Project:MACHINE lEARNING

DSBA

Submitted by:

Name: Hima Jose Paliakkara

Batch: 2021-2022

Table of Contents

Contents

List of figures

List of Tables

Problem 1:

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

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

Answer:

	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

Table 1.1.1 Summary of dataset

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

Table 1.1.1 Dataset info

Inferences:

- The dataset contains 1525 rows and 10 columns, in the 10 columns, 2 columns are object type and 8 columns are integer type.
- First column is and index (Unnamed: 0), as it is only serial numbers, we can remove it.
- After dropping index column, we have 1525 rows and 9 columns
- Age feature is the continuous variable

- Variables vote & gender: Nominal variables
- 6 variables: economic.cond.national, economic.cond.household , Blair, Hague, Europe, political.knowledge are ordinal variables.
- Dataset has no missing values
- There are 8 duplicates present in the dataset. We can remove these records from the dataset as they might not be adding any additional value.
- After dropping the records, we have 1517 rows and 9 columns.
- There are 2 types of vote parties- Labour and Conservative. Top party seems to be Labour
- There are 2 types of gender- Male and Female with Female being the top most voters
- Minimum age of individual voting is 24 years and maximum age is 93.
- Minimum assessment of current national economic conditions is 1 and a maximum assessment is 5 with an average assessment of 3.
- Minimum assessment of current household economic conditions 1 and a maximum assessment is 5 with an average assessment of 3.
- Minimum assessment of the Labour leader Blair is 1 and maximum assessment is 5 with an average assessment of 4.
- Minimum assessment of the Conservative leader Hague is 1 and maximum r assessment is 5 with an average assessment of 2
- 75% of the voters on a 11-point scale that measures respondents' attitudes toward European integration represent high 'Eurosceptic' sentiment with a maximum scale of 11 and a minimum scale of 1.
- An average knowledge of parties positions on European integration is 2. Approximately 25% of parties do not hold positions on European integration with a maximum holding of 3.

```
VOTE :  2
Conservative     460
Labour          1057
Name: vote, dtype: int64

GENDER :  2
female    808
male      709
Name: gender, dtype: int64

ECONOMIC.COND.NATIONAL :  5
1     37
2    256
3    604
4    538
5     82
Name: economic.cond.national, dtype: int64

ECONOMIC.COND.HOUSEHOLD :  5
1     65
2    280
3    645
4    435
5     92
```

```
Name: economic.cond.household, dtype: int64


BLAIR :  5
1     97
2    434
3      1
4    833
5    152
Name: Blair, dtype: int64

HAGUE :  5
1    233
2    617
3     37
4    557
5     73
Name: Hague, dtype: int64

POLITICAL.KNOWLEDGE :  4
0    454
1     38
2    776
3    249
Name: political.knowledge, dtype: int64

EUROPE :  11
1     109
2      77
3     128
4     126
5     123
6     207
7      86
8     111
9     111
10    101
11    338
Name: Europe, dtype: int64
```

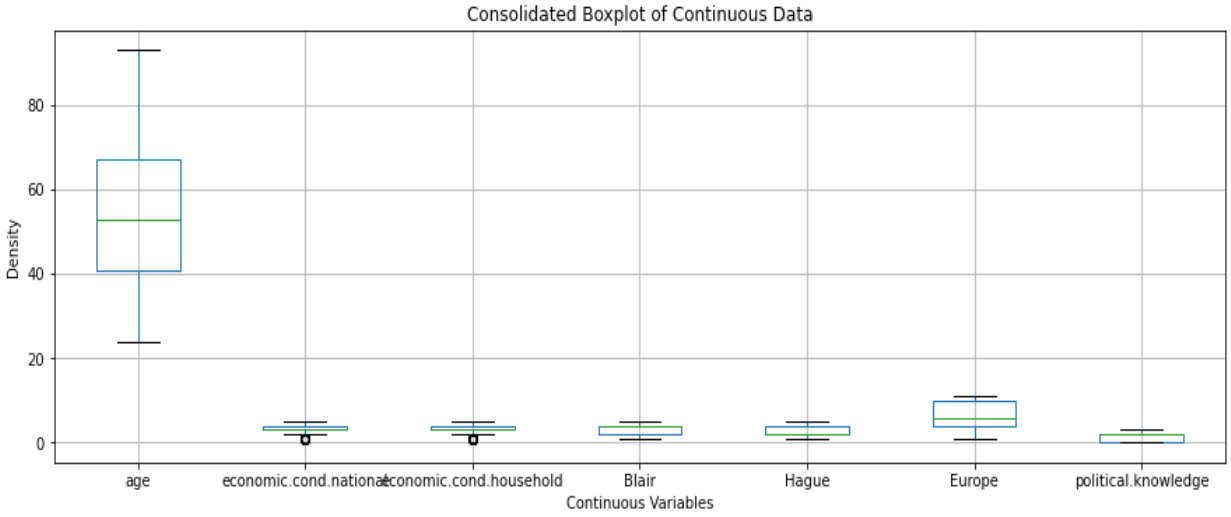


Figure 1.1.1 Boxplot for continuous variable

From the above graph we can find that there are outliers present in the variables economic.co nd.household and economic.cond.national.

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

Left skewed variables:
Europe,economic.cond.household,economic.cond.national,political.knowledge,Blair

Right skewed variables:
Hague, age

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

Answers:

EDA

Univariate Analysis

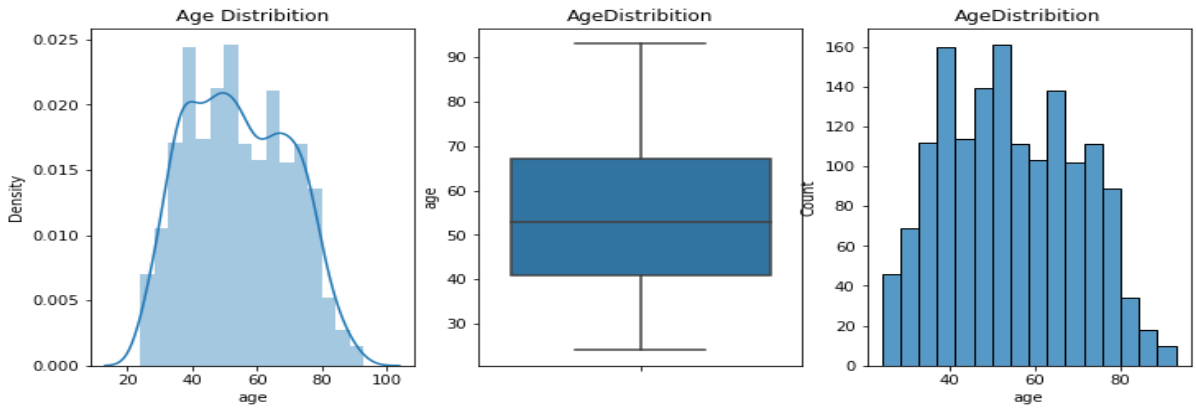


Figure 1.2.1 Age distribution

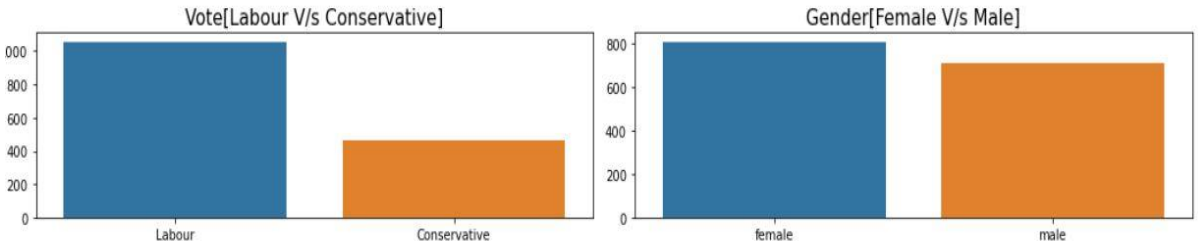


Figure 1.2.2 Nominal variables 'gender and vote' using barplot

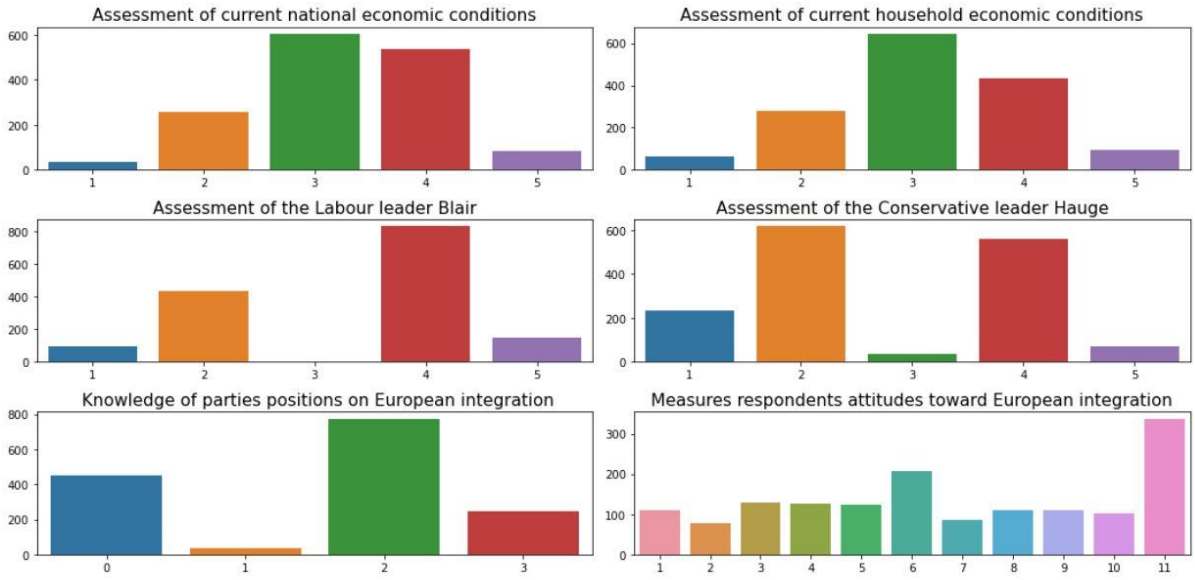


Figure 1.2.3 Ordinal variables using barplot

Inferences:

• Variable "age": Minimum voting age is 24 years and maximum voting age is 93 years. Mean voting age is 54 years.

• Variable "economic.cond.national" : Minimum assessment of current national economic conditions is 1 and a maximum assessment is 5 with an average assessment of 3.

• Variable "economic.cond.household" : Minimum assessment of current household economic conditions 1 and a maximum assessment is 5 with an average assessment of 3.

• Variable "Blair": Minimum assessment of the Labour leader Blair is 1 and maximum assessment is 5 with an average assessment of 4.

• Variable "Hague": Minimum assessment of the Conservative leader Hague is 1 and maximum r assessment is 5 with an average assessment of 2.

• For variable "Europe": 75% of the voters on a 11-point scale that measures respondents' attitudes toward European integration represent high 'Eurosceptic' sentiment with a maximum scale of 11 and a minimum scale of 1.

• On an average knowledge of parties' positions on European integration is 2. Approximately 25% of parties do not hold positions on European integration with a maximum holding of 3.

•There are outliers present in the variables: economic.cond.household and economic.cond.national.

Bivariate and Multivariate Analysis

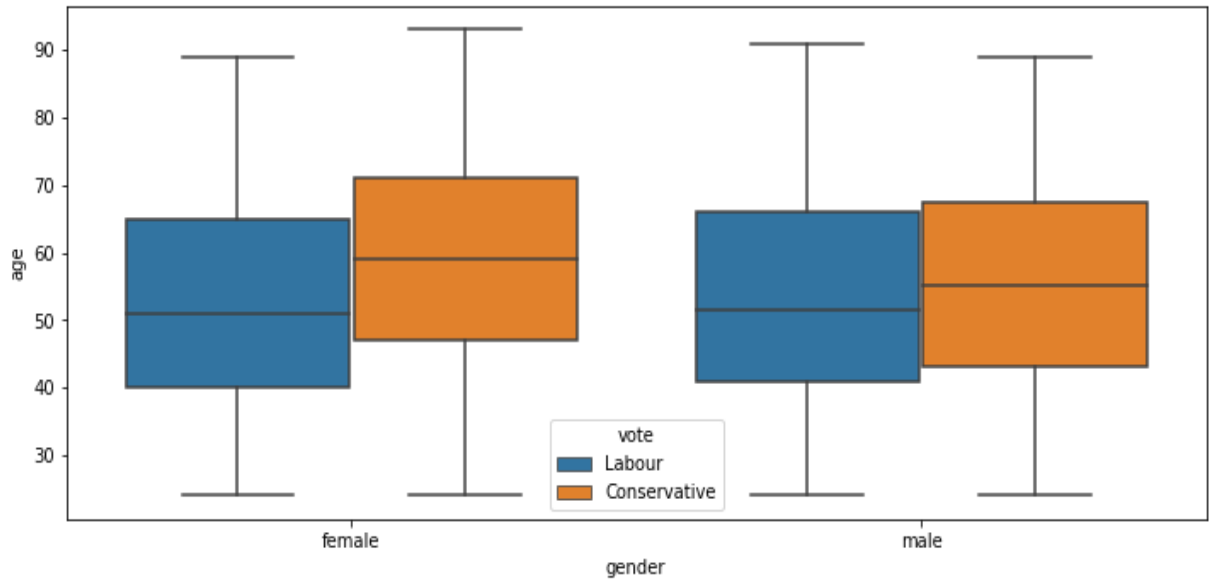


Figure 1.2.4 numeric v/s Nominal categorical variable

- Average age group close to 60 choose to elect conservative party whereas average age group near to 50 choose to elect Labour party.
- Middle 50% of the female people who fall under the age group within the range of 40 to 65 choose to elect Labour party.
- Middle 50% of the female people who fall under the age group within the range of 50 to 70 choose to elect conservative party.
- So, we can see from the boxplot that has the age increases people choose to elect conservative party

