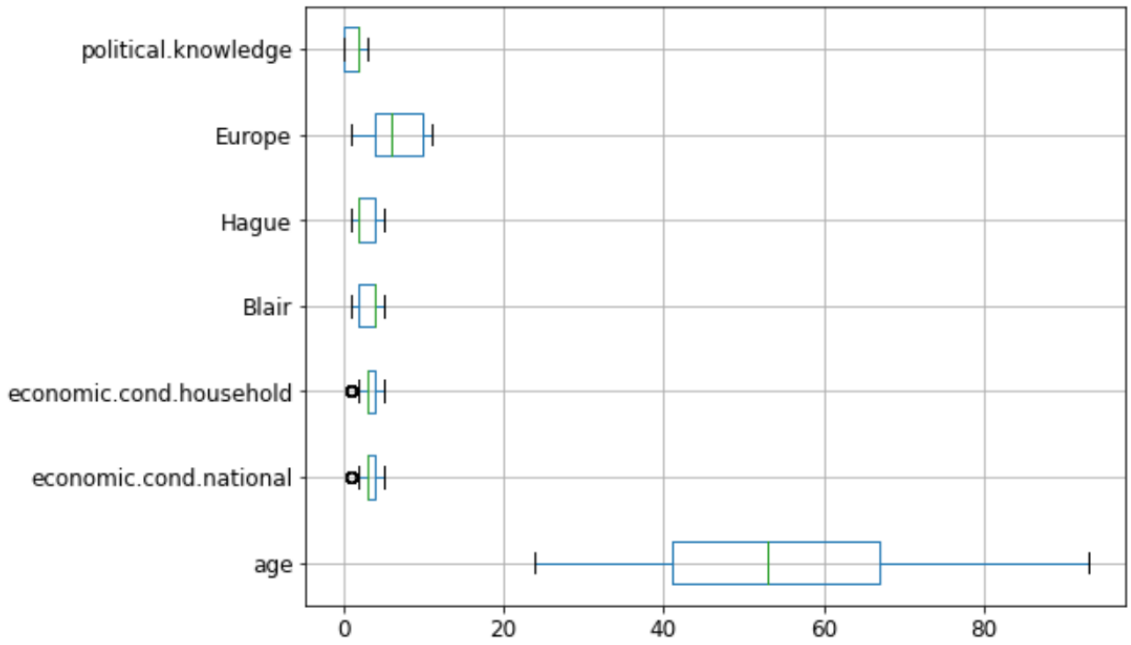
- It seems female voter contributions are higher than male voters.
- There are mainly two-party choice for variable "vote" (labour or conservative)> In sample data, we could see labour party has the upper hand then conservative party.

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

Outliers Check

Checking for Outliers(df.skew and df.boxplot())

```
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
dtype: float64
```

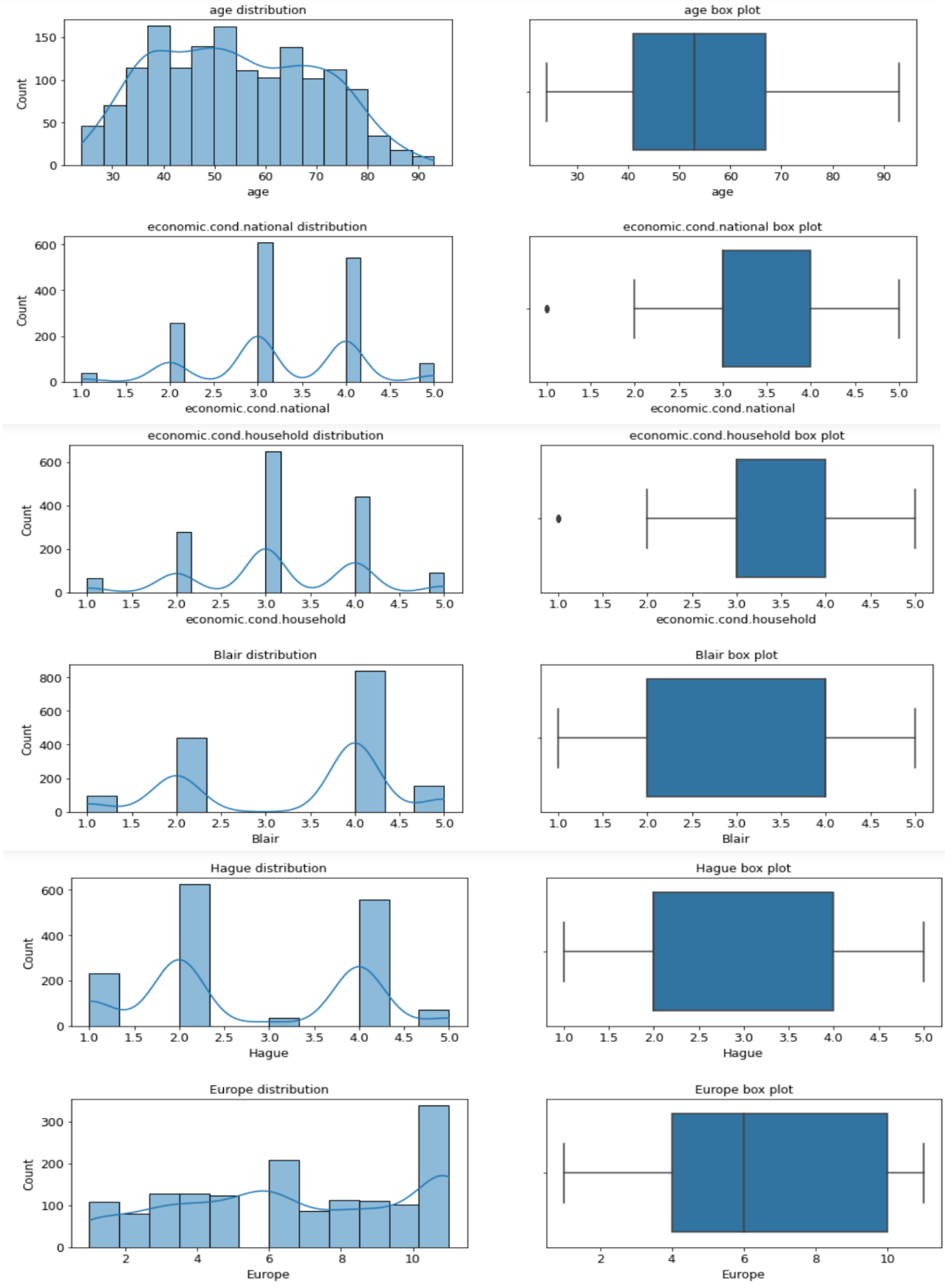


Insights from skewness and boxplot:

- We have very insignificant outliers in the dataset. So, we would not be doing any outlier treatment here.

Univariate Analysis (Numerical Variables)

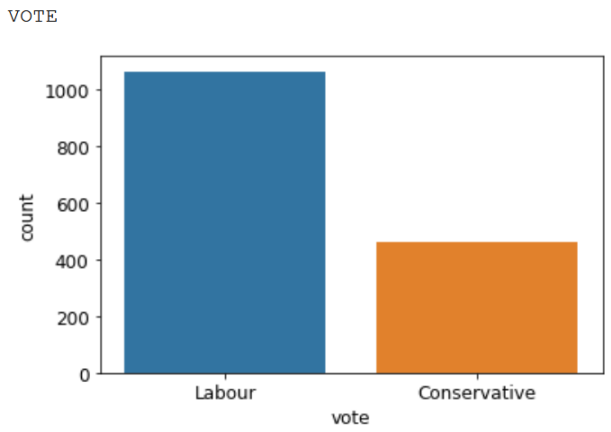
We would be using histogram/distribution plot for the inferences.

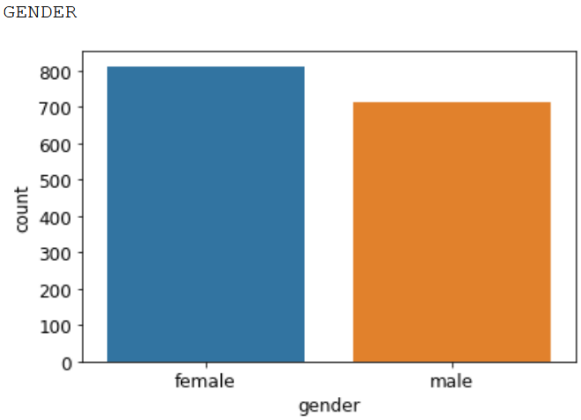


Insights:

- For "age", distribution looks not exactly normal, but box plot gives that it doesn't have any outliers.
- For "economic.cond.national" and "economic.cond.household" distribution looks normal and box plot shows very few outliers.
- "Blair" distribution looks left skewed and in box plot we don't have any outliers.
- "Hague" distribution looks normal and in box plot we don't have any outliers.
- For "Europe", distribution appears left skewed not exactly normal, but box plot gives that it doesn't have any outliers.
- "political.knowledge" distribution looks right skewed and in box plot we don't have any outliers.

Univariate Analysis (Categorical Variables)
We would be using count plot for the inferences.

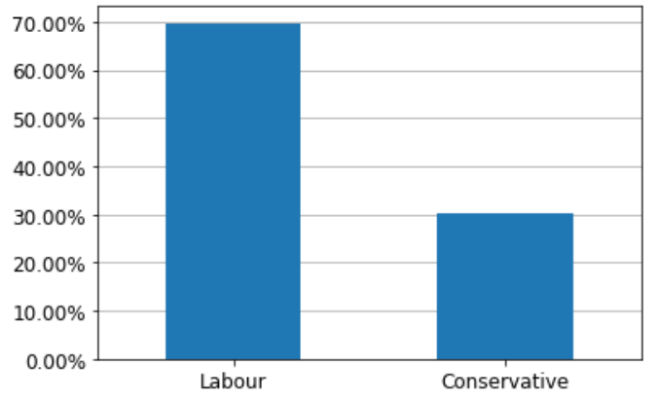




Insights:

- For "vote", labour party has the higher vote count than conservative.
- For "gender", female voter has the higher voting than male.

We further look at the distribution of the dependent variable “vote”.




```
Labour          0.697049
Conservative    0.302951
Name: vote, dtype: float64
```

Class imbalance and it's treatment

An imbalanced classification problem is an example of a classification problem where the distribution of examples across the known classes is biased or skewed.

The distribution can vary from a slight bias to a severe imbalance where there is one example in the minority class for hundreds, thousands, or millions of examples in the majority class or classes.

A classification problem may be a little skewed, such as if there is a slight imbalance. Alternately, the classification problem may have a severe imbalance where there might be hundreds or thousands of examples in one class and tens of examples in another class for a given training dataset.

Slight Imbalance: An imbalanced classification problem where the distribution of examples is uneven by a small amount in the training dataset (e.g., 4:6 or 3:7).

Severe Imbalance: An imbalanced classification problem where the distribution of examples is uneven by a large amount in the training dataset (e.g., 1:100 or more).

Here, we have 70%-30% distribution and a slight imbalance is often not a concern, and the problem can often be treated like a normal classification problem. So, we are proceeding AS-IS and not performing SMOTE etc on this dataset.

Bivariate Analysis (Numerical vs Numerical)

We have used covariance, correlation, pair plot and heat map to get the inferences.

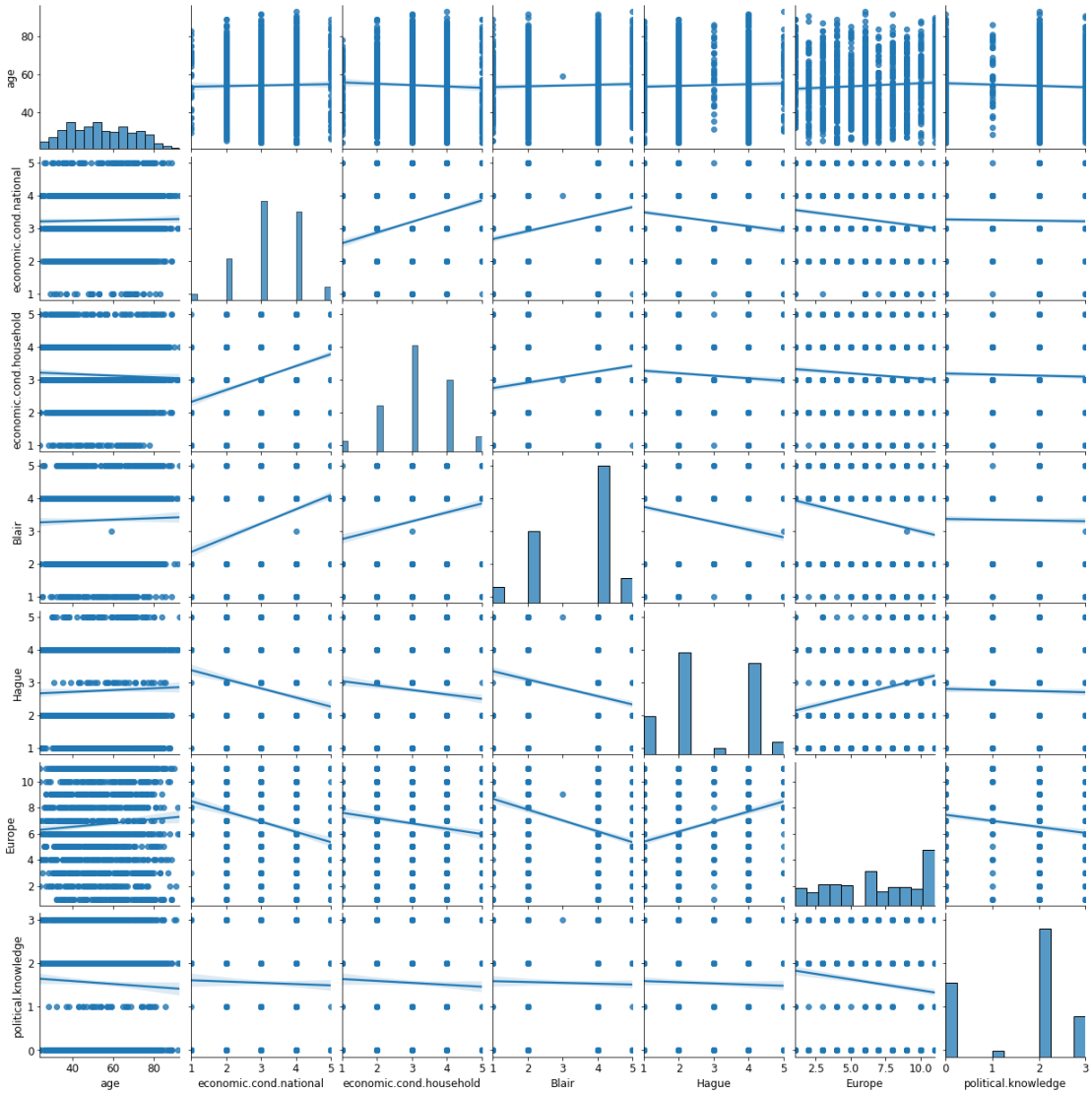
Covariance function:

	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge
age	246.842075	0.256981	-0.607619	0.557762	0.669531	3.568550	-0.825301
economic.cond.national	0.256981	0.776107	0.283712	0.338314	-0.216589	-0.608397	-0.022546
economic.cond.household	-0.607619	0.283712	0.864810	0.235192	-0.116689	-0.352299	-0.038091
Blair	0.557762	0.338314	0.235192	1.380212	-0.351648	-1.147341	-0.026621
Hague	0.669531	-0.216589	-0.116689	-0.351648	1.514631	1.166149	-0.040469
Europe	3.568550	-0.608397	-0.352299	-1.147341	1.166149	10.873759	-0.544285
political.knowledge	-0.825301	-0.022546	-0.038091	-0.026621	-0.040469	-0.544285	1.173571