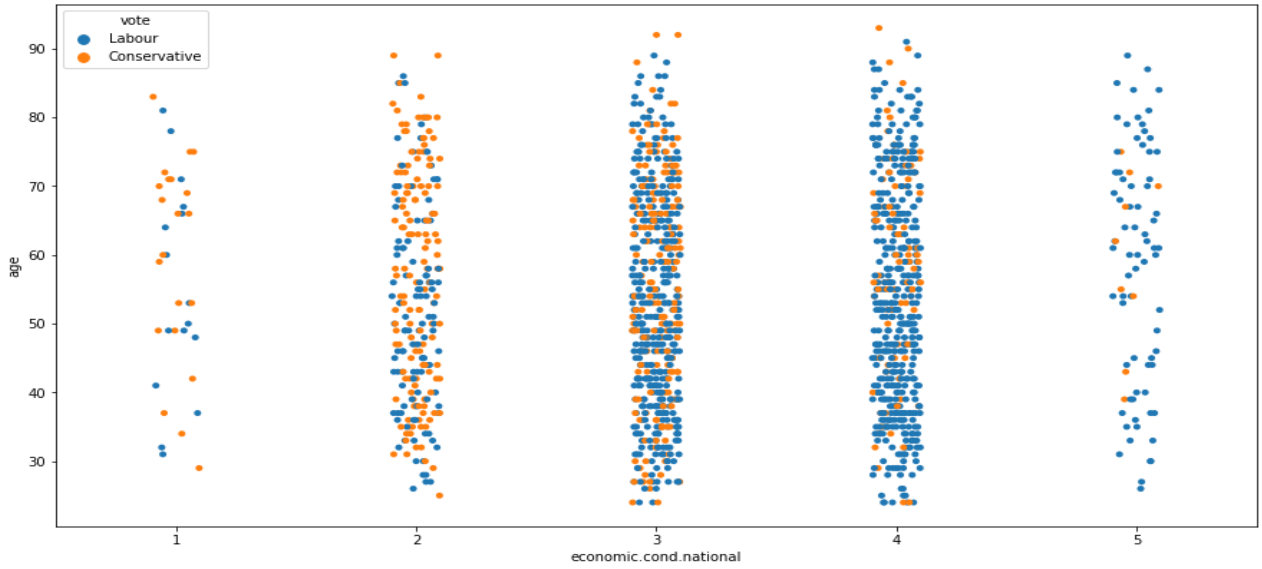


Figure 1.2.5 Target v/s Ordinal variable economic.cond.national

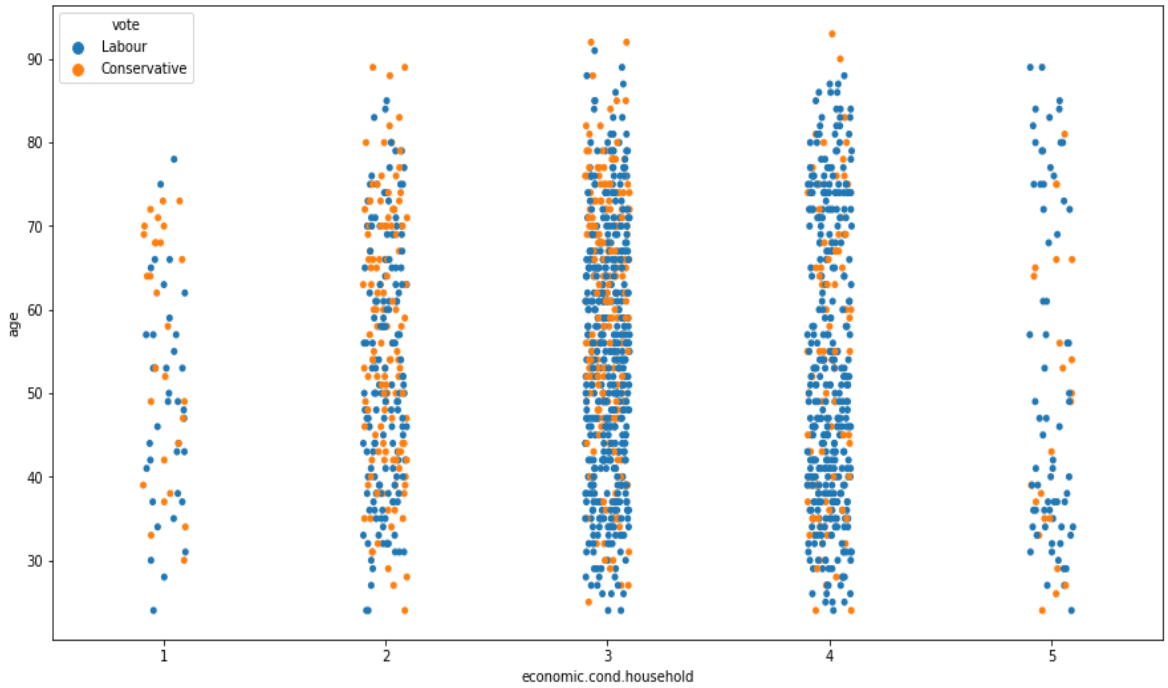


Figure 1.2.6 Target v/s Ordinal variable economic.cond.household

• From the above strip plot, we could see that rating 3 and rating 4 were given by majority of the voters for the assessment of economic national as well as household conditions and most of them choose to elect labour party.
• Also, we could say that voters who gave rating 1 or 2 choose select conservative party.

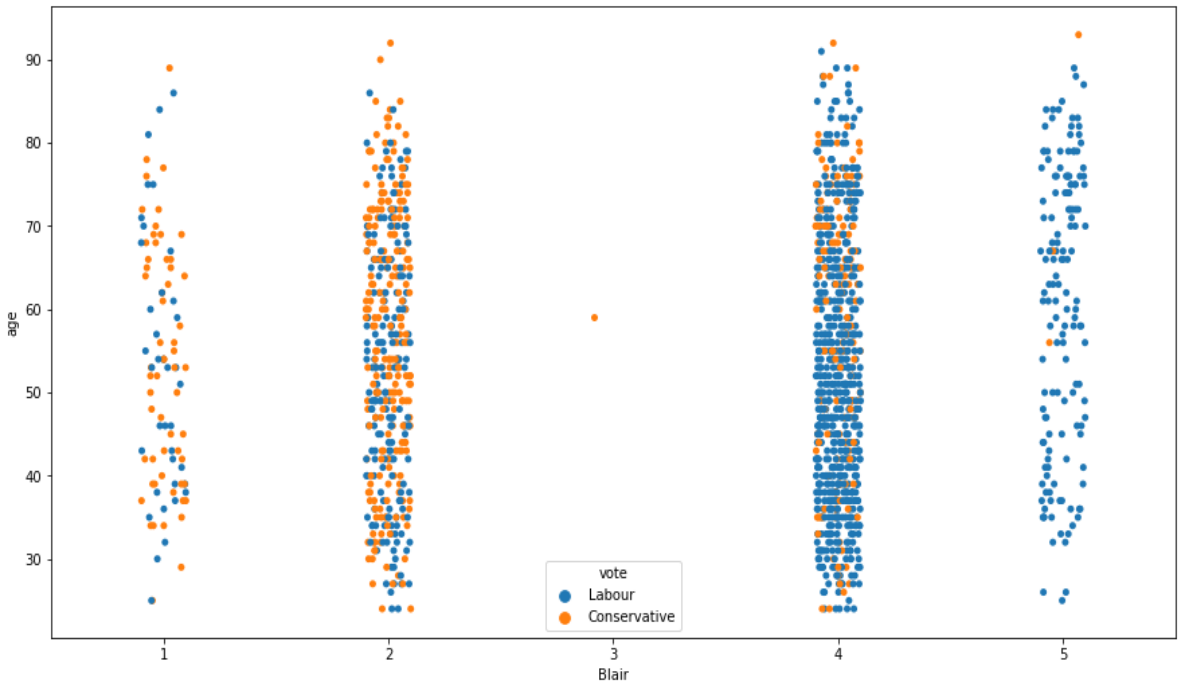


Figure 1.2.7 Target v/s Ordinal variable Blair

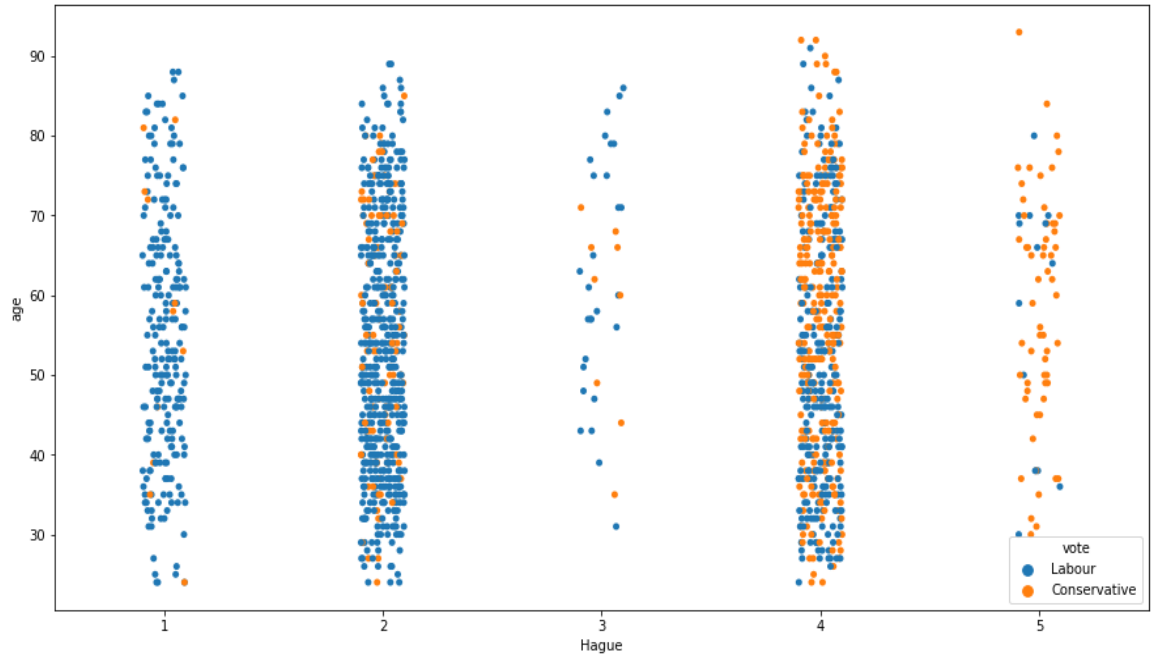


Figure 1.2.8 Target v/s Ordinal variable Hauge

•From the above strip plot, voters who gave rating 4 and 5 for the assessment of labour party leader Blair, choose to elect labour party.
• Also, it's very clear, voters who gave rating 3 and above for the assessment of conservative party leader Hague, choose to elect conservative party.

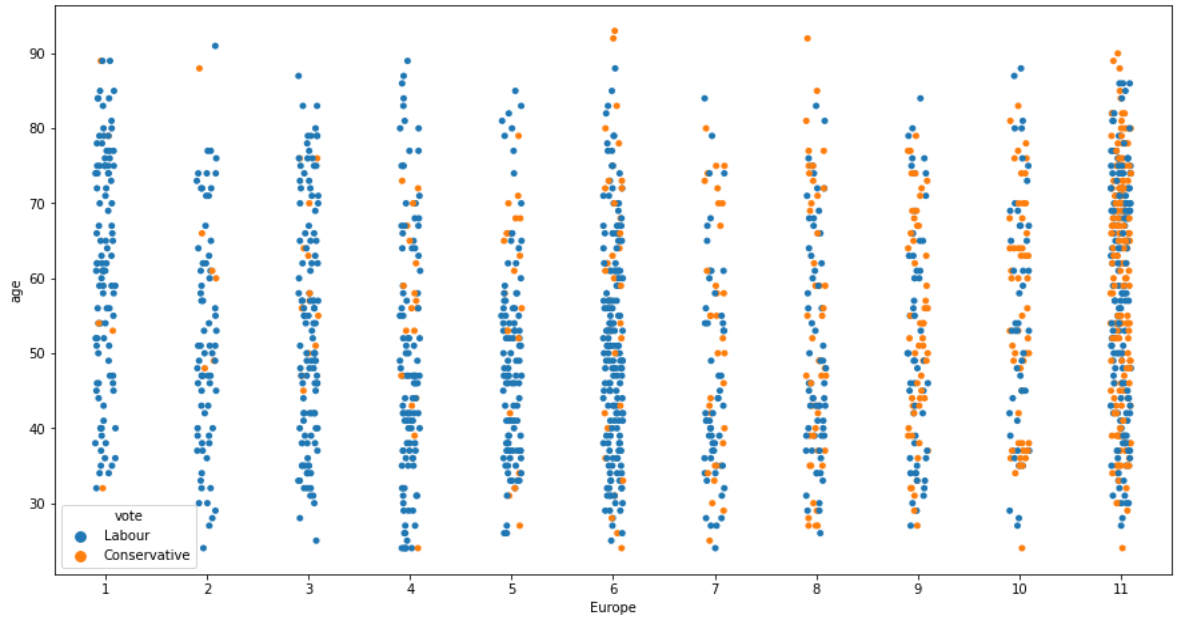


Figure 1.2.8 Target v/s Ordinal variable Europe

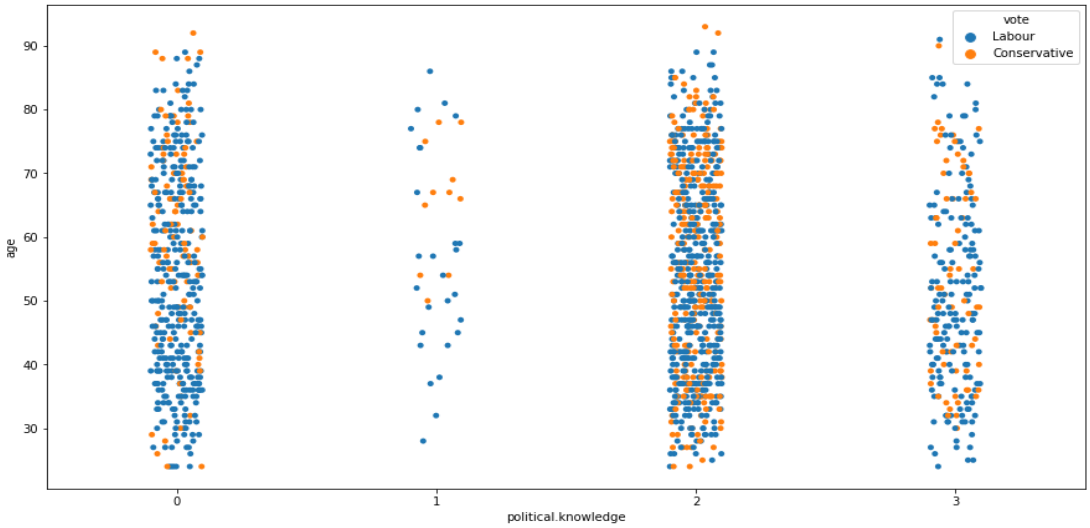


Figure 1.2.9 Target v/s Ordinal variable Political.knowledge

• Voters who choose to vote for labour party do not possess a very high Eurosceptic sentiment, whereas voters who represent high Eurosceptic sentiment vote for conservative party.

Pairplot

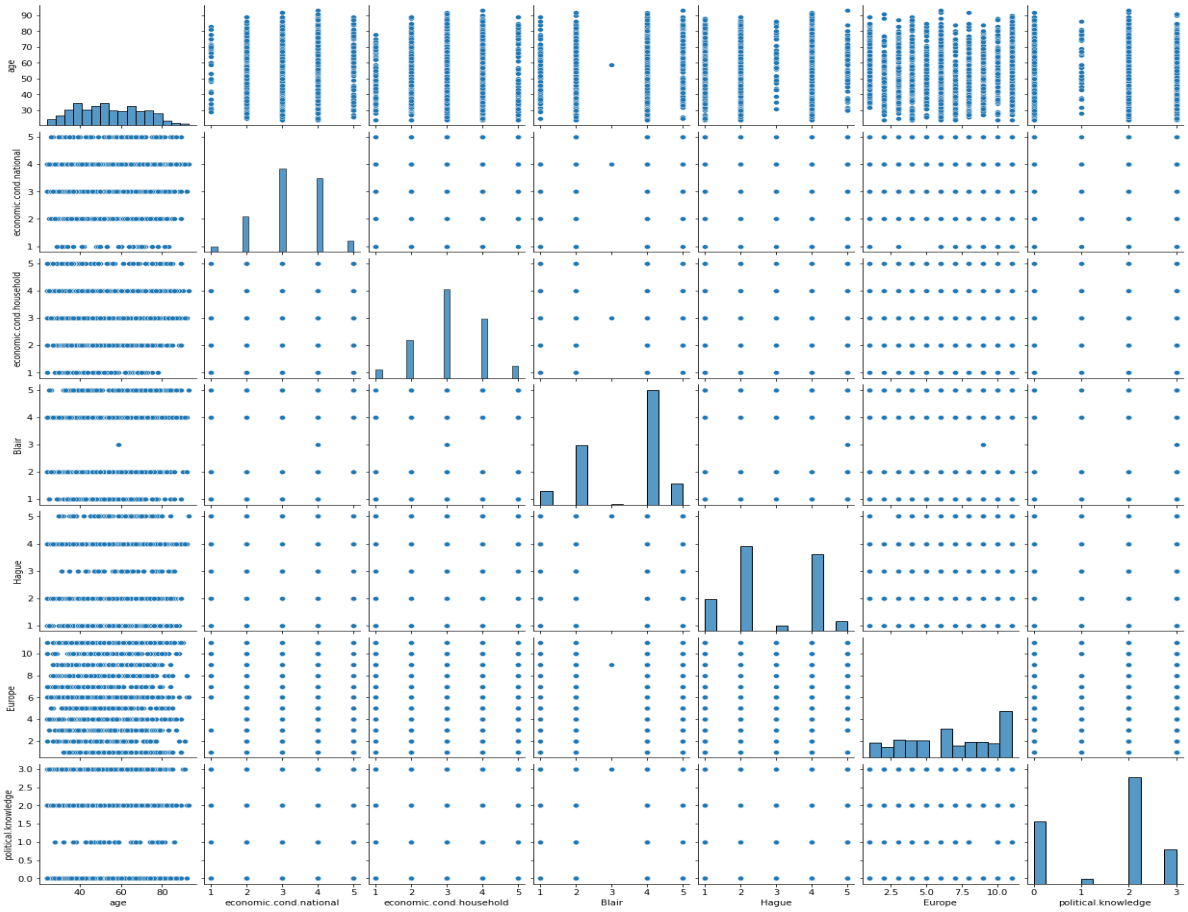


Figure 1.2.10 Pairplot

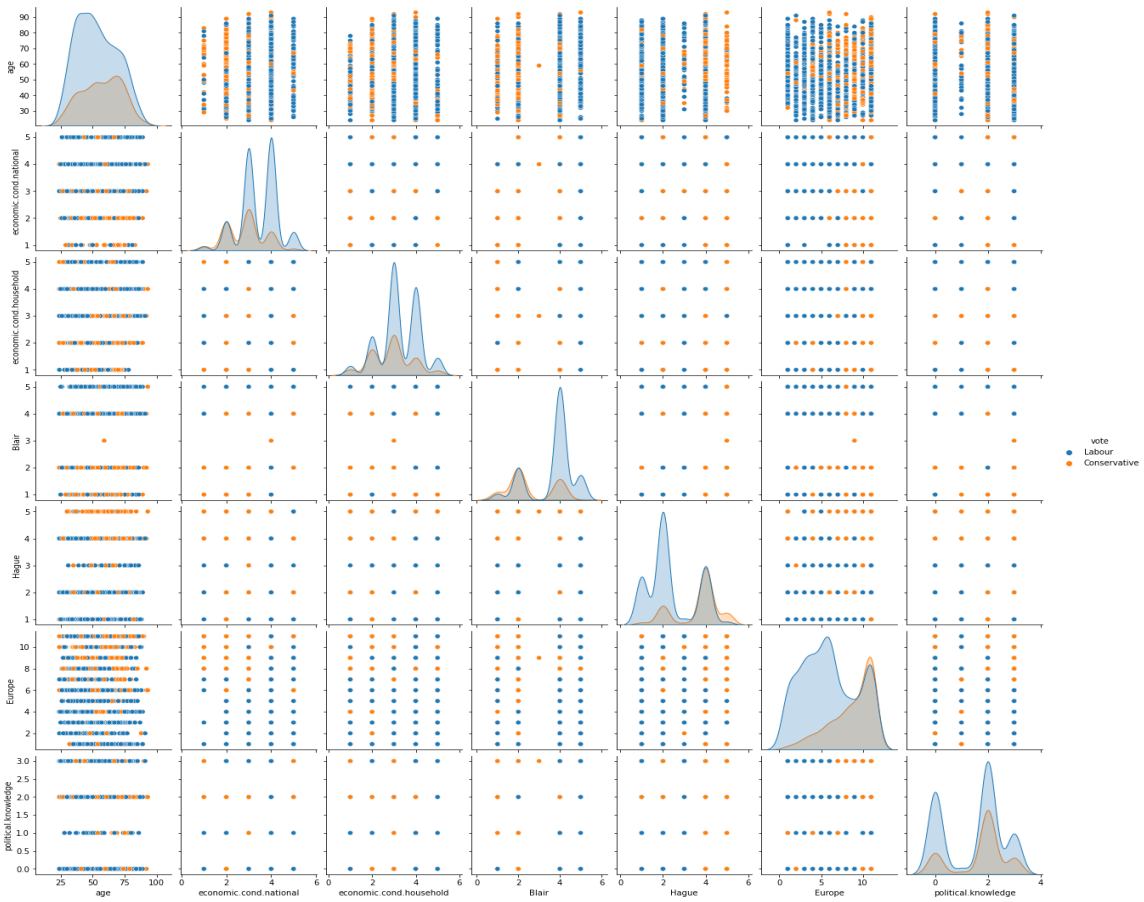


Figure 1.2.11 Pairplot with hue as target

Heatmap

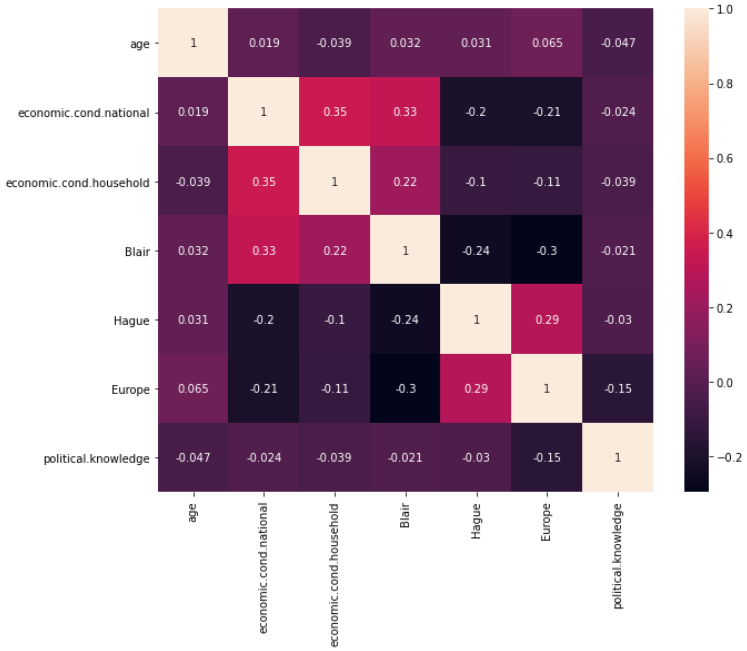


Figure 1.2.12 Heatmap

Inferences:

- There is not well correlation found for the continues variables.
- Negative correlation shows that the specific variable moves in opposite direction, i.e. whoever voting for Blair is not going to vote for Hauge (negative correlation)
- National economic conditions have very weak correlation with household economic condition.

Checking Outliers

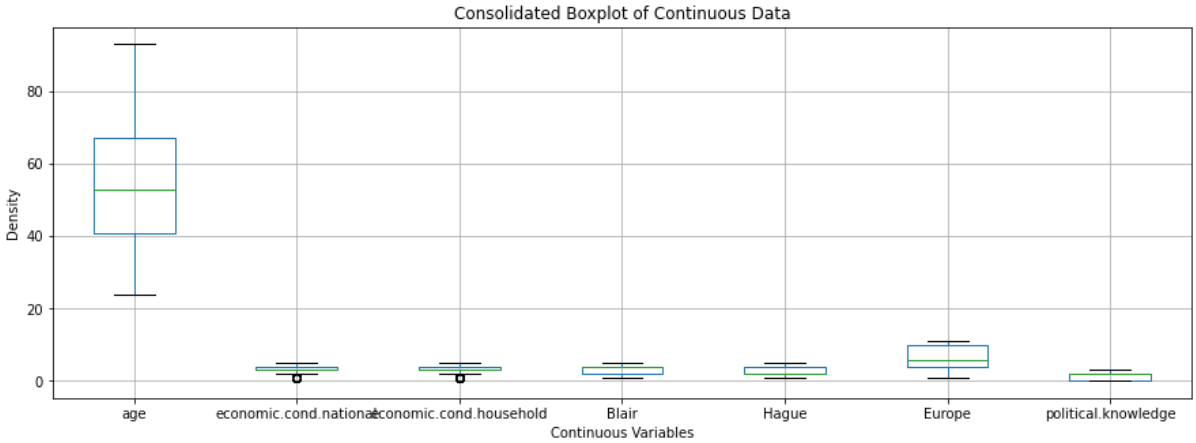


Figure 1.2.13 Checking outliers

Outliers are present in variables economic.cond.household and economic.cond.national.

economic.cond.national

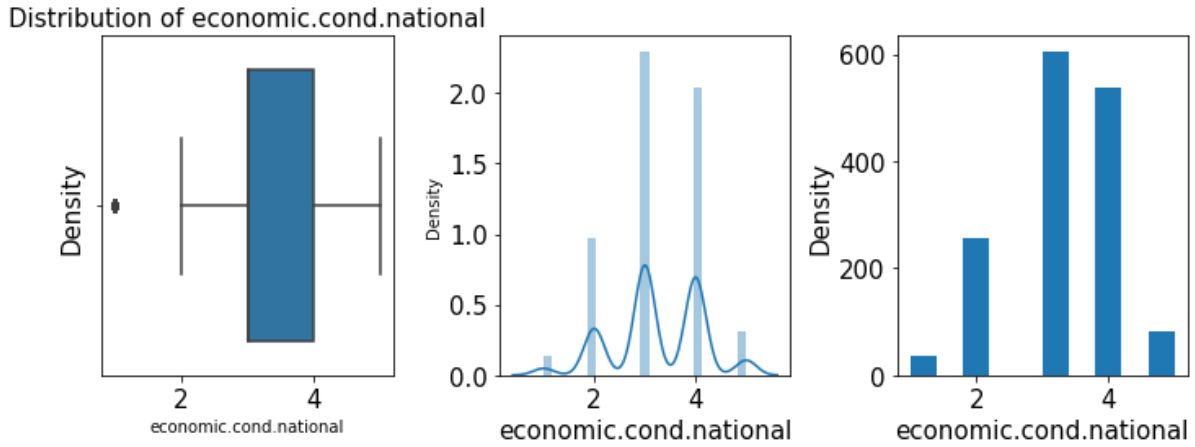


Figure 1.2.14 economic.cond.national distribution

Lower outliers in economic.cond.national	:	1.5
Upper outliers in economic.cond.national	:	5.5
Number of outliers in economic.cond.national upper	:	0
Number of outliers in economic.cond.national lower	:	37

economic.cond.household

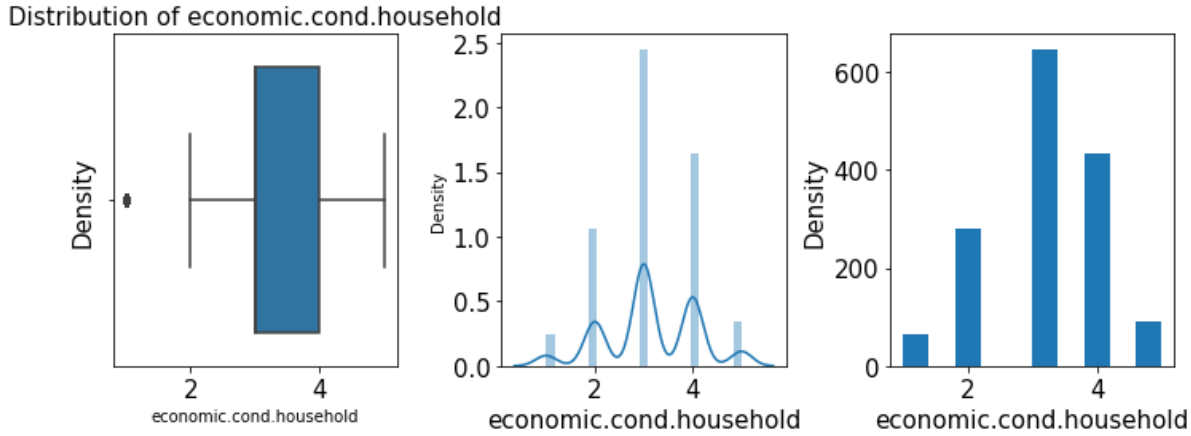


Figure 1.2.15 economic.cond.household distribution

Lower outliers in economic.cond.household: 1.5
Upper outliers in economic.cond.household: 5.5
Number of outliers in economic.cond.household upper : 0
Number of outliers in economic.cond.household lower : 65

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).

Answer:

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

Distribution of Target variable is 70:30%
Most of the voters elected to labour party. The ratio is almost 1:2

Encode the data:

- Converted the 'vote' Variable (target variable) into numeric by using the LabelEncoder functionality inside sklearn.

```
## Converting the 'vote' Variable into numeric by using the LabelEncoder
from sklearn.preprocessing import LabelEncoder

## Defining a Label Encoder object instance
LE=LabelEncoder()
```

```
## Applying the created Label Encoder object for the target class

df_ED['vote']=LE.fit_transform(df_ED['vote'])
```

```
df_ED.vote.value_counts()
```

```
1    1057
0     460
Name: vote, dtype: int64
```

- .get_dummies for the independent variable ‘gender’

After converting getting the info of dataset

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1517 entries, 0 to 1524
Data columns (total 9 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   vote                    1517 non-null   int32
 1   age                     1517 non-null   int64
 2   economic.cond.national  1517 non-null   int64
 3   economic.cond.household 1517 non-null   int64
 4   Blair                   1517 non-null   int64
 5   Hague                   1517 non-null   int64
 6   Europe                  1517 non-null   int64
 7   political.knowledge     1517 non-null   int64
 8   gender_male             1517 non-null   uint8
dtypes: int32(1), int64(7), uint8(1)
memory usage: 102.2 KB
```

Table 1.3 Info of dataset

Scaling

Scaling is performed when we are dealing with Gradient Descent Based algorithms (Linear and Logistic Regression, Neural Network) and Distance-based algorithms (KNN, K-means, SVM) as these are very sensitive to the range of the data points. Here, we are building a model, to predict which party a voter will vote on the basis of the given information and to create an exit poll that will help in predicting overall win and seats covered by a particular party. In this case we are expected to build model using Logistic Regression, LDA, KNN Model and Naïve Bayes Model. For now, we are not scaling the data and will do the scaling based on the models we will run ahead. Hence, as mentioned scaling might be necessary for Distance-based model such as KNN.

1.4 Apply Logistic Regression and LDA (linear discriminant analysis)

Answer:

Logistic Regression

Feature scaling doesn’t make a much effect for logistic regression test data accuracy; hence scaling is not performed for logistic regression.

Logistic regression is a linear model for classification rather than regression. It is also known as logit regression. In this model, the probabilities describing the possible outcomes of a single trial are modelled using a logistic function.

Running the model with the default parameters set to know the performance of model is good or not. Default parameters:

(*penalty='l2', *, dual=False, tol=0.0001, class_weight=None, random_state=None, n_jobs=None, solver='lbfgs', max_iter=100*)

- **Penalty:** This parameter is used to specify the norm (L1 or L2) used in penalization (regularization).
- **Dual:** It is used for dual or primal formulation whereas dual formulation is only implemented for L2 penalty.
- **Tol:** It represents the tolerance for stopping criteria.
- **Random_state** : This parameter represents the seed of the pseudo random number generated which is used while shuffling the data.
- **Solver**: This parameter represents which algorithm to use in the optimization problem.
- **max_iter :** (default = 100) it represents the maximum number of iterations taken for solvers to converge.
- **N_jobs**: controls the number of cores on which the package will attempt to run in parallel.

After fitting the model and running the matrix below are the output result for Train & Test data

AUC score for Training data: 0.8898167428438122

	precision	recall	f1-score	support
0	0.748120	0.648208	0.694590	307.000000
1	0.864151	0.911141	0.887024	754.000000
accuracy	0.835061	0.835061	0.835061	0.835061
macro avg	0.806136	0.779675	0.790807	1061.000000
weighted avg	0.830578	0.835061	0.831343	1061.000000

AUC score for Testing data: 0.8790310403589379

	precision	recall	f1-score	support
0	0.753425	0.718954	0.735786	153.000000
1	0.861290	0.881188	0.871126	303.000000
accuracy	0.826754	0.826754	0.826754	0.826754
macro avg	0.807357	0.800071	0.803456	456.000000
weighted avg	0.825099	0.826754	0.825716	456.000000

Table 1.4.1 Logistic Regression Train & Test data

Here we can see the accuracy for Train is 84% and Test is 83%.The model is a valid since both the train and test set accuracy are almost similar and the difference between both the set lies within the range of 10%(not overfit or underfit).

We will further fine tune the model later using Grid Search to understand if there is any improvement in the performance metrics of the model question 1.6 and 1.7

LDA (linear discriminant analysis)

Feature scaling doesn't make a much effect for LDA test data accuracy, hence scaling is not performed for LDA model.

Linear Discriminant Analysis or LDA is a dimensionality reduction technique. It is used as a pre-processing step in Machine Learning and applications of pattern classification. LDA is a

supervised classification technique that is considered a part of crafting competitive machine learning models.

Running the model with the default parameters set to know the performance of model is good or not. Default parameters:

(*solver='svd', shrinkage=None, priors=None, n_components=None, tol=0.0001, covariance_estimator=None*)

- **Solver**: This parameter represents which algorithm to use in the optimization problem.
- **Shrinkage:** This should be left to None if covariance_estimator is used.
- **Priors:** The class prior probabilities. By default, the class proportions are inferred from the training data.
- **n_components:** Number of components (<= min(n_classes - 1, n_features)) for dimensionality reduction. If None, will be set to min(n_classes - 1, n_features). This parameter only affects the transform method.
- **Tol:** It represents the tolerance for stopping criteria.
- **covariance_estimator:** if None the shrinkage parameter drives the estimate. Note that covariance_estimator works only with 'lsqr' and 'eigen' solvers.

After fitting the model and running the matrix below are the output result for Train & Test data