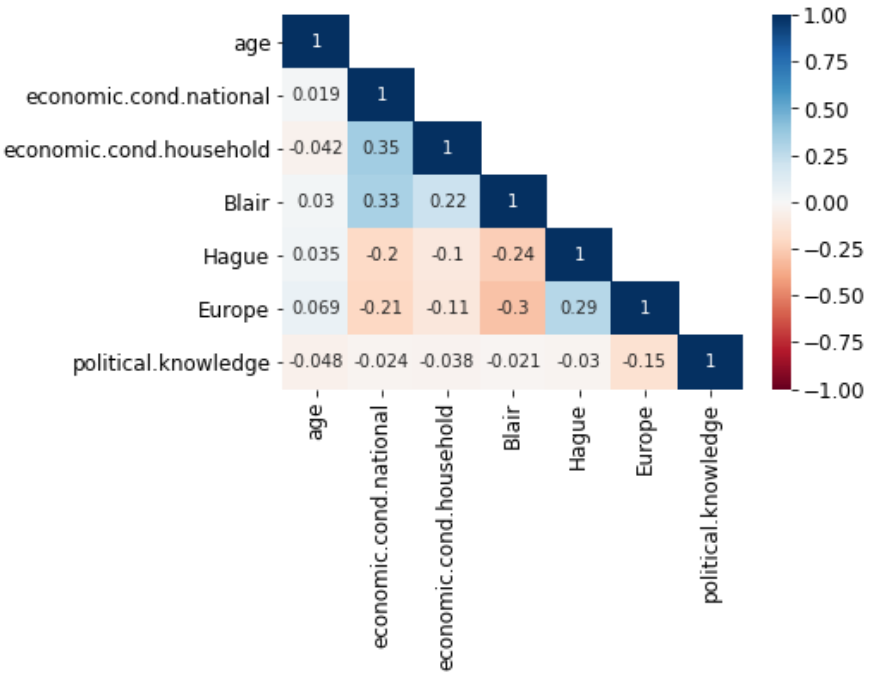
Correlation function:

	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge
age	1.000000	0.018567	-0.041587	0.030218	0.034626	0.068880	-0.048490
economic.cond.national	0.018567	1.000000	0.346303	0.326878	-0.199766	-0.209429	-0.023624
economic.cond.household	-0.041587	0.346303	1.000000	0.215273	-0.101956	-0.114885	-0.037810
Blair	0.030218	0.326878	0.215273	1.000000	-0.243210	-0.296162	-0.020917
Hague	0.034626	-0.199766	-0.101956	-0.243210	1.000000	0.287350	-0.030354
Europe	0.068880	-0.209429	-0.114885	-0.296162	0.287350	1.000000	-0.152364
political.knowledge	-0.048490	-0.023624	-0.037810	-0.020917	-0.030354	-0.152364	1.000000

Pair plot:


age
economic.cond.national
economic.cond.household
Blair
Hague
Europe
political.knowledge

Heatmap:


age
economic.cond.national
economic.cond.household
Blair
Hague
Europe
political.knowledge
1
0.019 1
-0.042 0.35 1
0.03 0.33 0.22 1
0.035 -0.2 -0.1 -0.24 1
0.069 -0.21 -0.11 -0.3 0.29 1
-0.048 -0.024 -0.038 -0.021 -0.03 -0.15 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

We then sorted the correlations in descending order:

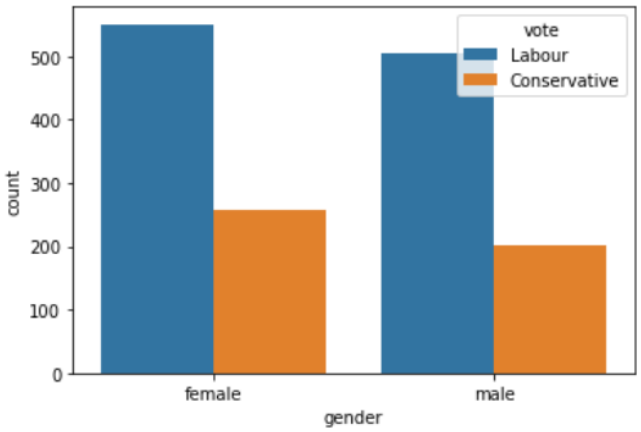
```
economic.cond.household  economic.cond.national     0.35
economic.cond.national   economic.cond.household    0.35
                         Blair                      0.33
Blair                    economic.cond.national     0.33
                         Europe                     0.30
Europe                   Blair                      0.30
                         Hague                      0.29
Hague                    Europe                     0.29
                         Blair                      0.24
Blair                    Hague                      0.24
economic.cond.household  Blair                      0.22
Blair                    economic.cond.household    0.22
economic.cond.national   Europe                     0.21
Europe                   economic.cond.national     0.21
economic.cond.national   Hague                      0.20
dtype: float64
```

Insights:

- In Bivariate analysis(Numerical variables), we have used scatter plot & heatmap and found that in scatter plot and heat map both stats that none of the variables shows the strong relationship.

Bivariate Analysis (Categorical vs Categorical)

We have used count plot for Party choice Vs Gender to get inferences.

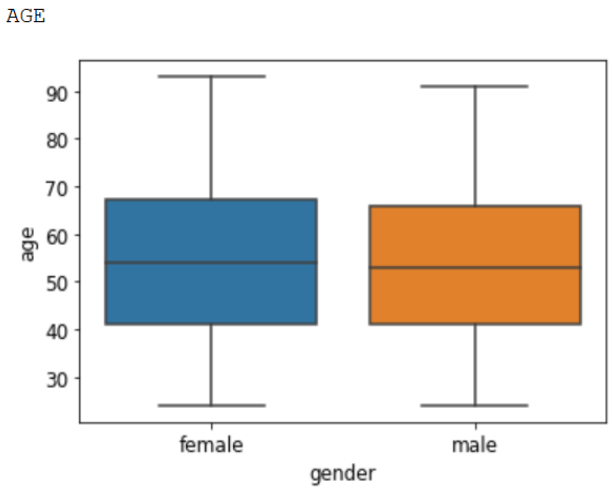


Insights:

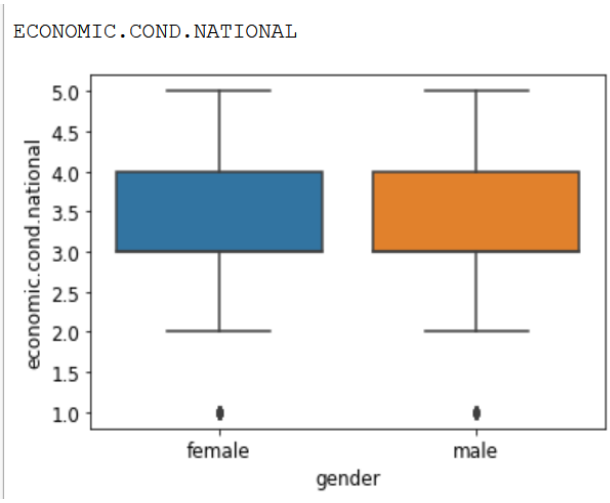
- Casting of vote is similar gender wise and Labour party is the choice for both the genders.
- Female voters are higher than male voters

Bivariate Analysis (Categorical vs Numerical)

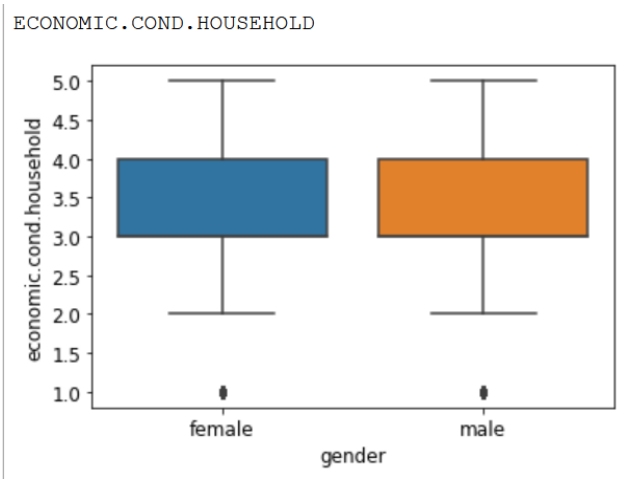
Gender Vs Age



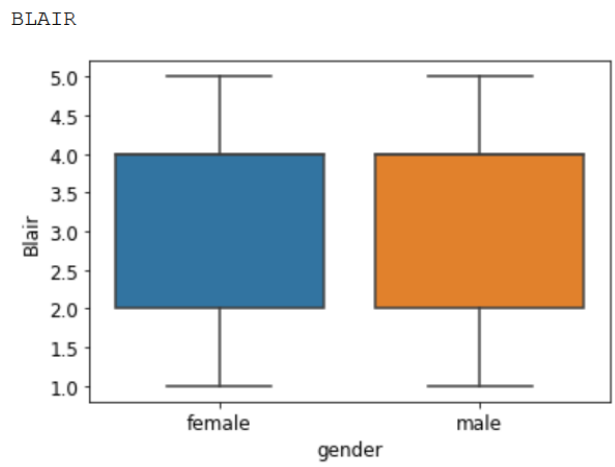
Gender Vs Economic Cond National



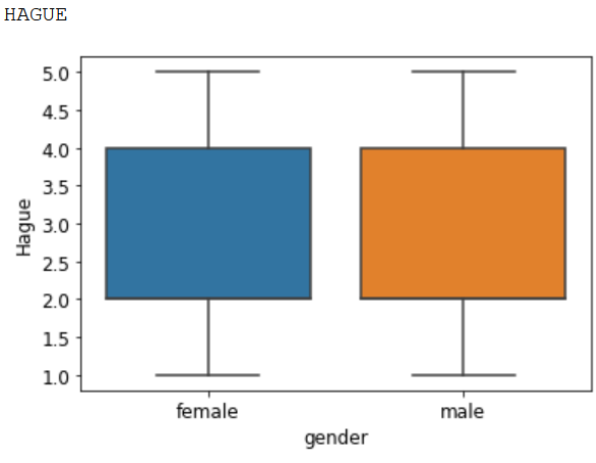
Gender Vs Economic Cond Household



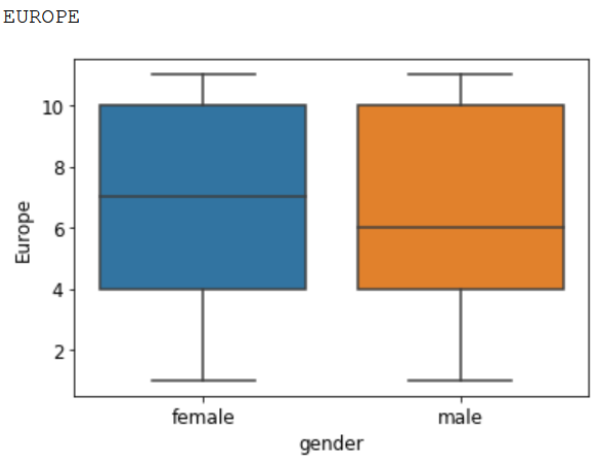
Gender Vs Blair



Gender Vs Economic Cond Household

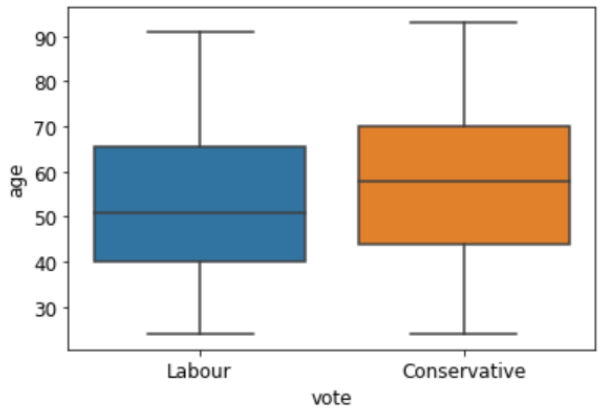


Gender Vs Blair



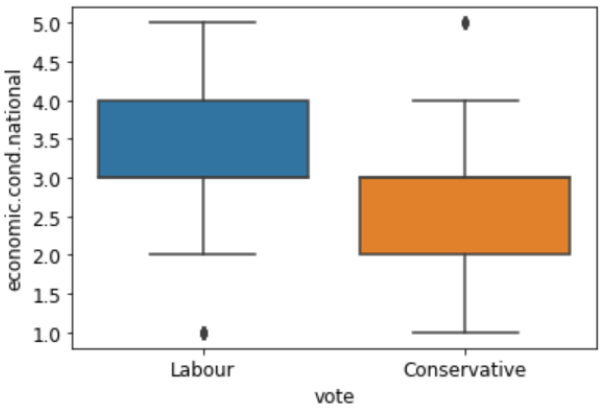
Vote Vs Age

AGE



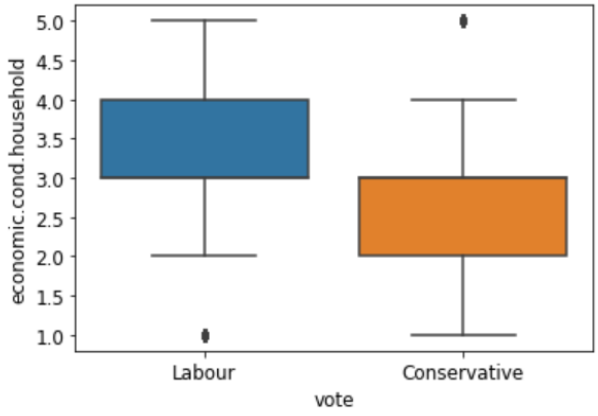
Vote Vs Economic Cond National

ECONOMIC.COND.NATIONAL



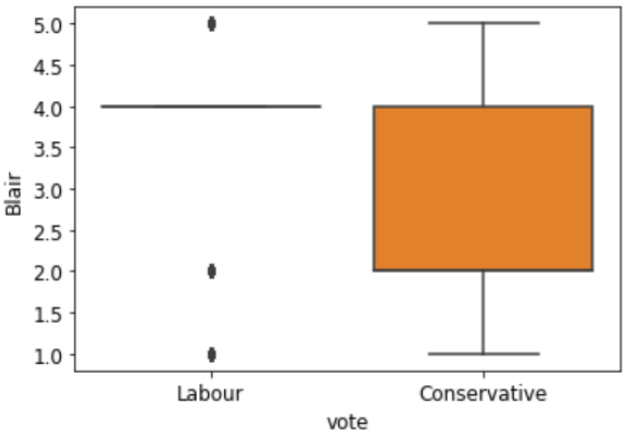
Vote Vs Economic Cond Household

ECONOMIC.COND.HOUSEHOLD



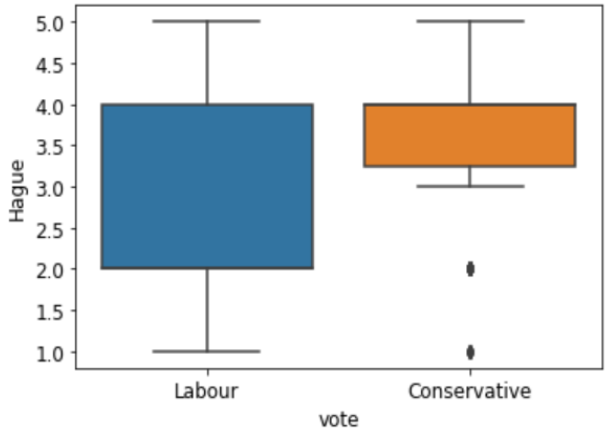
Vote Vs Blair

BLAIR



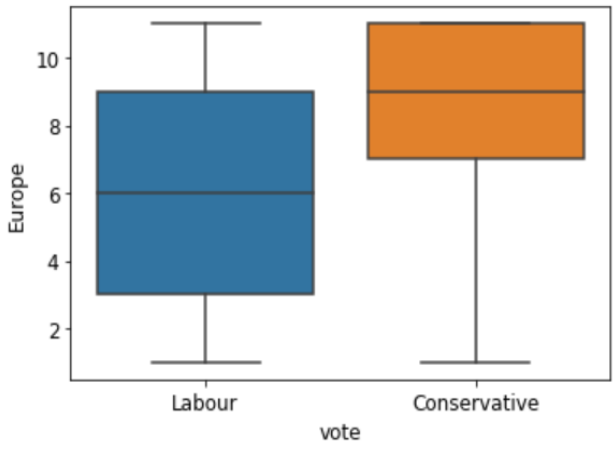
Vote Vs Hague

HAGUE



Vote Vs Europe

EUROPE



Insights:

- The spread of age in casting vote both to Labour and Conservative is almost even. The age slab only is the variation for party choice.
- Population above age 60 is inclined towards conservative party and below 60 is inclined towards Labour party.
- With respect to economic condition nation wise or household wise, Labour party is the choice for all the categories.
- Population who has good political knowledge and who have Eurosceptic sentiments are inclined towards Labour party.
- With respect to gender, there is no significant difference in vote casting by female and male; however, both the genders are inclined towards labour party.
- With respect to 'Eurosceptic' sentiment, the highest count for both the party indicates in the 11th score with an exception the labour party is highest in the 6th score as well.