```
AUC score for Training data: 0.8893674560865394
```

	precision	recall	f1-score	support
0	0.743494	0.651466	0.694444	307.000000
1	0.864899	0.908488	0.886158	754.000000
accuracy	0.834119	0.834119	0.834119	0.834119
macro avg	0.804197	0.779977	0.790301	1061.000000
weighted avg	0.829771	0.834119	0.830686	1061.000000

```
AUC score for Testing data: 0.8876377833861817
```

	precision	recall	f1-score	support
0	0.765517	0.725490	0.744966	153.000000
1	0.864952	0.887789	0.876221	303.000000
accuracy	0.833333	0.833333	0.833333	0.833333
macro avg	0.815235	0.806639	0.810594	456.000000
weighted avg	0.831589	0.833333	0.832182	456.000000

Table 1.4.2 LDA Train & Test data

Here we can see the accuracy for Train is 83% and Test is 83%.The model is a valid since both the train and test set accuracy are almost similar and the difference between both the set lies within the range of 10%(not overfit or underfit).We will further fine tune the model later using Grid Search to understand if there is any improvement in the performance metrics of the model question 1.6 and 1.7.

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

Answer:

Naïve Bayes

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

Performing a feature scaling in these algorithms may not have much effect.

Running the model with the default parameters set to know the performance of model is good or not. Default parameters:

(*priors=None, var_smoothing=1e-09)*

- **priors:** Prior probabilities of the classes. If specified the priors are not adjusted according to the data.
- **var_smoothing:** Portion of the largest variance of all features that is added to variances for calculation stability.

After fitting the model and running the matrix below are the output result for Train & Test data

	precision	recall	f1-score	support
0	0.727586	0.687296	0.706868	307.000000
1	0.875486	0.895225	0.885246	754.000000
accuracy	0.835061	0.835061	0.835061	0.835061
macro avg	0.801536	0.791261	0.796057	1061.000000
weighted avg	0.832692	0.835061	0.833632	1061.000000

	precision	recall	f1-score	support
0	0.736842	0.732026	0.734426	153.000000
1	0.865132	0.867987	0.866557	303.000000
accuracy	0.822368	0.822368	0.822368	0.822368
macro avg	0.800987	0.800006	0.800492	456.000000
weighted avg	0.822087	0.822368	0.822224	456.000000

Table 1.5.1 Naïve Bayes Train & Test data

Here we can see the accuracy for Train is 84% and Test is 82%. The model is a valid since both the train and test set accuracy are almost similar and the difference between both the set lies within the range of 10% (not overfit or underfit).

We will further fine tune the model later using Grid Search to understand if there is any improvement in the performance metrics of the model question 1.6 and 1.7.

KNN Model

K Nearest Neighbor is a versatile algorithm used for imputing missing values and resampling datasets. As the name (K Nearest Neighbor) suggests it considers K Nearest Neighbors (Data points) to predict the class or continuous value for the new Datapoint.

In order to have good KNN model, data requires pre-processing to make all independent variables similarly scaled and centred. Hence need to perform scaling on all numeric attributes in models that calculate distance and see the performance for KNN.

Default value of **n_neighbors is equal to 5**. First, we will build KNN Model with k=5

*(n_neighbors=5, *, weights='uniform', algorithm='auto', p=2, metric='minkowski')*

- **n_neighbors:** Number of neighbors to use by default for kneighbors queries

- **weights:** uniform': uniform weights. All points in each neighborhood are weighted equally.
- **Algorithm:** 'auto' will attempt to decide the most appropriate algorithm based on the values passed to fit method
- **P:** Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.
- **Metric:** The distance metric to use for the tree

After fitting the model and running the matrix below are the output result for Train & Test data

AUC score for Training data: 0.9278765152627896

	precision	recall	f1-score	support
0	0.769504	0.706840	0.736842	307.000000
1	0.884467	0.913793	0.898891	754.000000
accuracy	0.853911	0.853911	0.853911	0.853911
macro avg	0.826985	0.810317	0.817867	1061.000000
weighted avg	0.851203	0.853911	0.852002	1061.000000

AUC score for Testing data: 0.8672857481826615

	precision	recall	f1-score	support
0	0.755245	0.705882	0.729730	153.000000
1	0.856230	0.884488	0.870130	303.000000
accuracy	0.824561	0.824561	0.824561	0.824561
macro avg	0.805737	0.795185	0.799930	456.000000
weighted avg	0.822347	0.824561	0.823022	456.000000

Table 1.5.2 KNN Train & Test data

Here we can see the accuracy for Train is 85% and Test is 82%. The model is a valid since both the train and test set accuracy are almost similar and the difference between both the set lies within the range of 10% (not overfit or underfit).

We will further fine tune the model later using Grid Search to understand if there is any improvement in the performance metrics of the model question 1.6 and 1.7

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

Answer:

Model Tuning is the process of maximizing a model's performance without overfitting or creating too high of a variance. It allows you to customize your models so they generate the most accurate outcomes and give you highly valuable insights into your data, enabling you to make the most effective business decisions. This is accomplished by selecting appropriate "hyperparameters", these parameters are set manually. Below mentioned are the three most commonly used approaches:

1. Grid Search: Grid search also known as parameter sweeping. This method involves manually defining a subset of the hyperparametric space and exhausting all combinations of the specified hyperparameter subsets. Each combination's performance is then evaluated, typically using cross-validation, and the best performing hyperparametric combination is chosen.

2. Random Search: Random search methods resemble grid search methods but tend to be less expensive and time consuming because they do not examine every possible combination of parameters. Instead of testing on a predetermined subset of hyperparameters, random search, as its name implies, randomly selects a chosen number of hyperparametric pairs from a given domain and tests only those. This greatly simplifies the analysis without significantly sacrificing optimization. For example, if the region of hyperparameters that are near optimal occupies at least 5% of the grid, then random search with 60 trials will find that region with high probability (95%).

3. Bayesian Optimization: This process builds a probabilistic model for a given function and analyzes this model to make decisions about where to next evaluate the function. It offers an efficient framework for optimising the highly expensive black-box functions without knowing its form.It is an efficient tool for hyperparameter tuning for complex models like deep neural networks.

Here I am using GridSearchCV Method for the tuning.

Logistic Regression with Tuning - GridSearchCV

First, we will define the library required for grid search followed by defining all the parameters or the combination that we want to test out on the model. Building a Logistic regression model using a grid search cross validation to get best parameter/ estimators for a given dataset. The given tuning parameters are below:

```
GridSearchCV(cv=3, estimator=LogisticRegression(max_iter=10000, n_jobs=2),
             n_jobs=-1,
             param_grid={'penalty': ['l2', 'none'],
                         'solver': ['liblinear', 'lbfgs', 'sag', 'newton-cg'],
                         'tol': [0.0001, 1e-06]},
             scoring='f1')
```

- **estimator:** In this we have to pass the models or functions on which we want to use GridSearchCV
- **param_grid:** Dictionary or list of parameters of models or function in which GridSearchCV have to select the best.
- **Cs:** Each of the values in Cs describes the inverse of regularization strength. If Cs is as an int, then a grid of Cs values are chosen in a logarithmic scale between 1e-4 and 1e4. Like in support vector machines, smaller values specify stronger regularization.
- **Scoring**: It is used as a evaluating metric for the model performance to decide the best hyperparameters, if not especified then it uses estimator score.
- **solver:** This parameter represents which algorithm to use in the optimization problem
- **max_iter:**Defines the maximum number of iterations by the solver during model fitting.
- **penalty:** It imposes a penalty to the logistic model for having too many variables. This results in shrinking the coefficients of the less contributive variables toward zero.

- **n_jobs:** controls the number of cores on which the package will attempt to run in parallel.
- **cv:** cross validation generator or an iterable
- **scoring:** choosing scoring F1 since it computes the Harmonic Mean between Recall and Precision, it tells us whether both Type I and Type II error is low or high on an average

The best parameters after fitting the model is:

```
LogisticRegression(C=0.01, max_iter=10000, n_jobs=2, penalty='none',
                   solver='sag', tol=1e-06)
```

After fitting the model and running the matrix below are the output result for Train & Test data

AUC score for Training data: 0.8900068257026587

	precision	recall	f1-score	support
0	0.742424	0.638436	0.686515	307.000000
1	0.860728	0.909814	0.884591	754.000000
accuracy	0.831291	0.831291	0.831291	0.831291
macro avg	0.801576	0.774125	0.785553	1061.000000
weighted avg	0.826497	0.831291	0.827277	1061.000000

AUC score for Testing data: 0.8826549321598827

	precision	recall	f1-score	support
0	0.760274	0.725490	0.742475	153.00000
1	0.864516	0.884488	0.874388	303.00000
accuracy	0.831140	0.831140	0.831140	0.83114
macro avg	0.812395	0.804989	0.808432	456.00000
weighted avg	0.829540	0.831140	0.830128	456.00000

Table 1.6.1 Logistic regression with GridSearch Train & Test data

Here we can see the accuracy for Train is 83% and Test is 83%. After fine tuning the model we can see that model has given mostly the same performance with a very slight improvement in few parameters. Hence, we can say that fine tuning this particular model does not make much of a difference the model performance. Model is valid since both the train and test set accuracy are almost similar and the difference between both the set lies within the range of 10% (not overfit or underfit).

LDA with GridSerachCV

First, we will define the library required for grid search followed by defining all the parameters or the combination that we want to test out on the model. Building a LDA model using a grid search cross validation to get best parameter/ estimators for a given dataset. The given tuning parameters are below:

```
GridSearchCV(cv=2, estimator=LinearDiscriminantAnalysis(), n_jobs=4,
             param_grid={'solver': ['svd', 'lsqr', 'eigen'],
                         'tol': [0.0001, 0.0002, 0.0003]},
             scoring='accuracy', verbose=1)
```

- **estimator:** In this we have to pass the models or functions on which we want to use GridSearchCV
- **param_grid:** Dictionary or list of parameters of models or function in which GridSearchCV have to select the best.
- **n_jobs:** controls the number of cores on which the package will attempt to run in parallel.
- **cv:** cross validation generator or an iterable
- **Scoring**: It is used as a evaluating metric for the model performance to decide the best hyperparameters, if not especified then it uses estimator score
- **Verbose:** For the 'liblinear', 'sag' and 'lbfgs' solvers set verbose to any positive number for verbosity.

The best parameters after fitting the model are:

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

After fitting the model and running the matrix below are the output result for Train & Test data

```
AUC score for Training data: 0.8893674560865394
```

	precision	recall	f1-score	support
0	0.743494	0.651466	0.694444	307.000000
1	0.864899	0.908488	0.886158	754.000000
accuracy	0.834119	0.834119	0.834119	0.834119
macro avg	0.804197	0.779977	0.790301	1061.000000
weighted avg	0.829771	0.834119	0.830686	1061.000000

```
AUC score for Testing data: 0.8876377833861817
```

	precision	recall	f1-score	support
0	0.765517	0.725490	0.744966	153.000000
1	0.864952	0.887789	0.876221	303.000000
accuracy	0.833333	0.833333	0.833333	0.833333
macro avg	0.815235	0.806639	0.810594	456.000000
weighted avg	0.831589	0.833333	0.832182	456.000000

Table 1.6.2 LDA with GridSearch Train & Test data

Here we can see the accuracy for Train is 83% and Test is 83%. After fine tuning the model, we can see that model has given mostly the same performance. Hence, we can say that fine tuning this particular model does not make much of a difference the model performance. Model is valid since both the train and test set accuracy are almost similar and the difference between both the set lies within the range of 10% (not overfit or underfit).

Naïve Bayes with GridSerachCV

First, we will define the library required for grid search followed by defining all the parameters or the combination that we want to test out on the model. Building a Naïve Bayes model using a grid search cross validation to get best parameter/ estimators for a given dataset. The given tuning parameters are below:

```
param_grid_NB = {'var_smoothing': np.logspace(0,-9, num=100)}

GridSearchCV(estimator=GaussianNB(),param_grid=param_grid_NB,
verbose=1,scoring='accuracy',cv=3)
```

- **var_smoothing:** is a stability calculation to widen (or smooth) the curve and therefore account for more samples that are further away from the distribution mean. In this case, np.logspace returns numbers spaced evenly on a log scale, starts from 0, ends at -9, and generates 100 samples.
- **estimator** is the machine learning model of interest, provided the model has a scoring function; in this case, the model assigned is GaussianNB().
- **param_grid** is a dictionary with parameters names (string) as keys and lists of parameter settings to try as values; this enables searching over any sequence of parameter settings.
- **verbose** is the verbosity: the higher, the more messages; in this case, it is set to 1.
- **cv** is the cross-validation generator or an iterable, in this case, there is a 10-fold cross-validation.

The best parameters after fitting the model are:

```
GaussianNB(var_smoothing=0.0006579332246575676)
```

After fitting the model and running the matrix below are the output result for Train & Test data

AUC score for Training data: 0.887302465029074

	precision	recall	f1-score	support
0	0.742049	0.684039	0.711864	307.000000
1	0.875321	0.903183	0.889034	754.000000
accuracy	0.839774	0.839774	0.839774	0.839774
macro avg	0.808685	0.793611	0.800449	1061.000000
weighted avg	0.836759	0.839774	0.837770	1061.000000

AUC score for Testing data: 0.8786859078064668

	precision	recall	f1-score	support
0	0.748299	0.718954	0.733333	153.000000
1	0.860841	0.877888	0.869281	303.000000
accuracy	0.824561	0.824561	0.824561	0.824561
macro avg	0.804570	0.798421	0.801307	456.000000
weighted avg	0.823081	0.824561	0.823667	456.000000

Table 1.6.3 Naïve Bayes with GridSearch Train & Test data

Here we can see the accuracy for Train is 84% and Test is 82%. After fine tuning the model, we can see that model has given mostly the same performance with a very slight improvement in few parameters. Hence, we can say that fine tuning this particular model does not make much of a difference the model performance. Model is valid since both the train and test set accuracy are almost similar and the difference between both the set lies within the range of 10% (not overfit or underfit).

KNN model with GridSerachCV'

First, we will define the library required for grid search followed by defining all the parameters or the combination that we want to test out on the model. Building a KNN model using a grid search cross validation to get best parameter/ estimators for a given dataset. The given tuning parameters are below:

```
knn = KNeighborsClassifier()
k_range    = range(1,20)
weights    = ['uniform' , 'distance']
metrics = ['minkowski','euclidean','manhattan']
```

```
param_grid = dict(n_neighbors = list(k_range), weights = weights, metric=metrics)
```

```
grid_knn = GridSearchCV(knn, param_grid, cv=3,verbose = 1, n_jobs = -1)
```

- **n_neighbors:** Number of neighbors to use by default for kneighbors queries
- **weights:** uniform': uniform weights. All points in each neighborhood are weighted equally.
- **Algorithm:** 'auto' will attempt to decide the most appropriate algorithm based on the values passed to fit method
- **Metric:** The distance metric to use for the tree
- **K_range**: giving a range of values from 1-20 to select the best k value.

The best parameters after fitting the model are:

```
{'metric': 'manhattan', 'n_neighbors': 19, 'weights': 'distance'}
```

After fitting the model and running the matrix below are the output result for Train & Test data

```
AUC score for Training data: 1.0
```

```
AUC score for Testing data: 0.8885006147673591
```

	precision	recall	f1-score	support
0	1.0	1.0	1.0	307.0
1	1.0	1.0	1.0	754.0
accuracy	1.0	1.0	1.0	1.0
macro avg	1.0	1.0	1.0	1061.0
weighted avg	1.0	1.0	1.0	1061.0

	precision	recall	f1-score	support
0	0.796875	0.666667	0.725979	153.00000
1	0.844512	0.914191	0.877971	303.00000
accuracy	0.831140	0.831140	0.831140	0.83114
macro avg	0.820694	0.790429	0.801975	456.00000
weighted avg	0.828529	0.831140	0.826974	456.00000

Table 1.6.4 KNN with GridSearch Train & Test data

After performing the tuning, we can see that the accuracy for Train is 100% and Test is 83%. Model is not valid since both the train and test set accuracy between both the set lies not within the range of 10%, it's an overfit model.

Further, to find the optimal value of k we will look at the K=1,3,5,7....19 and store the train and test scores in a Dataframe (ac_score) and using these scores, we will calculate the Misclassification error (MCE) and find the model with lowest Misclassification error (MCE) using the below mentioned formula: Misclassification error (MCE) = 1 - Test accuracy score

Accuracy Scores:

```
[0.7828947368421053,
 0.7828947368421053,
 0.7960526315789473,
 0.8026315789473685,
 0.8179824561403509,
 0.8179824561403509,
 0.8179824561403509,
 0.8179824561403509,
 0.8157894736842105,
 0.8289473684210527,
 0.8289473684210527,
 0.8355263157894737,
 0.8267543859649122,
 0.8355263157894737,
 0.8267543859649122,
 0.8333333333333334,
 0.8289473684210527,
 0.8355263157894737,
 0.8355263157894737]
```

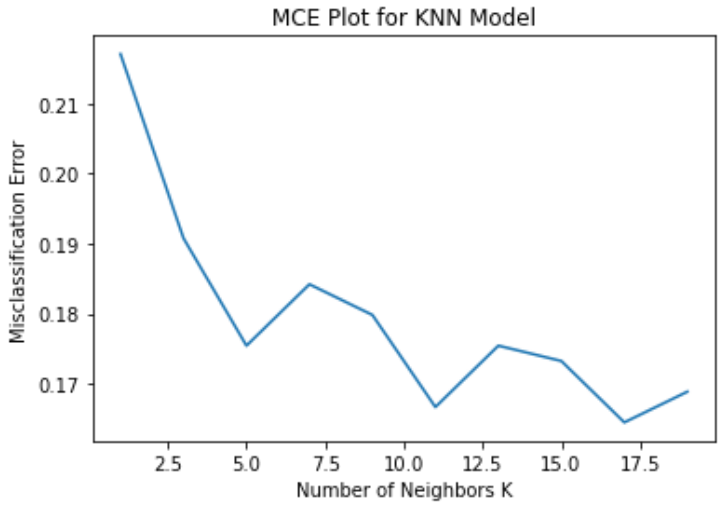


Table 1.6.5 MCE plot for KNN model

Hence, we can say that the lowest value of Misclassification Error is at k=17. Also, we have seen above that accuracy score for KNN Model at k=17 is 84% which is considered a good accuracy score and the difference between train and test accuracies is less than 10%, it is a valid model. Therefore, we can say that the optimal value of k is 17 for this particular model.

Train and Test results after fitting the model with n_neighbors=17

```
AUC score for Training data: 0.9046302456388944
```

	precision	recall	f1-score	support
0	0.740072	0.667752	0.702055	307.000000
1	0.869898	0.904509	0.886866	754.000000
accuracy	0.836004	0.836004	0.836004	0.836004
macro avg	0.804985	0.786131	0.794460	1061.000000
weighted avg	0.832333	0.836004	0.833391	1061.000000

```
AUC score for Testing data: 0.8877456373088289
```

	precision	recall	f1-score	support
0	0.809524	0.666667	0.731183	153.000000
1	0.845455	0.920792	0.881517	303.000000
accuracy	0.835526	0.835526	0.835526	0.835526
macro avg	0.827489	0.793729	0.806350	456.000000
weighted avg	0.833399	0.835526	0.831076	456.000000

Table 1.6.6 KNN model with K=17 Train & Test data

Random Forest

Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset. Running the model with the default parameters set to know the performance of model is good or not.

After fitting the model and running the matrix below are the output result for Train & Test data

```
AUC score for Training data: 1.0
```

	precision	recall	f1-score	support
0	1.0	1.0	1.0	307.0
1	1.0	1.0	1.0	754.0
accuracy	1.0	1.0	1.0	1.0
macro avg	1.0	1.0	1.0	1061.0
weighted avg	1.0	1.0	1.0	1061.0

```
AUC score for Testing data: 0.890668478612567
```

	precision	recall	f1-score	support
0	0.779412	0.69281	0.733564	153.00000
1	0.853125	0.90099	0.876404	303.00000
accuracy	0.831140	0.83114	0.831140	0.83114
macro avg	0.816268	0.79690	0.804984	456.00000
weighted avg	0.828392	0.83114	0.828478	456.00000

Table 1.6.7 randomForest model Train & Test data

After performing the tuning, we can see that the accuracy for Train is 100% and Test is 83%. Model is not valid since both the train and test set accuracy between both the set lies not within the range of 10%, its an overfit model.

Random Forest with GridSerachCV

First, we will define the library required for grid search followed by defining all the parameters or the combination that we want to test out on the model. Building a RandomForest classifier using a grid search cross validation to get best parameter/ estimators for a given dataset. The given tuning parameters are below:

```
GridSearchCV(cv=3, estimator=RandomForestClassifier(random_state=1),
             param_grid={'max_depth': [4, 5, 6], 'max_features': [4, 5, 6],
                         'min_samples_leaf': [8, 10, 15],
                         'min_samples_split': [40, 50, 60],
                         'n_estimators': [300]})
```

- **random_state*:*** *int, RandomState instance or None, optional (default=None).* Random state ensures that the splits that you generate are reproducible. Scikit-learn uses random permutations to generate the splits. The random state that you provide is used as a seed to the random number generator. This ensures that the random numbers are generated in the same order.
- **cv:** number of cross-validation you have to try for each selected set of hyperparameters.
- **max_depth:** *int or None, optional (default=None).* The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure. The max_depth value should be adjusted according no to high/low to avoid overfitting/underfitting.

- **min_samples_split:** *int, float, optional (default=2).* min_samples_split is used to control over-fitting. depending on the level of underfitting or overfitting, you can tune the values for min_samples_split.
- **min_samples_leaf***: int, float, optional (default=1).* The minimum number of samples required to be at a leaf node. min_samples_leaf is also used to control over-fitting by defining that each leaf has more than one element. Thus ensuring that the tree cannot overfit the training dataset.
- **max_features:** These are the maximum number of features Random Forest is allowed to try in individual tree. Increasing max_features generally improve the performance of the model as at each node now we have a higher number of options to be considered.
- **n_estimators:** This is the number of trees you want to build before taking the maximum voting or averages of predictions. Higher number of trees give you better performance but makes your code slower. You should choose as high value as your processor can handle because this makes your predictions stronger and more stable.

The best parameters after fitting the model is**:**

```
{'max_depth': 4,
 'max_features': 4,
 'min_samples_leaf': 10,
 'min_samples_split': 40,
 'n_estimators': 300}
```

After fitting the model and running the matrix below are the output result for Train & Test data

```
AUC score for Training data: 0.9093218362004165
```

	precision	recall	f1-score	support
0	0.787072	0.674267	0.726316	307.000000
1	0.874687	0.925729	0.899485	754.000000
accuracy	0.852969	0.852969	0.852969	0.852969
macro avg	0.830879	0.799998	0.812900	1061.000000
weighted avg	0.849335	0.852969	0.849378	1061.000000

```
AUC score for Testing data: 0.8898487888004486
```

	precision	recall	f1-score	support
0	0.790698	0.666667	0.723404	153.000000
1	0.844037	0.910891	0.876190	303.000000
accuracy	0.828947	0.828947	0.828947	0.828947
macro avg	0.817367	0.788779	0.799797	456.000000
weighted avg	0.826140	0.828947	0.824927	456.000000

Table 1.6.8 randomForest model with GridSearch Train & Test data

Here we can see the accuracy for Train is 85% and Test is 83%. After fine tuning the model, we can see that model has improved. Hence, we can say that fine tuning this particular model make much difference in model performance. Model is valid since both the train and test set accuracy are almost similar and the difference between both the set lies within the range of 10% (not overfit or underfit).

Bagging

Bootstrap Aggregation (or Bagging for short), is a simple and very powerful ensemble method. It is a general procedure that can be used to reduce the variance for those algorithms that have high variance. Hence, we will build the Bagging model using Random Forest as the base estimator and then fit the model.

```
BaggingClassifier(base_estimator=RandomForestClassifier(), n_estimators
=50, random_state=1)
```

- **base_estimator:** The base estimator to fit on random subsets of the dataset. Here, we are using RandomForest Classifier to improve accuracy and reduce variance.
- **n_estimators:**The number of base estimators in the ensemble. Here we are taking 100.
- **random_state:** Controls the random resampling of the original dataset (sample wise and feature wise).

After fitting the model to the classifier below are the outputs for Train & Test data

```
AUC score for Training data: 0.9971660373772022
```

	precision	recall	f1-score	support
0	0.975265	0.899023	0.935593	307.000000
1	0.960154	0.990716	0.975196	754.000000
accuracy	0.964185	0.964185	0.964185	0.964185
macro avg	0.967710	0.944869	0.955395	1061.000000
weighted avg	0.964527	0.964185	0.963737	1061.000000

```
AUC score for Testing data: 0.8954787635626309
```

	precision	recall	f1-score	support
0	0.792308	0.673203	0.727915	153.00000
1	0.846626	0.910891	0.877583	303.00000
accuracy	0.831140	0.831140	0.831140	0.83114
macro avg	0.819467	0.792047	0.802749	456.00000
weighted avg	0.828401	0.831140	0.827366	456.00000

Table 1.6.9 Bagging model Train & Test data

Here we can see the accuracy for Train is 96% and test is 83%. Model is not valid since both the train and test set accuracy between both the set lies not within the range of 10%, its an overfit model. Hence performing a tuning for the Bagging model below to get a better accuracy score.

Bagging with GridSearchCV

First, we will define the library required for grid search followed by defining all the parameters or the combination that we want to test out on the model. Building a bagging model using a grid search cross validation to get best parameter/ estimators for a given dataset. The given tuning parameters are below:

```
GridSearchCV(cv=3, estimator=BaggingClassifier(random_state=1),
             param_grid={'base_estimator': [RandomForestClassifier()],
                         'bootstrap': [True, False],
                         'bootstrap_features': [True, False],
                         'max_samples': [0.5, 1.0],
                         'n_estimators': [50, 80, 100]})
```

- **base_estimator:** The base estimator to fit on random subsets of the dataset. Here, we are using RandomForest Classifier to improve accuracy and reduce variance.
- **n_estimators:**The number of base estimators in the ensemble. Here we are taking 100.
- **random_state:** Controls the random resampling of the original dataset (sample wise and feature wise).
- **max_samples:** The number of features to draw from X to train each base estimator
- **bootstrap:** Whether samples are drawn with replacement. If False, sampling without replacement is performed

- **bootstrap_features:** Whether features are drawn with replacement.

The best paramters after the fitting the model are:

```
BaggingClassifier(base_estimator=RandomForestClassifier(), max_samples=
0.5,n_estimators=80, random_state=1)
```

After fitting the model to the classifier below are the outputs for Train & Test data

```
AUC score for Training data: 0.9798166564425128
```

	precision	recall	f1-score	support
0	0.893773	0.794788	0.841379	307.000000
1	0.920051	0.961538	0.940337	754.000000
accuracy	0.913289	0.913289	0.913289	0.913289
macro avg	0.906912	0.878163	0.890858	1061.000000
weighted avg	0.912447	0.913289	0.911704	1061.000000

```
AUC score for Testing data: 0.8975711296619858
```

	precision	recall	f1-score	support
0	0.789474	0.686275	0.734266	153.000000
1	0.851393	0.907591	0.878594	303.000000
accuracy	0.833333	0.833333	0.833333	0.833333
macro avg	0.820433	0.796933	0.806430	456.000000
weighted avg	0.830618	0.833333	0.830168	456.000000

Table 1.6.10 Bagging with GridSearch Train & Test data

Here we can see the accuracy for Train is 91% and Test is 83%. After fine tuning the model, we can see that model has improved. Hence, we can say that fine tuning this particular model make much difference in model performance. Model is valid since both the train and test set accuracy are almost similar and the difference between both the set lies within the range of 10% (not overfit or underfit).

We will now try to build the model using Boosting. While bagging and boosting are both ensemble methods, they approach the problem from opposite directions. Bagging uses complex base models and tries to "smooth out" their predictions, while boosting uses simple base models and tries to "boost" their aggregate complexity.

Boosting

In Boosting, Base estimators are built sequentially and one tries to reduce the bias of the combined estimator. The idea is to combine several weak models to produce a powerful ensemble. It makes the boosting algorithms prone to overfitting. Examples: AdaBoost, Gradient Tree Boosting.

There are three types of Boosting Algorithms which are as follows:

1. AdaBoost (Adaptive Boosting) algorithm.
2. Gradient Boosting algorithm.
3. XG Boost algorithm.

Here, we will use Adaboost and Gradient Boosting algorithm.

AdaBoost (Adaptive Boosting)

It fits a sequence of weak learners on different weighted training data. It starts by predicting original data set and gives equal weight to each observation. If prediction is incorrect using the first learner, then it gives higher weight to observation which have been predicted incorrectly. Being an iterative process, it continues to add learner(s) until a limit is reached in the number of models or accuracy.

Building the AdaBoost classifier with its default parameters.

*(base_estimator=None, *, n_estimators=50, learning_rate=1.0)*

- **base_estimator:** The base estimator from which the boosted ensemble is built. By default the base estimator is a decision tree with max_depth=1
- **n_estimators:** The maximum number of estimators at which boosting is terminated. Default value is 50.
- **learning_rate:** Learning rate shrinks the contribution of each classifier by learning_rate. There is a trade-off between learning_rate and n_estimators.

After fitting the model to the classifier below are the outputs for Train & Test data

AUC score for Training data: 0.9118987549572745

	precision	recall	f1-score	support
0	0.760870	0.684039	0.720412	307.000000
1	0.876433	0.912467	0.894087	754.000000
accuracy	0.846371	0.846371	0.846371	0.846371
macro avg	0.818651	0.798253	0.807249	1061.000000
weighted avg	0.842995	0.846371	0.843834	1061.000000

AUC score for Testing data: 0.8805517806682629

	precision	recall	f1-score	support
0	0.739437	0.686275	0.711864	153.000000
1	0.847134	0.877888	0.862237	303.000000
accuracy	0.813596	0.813596	0.813596	0.813596
macro avg	0.793285	0.782081	0.787051	456.000000
weighted avg	0.810999	0.813596	0.811783	456.000000

Table 1.6.11 AdaBoost model Train & Test data

Here we can see the accuracy for Train is 85% and Test is 81%. The model is a valid since both the train and test set accuracy are almost similar and the difference between both the set lies within the range of 10%(not overfit or underfit). We will further tune the model and see the model can perform better after the tuning.

AdaBoost (Adaptive Boosting) with GridSearchCV

First, we will define the library required for grid search followed by defining all the parameters or the combination that we want to test out on the model. Building a AdaBoost model using a grid search cross validation to get best parameter/ estimators for a given dataset. The given tuning parameters are below:

```
params_ada = { 'n_estimators': np.arange(10,110,10),
               'learning_rate':np.arange (0.1, 2, 0.1) }
acc_scorer = metrics.make_scorer(metrics.recall_score)
grid_cv_ada=GridSearchCV(AdaBoostClassifier(random_state=1), param_grid= params_ada,
cv=5, scoring=acc_scorer)
```

- **base_estimator:** The base estimator from which the boosted ensemble is built.
- **n_estimators:** The maximum number of estimators at which boosting is terminated. Default value is 50.
- **learning_rate:** Learning rate shrinks the contribution of each classifier by learning_rate. There is a trade-off between learning_rate and n_estimators.
- **random_state**: Controls the random seed given at each base_estimator at each boosting iteration.
- **Acc_scorer:** Type of scoring used to compare parameter combinations

The best parameters after fitting the model are:

```
{'learning_rate': 0.1, 'n_estimators': 20}
```

After fitting the model to the classifier below are the outputs for Train & Test data

```
AUC score for Training data: 0.8748628379370826
```

	precision	recall	f1-score	support
0	0.790850	0.394137	0.526087	307.000000
1	0.795154	0.957560	0.868833	754.000000
accuracy	0.794533	0.794533	0.794533	0.794533
macro avg	0.793002	0.675848	0.697460	1061.000000
weighted avg	0.793909	0.794533	0.769659	1061.000000

```
AUC score for Testing data: 0.8585172242714467
```

	precision	recall	f1-score	support
0	0.851351	0.411765	0.555066	153.000000
1	0.764398	0.963696	0.852555	303.000000
accuracy	0.778509	0.778509	0.778509	0.778509
macro avg	0.807875	0.687731	0.703810	456.000000
weighted avg	0.793573	0.778509	0.752739	456.000000

Table 1.6.12 AdaBoost model with GridSearch Train & Test data

Here we can see the accuracy for Train is 79% and Test is 78%. After fine tuning the model, we can see that model is performing not well than the base model. Hence, we can say that fine tuning this particular model does not make difference in model performance. Model is valid since both the train and test set accuracy are almost similar and the difference between both the set lies within the range of 10% (not overfit or underfit).

Below are the **Important Features** for the tuned AdaBoost Classifier:

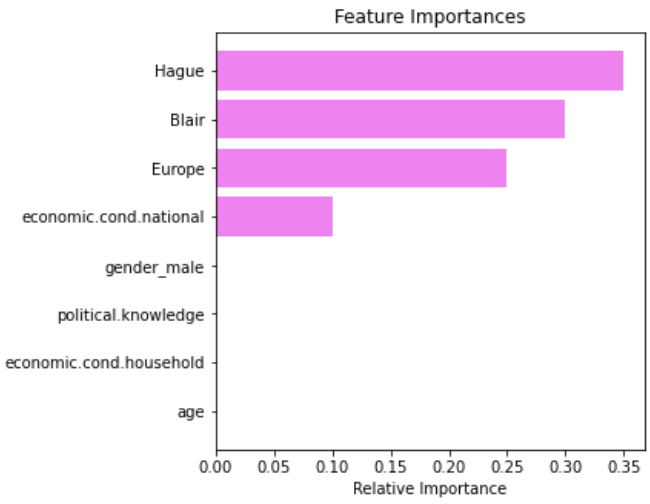


Table 1.6.13 AdaBoost feature importance

Hague is the most important feature as per the tuned AdaBoost model.

Gradient Boosting algorithm

Gradient Boosting is a popular boosting algorithm. In gradient boosting, each predictor corrects its predecessor's error. In contrast to Adaboost, the weights of the training instances are not tweaked, instead, each predictor is trained using the residual errors of predecessor as labels.

Building the Gradient Booster classifier with its default parameters.

(*n_estimators=100, learning_rate=0.1, random_state=1, loss='deviance'*)

- **n_estimators:** The number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting so a large number usually results in better performance.
- **learning_rate:** Learning rate shrinks the contribution of each classifier by learning_rate. There is a trade-off between learning_rate and n_estimators.
- **random_state**: Controls the random seed given to each Tree estimator at each boosting iteration.
- **Loss:** The loss function to be optimized. 'deviance' refers to deviance (= logistic regression) for classification with probabilistic outputs. For loss 'exponential' gradient boosting recovers the AdaBoost algorithm.

After fitting the model to the classifier below are the outputs for Train & Test data

AUC score for Training data: 0.951155185373988

	precision	recall	f1-score	support
0	0.838596	0.778502	0.807432	307.000000
1	0.912371	0.938992	0.925490	754.000000
accuracy	0.892554	0.892554	0.892554	0.892554
macro avg	0.875484	0.858747	0.866461	1061.000000
weighted avg	0.891024	0.892554	0.891330	1061.000000

AUC score for Testing data: 0.8993615047779289

	precision	recall	f1-score	support
0	0.795455	0.686275	0.736842	153.000000
1	0.851852	0.910891	0.880383	303.000000
accuracy	0.835526	0.835526	0.835526	0.835526
macro avg	0.823653	0.798583	0.808612	456.000000
weighted avg	0.832929	0.835526	0.832221	456.000000

Table 1.6.14 Gradient Boosting model Train & Test data

Here we can see the accuracy for Train is 89% and Test is 83%. The model is a valid since both the train and test set accuracy are almost similar and the difference between both the set lies within the range of 10% (not overfit or underfit). We will further tune the model and see the model can perform better after the tuning.

Gradient Boosting algorithm with GridSearchCV

First, we will define the library required for grid search followed by defining all the parameters or the combination that we want to test out on the model. Building a Gradient Boost model using a grid search cross validation to get best parameter/ estimators for a given dataset. The given tuning parameters are below:

```
# Choose the type of classifier.
gbc_tuned = GradientBoostingClassifier(random_state=1)

parameters = {
    "n_estimators": [100,150,200,250],
    "subsample":[0.8,0.9,1],
    "max_features":[0.7,0.8,0.9,1]
}

# Type of scoring used to compare parameter combinations
acc_scorer = metrics.make_scorer(metrics.recall_score)
grid_obj = GridSearchCV(gbc_tuned, parameters, scoring=acc_scorer,cv=5)
```

- **n_estimators:** The number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting so a large number usually results in better performance.
- **random_state**: Controls the random seed given to each Tree estimator at each boosting iteration.
- **Subsample:** The fraction of samples to be used for fitting the individual base learners. If smaller than 1.0 this results in Stochastic Gradient Boosting. subsample interacts with the parameter n_estimators. Choosing subsample < 1.0 leads to a reduction of variance and an increase in bias.
- **max_features:** The number of features to consider when looking for the best split
- **Acc_scorer:** Type of scoring used to compare parameter combinations

The best parameters after fitting the model are:

```
GradientBoostingClassifier(max_features=1, random_state=1, subsample
=1)
```

After fitting the model to the classifier below are the outputs for Train & Test data

AUC score for Training data: 0.9325594656943641

	precision	recall	f1-score	support
0	0.814286	0.742671	0.776831	307.000000
1	0.898848	0.931034	0.914658	754.000000
accuracy	0.876532	0.876532	0.876532	0.876532
macro avg	0.856567	0.836853	0.845745	1061.000000
weighted avg	0.874380	0.876532	0.874778	1061.000000

AUC score for Testing data: 0.8942060872753943

	precision	recall	f1-score	support
0	0.796992	0.692810	0.741259	153.000000
1	0.854489	0.910891	0.881789	303.000000
accuracy	0.837719	0.837719	0.837719	0.837719
macro avg	0.825741	0.801851	0.811524	456.000000
weighted avg	0.835198	0.837719	0.834637	456.000000

Table 1.6.15 Gradient Boosting with GridSerach model Train & Test data

Here we can see the accuracy for Train is 88% and Test is 84%. After fine tuning the model, we can see that model has improved. Hence, we can say that fine tuning this particular model make difference in model performance. Model is valid since both the train and test set accuracy are almost similar and the difference between both the set lies within the range of 10% (not overfit or underfit).

Below are the **Important Features** for the tuned Gradient Boosting Classifier:

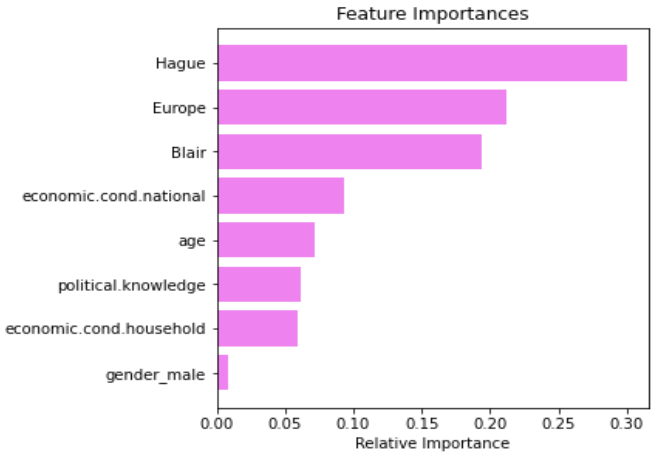


Table 1.6.16 Gradient Boosting with GridSerach Important features

Hague is the most important feature, followed by Europe and Blair, as per the tuned gradient boosting model.

On comparing accuracy of Gradient Boosting and AdaBoost models, the difference between the train and test set is very low for Gradient Boosting compared to Adaboost model, Gradient Boosting model performs well for predicting Labour or conservative party

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.

Answer:

Logistic regression Base model performance

Confusion matrix, Classification report & ROC_AUC score for Training and Testing dataset

Classification report of Training Data:

	precision	recall	f1-score	support
0	0.75	0.65	0.69	307
1	0.86	0.91	0.89	754
accuracy			0.84	1061
macro avg	0.81	0.78	0.79	1061
weighted avg	0.83	0.84	0.83	1061

Classification report of Testing Data:

	precision	recall	f1-score	support
0	0.75	0.72	0.74	153
1	0.86	0.88	0.87	303
accuracy			0.83	456
macro avg	0.81	0.80	0.80	456
weighted avg	0.83	0.83	0.83	456

Figure 1.7.1 Logistic Regression Confusion Matrix & Classification report

```
AUC score for Training data: 0.890
AUC score for Testing data: 0.879
```

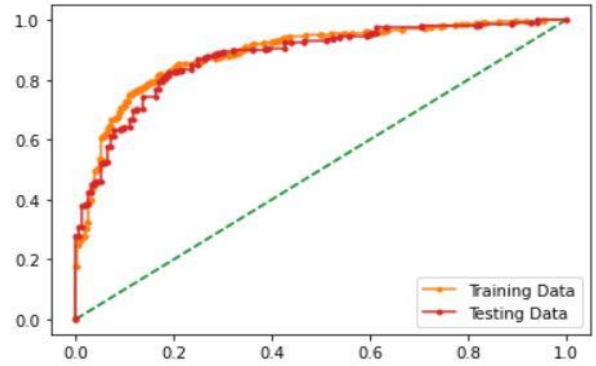


Figure 1.7.2 Logistic Regression AUC/ROC curve

Logistic regression GridSearchCV model performance

Confusion matrix, Classification report & ROC_AUC score for Training and Testing dataset


```
Classification report of Training Data:
              precision    recall  f1-score   support

           0       0.74      0.64      0.69       307
           1       0.86      0.91      0.88       754

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

Classification report of Testing Data:
              precision    recall  f1-score   support

           0       0.76      0.73      0.74       153
           1       0.86      0.88      0.87       303

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

Figure 1.7.3 Logistic Regression(GridSearch) Confusion Matrix & Classification report

```
AUC score for Training data: 0.890
AUC score for Testing data: 0.883
```

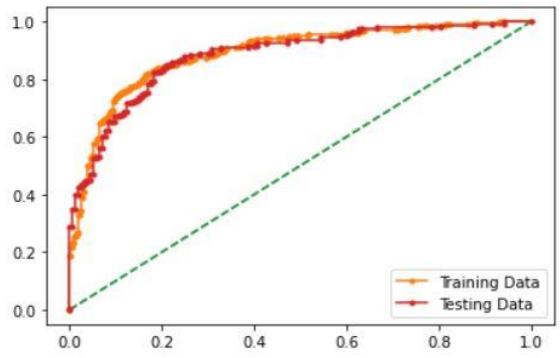
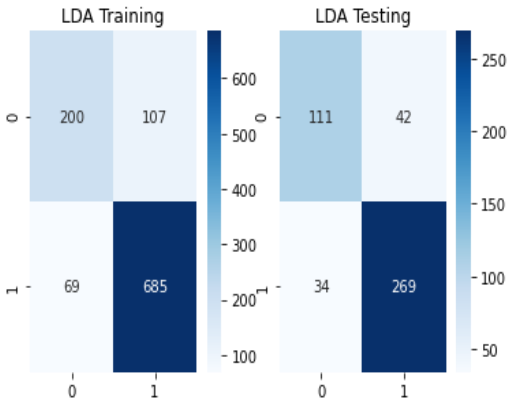


Figure 1.7.4 Logistic Regression(GridSearch) AUC/ROC curve

LDA Base model performance

Confusion matrix, Classification report & ROC_AUC score for Training and Testing dataset



Classification report of Training Data:

	precision	recall	f1-score	support
0	0.74	0.65	0.69	307
1	0.86	0.91	0.89	754
accuracy			0.83	1061
macro avg	0.80	0.78	0.79	1061
weighted avg	0.83	0.83	0.83	1061

Classification report of Testing Data:

	precision	recall	f1-score	support
0	0.77	0.73	0.74	153
1	0.86	0.89	0.88	303
accuracy			0.83	456
macro avg	0.82	0.81	0.81	456
weighted avg	0.83	0.83	0.83	456

Figure 1.7.5 LDA Confusion Matrix & Classification report

AUC score for Training data: 0.889
AUC score for Testing data: 0.888

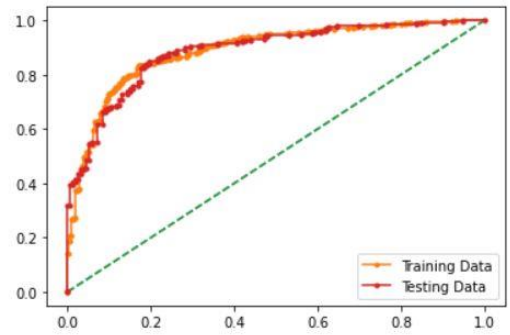


Figure 1.7.6 LDA AUC/ROC curve

LDA GridSerachCV model performance

Confusion matrix, Classification report & ROC_AUC score for Training and Testing dataset

Classification report of Training Data:

	precision	recall	f1-score	support
0	0.74	0.65	0.69	307
1	0.86	0.91	0.89	754
accuracy			0.83	1061
macro avg	0.80	0.78	0.79	1061
weighted avg	0.83	0.83	0.83	1061

Classification report of Testing Data:

	precision	recall	f1-score	support
0	0.77	0.73	0.74	153
1	0.86	0.89	0.88	303
accuracy			0.83	456
macro avg	0.82	0.81	0.81	456
weighted avg	0.83	0.83	0.83	456

Figure 1.7.7 LDA(GridSearch) Confusion Matrix & Classification report

```
AUC score for Training data: 0.889
AUC score for Testing data: 0.888
```