We have completed EDA portion, now let's move to next question.

2. Data Preparation

We imported all necessary libraries before we load the dataset.

2.1 Encode the data (having string values) for Modelling

Let's convert string/categorical variables into numerical using get dummies function.

	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	vote_Labour	gender_male
0	43	3	3	4	1	2	2	1	0
1	36	4	4	4	4	5	2	1	1

Note: *Here, we could refer* ***"vote_labour = 1"*** *to vote was given to* ***"labour party"*** *and* ***"vote_labour = 0"*** *means vote given to* ***"conservative party"***

2.2 Train – Test Split

Then we split the data into train and test with 70:30 split and used parameter stratify as "yes" which ensures that both the train and test sets have the proportion of examples in each class that is present in the provided "y" array.

```
# train and test split with 70/30 ratio and used stratify to allocate same % of vote_labour class into train and test.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 1, stratify = y)
```

```
X_train: (1067, 8)
Y_train: (1067,)
X_test: (458, 8)
Y_test: (458,)
Total Count: 1525
```

```
y_train.value_counts(1)
```

```
1    0.697282
0    0.302718
Name: vote_labour, dtype: float64
```

```
y_test.value_counts(1)
```

```
1    0.696507
0    0.303493
Name: vote_labour, dtype: float64
```

2.3 Data Scaling

For this dataset, We are going to use several models and we get some benefit from scaling. Hence, let's scale the data for 'age' variable as other variables are ranking based

We have used standard scaler function. Further,

- Train data: We used fit_transform() on the train data so that we learn the parameters of scaling on the train data and at the same time we scale the train data.
- Test data: We only use transform() on the test data because **we use the scaling parameters learned on the train data to scale the test data**.

Before Scaling

	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	gender_male
1493	34	3	1	4	2	6	2	0
1431	42	3	4	4	4	3	2	0

After Scaling

	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	gender_male
1493	-1.266219	3	1	4	2	6	2	0
1431	-0.757323	3	4	4	4	3	2	0

Now, dataset is ready for data modelling.

3. Modelling

3.1 Apply Logistic Regression and LDA (linear discriminant analysis). Interpret the results

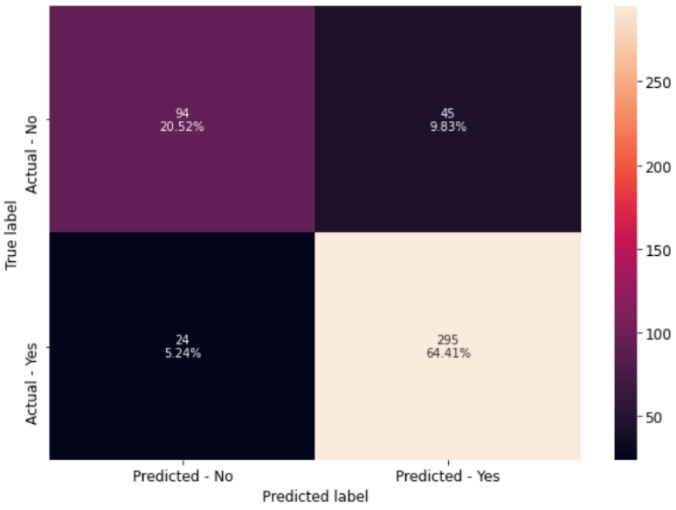
Logistic Regression

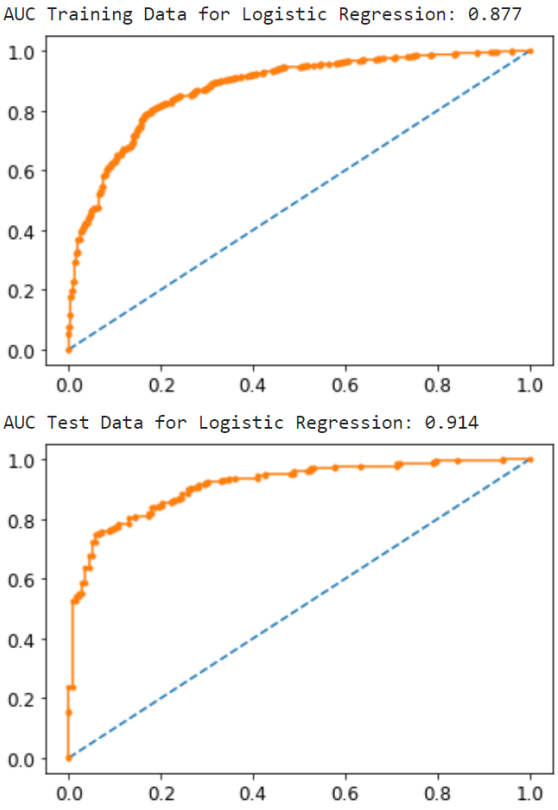
- We have imported *LogisticRegression* model from sklearn library.
- Load the model and fit the model in train data
- Predicted the values for train and test data to check accuracy of the model on train data & test data.

- We have further included classification report, confusion matrix and AUC score.

```
Train Accuracy: 83%
Test Accuracy: 85%


Train Classification Report

              precision    recall  f1-score   support

           0       0.75      0.65      0.70       323
           1       0.86      0.91      0.88       744

    accuracy                           0.83      1067
   macro avg       0.80      0.78      0.79      1067
weighted avg       0.82      0.83      0.82      1067

Test Classification Report

              precision    recall  f1-score   support

           0       0.80      0.68      0.73       139
           1       0.87      0.92      0.90       319

    accuracy                           0.85       458
   macro avg       0.83      0.80      0.81       458
weighted avg       0.85      0.85      0.85       458
```

```
Accuracy on training set :83%
Accuracy on test set : 85%
Recall on training set : 91%
Recall on test set : 92%
Precision on training set : 86%
Precision on test set : 87%
F1 score on training set : 88%
F1 score on test set : 90%
```





Interpretation (Logistic Regression):

- Accuracy for train and test is **almost same** (i.e., train: ~83%, test: ~85%)
- Recall for train and test is **almost same** (i.e., train: ~91%, test: ~92%)
- Precision for train and test is **almost same** (i.e., train: ~86%, test: ~87%)
- F1-score for train and test is **almost same** (i.e., train: ~88%, test: ~90%)
- AUC for train and test is **almost same** (i.e., train: ~88%, test: ~91%)

Let's evaluate the other models.

Linear Discriminant Analysis (LDA)

- We have imported *LinearDiscriminantAnalysis* model from sklearn library.
- Load the model and fit the model in train data
- Predicted the values for train and test data to check accuracy of the model on train data & test data.
- We have further included classification report, confusion matrix and AUC score

```
Train Accuracy: 83%
Test Accuracy: 84%
```

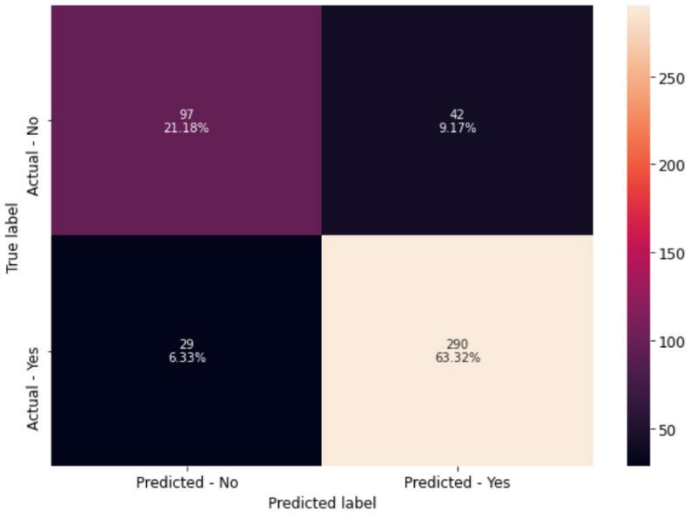
Train Classification Report

	precision	recall	f1-score	support
0	0.73	0.67	0.70	323
1	0.86	0.90	0.88	744
accuracy			0.83	1067
macro avg	0.80	0.78	0.79	1067
weighted avg	0.82	0.83	0.82	1067

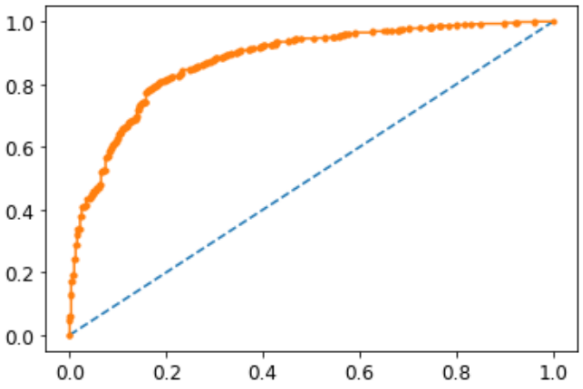
Test Classification Report

	precision	recall	f1-score	support
0	0.77	0.70	0.73	139
1	0.87	0.91	0.89	319
accuracy			0.84	458
macro avg	0.82	0.80	0.81	458
weighted avg	0.84	0.84	0.84	458

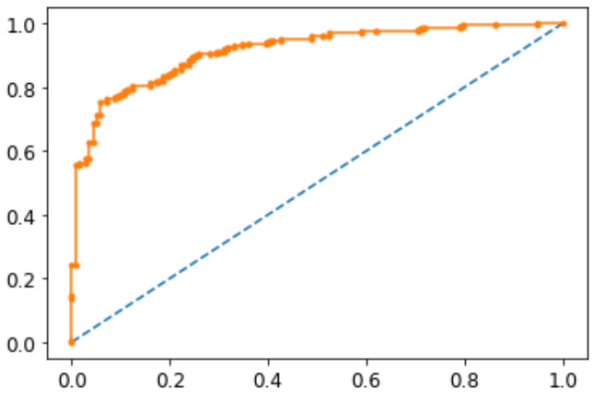
Accuracy on training set :83%
Accuracy on test set : 84%
Recall on training set : 90%
Recall on test set : 91%
Precision on training set : 86%
Precision on test set : 87%
F1 score on training set : 88%
F1 score on test set : 89%



AUC Training Data for LDA: 0.876



AUC Test Data for LDA: 0.915



	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision	Train_F1 score	Test_F1 score
0	Logistic Regression	0.83	0.85	0.91	0.92	0.86	0.87	0.88	0.90
1	LDA	0.83	0.84	0.90	0.91	0.86	0.87	0.88	0.89

Interpretation (LDA):

- Accuracy for train and test is **almost same** (i.e., train: ~83%, test: ~84%)
- Recall for train and test is **almost same** (i.e., train: ~90%, test: ~91%)
- Precision for train and test is **almost same** (i.e., train: ~86%, test: ~87%)
- F1-score for train and test is **almost same** (i.e., train: ~88%, test: ~89%)
- AUC for train and test is **almost same** (i.e., train: ~88%, test: ~92%)

So, Logistic Regression LDA gave almost the same results

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

K-nearest neighbours (KNN)

- We have imported *KNeighborsClassifier* model from sklearn library.
- Load the model and fit the model in train data
- Predicted the values for train and test data to check accuracy of the model on train data & test data.
- We have further included classification report, confusion matrix and AUC score

```
Train Accuracy: 86%
Test Accuracy: 84%
```

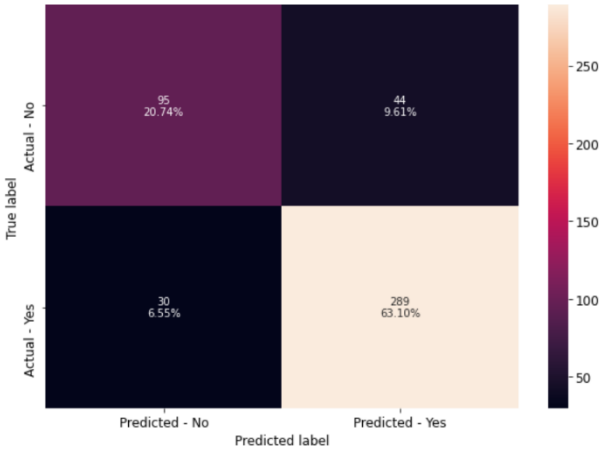
Train Classification Report

	precision	recall	f1-score	support
0	0.78	0.74	0.76	323
1	0.89	0.91	0.90	744
accuracy			0.86	1067
macro avg	0.84	0.83	0.83	1067
weighted avg	0.86	0.86	0.86	1067

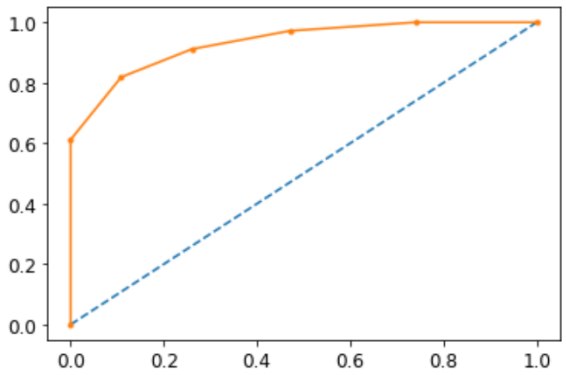
Test Classification Report

	precision	recall	f1-score	support
0	0.76	0.68	0.72	139
1	0.87	0.91	0.89	319
accuracy			0.84	458
macro avg	0.81	0.79	0.80	458
weighted avg	0.84	0.84	0.84	458

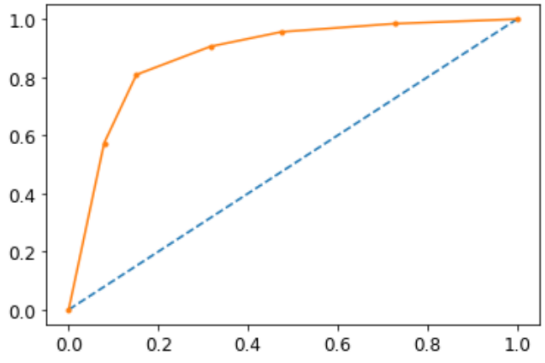
```
Accuracy on training set :86%
Accuracy on test set : 84%
Recall on training set : 91%
Recall on test set : 91%
Precision on training set : 89%
Precision on test set : 87%
F1 score on training set : 90%
F1 score on test set : 89%
```



AUC Training Data for KNN: 0.933



AUC Test Data for KNN: 0.877



Interpretation (KNN):

- Accuracy for train and test is **almost same** (i.e., train: ~86%, test: ~84%)
- Recall for train and test is **almost same** (i.e., train: ~91%, test: ~91%)
- Precision for train and test is **almost same** (i.e., train: ~89%, test: ~87%)
- F1-score for train and test is **almost same** (i.e., train: ~90%, test: ~89%)
- AUC for train and test has **slight overfit but manageable** (i.e., train: ~93%, test: ~88%)

Naïve Bayes

- We have imported *GaussianNB* model from sklearn library.
- Load the model and fit the model in train data
- Predicted the values for train and test data to check accuracy of the model on train data & test data.
- We have further included classification report, confusion matrix and AUC score

```
Train Accuracy: 82%
Test Accuracy: 85%
```

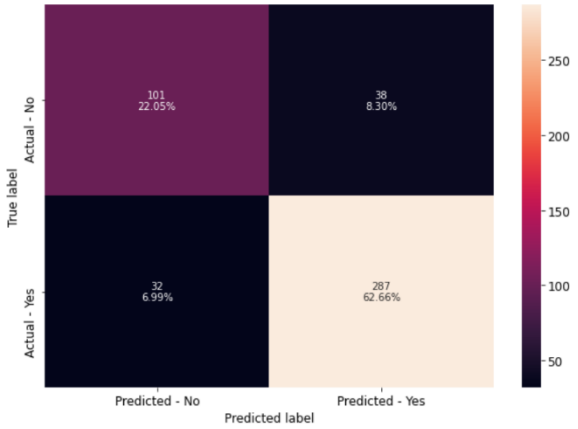
Train Classification Report

	precision	recall	f1-score	support
0	0.71	0.69	0.70	323
1	0.87	0.88	0.87	744
accuracy			0.82	1067
macro avg	0.79	0.78	0.79	1067
weighted avg	0.82	0.82	0.82	1067

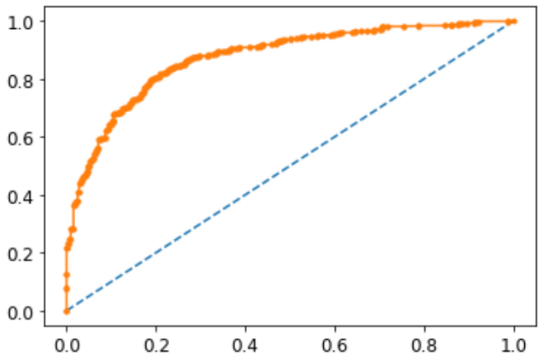
Test Classification Report

	precision	recall	f1-score	support
0	0.76	0.73	0.74	139
1	0.88	0.90	0.89	319
accuracy			0.85	458
macro avg	0.82	0.81	0.82	458
weighted avg	0.85	0.85	0.85	458

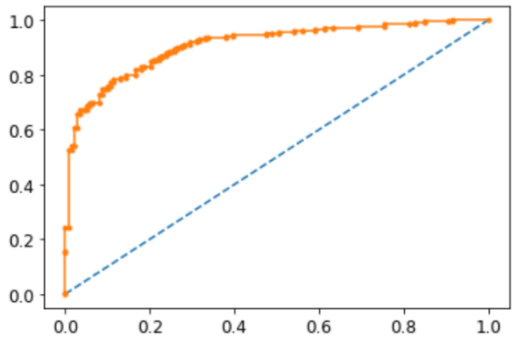
```
Accuracy on training set :82%
Accuracy on test set : 85%
Recall on training set : 88%
Recall on test set : 90%
Precision on training set : 87%
Precision on test set : 88%
F1 score on training set : 87%
F1 score on test set : 89%
```



AUC Training Data for Naive Bayes: 0.874



AUC Test Data for Naive Bayes: 0.910



Interpretation (Naïve Bayes):

- Accuracy for train and test is **almost same** (i.e., train: ~82%, test: ~85%)
- Recall for train and test is **almost same** (i.e., train: ~88%, test: ~90%)
- Precision for train and test is **almost same** (i.e., train: ~87%, test: ~88%)
- F1-score for train and test is **almost same** (i.e., train: ~87%, test: ~89%)
- AUC for train and test is **almost same** (i.e., train: ~87%, test: ~91%)

So, Let's go for bagging and boosting models and interpret the results

3.3 Bagging *(Random Forest should be applied for Bagging)* & Boosting models. Interpret the results

Bagging Classifier

- We have imported *BaggingClassifier and RandomForestClassifier* model from sklearn library.
- Considered *RandomForestClassifier* model as base estimator
- Load the model and fit the model in train data
- Predicted the values for train and test data to check accuracy of the model on train data & test data.
- We have further included classification report, confusion matrix and AUC score

```
Train Accuracy: 96%
Test Accuracy: 83%
```

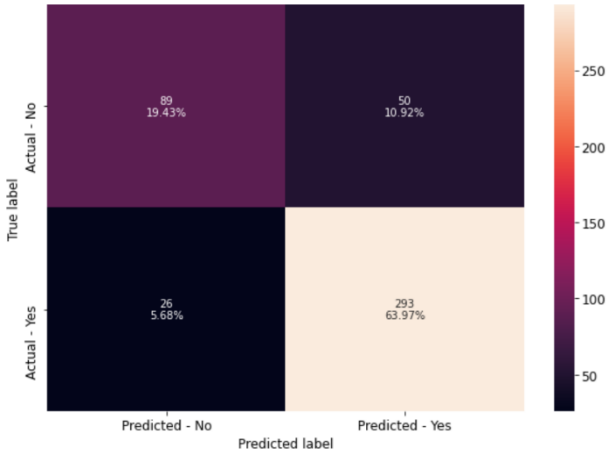
Train Classification Report

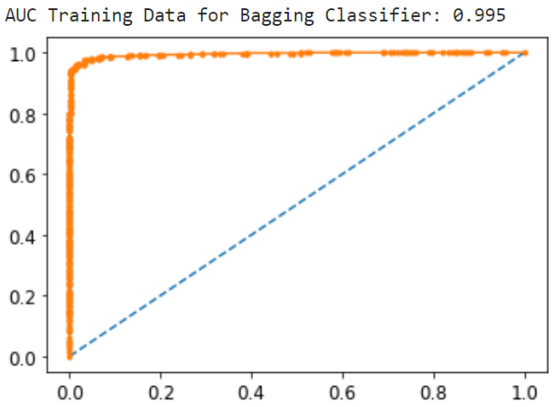
	precision	recall	f1-score	support
0	0.97	0.91	0.94	323
1	0.96	0.99	0.97	744
accuracy			0.96	1067
macro avg	0.96	0.95	0.96	1067
weighted avg	0.96	0.96	0.96	1067

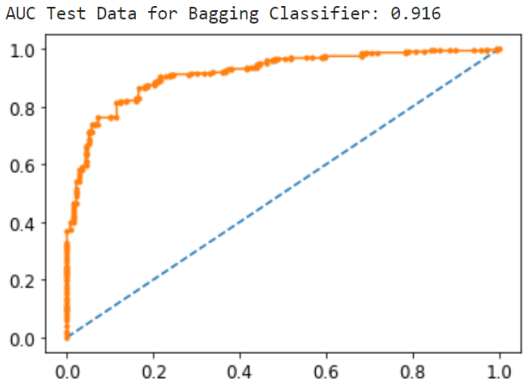
Test Classification Report

	precision	recall	f1-score	support
0	0.77	0.64	0.70	139
1	0.85	0.92	0.89	319
accuracy			0.83	458
macro avg	0.81	0.78	0.79	458
weighted avg	0.83	0.83	0.83	458

```
Accuracy on training set :96%
Accuracy on test set : 83%
Recall on training set : 99%
Recall on test set : 92%
Precision on training set : 96%
Precision on test set : 85%
F1 score on training set : 97%
F1 score on test set : 89%
```







Interpretation (Bagging Classifier(RF as base model)):

- Accuracy for train and test having **overfit of ~13%** (i.e., train: 96%, test: 83%)
- Recall for train and test having **overfit of ~7%** (i.e., train: 99%, test: 92%)
- Precision for train and test having **overfit of ~11%** (i.e., train: 96%, test: 85%)
- F1-score for train and test having **overfit of ~8%** (i.e., train: 97%, test: 89%)
- AUC for train and test is ~100% and ~92% respectively.

So, it looks like with using RF as base model bagging classifier model gives overfit in all parameter. We need to do model tuning for sure to get rid of overfit.

Boosting Model (1. Adaboost)

- We have imported *AdaBoostClassifier* from sklearn library.
- Load the model and fit the model in train data with learning rate equals to 0.01
- Predicted the values for train and test data to check accuracy of the model on train data & test data.
- We have further included classification report, confusion matrix and AUC score

```
Train Accuracy: 83%
Test Accuracy: 84%
```

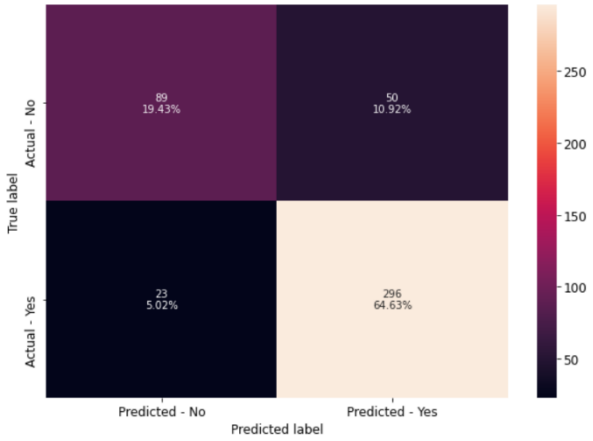
Train Classification Report

	precision	recall	f1-score	support
0	0.76	0.65	0.70	323
1	0.86	0.91	0.88	744
accuracy			0.83	1067
macro avg	0.81	0.78	0.79	1067
weighted avg	0.83	0.83	0.83	1067

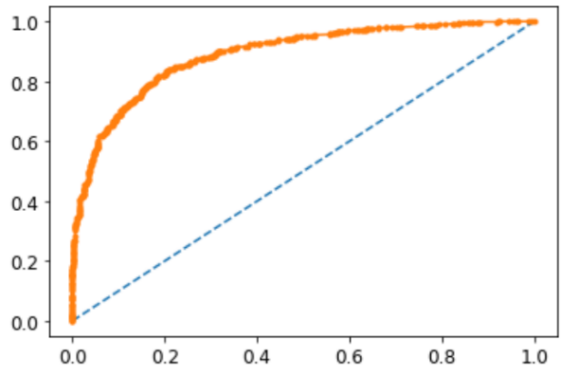
Test Classification Report

	precision	recall	f1-score	support
0	0.79	0.64	0.71	139
1	0.86	0.93	0.89	319
accuracy			0.84	458
macro avg	0.83	0.78	0.80	458
weighted avg	0.84	0.84	0.84	458

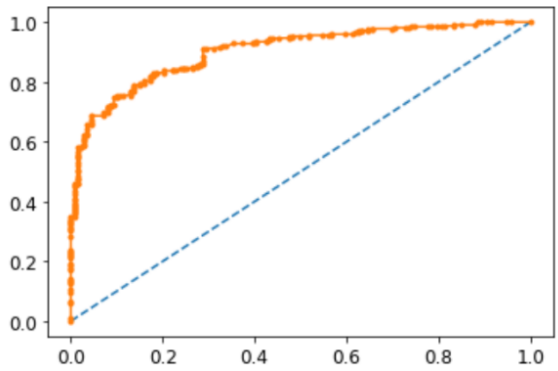
```
Accuracy on training set :83%
Accuracy on test set : 84%
Recall on training set : 91%
Recall on test set : 93%
Precision on training set : 86%
Precision on test set : 86%
F1 score on training set : 88%
F1 score on test set : 89%
```



AUC Training Data for AdaBoost: 0.891



AUC Test Data for AdaBoost: 0.905



Interpretation (Adaboost):

- Accuracy for train and test is **almost same** (i.e., train: ~83%, test: ~84%)
- Recall for train and test is **almost same** (i.e., train: ~91%, test: ~93%)
- Precision for train and test is **same** (i.e., train: ~86%, test: ~86%)
- F1-score for train and test is **almost same** (i.e., train: ~88%, test: ~89%)
- AUC for train and test is **almost same** (i.e., train: ~89%, test: ~91%)

Boosting Model (2. Gradient Boosting)

- We have imported *GradientBoostingClassifier* from sklearn library.
- Load the model and fit the model in train data
- Predicted the values for train and test data to check accuracy of the model on train data & test data.
- We have further included classification report, confusion matrix and AUC score

```
Train Accuracy: 89%
Test Accuracy: 84%
```

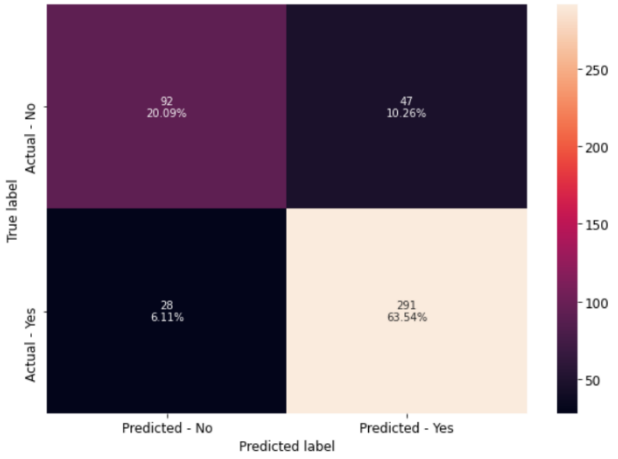
Train Classification Report

	precision	recall	f1-score	support
0	0.85	0.76	0.80	323
1	0.90	0.94	0.92	744
accuracy			0.89	1067
macro avg	0.88	0.85	0.86	1067
weighted avg	0.89	0.89	0.89	1067

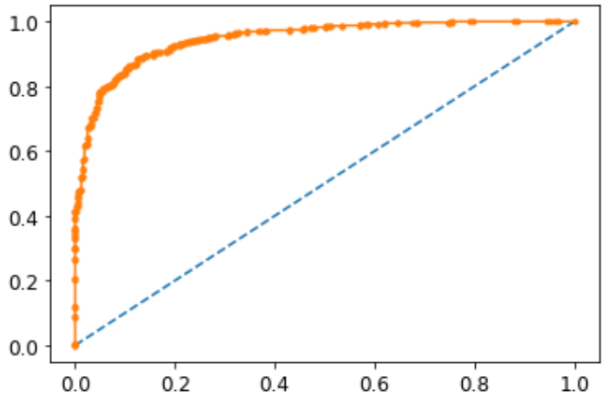
Test Classification Report

	precision	recall	f1-score	support
0	0.77	0.66	0.71	139
1	0.86	0.91	0.89	319
accuracy			0.84	458
macro avg	0.81	0.79	0.80	458
weighted avg	0.83	0.84	0.83	458

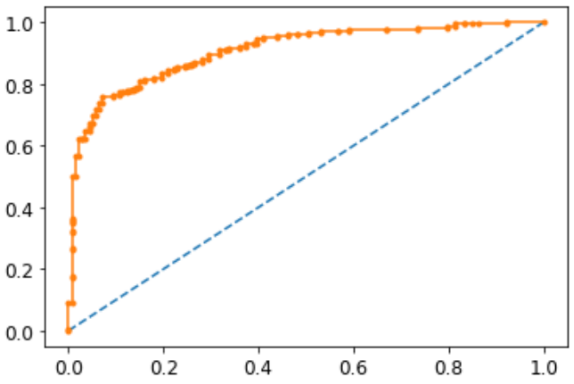
```
Accuracy on training set :89%
Accuracy on test set : 84%
Recall on training set : 94%
Recall on test set : 91%
Precision on training set : 90%
Precision on test set : 86%
F1 score on training set : 92%
F1 score on test set : 89%
```



AUC Training Data for Gradient Boosting: 0.948



AUC Test Data for Gradient Boosting: 0.908



Interpretation (Gradient Boosting):

- Accuracy for train and test having **slight overfit** (i.e., train: ~89%, test: ~84%)
- Recall for train and test is **almost same** (i.e., train: ~94%, test: ~91%)
- Precision for train and test is **almost same** (i.e., train: ~90%, test: ~86%)
- F1-score for train and test is **almost same** (i.e., train: ~92%, test: ~89%)
- AUC for train and test is **almost same** (i.e., train: ~95%, test: ~91%)

Boosting Model (3. XGboost)

- We have imported *XGBClassifier* from xgboost library.
- Load the model and fit the model in train data
- Predicted the values for train and test data to check accuracy of the model on train data & test data.
- We have further included classification report, confusion matrix and AUC score

```
Train Accuracy: 99%
Test Accuracy: 83%
```