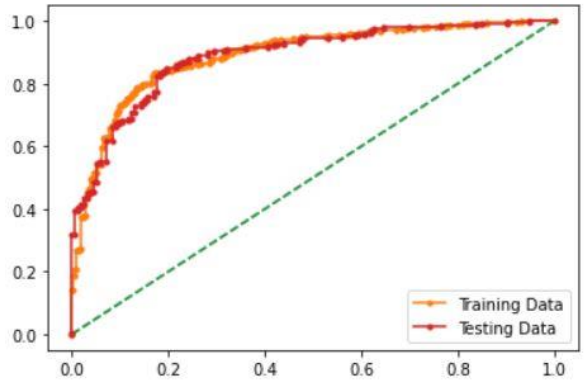
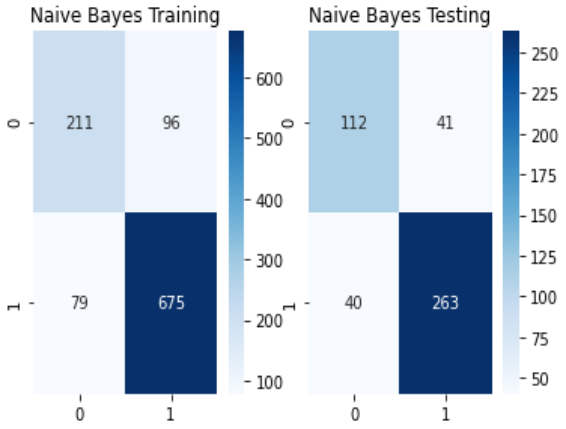


Figure 1.7.8 LDA(GridSearch) AUC/ROC curve

Naïve Bayes Model Performance

Confusion matrix, Classification report & ROC_AUC score for Training and Testing dataset




```
Classification report of Training Data:
              precision    recall  f1-score   support

           0       0.73      0.69      0.71       307
           1       0.88      0.90      0.89       754

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

Classification report of Testing Data:
              precision    recall  f1-score   support

           0       0.74      0.73      0.73       153
           1       0.87      0.87      0.87       303

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

Figure 1.7.9 Naïve Bayes Confusion Matrix & Classification report

```
AUC score for Training data: 0.888
AUC score for Testing data: 0.876
```

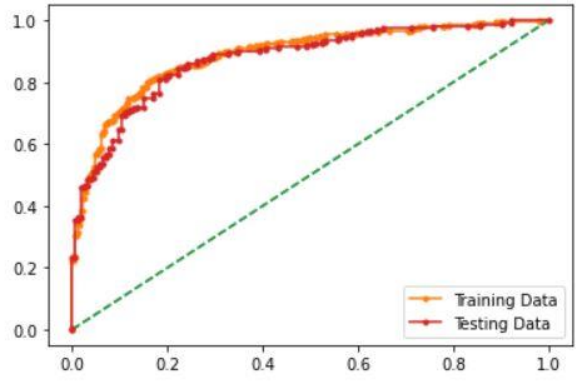


Figure 1.7.10 Naïve Bayes AUC/ROC curve

Naïve Bayes GridSearchCV Model Performance

Confusion matrix, Classification report & ROC_AUC score for Training and Testing dataset

Classification report of Training Data:

	precision	recall	f1-score	support
0	0.74	0.68	0.71	307
1	0.88	0.90	0.89	754
accuracy			0.84	1061
macro avg	0.81	0.79	0.80	1061
weighted avg	0.84	0.84	0.84	1061

Classification report of Testing Data:

	precision	recall	f1-score	support
0	0.75	0.72	0.73	153
1	0.86	0.88	0.87	303
accuracy			0.82	456
macro avg	0.80	0.80	0.80	456
weighted avg	0.82	0.82	0.82	456

Figure 1.7.11 Naïve Bayes(GridSerach) Confusion Matrix & Classification report

AUC score for Training data: 0.887
AUC score for Testing data: 0.879

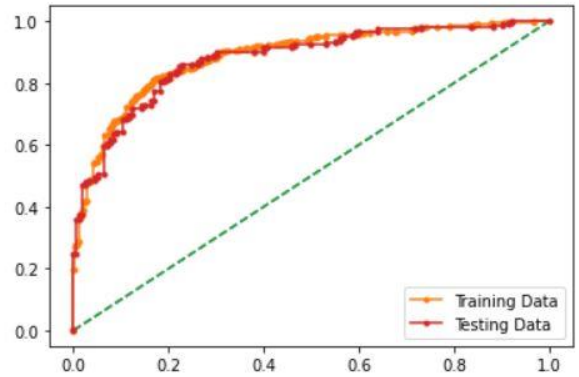
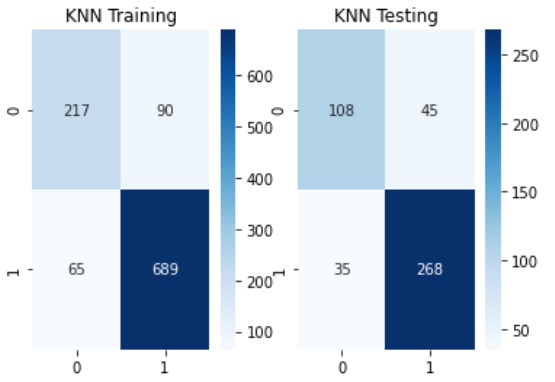


Figure 1.7.12 Naïve Bayes(GridSerach) AUC/ROC curve

KNN model performance

Confusion matrix, Classification report & ROC_AUC score for Training and Testing dataset



Classification report of Training Data:

	precision	recall	f1-score	support
0	0.77	0.71	0.74	307
1	0.88	0.91	0.90	754
accuracy			0.85	1061
macro avg	0.83	0.81	0.82	1061
weighted avg	0.85	0.85	0.85	1061

Classification report of Testing Data:

	precision	recall	f1-score	support
0	0.76	0.71	0.73	153
1	0.86	0.88	0.87	303
accuracy			0.82	456
macro avg	0.81	0.80	0.80	456
weighted avg	0.82	0.82	0.82	456

Figure 1.7.13 KNN base model Confusion Matrix & Classification report

```
AUC score for Training data: 0.928
AUC score for Testing data: 0.867
```

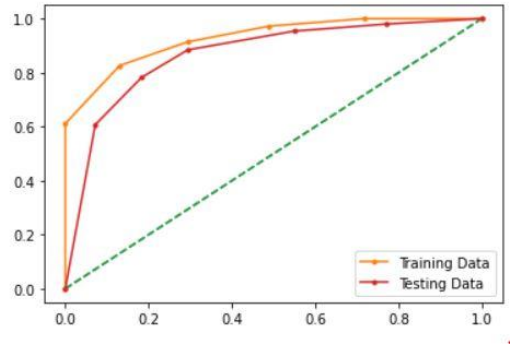
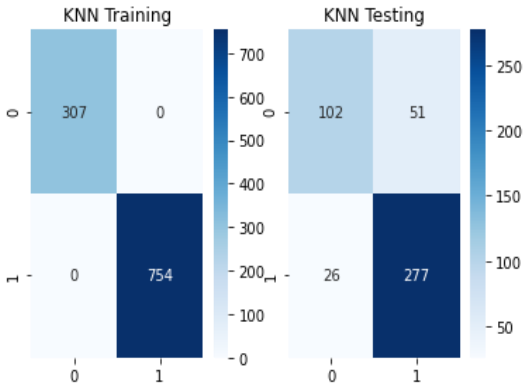


Figure 1.7.14 KNN base model AUC/ROC curve

KNN with GridSeracgCV model performance

Confusion matrix, Classification report & ROC_AUC score for Training and Testing dataset




```
Classification report of Training Data:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       307
           1       1.00      1.00      1.00       754

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

Classification report of Testing Data:
              precision    recall  f1-score   support

           0       0.80      0.67      0.73       153
           1       0.84      0.91      0.88       303

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

Figure 1.7.15 KNN(GridSerach) Confusion Matrix & Classification report

```
AUC score for Training data: 1.000
AUC score for Testing data: 0.889
```

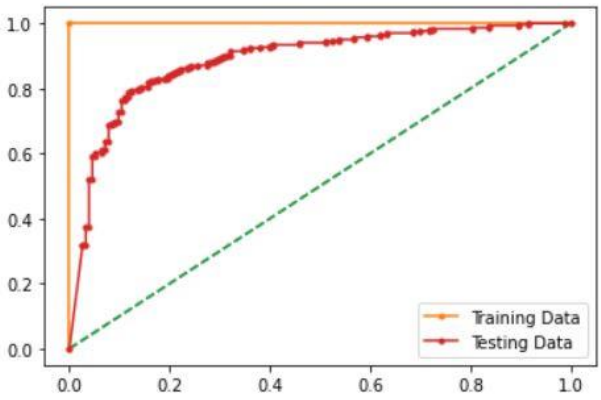
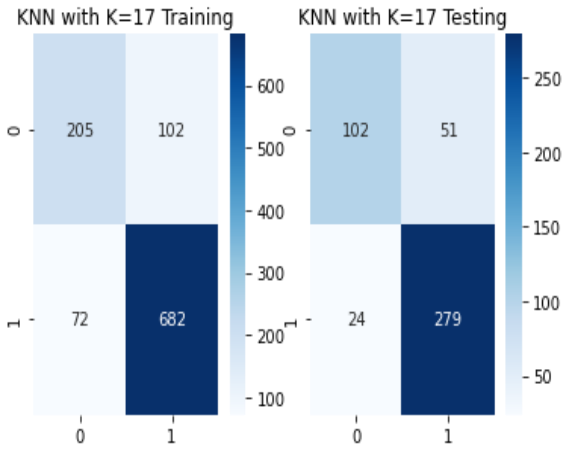


Figure 1.7.16 KNN(GridSerach) AUC/ROC curve

KNN with K=17 model performance

Confusion matrix, Classification report & ROC_AUC score for Training and Testing dataset



Classification report of Training Data:

	precision	recall	f1-score	support
0	0.74	0.67	0.70	307
1	0.87	0.90	0.89	754
accuracy			0.84	1061
macro avg	0.80	0.79	0.79	1061
weighted avg	0.83	0.84	0.83	1061

Classification report of Testing Data:

	precision	recall	f1-score	support
0	0.81	0.67	0.73	153
1	0.85	0.92	0.88	303
accuracy			0.84	456
macro avg	0.83	0.79	0.81	456
weighted avg	0.83	0.84	0.83	456

Figure 1.7.17 KNN(K=17) Confusion Matrix & Classification report

AUC score for Training data: 0.905
AUC score for Testing data: 0.888

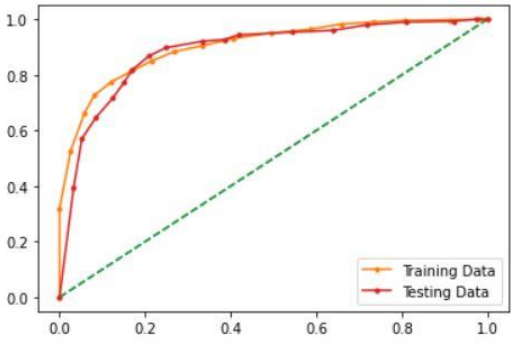


Figure 1.7.18 KNN(K=17) AUC/ROC curve

Random Forest base model performance

Confusion matrix, Classification report & ROC_AUC score for Training and Testing dataset

Classification report of Training Data:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	307
1	1.00	1.00	1.00	754
accuracy			1.00	1061
macro avg	1.00	1.00	1.00	1061
weighted avg	1.00	1.00	1.00	1061

Classification report of Testing Data:

	precision	recall	f1-score	support
0	0.78	0.69	0.73	153
1	0.85	0.90	0.88	303
accuracy			0.83	456
macro avg	0.82	0.80	0.80	456
weighted avg	0.83	0.83	0.83	456

Figure 1.7.19 Random Forest Confusion Matrix & Classification report

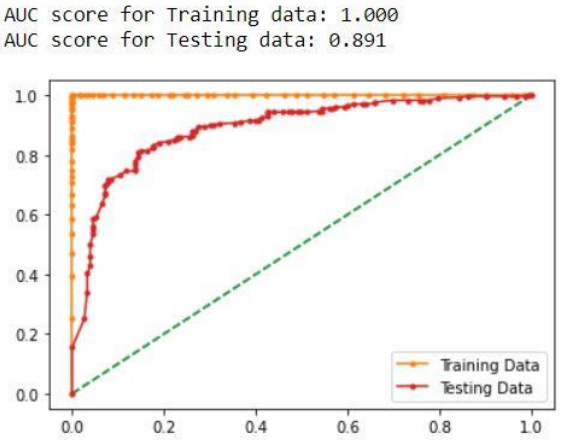


Figure 1.7.20 Random Forest AUC/ROC curve

Random Forest with GridSerachCV model performance

Confusion matrix, Classification report & ROC_AUC score for Training and Testing dataset

Classification report of Training Data:

	precision	recall	f1-score	support
0	0.79	0.67	0.73	307
1	0.87	0.93	0.90	754
accuracy			0.85	1061
macro avg	0.83	0.80	0.81	1061
weighted avg	0.85	0.85	0.85	1061

Classification report of Testing Data:

	precision	recall	f1-score	support
0	0.79	0.67	0.72	153
1	0.84	0.91	0.88	303
accuracy			0.83	456
macro avg	0.82	0.79	0.80	456
weighted avg	0.83	0.83	0.82	456

Figure 1.7.21 Random Forest (GridSerach) Confusion Matrix & Classification report

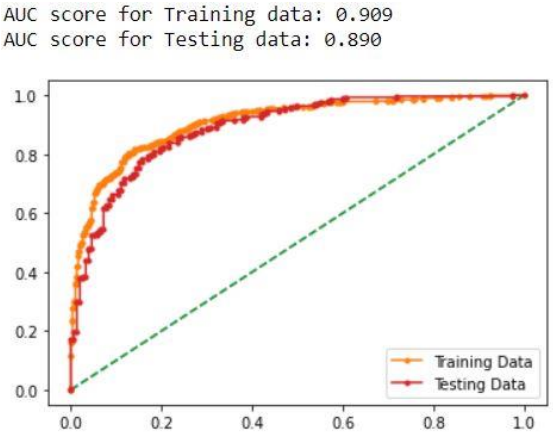


Figure 1.7.22 Random Forest (GridSerach) AUC/ROC curve

Bagging base model performance

Confusion matrix, Classification report & ROC_AUC score for Training and Testing dataset

Classification report of Training Data:

	precision	recall	f1-score	support
0	0.98	0.90	0.94	307
1	0.96	0.99	0.98	754
accuracy			0.96	1061
macro avg	0.97	0.94	0.96	1061
weighted avg	0.96	0.96	0.96	1061

Classification report of Testing Data:

	precision	recall	f1-score	support
0	0.79	0.67	0.73	153
1	0.85	0.91	0.88	303
accuracy			0.83	456
macro avg	0.82	0.79	0.80	456
weighted avg	0.83	0.83	0.83	456

Figure 1.7.23 Bagging Confusion Matrix & Classification report

AUC score for Training data: 0.997
AUC score for Testing data: 0.895

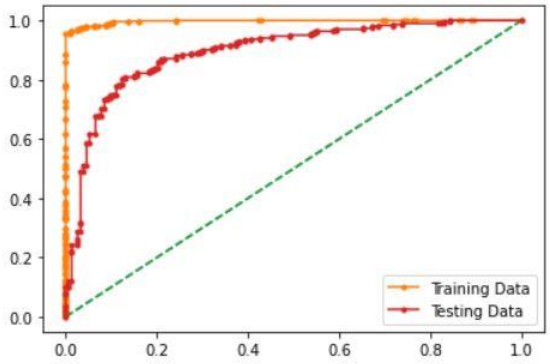


Figure 1.7.24 Bagging AUC/ROC curve

Bagging with GridSearchCV model performance

Confusion matrix, Classification report & ROC_AUC score for Training and Testing dataset

Classification report of Training Data:

	precision	recall	f1-score	support
0	0.89	0.79	0.84	307
1	0.92	0.96	0.94	754
accuracy			0.91	1061
macro avg	0.91	0.88	0.89	1061
weighted avg	0.91	0.91	0.91	1061