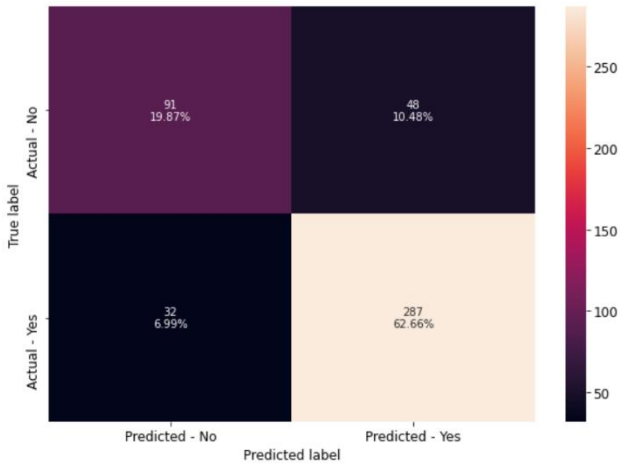
Train Classification Report

	precision	recall	f1-score	support
0	0.99	0.98	0.98	323
1	0.99	1.00	0.99	744
accuracy			0.99	1067
macro avg	0.99	0.99	0.99	1067
weighted avg	0.99	0.99	0.99	1067

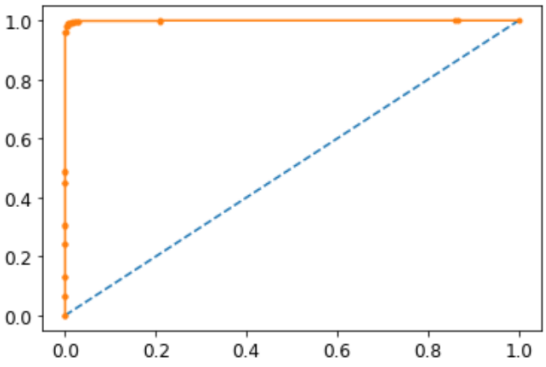
Test Classification Report

	precision	recall	f1-score	support
0	0.74	0.65	0.69	139
1	0.86	0.90	0.88	319
accuracy			0.83	458
macro avg	0.80	0.78	0.79	458
weighted avg	0.82	0.83	0.82	458

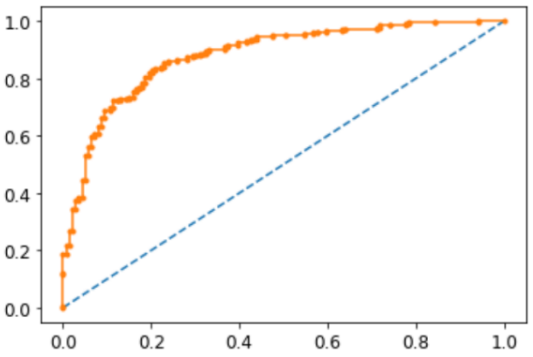
```
Accuracy on training set :99%
Accuracy on test set : 83%
Recall on training set : 100%
Recall on test set : 90%
Precision on training set : 99%
Precision on test set : 86%
F1 score on training set : 99%
F1 score on test set : 88%
```



AUC Training Data for XGBoost: 0.999



AUC Test Data for XGBoost: 0.881



Interpretation (XGboost):

- Accuracy for train and test having **overfit of ~16%** (i.e., train: ~99%, test: ~83%)
- Recall for train and test having **overfit of ~10%** (i.e., train: ~100%, test: ~90%)
- Precision for train and test having **overfit of ~13%** (i.e., train: ~99%, test: ~86%)
- F1-score for train and test having **overfit of ~11%** (i.e., train: ~99%, test: ~88%)
- AUC for train and test is ~100% and ~88% respectively.

We have good amount of overfit with default parameter of XGboost. Model tunning would be helpful to get rid of the same.

Boosting Model (4. SVM (Support Vector Machine))

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

Steps:

- We have imported *svm* from sklearn library.
- Load the model and fit the model in train data with probability equals to true.
- Predicted the values for train and test data to check accuracy of the model on train data & test data.

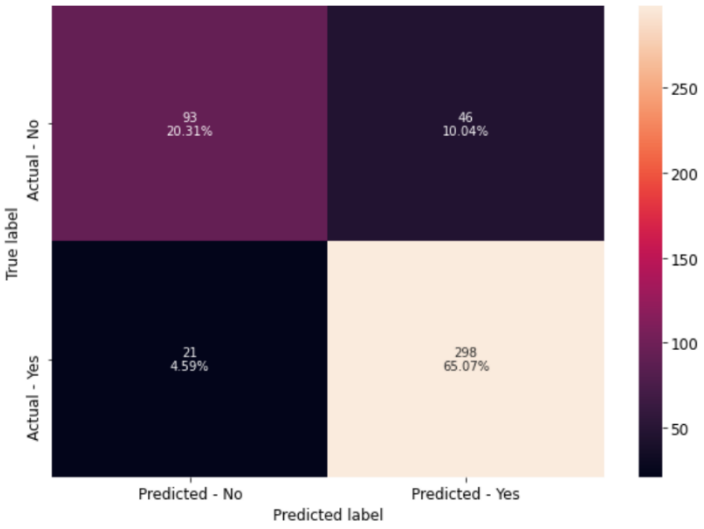
```
Train Accuracy: 83%
Test Accuracy: 85%


Train Classification Report
```

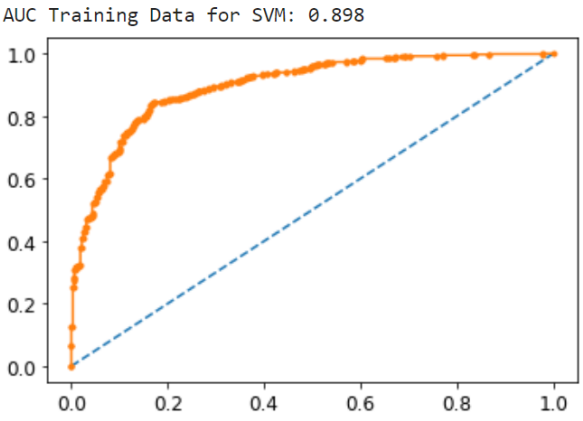
	precision	recall	f1-score	support
0	0.77	0.64	0.70	323
1	0.86	0.92	0.89	744
accuracy			0.83	1067
macro avg	0.81	0.78	0.79	1067
weighted avg	0.83	0.83	0.83	1067

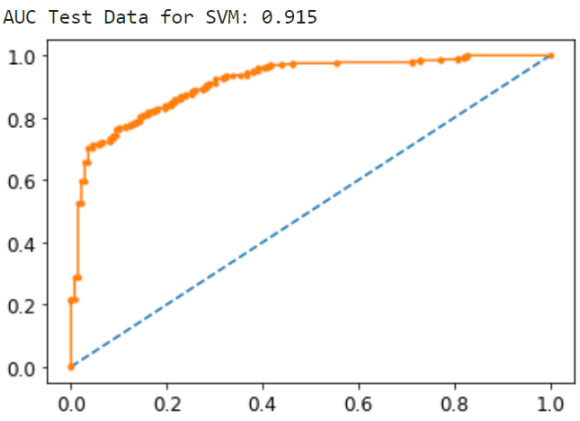
```
Test Classification Report
```

	precision	recall	f1-score	support
0	0.82	0.67	0.74	139
1	0.87	0.93	0.90	319
accuracy			0.85	458
macro avg	0.84	0.80	0.82	458
weighted avg	0.85	0.85	0.85	458

```
Accuracy on training set :83%
Accuracy on test set : 85%
Recall on training set : 92%
Recall on test set : 93%
Precision on training set : 86%
Precision on test set : 87%
F1 score on training set : 89%
F1 score on test set : 90%
```







Interpretation (SVM):

- Accuracy for train and test is **almost same** (i.e., train: ~83%, test: ~85%)
- Recall for train and test is **almost same** (i.e., train: ~92%, test: ~93%)
- Precision for train and test is **almost same** (i.e., train: ~86%, test: ~87%)
- F1-score for train and test having **almost same** (i.e., train: ~89%, test: ~90%)
- AUC for train and test is **almost same** (i.e., train: ~89%, test: ~92%)

Model Summary:

	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision	Train_F1 score	Test_F1 score
0	Logistic Regression	0.83	0.85	0.91	0.92	0.86	0.87	0.88	0.90
1	LDA	0.83	0.84	0.90	0.91	0.86	0.87	0.88	0.89
2	KNN	0.86	0.84	0.91	0.91	0.89	0.87	0.90	0.89
3	Naive Bayes	0.82	0.85	0.88	0.90	0.87	0.88	0.87	0.89
4	Bagging	0.96	0.83	0.99	0.92	0.96	0.85	0.97	0.89
5	AdaBoost	0.83	0.84	0.91	0.93	0.86	0.86	0.88	0.89
6	Gradient Boosting	0.89	0.84	0.94	0.91	0.90	0.86	0.92	0.89
7	XGBoost	0.99	0.83	1.00	0.90	0.99	0.86	0.99	0.88
8	Support Vector Machine(SVM)	0.83	0.85	0.92	0.93	0.86	0.87	0.89	0.90

Based on above models **without tunning, SVM is the better model on all parameters BUT** *Let's do model tunning and try to remove overfit or underfit and find the best model.*

3.4 Model tuning and Interpret the results

For Model tuning we would be using GridSearchCV using hyperparameters. GridSearchCV is a library function that is a member of sklearn's model_selection package. It helps to loop through predefined hyperparameters and fit your estimator (model) on your training set.

We are trying to use hyperparameters such a way that it will reduce the overfit and gives the best result. So, Let's start model tuning (import the *GridSearchCV* from sklearn)

Logistic Regression (model tuning)

- Define the model as *LogisticRegression(random_state=1)*
- Define below parameters for GridSearch
    - solvers = ['newton-cg', 'lbfgs', 'liblinear','sag','saga']
    - penalty = ['l2']
    - c_values = [100, 10, 1.0, 0.1, 0.01]
- Define the Grid and fit the model in train data
- Find the best parameter using *grid_search_lr.best_params_* function

```
{'C': 100, 'penalty': 'l2', 'solver': 'saga'}
```

- Save this parameter as best grid and predicted the values for train and test data to check accuracy of the model on train data & test data.
- We have further included classification report, confusion matrix and AUC score

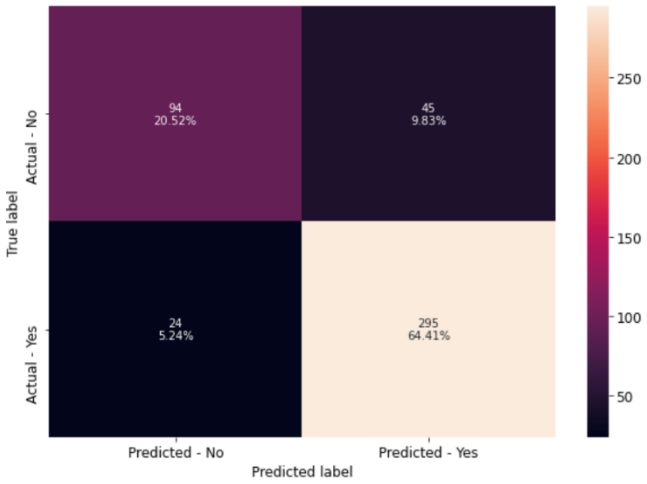
```
Train Accuracy: 83%
Test Accuracy: 85%
```

```
Train Classification Report
```

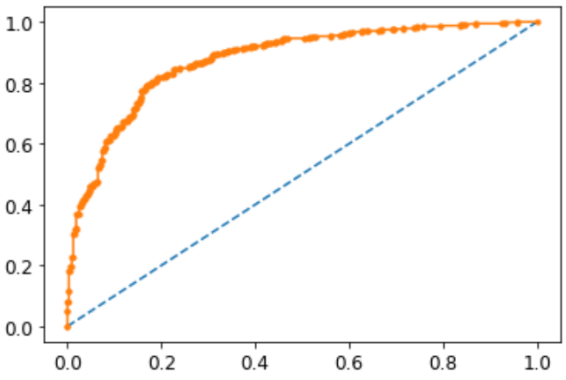
	precision	recall	f1-score	support
0	0.75	0.65	0.70	323
1	0.86	0.91	0.88	744
accuracy			0.83	1067
macro avg	0.80	0.78	0.79	1067
weighted avg	0.83	0.83	0.83	1067

```
Test Classification Report
```

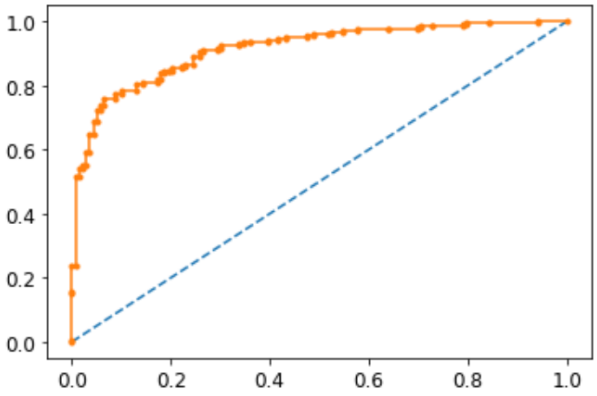
	precision	recall	f1-score	support
0	0.80	0.68	0.73	139
1	0.87	0.92	0.90	319
accuracy			0.85	458
macro avg	0.83	0.80	0.81	458
weighted avg	0.85	0.85	0.85	458




```
AUC Training Data for Logistic Regression(tuned): 0.877
```




```
AUC Test Data for Logistic Regression(tuned): 0.914
```



	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision	Train_F1 score	Test_F1 score
0	Logistic Regression	0.83	0.85	0.91	0.92	0.86	0.87	0.88	0.9
1	Logistic Regression (Tuned)	0.83	0.85	0.91	0.92	0.86	0.87	0.88	0.9

Interpretation (Logistic Regression - tuned):

- After model tuning, we don't see any improvement in performance metrics.

LDA (model tuning)

- Define the model as *LinearDiscriminantAnalysis()*
- Define below parameters for GridSearch
  - solvers = ['svd', 'lsqr', 'eigen']
  - tol : [0.0001,0.0002,0.0003]
- Define the Grid and fit the model in train data
- Find the best parameter using *grid_search_lr.best_params_* function

```
{'solver': 'svd', 'tol': 0.0001}
```

- Save this parameter as best grid and predicted the values for train and test data to check accuracy of the model on train data & test data.
- We have further included classification report, confusion matrix and AUC score

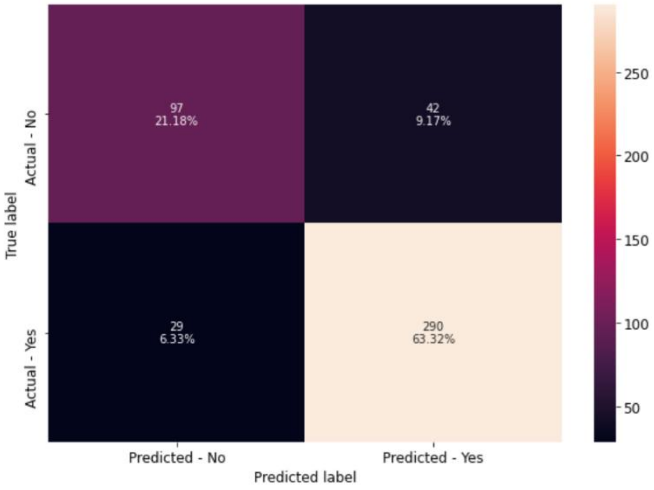
```
Train Accuracy: 83%
Test Accuracy: 84%
```

```
Train Classification Report
```

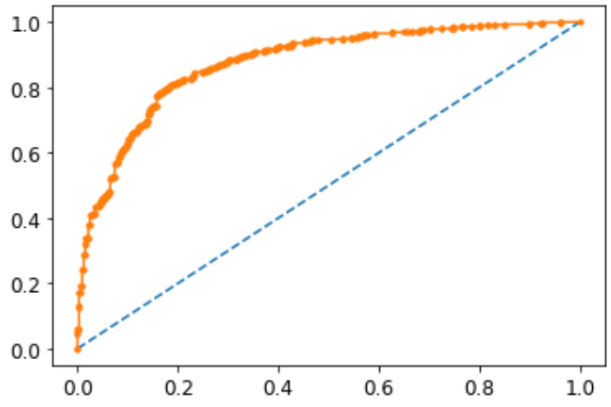
	precision	recall	f1-score	support
0	0.73	0.67	0.70	323
1	0.86	0.90	0.88	744
accuracy			0.83	1067
macro avg	0.80	0.78	0.79	1067
weighted avg	0.82	0.83	0.82	1067

```
Test Classification Report
```

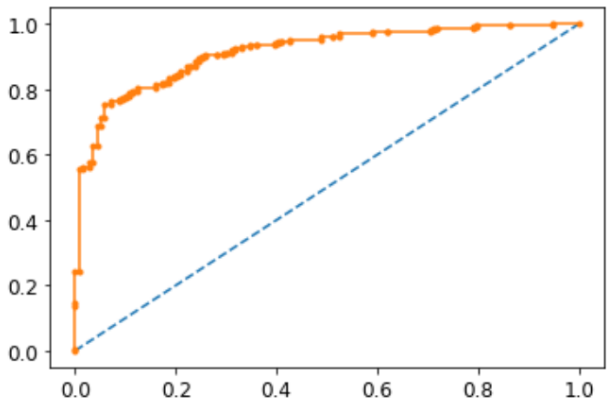
	precision	recall	f1-score	support
0	0.77	0.70	0.73	139
1	0.87	0.91	0.89	319
accuracy			0.84	458
macro avg	0.82	0.80	0.81	458
weighted avg	0.84	0.84	0.84	458



AUC Training Data for LDA(tuned): 0.876



AUC Test Data for LDA(tuned): 0.915



	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision	Train_F1 score	Test_F1 score
0	LDA	0.83	0.84	0.9	0.91	0.86	0.87	0.88	0.89
1	LDA (Tuned)	0.83	0.84	0.9	0.91	0.86	0.87	0.88	0.89

Interpretation (LDA - tuned):

- After model tuning, we don't see any improvement in performance metrics.

KNN (model tuning)

- Define the model as *KNeighborsClassifier()*
- Define below parameters for GridSearch
  - n_neighbors: [1,2,3,5,7,9,11,19]
  - weights: ['uniform', 'distance']
  - metric :['euclidean','manhattan']
  - leaf_size: [1,2,5,10,20,30,50]
- Define the Grid and fit the model in train data
- Find the best parameter using *grid_search_lr.best_params_* function

```
{'leaf_size': 20,
 'metric': 'manhattan',
 'n_neighbors': 19,
 'weights': 'uniform'}
```

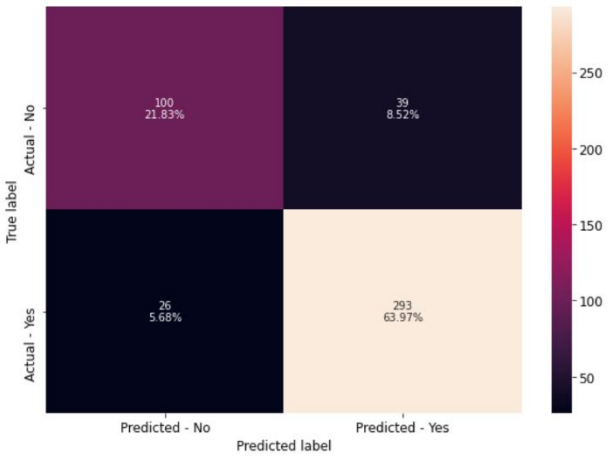
- Save this parameter as best grid and predicted the values for train and test data to check accuracy of the model on train data & test data.
- We have further included classification report, confusion matrix and AUC score

```
Train Accuracy: 84%
Test Accuracy: 86%


Train Classification Report
```

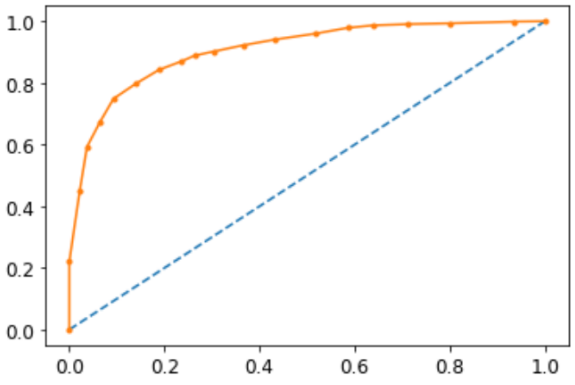
	precision	recall	f1-score	support
0	0.76	0.70	0.72	323
1	0.87	0.90	0.89	744
accuracy			0.84	1067
macro avg	0.81	0.80	0.81	1067
weighted avg	0.84	0.84	0.84	1067

```
Test Classification Report
```

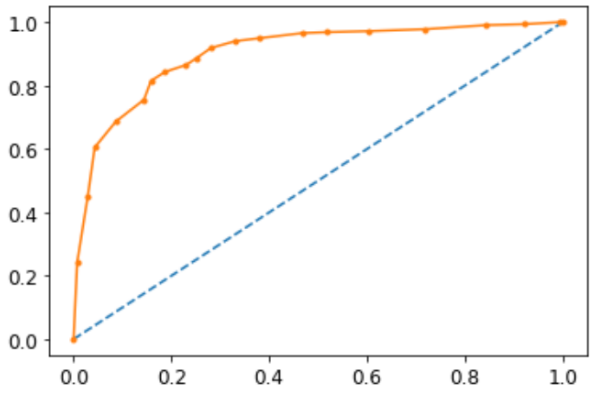
	precision	recall	f1-score	support
0	0.79	0.72	0.75	139
1	0.88	0.92	0.90	319
accuracy			0.86	458
macro avg	0.84	0.82	0.83	458
weighted avg	0.86	0.86	0.86	458



AUC Training Data for KNN(tuned): 0.907



AUC Test Data for KNN(tuned): 0.902



	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision	Train_F1 score	Test_F1 score
0	KNN	0.86	0.84	0.91	0.91	0.89	0.87	0.90	0.89
1	KNN (Tuned)	0.84	0.86	0.90	0.92	0.87	0.88	0.89	0.90

Interpretation (KNN - tuned):

- After model tuning, we got a *slight improvement* in all performance metrics parameters.

Naïve Bayes (model tuning)

- Define the model as *GaussianNB()*
- Define below parameters for GridSearch
  - var_smoothing: np.logspace(0,-9, num=100)
- Define the Grid and fit the model in train data
- Find the best parameter using *grid_search_lr.best_params_* function

```
{'var_smoothing': 0.0023101297000831605}
```

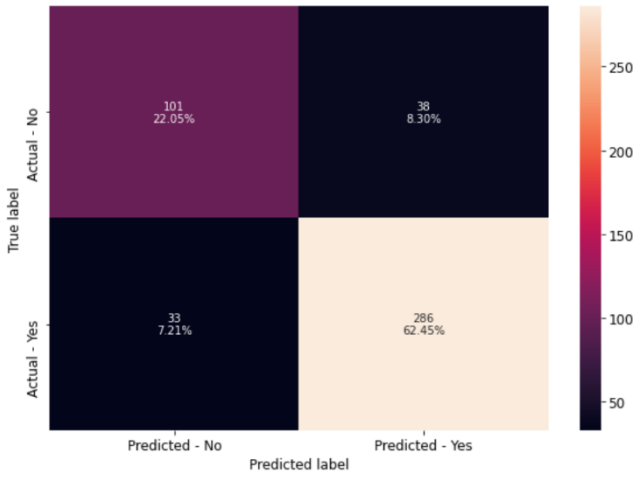
- Save this parameter as best grid and predicted the values for train and test data to check accuracy of the model on train data & test data.
- We have further included classification report, confusion matrix and AUC score

```
Train Accuracy: 82%
Test Accuracy: 84%


Train Classification Report
```

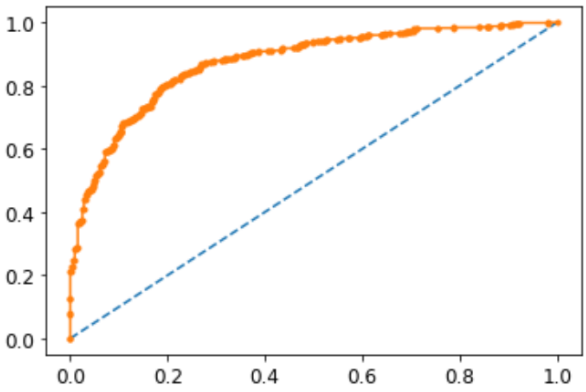
	precision	recall	f1-score	support
0	0.71	0.69	0.70	323
1	0.87	0.88	0.87	744
accuracy			0.82	1067
macro avg	0.79	0.79	0.79	1067
weighted avg	0.82	0.82	0.82	1067

```
Test Classification Report
```

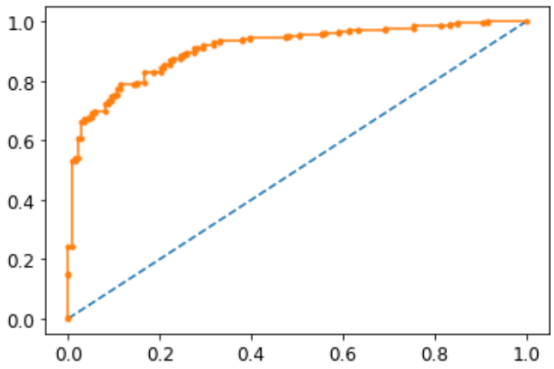
	precision	recall	f1-score	support
0	0.75	0.73	0.74	139
1	0.88	0.90	0.89	319
accuracy			0.84	458
macro avg	0.82	0.81	0.81	458
weighted avg	0.84	0.84	0.84	458



AUC Training Data for Naive Bayes(tuned): 0.874



AUC Test Data for Naive Bayes(tuned): 0.910



	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision	Train_F1 score	Test_F1 score
0	Naive Bayes	0.82	0.85	0.88	0.9	0.87	0.88	0.87	0.89
1	Naive Bayes (Tuned)	0.82	0.84	0.88	0.9	0.87	0.88	0.87	0.89

Interpretation (Naïve Bayes - tuned):

- After model tuning, we don't see any improvement in performance metrics parameters.

Bagging (model tuning)

- Define the model as *RandomForestClassifier(random_state=1)*
- Define below parameters for GridSearch
  - 'bootstrap': [True],
  - 'max_depth': [1,2,5,10],
  - 'max_features': [3,4],
  - 'min_samples_leaf': [4,5],
  - 'min_samples_split': [2,4,6],
  - 'n_estimators': [100, 200, 300]
- Define the Grid and fit the model in train data
- Find the best parameter using *grid_search_lr.best_params_* function

```
{'bootstrap': True,
 'max_depth': 5,
 'max_features': 4,
 'min_samples_leaf': 5,
 'min_samples_split': 2,
 'n_estimators': 300}
```

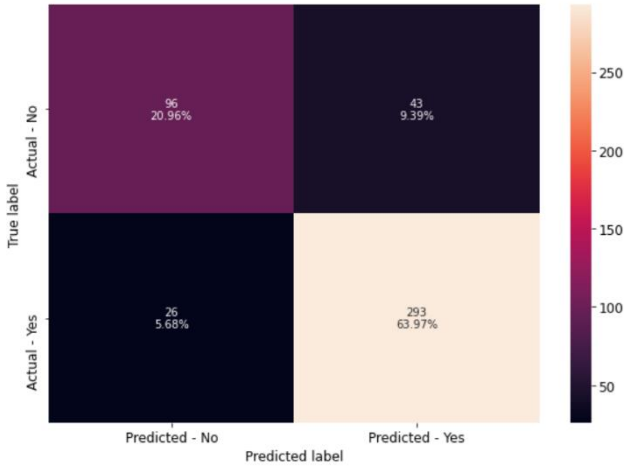
- Save this parameter as best grid and predicted the values for train and test data to check accuracy of the model on train data & test data.
- We have further included classification report, confusion matrix and AUC score

```
Train Accuracy: 87%
Test Accuracy: 85%


Train Classification Report
```

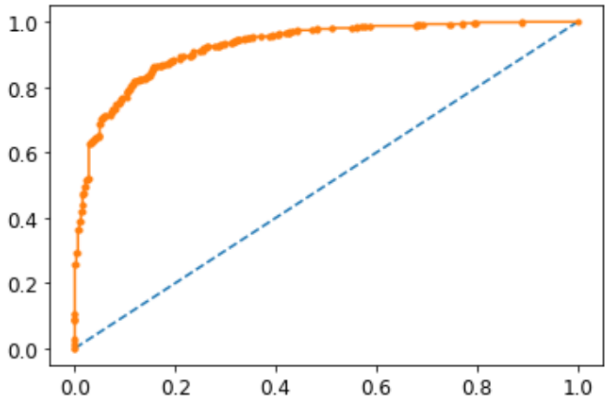
	precision	recall	f1-score	support
0	0.80	0.74	0.77	323
1	0.89	0.92	0.91	744
accuracy			0.87	1067
macro avg	0.85	0.83	0.84	1067
weighted avg	0.86	0.87	0.87	1067

```
Test Classification Report
```

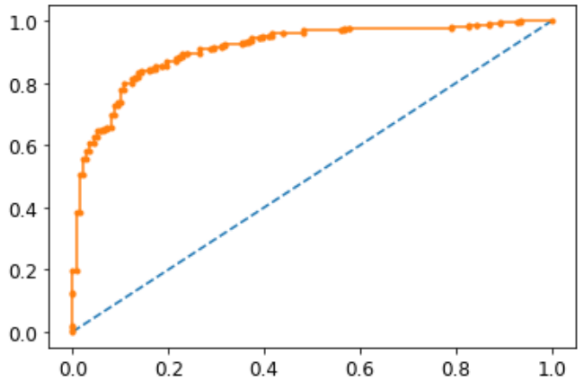
	precision	recall	f1-score	support
0	0.79	0.69	0.74	139
1	0.87	0.92	0.89	319
accuracy			0.85	458
macro avg	0.83	0.80	0.82	458
weighted avg	0.85	0.85	0.85	458



AUC Training Data for Bagging Classifier(tuned): 0.926



AUC Test Data for Bagging Classifier(tuned): 0.912



	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision	Train_F1 score	Test_F1 score
0	Bagging Classifier	0.96	0.83	0.99	0.92	0.96	0.85	0.97	0.89
1	Bagging Classifier (Tuned)	0.87	0.85	0.92	0.92	0.89	0.87	0.91	0.89

Interpretation (Bagging Classifier - tuned):

- We could see there is no more overfit, and model looks more promising.

Adaboost (model tuning)

- Define the model as *AdaBoostClassifier(random_state=1)*
- Define below parameters for GridSearch
  - "base_estimator":[DecisionTreeClassifier(max_depth=1), DecisionTreeClassifier(max_depth=2), DecisionTreeClassifier(max_depth=3)],
  - "n_estimators": np.arange(10,120,20),
  - "learning_rate": np.arange(0.1,2,0.1)
- Define the Grid and fit the model in train data
- Find the best parameter using *grid_search_lr.best_params_* function

```
{'base_estimator': DecisionTreeClassifier(max_depth=3),
 'learning_rate': 0.2,
 'n_estimators': 10}
```

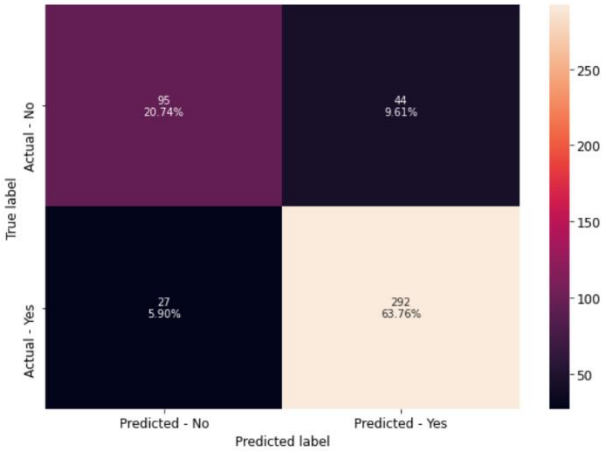
- Save this parameter as best grid and predicted the values for train and test data to check accuracy of the model on train data & test data.
- We have further included classification report, confusion matrix and AUC score

```
Train Accuracy: 86%
Test Accuracy: 84%


Train Classification Report
```

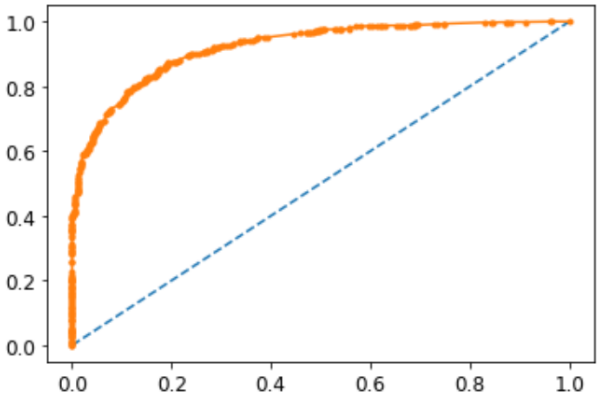
	precision	recall	f1-score	support
0	0.79	0.71	0.75	323
1	0.88	0.92	0.90	744
accuracy			0.86	1067
macro avg	0.84	0.81	0.82	1067
weighted avg	0.85	0.86	0.85	1067

```
Test Classification Report
```

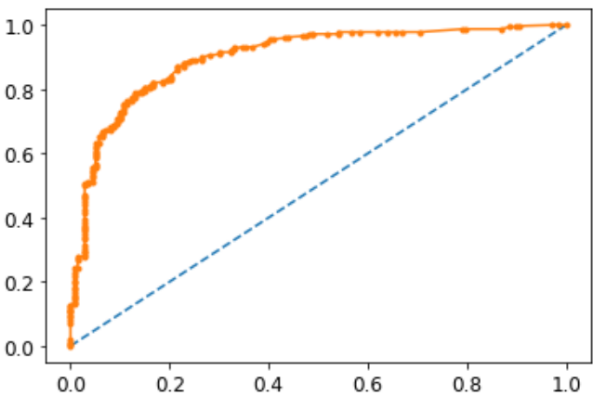
	precision	recall	f1-score	support
0	0.78	0.68	0.73	139
1	0.87	0.92	0.89	319
accuracy			0.84	458
macro avg	0.82	0.80	0.81	458
weighted avg	0.84	0.84	0.84	458




```
AUC Training Data for AdaBoost(tuned): 0.922
```




```
AUC Test Data for AdaBoost(tuned): 0.901
```



	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision	Train_F1 score	Test_F1 score
0	Adaboost	0.83	0.84	0.91	0.93	0.86	0.86	0.88	0.89
1	Adaboost (Tuned)	0.86	0.84	0.92	0.92	0.88	0.87	0.90	0.89

Interpretation (Adaboost - tuned):

- After model tuning, we don't see much improvement performance metrics parameters.

Gradient Boosting (model tuning : init parameter)
Most of the hyperparameters available are same as random forest classifier.

init: An estimator object that is used to compute the initial predictions. If 'zero', the initial raw predictions are set to zero. By default, a DummyEstimator predicting the classes priors is used.

There is no class_weights parameter in gradient boosting.

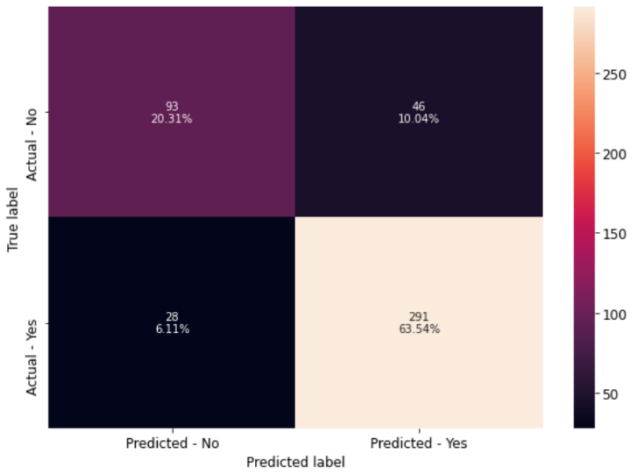
- Define the model as *GradientBoostingClassifier(init=AdaBoostClassifier(random_state=1),random_state=1)* and use init parameter as *AdaBoostClassifier*.
- Fit the model in train data
- Predict the values for train and test data to check accuracy of the model on train data & test data.
- We have further included classification report, confusion matrix and AUC score

```
Train Accuracy: 89%
Test Accuracy: 84%


Train Classification Report
```

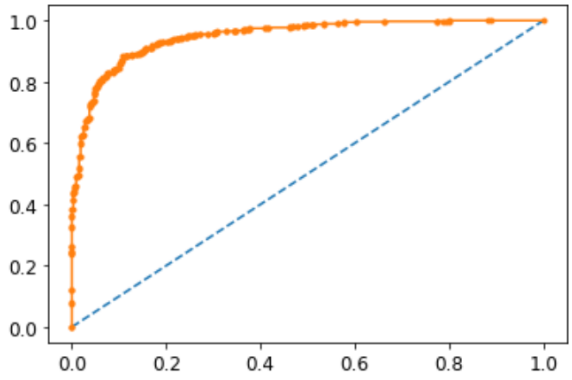
	precision	recall	f1-score	support
0	0.85	0.77	0.81	323
1	0.90	0.94	0.92	744
accuracy			0.89	1067
macro avg	0.88	0.86	0.87	1067
weighted avg	0.89	0.89	0.89	1067

```
Test Classification Report
```

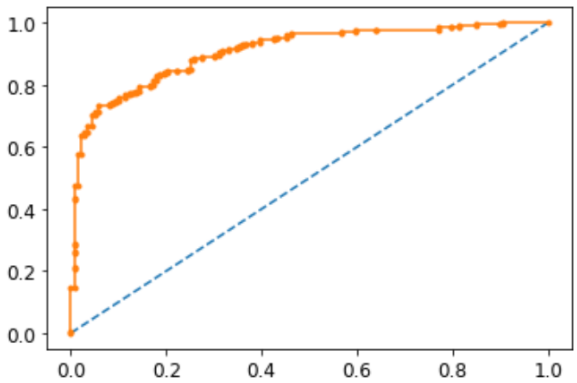
	precision	recall	f1-score	support
0	0.77	0.67	0.72	139
1	0.86	0.91	0.89	319
accuracy			0.84	458
macro avg	0.82	0.79	0.80	458
weighted avg	0.83	0.84	0.84	458




```
AUC Training Data for Logistic Regression(tuned): 0.949
```




```
AUC Test Data for Logistic Regression(tuned): 0.909
```



	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision	Train_F1 score	Test_F1 score
0	Gradient Boosting	0.89	0.84	0.94	0.91	0.9	0.86	0.92	0.89
1	Gradient Boosting (init)	0.89	0.84	0.94	0.91	0.9	0.86	0.92	0.89

Interpretation (Gradient Boosting – init parameter):

- After model tuning, we don't see much improvement performance metrics parameters.

Gradient Boosting (model tuning)

- Define the model as *GradientBoostingClassifier(init=AdaBoostClassifier(random_state=1),random_state=1)* and use init parameter as *AdaBoostClassifier*.
- Define below parameters for GridSearch
    - "n_estimators": [100,150,200,250],
    - "subsample":[0.7,0.8,0.9,1],
    - "max_features":[0.7,0.8,0.9,1]
- Define the Grid and fit the model in train data
- Find the best parameter using *grid_search_lr.best_params_* function

```
{'max_features': 1, 'n_estimators': 250, 'subsample': 0.9}
```

- Save this parameter as best grid and predicted the values for train and test data to check accuracy of the model on train data & test data.

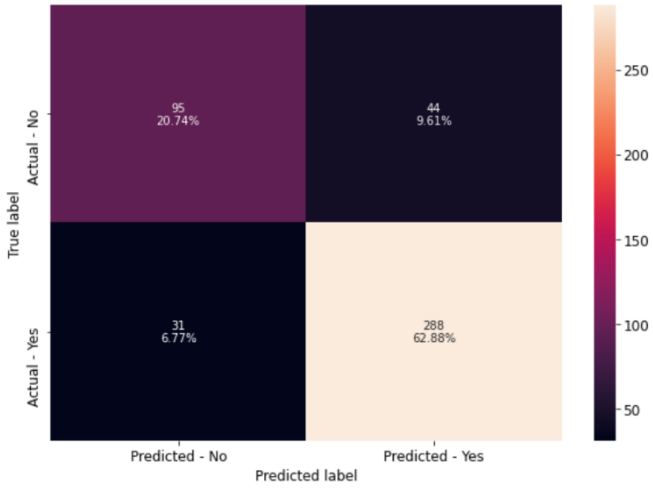
- We have further included classification report, confusion matrix and AUC score

```
Train Accuracy: 91%
Test Accuracy: 84%


Train Classification Report
```

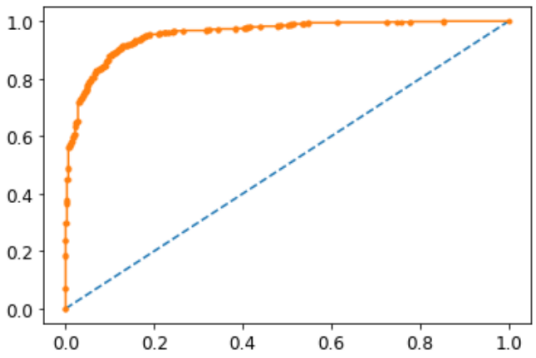
	precision	recall	f1-score	support
0	0.88	0.82	0.85	323
1	0.92	0.95	0.94	744
accuracy			0.91	1067
macro avg	0.90	0.89	0.89	1067
weighted avg	0.91	0.91	0.91	1067

```
Test Classification Report
```

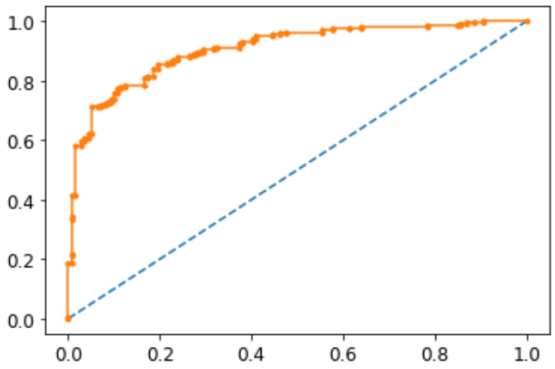
	precision	recall	f1-score	support
0	0.75	0.68	0.72	139
1	0.87	0.90	0.88	319
accuracy			0.84	458
macro avg	0.81	0.79	0.80	458
weighted avg	0.83	0.84	0.83	458



AUC Training Data for Gradient Boosting(tuned): 0.956



AUC Test Data for Gradient Boosting(tuned): 0.907



	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision	Train_F1 score	Test_F1 score
0	Gradient Boosting (init)	0.89	0.84	0.94	0.91	0.90	0.86	0.92	0.89
1	Gradient Boosting (Tuned)	0.91	0.84	0.95	0.90	0.92	0.87	0.94	0.88

Interpretation (Gradient Boosting – tuned):

- After model tuning, we could see overfit has increased.

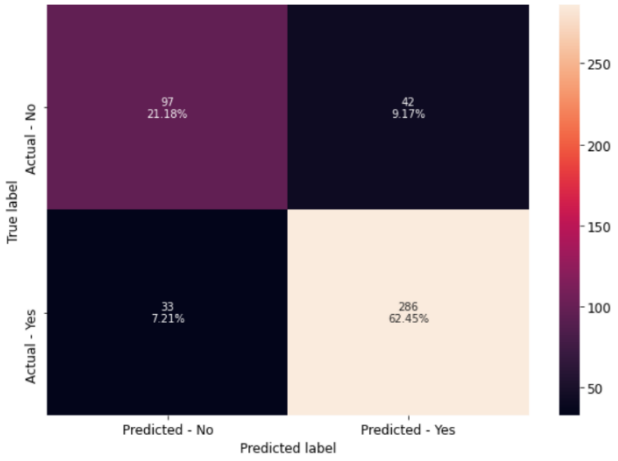
XGboost (model tuning)

- Define the model as *XGBClassifier(random_state=1)*
- Define below parameters for GridSearch
  - "n_estimators": np.arange(10,100,20),
  - "scale_pos_weight":[0,1,2],
  - "subsample":[0.5,0.7],
  - "learning_rate":[0.01,0.1,0.2,0.05],
  - "gamma":[0,1],
  - "colsample_bytree":[0.5,0.7,0.9,1],
  - "colsample_bylevel":[0.5,0.7,0.9,1]
- Define the Grid and fit the model in train data
- Find the best parameter using *grid_search_lr.best_params_* function

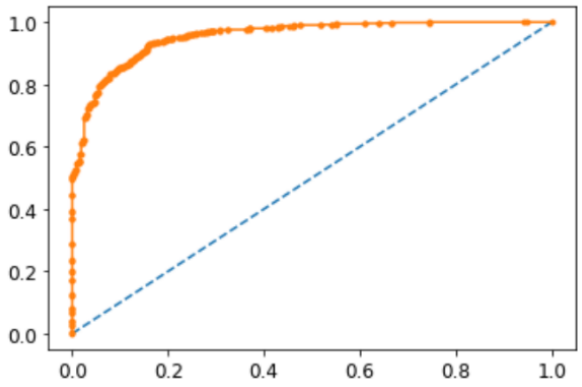
```
{'colsample_bylevel': 0.5,
 'colsample_bytree': 1,
 'gamma': 1,
 'learning_rate': 0.2,
 'n_estimators': 30,
 'scale_pos_weight': 1,
 'subsample': 0.5}
```

- Save this parameter as best grid and predicted the values for train and test data to check accuracy of the model on train data & test data.
- We have further included classification report, confusion matrix and AUC score

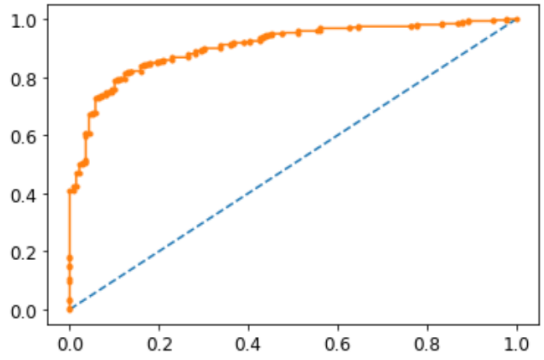
```
Train Accuracy: 90%
Test Accuracy: 84%


Train Classification Report

              precision    recall  f1-score   support

           0       0.86      0.80      0.83       323
           1       0.92      0.94      0.93       744

    accuracy                           0.90      1067
   macro avg       0.89      0.87      0.88      1067
weighted avg       0.90      0.90      0.90      1067

Test Classification Report

              precision    recall  f1-score   support

           0       0.75      0.70      0.72       139
           1       0.87      0.90      0.88       319

    accuracy                           0.84       458
   macro avg       0.81      0.80      0.80       458
weighted avg       0.83      0.84      0.83       458
```



AUC Training Data for XGBoost(tuned): 0.955



AUC Test Data for XGBoost(tuned): 0.906



	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision	Train_F1 score	Test_F1 score
0	XGBoost	0.99	0.83	1.00	0.9	0.99	0.86	0.99	0.88
1	XGBoost (Tuned)	0.90	0.84	0.94	0.9	0.92	0.87	0.93	0.88

Interpretation (XGboost – tuned):

- After model tuning, we could see improvement in overfit and model looks promising.

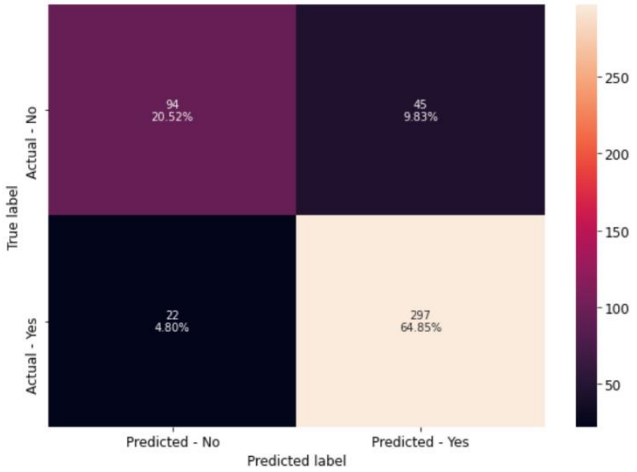
SVM (model tuning)

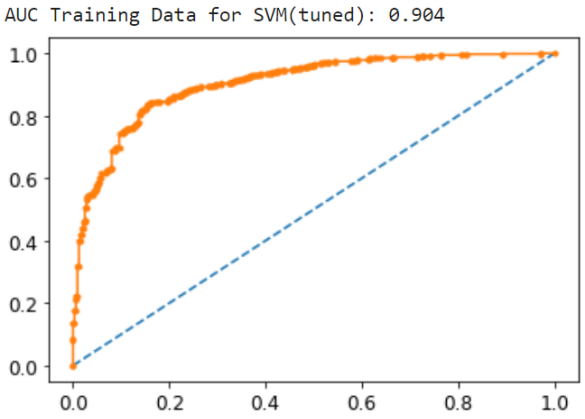
- Define the model as *svm.SVC(random_state=1,probability=True)*
- Define below parameters for GridSearch
  - 'C': [0.1,1, 10, 100],
  - 'gamma': [1,0.1,0.01,0.001],
  - 'kernel': ['rbf']
- Define the Grid and fit the model in train data
- Find the best parameter using *grid_search_lr.best_params_* function

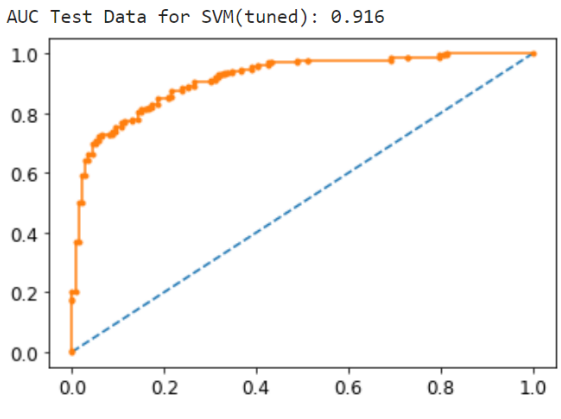
```
{'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
```

- Save this parameter as best grid and predicted the values for train and test data to check accuracy of the model on train data & test data.
- We have further included classification report, confusion matrix and AUC score

```
Train Accuracy: 84%
Test Accuracy: 85%


Train Classification Report


              precision    recall  f1-score   support

           0       0.77      0.65      0.70       323
           1       0.86      0.92      0.89       744

    accuracy                           0.84      1067
   macro avg       0.82      0.78      0.80      1067
weighted avg       0.83      0.84      0.83      1067

Test Classification Report


              precision    recall  f1-score   support

           0       0.81      0.68      0.74       139
           1       0.87      0.93      0.90       319

    accuracy                           0.85       458
   macro avg       0.84      0.80      0.82       458
weighted avg       0.85      0.85      0.85       458
```







	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision	Train_F1 score	Test_F1 score
0	SVM	0.83	0.85	0.92	0.93	0.86	0.87	0.89	0.9
1	SVM (Tuned)	0.84	0.85	0.92	0.93	0.86	0.87	0.89	0.9

Interpretation (SVM - tuned):

- After model tuning, we don't see any improvement in performance metrics parameters.

3.5 Performance Metrics

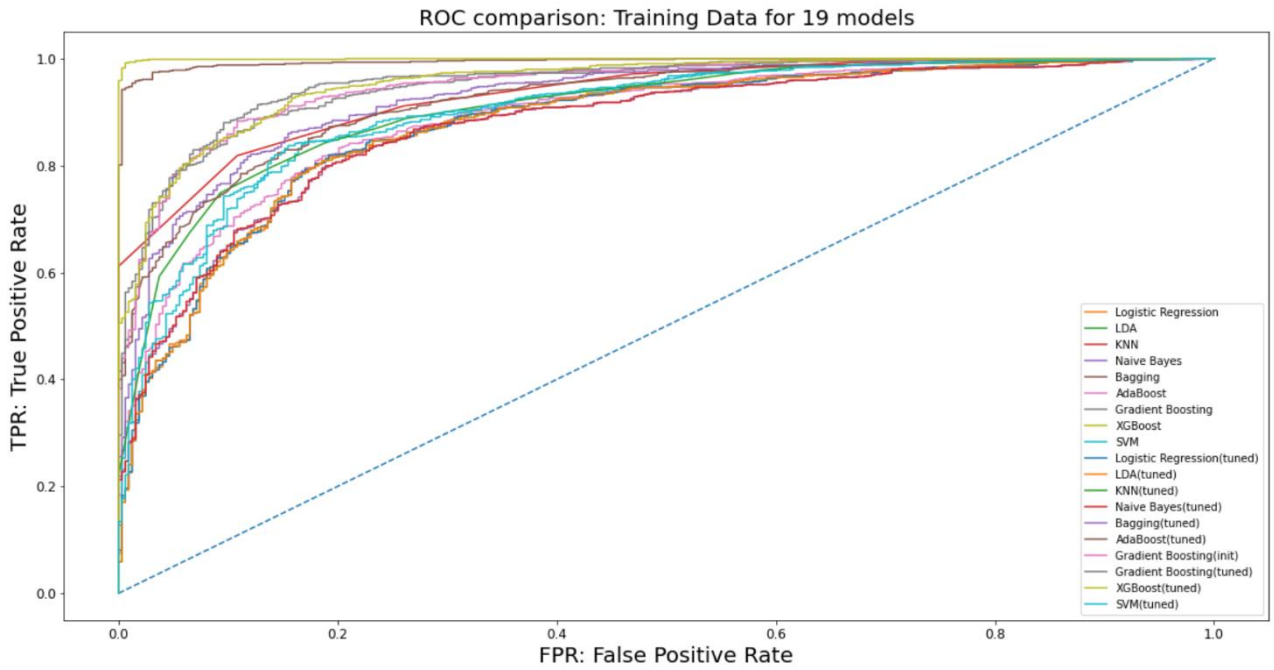
At the time of model building and model tuning we have tried to capture all the performance metrics on train and test dataset BUT to identify which model or models is/are performing best we have combined all the models in one table and for AUC curve as well. Please refer below table and graph for further inferences.

All models in one table

	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision	Train_F1 score	Test_F1 score
0	Logistic Regression	0.83	0.85	0.91	0.92	0.86	0.87	0.88	0.90
1	LDA	0.83	0.84	0.90	0.91	0.86	0.87	0.88	0.89
2	KNN	0.86	0.84	0.91	0.91	0.89	0.87	0.90	0.89
3	Naive Bayes	0.82	0.85	0.88	0.90	0.87	0.88	0.87	0.89
4	Bagging	0.96	0.83	0.99	0.92	0.96	0.85	0.97	0.89
5	AdaBoost	0.83	0.84	0.91	0.93	0.86	0.86	0.88	0.89
6	Gradient Boosting	0.89	0.84	0.94	0.91	0.90	0.86	0.92	0.89
7	XGBoost	0.99	0.83	1.00	0.90	0.99	0.86	0.99	0.88
8	Support Vector Machine(SVM)	0.83	0.85	0.92	0.93	0.86	0.87	0.89	0.90
9	Logistic Regression(tuned)	0.83	0.85	0.91	0.92	0.86	0.87	0.88	0.90
10	LDA(tuned)	0.83	0.84	0.90	0.91	0.86	0.87	0.88	0.89
11	KNN(tuned)	0.84	0.86	0.90	0.92	0.87	0.88	0.89	0.90
12	Naive Bayes(tuned)	0.82	0.84	0.88	0.90	0.87	0.88	0.87	0.89
13	Bagging(tuned)	0.87	0.85	0.92	0.92	0.89	0.87	0.91	0.89
14	AdaBoost(tuned)	0.86	0.84	0.92	0.92	0.88	0.87	0.90	0.89
15	Gradient Boosting(init)	0.89	0.84	0.94	0.91	0.90	0.86	0.92	0.89
16	Gradient Boosting(tuned)	0.91	0.84	0.95	0.90	0.92	0.87	0.94	0.88
17	XGBoost(tuned)	0.90	0.84	0.94	0.90	0.92	0.87	0.93	0.88
18	SVM(tuned)	0.84	0.85	0.92	0.93	0.86	0.87	0.89	0.90

AUC Curve – Training data

```
AUC Training Data for Logistic Regression: 0.877
AUC Training Data for LDA: 0.876
AUC Training Data for KNN: 0.933
AUC Training Data for Naive Bayes: 0.874
AUC Training Data for Bagging: 0.995
AUC Training Data for AdaBoost: 0.891
AUC Training Data for Gradient Boosting: 0.948
AUC Training Data for XGBoost: 0.999
AUC Training Data for SVM: 0.898
AUC Training Data for Logistic Regression(tuned): 0.877
AUC Training Data for LDA(tuned): 0.876
AUC Training Data for KNN(tuned): 0.907
AUC Training Data for Naive Bayes(tuned): 0.874
AUC Training Data for Bagging(tuned): 0.926
AUC Training Data for AdaBoost(tuned): 0.922
AUC Training Data for Gradient Boosting(init): 0.949
AUC Training Data for Gradient Boosting(tuned): 0.956
AUC Training Data for XGBoost(tuned): 0.955
AUC Training Data for SVM(tuned): 0.904
```



AUC Curve – Test data

```
AUC Testing Data for Logistic Regression: 0.914
AUC Testing Data for LDA: 0.915
AUC Testing Data for KNN: 0.877
AUC Testing Data for Naive Bayes: 0.910
AUC Testing Data for Bagging: 0.916
AUC Testing Data for AdaBoost: 0.905
AUC Testing Data for Gradient Boosting: 0.908
AUC Testing Data for XGBoost: 0.881
AUC Testing Data for SVM: 0.915
AUC Testing Data for Logistic Regression(tuned): 0.914
AUC Testing Data for LDA(tuned): 0.915
AUC Testing Data for KNN(tuned): 0.902
AUC Testing Data for Naive Bayes(tuned): 0.910
AUC Testing Data for Bagging(tuned): 0.912
AUC Testing Data for AdaBoost(tuned): 0.901
AUC Testing Data for Gradient Boosting(init): 0.909
AUC Testing Data for Gradient Boosting(tuned): 0.907
AUC Testing Data for XGBoost(tuned): 0.906
AUC Testing Data for SVM(tuned): 0.916
```

Best model identification:

1. SVM (tuned) model considered to be best model followed by,
2. Adaboost (tuned) model and,
3. KNN (tuned)

#	Model	Accuracy			Recall			Precision			F1			AUC			Best Model
		Train	Test	Diff.	Train	Test	Diff	Train	Test	Diff.	Train	Test	Diff	Train	Test	Diff.	
1	Logistic Regression	83%	85%	2%	91%	92%	1%	86%	87%	1%	88%	90%	2%	88%	91%	3%	
2	LDA	83%	84%	1%	90%	91%	1%	86%	87%	1%	88%	89%	1%	88%	92%	4%	
3	KNN	86%	84%	-2%	91%	91%	0%	89%	87%	-2%	90%	89%	-1%	93%	88%	-5%	
4	Naïve Bayes	82%	85%	3%	88%	90%	2%	87%	88%	1%	87%	89%	2%	87%	91%	4%	
5	Bagging Classifier	96%	83%	-13%	99%	92%	-7%	96%	85%	-11%	97%	89%	-8%	100%	92%	-8%	
6	AdaBoost	83%	84%	1%	91%	93%	2%	86%	86%	0%	88%	89%	1%	89%	91%	2%	
7	Gradient Boosting	89%	84%	-5%	94%	91%	-3%	90%	86%	-4%	92%	89%	-3%	95%	91%	-4%	
8	XGBoost	99%	83%	-16%	100%	90%	-10%	99%	86%	-13%	99%	88%	-11%	100%	88%	-12%	
9	SVM	83%	85%	2%	92%	93%	1%	86%	87%	1%	89%	90%	1%	90%	92%	2%	
10	Logistic Regression (tuned)	83%	85%	2%	91%	92%	1%	86%	87%	1%	88%	90%	2%	88%	91%	3%	
11	LDA (tuned)	83%	84%	1%	90%	91%	1%	86%	87%	1%	88%	89%	1%	88%	92%	4%	
12	**KNN (tuned)**	**84%**	**86%**	**2%**	**90%**	**92%**	**2%**	**87%**	**88%**	**1%**	**89%**	**90%**	**1%**	**91%**	**91%**	**0%**	**3**
13	Naïve Bayes (tuned)	82%	84%	2%	88%	90%	2%	87%	88%	1%	87%	89%	2%	87%	91%	4%	
14	Bagging Classifier (tuned)	87%	85%	-2%	92%	92%	0%	89%	87%	-2%	91%	89%	-2%	93%	90%	-3%	
15	**AdaBoost (tuned)**	**86%**	**84%**	**-2%**	**92%**	**92%**	**0%**	**88%**	**87%**	**-1%**	**90%**	**89%**	**-1%**	**92%**	**91%**	**-1%**	**2**
16	Gradient Boosting (init)	89%	84%	-5%	94%	91%	-3%	90%	86%	-4%	92%	89%	-3%	95%	91%	-4%	
17	Gradient Boosting (tuned)	91%	84%	-7%	95%	90%	-5%	92%	87%	-5%	94%	88%	-6%	96%	91%	-5%	
18	XGBoost (tuned)	90%	84%	-6%	94%	90%	-4%	92%	87%	-5%	93%	88%	-5%	96%	91%	-5%	
19	**SVM (tuned)**	**84%**	**85%**	**1%**	**92%**	**93%**	**1%**	**86%**	**87%**	**1%**	**89%**	**90%**	**1%**	**90%**	**92%**	**2%**	**1**

4. Inference & Insights

4.1 Inferences

i. In this dataset we have used total of 19 models including tuning (main 9 models).
ii. It was great exposure understanding all the models and its hyper tuning parameters in details.
iii. Further, we also found that border class imbalance dataset could also be treated with same distribution.
iv. We tried applying SMOTE as well in the dataset *(in a separate Jupyter notebook, which can be shared if required)* and evaluated the result based on F1-score and precision and we found:
    - Slight change in performance metrics parameters BUT found more overfit and underfit even though applying multiple iterations in hyper tuning in Grid Search.
    - Hence, it proved that slight class imbalance (i.e., 4:6 OR 3:7) could be processed with AS IS.
v. Age has the highest importance to the prediction of the class and inversely related to class 0. Higher the Age more chances of voting to conservative party.
vi. More the rating to the Blair, higher the chance of voting to labour party.
vii. More the rating to the Hague, higher the chance to vote to conservative party.
viii. In case of more Eurosceptic sentiment(high rating in Europe) more the voter inclined to conservative party compared to labour party
ix. Voter who have more political knowledge and high rating in Europe(Eurosceptic sentiment) more inclined to conservative party over Labour party.

4.2 Insights

i. A Cost Function quantifies the error between predicted values and expected values and presents it in the form of a single real number.
ii. The voting prediction model could be choose depending on their cost function which they want to minimize. The cost function can depend on the geographical and economic conditions as well.
iii. Main aim would be to balance the trade-off between losing an opportunity (to gain vote by getting right publicity and work) in case of FP and losing the votes in case of FN.
iv. Based on above model's poll comes out that "labour" party will get the higher seats than "conventional" and top3 models which would help to predict are:
    - SVM OR SVM (tuned)
    - AdaBoost (tuned)
    - KNN (tuned)