Classification report of Testing Data:

	precision	recall	f1-score	support
0	0.79	0.69	0.73	153
1	0.85	0.91	0.88	303
accuracy			0.83	456
macro avg	0.82	0.80	0.81	456
weighted avg	0.83	0.83	0.83	456

Figure 1.7.25 Bagging(GridSerach) Confusion Matrix & Classification report

```
AUC score for Training data: 0.980
AUC score for Testing data: 0.898
```

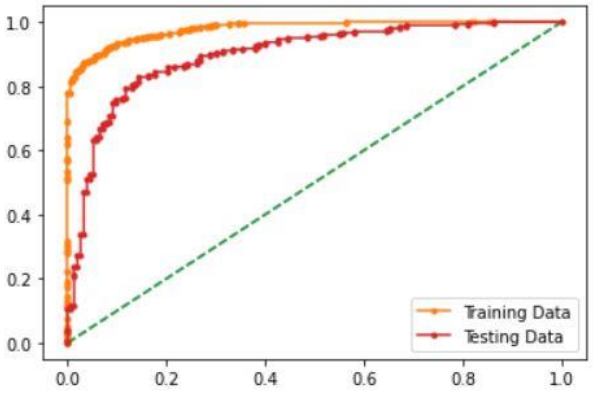
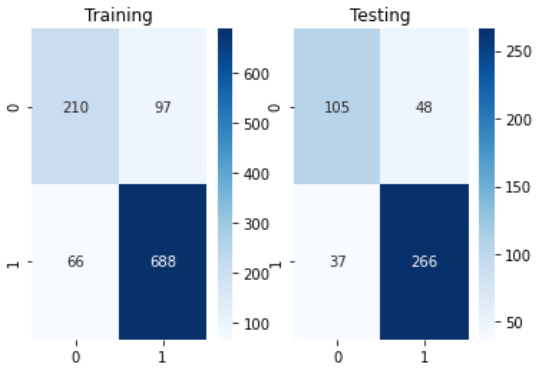


Figure 1.7.26 Bagging(GridSerach) AUC/ROC curve

AdaBoost base model performance

Confusion matrix, Classification report & ROC_AUC score for Training and Testing dataset




```
Classification report of Training Data:
              precision    recall  f1-score   support

           0       0.76      0.68      0.72       307
           1       0.88      0.91      0.89       754

    accuracy                           0.85      1061
   macro avg       0.82      0.80      0.81      1061
weighted avg       0.84      0.85      0.84      1061

Classification report of Testing Data:
              precision    recall  f1-score   support

           0       0.74      0.69      0.71       153
           1       0.85      0.88      0.86       303

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

Figure 1.7.27 AdaBoost Confusion Matrix & Classification report

```
AUC score for Training data: 0.912
AUC score for Testing data: 0.881
```

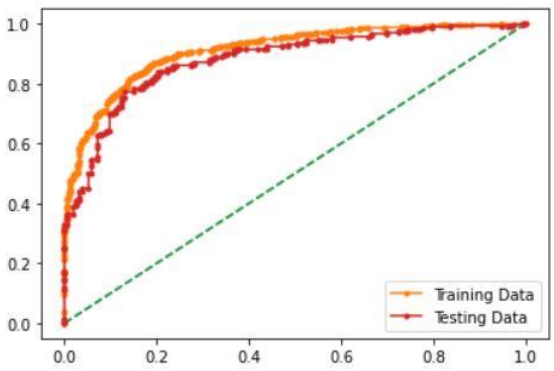
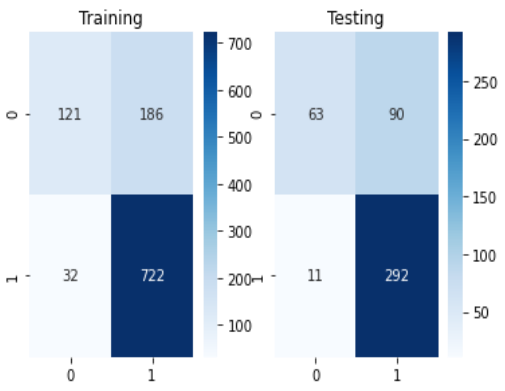


Figure 1.7.28 AdaBoost AUC/ROC curve

AdaBoost GridSerachCV model performance

Confusion matrix, Classification report & ROC_AUC score for Training and Testing dataset



Classification report of Training Data:

	precision	recall	f1-score	support
0	0.79	0.39	0.53	307
1	0.80	0.96	0.87	754
accuracy			0.79	1061
macro avg	0.79	0.68	0.70	1061
weighted avg	0.79	0.79	0.77	1061

Classification report of Testing Data:

	precision	recall	f1-score	support
0	0.85	0.41	0.56	153
1	0.76	0.96	0.85	303
accuracy			0.78	456
macro avg	0.81	0.69	0.70	456
weighted avg	0.79	0.78	0.75	456

Figure 1.7.29 AdaBoost(GridSerach) Confusion Matrix & Classification report

AUC score for Training data: 0.875
AUC score for Testing data: 0.859

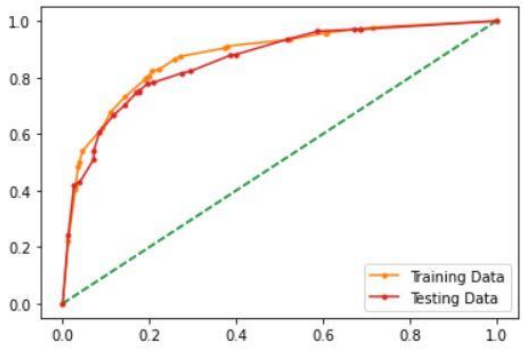


Figure 1.7.30 AdaBoost(GridSerach) AUC/ROC curve

Gradient Boosting base model performance

Confusion matrix, Classification report & ROC_AUC score for Training and Testing dataset

Classification report of Training Data:

	precision	recall	f1-score	support
0	0.84	0.78	0.81	307
1	0.91	0.94	0.93	754
accuracy			0.89	1061
macro avg	0.88	0.86	0.87	1061
weighted avg	0.89	0.89	0.89	1061

Classification report of Testing Data:

	precision	recall	f1-score	support
0	0.80	0.69	0.74	153
1	0.85	0.91	0.88	303
accuracy			0.84	456
macro avg	0.82	0.80	0.81	456
weighted avg	0.83	0.84	0.83	456

Figure 1.7.31 Gradient Boosting Confusion Matrix & Classification report

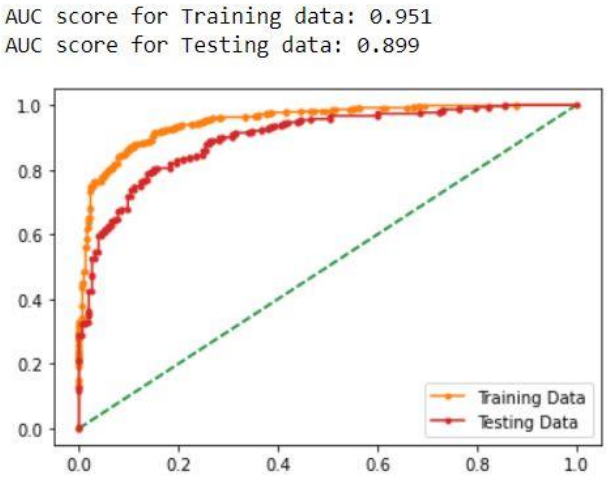
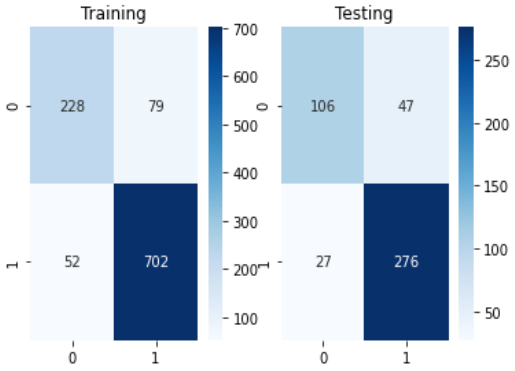


Figure 1.7.32 Gradient Boosting AUC/ROC curve

Gradient Boosting GridSearchCV model performance

Confusion matrix, Classification report & ROC_AUC score for Training and Testing dataset



Classification report of Training Data:

	precision	recall	f1-score	support
0	0.81	0.74	0.78	307
1	0.90	0.93	0.91	754
accuracy			0.88	1061
macro avg	0.86	0.84	0.85	1061
weighted avg	0.87	0.88	0.87	1061

Classification report of Testing Data:

	precision	recall	f1-score	support
0	0.80	0.69	0.74	153
1	0.85	0.91	0.88	303
accuracy			0.84	456
macro avg	0.83	0.80	0.81	456
weighted avg	0.84	0.84	0.83	456

Figure 1.7.33 Gradient Boosting (GridSerach) Confusion Matrix & Classification report

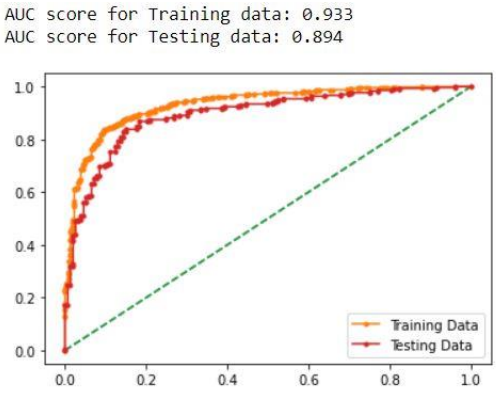


Figure 1.7.34 Gradient Boosting (GridSerach) AUC/ROC curve

Final Model Comparison

	Accuracy	AUC	Recall	Precision	F1 Score
LR Train	0.84	0.890	0.91	0.86	0.89
LR Test	0.83	0.879	0.88	0.86	0.87
LR Grid Train	0.83	0.890	0.91	0.86	0.88
LR Grid Test	0.83	0.883	0.88	0.86	0.87
LDA Train	0.83	0.889	0.91	0.86	0.89
LDA Test	0.83	0.888	0.89	0.86	0.88
LDA Grid Train	0.83	0.889	0.91	0.86	0.89
LDA Grid Test	0.83	0.888	0.89	0.86	0.88
NB Train	0.84	0.888	0.90	0.88	0.89
NB Test	0.82	0.876	0.87	0.87	0.87
NB Grid Train	0.84	0.887	0.90	0.88	0.89
NB Grid Test	0.82	0.879	0.88	0.86	0.87
KNN Train	0.85	0.928	0.91	0.88	0.90
KNN Test	0.82	0.867	0.88	0.86	0.87
KNN Grid Train	1.00	1.000	1.00	1.00	1.00
KNN Grid Test	0.83	0.889	0.91	0.84	0.88
KNN K=17 Train	0.84	0.905	0.90	0.87	0.89
KNN K=17 Test	0.84	0.888	0.92	0.85	0.88
RainForest Train	1.00	1.000	1.00	1.00	1.00
RainForest Test	0.83	0.891	0.90	0.85	0.88
RainForest Grid Train	0.85	0.909	0.93	0.87	0.90
RainForest Grid Test	0.83	0.890	0.91	0.84	0.88
Bagging Train	0.96	0.997	0.99	0.96	0.98
Bagging Test	0.83	0.897	0.91	0.85	0.88
Bagging Grid Train	0.91	0.980	0.96	0.92	0.94
Bagging Grid Test	0.83	0.898	0.91	0.85	0.88
AdaBoost Train	0.85	0.912	0.91	0.88	0.89
AdaBoost Test	0.81	0.881	0.88	0.85	0.86
AdaBoost Grid Train	0.79	0.875	0.96	0.80	0.87
AdaBoost Grid Test	0.78	0.859	0.96	0.76	0.85
Gradient-Boost Train	0.89	0.951	0.94	0.91	0.93
Gradient-Boost Test	0.84	0.899	0.91	0.85	0.88
Gradient-Boost Grid Train	0.88	0.933	0.93	0.90	0.91
Gradient-Boost Grid Test	0.84	0.894	0.91	0.85	0.88

Table 1.7 Final model comparison

After comparing all the model we can find that the performance of KNN with optimal value k=17 and Gradient Boosting GridSerachCV model perform in Train and test compared to rest other model. Both Recall and Precision can be equally important in this case. Difference between in all Train & test values of Accuracy, Recall, Precision & F1 score are less, which confirms that the model performance is good.

- The test data recall of KNN(K=17) & Gradient Boosting (with Gridsearch) is 91% i.e only 9% of the people who is in favour of labour party, automatically he or she will be voting against the labour party.
- The test data precision of KNN(K=17) & Gradient Boosting (with Gridsearch) is 90% i.e only 10% of the people were made false predictions that the votes to be in favour were actually predicted against the labour party.

1.8 Based on these predictions, what are the insights?

Answer:

Insights:

- Comparing all the models, KNN with K=17 model and Gradient Boosting tuned model is performing best. Both Train and test results are performing good for these models. Parameters such as Recall, Precision, F1-score and AUC/ROC are good in these models.
- Labour Party is performing better than Conservative from huge margin
- Female voters turn out is greater than male voters.
- Those who have better national economic conditions & household are performing to vote for labour party.
- Persons having higher Eurosceptic sentiments conservative party is preferring to vote for conservative party.
- The age increases people choose to elect conservative party
- Those who have higher political knowledge has voted for labour party
- Looking at the assessments for both leaders, Labour leader Blair is performing well as he got better rating.

Recommendations:

- Conservative party should target on more female voters for their next camping elections.
- Labour party should give more focus on people with higher Eurosceptic sentiments so that they will get more votes from those peoples.
- Conservative party should get attention of more middle age people to get more votes.
- Labour party should satisfy the needs of old aged people also so that they will prefer for labour party during election.

Problem 2

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

1. President Franklin D. Roosevelt in 1941
2. President John F. Kennedy in 1961
3. President Richard Nixon in 1973

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

Answer:

Number of Characters, Words and Sentences for document President Franklin D.Roosevelt in 1941:

Number of all words in text 1941-Roosevelt is : 1536

Number of all sentences in text 1941-Roosevelt is : 68

No of characters in document President 1941-Franklin D. Roosevelt(with space) :7571
No of characters in document President 1941-Franklin D. Roosevelt(without space) :6249

Number of Characters, Words and Sentences for document President John F.Kennedy i n 1961:
Number of all words in text 1961-Kennedy is :1546

Number of all sentences in text 1961-Kennedy is : 52

No of characters in document President 1961-John F. Kennedy (with space) : 7618
No of characters in document President 1961-John F. Kennedy (without space) : 6255

Number of Characters, Words and Sentences for document President Richard Nixon in 1973:

Number of all words in text 1973-Nixon is : 2028

Number of all sentences in text 1973-Nixon is : 69

No of characters in document President Richard Nixon in 1973(with space) : 9991
No of characters in document President Richard Nixon in 1973(without space) : 8223

2.2 Remove all the stopwords from all three speeches.

Answer:

For President Franklin D. Roosevelt in 1941:

Creating the speech into a dataframe

	speech
0	On each national day of inauguration since 178...
1	In Washington's day the task of the people was...
2	In Lincoln's day the task of the people was to...
3	In this day the task of the people is to save ...
4	To us there has come a time, in the midst of s...
5	Lives of nations are determined not by the cou...
6	There are men who doubt this. There are men wh...
7	But we Americans know that this is not true.
8	Eight years ago, when the life of this Republi...
9	These later years have been living years -- fr...

Table 2.2.1 Roosevelt dataframe

Done the Basic pre-processing such as

Lower-case conversion

```
0    on each national day of inauguration since 178...
1    in washington's day the task of the people was...
2    in lincoln's day the task of the people was to...
3    in this day the task of the people is to save ...
4    to us there has come a time, in the midst of s...
Name: speech, dtype: object
```

Removal of punctuation

```
0    on each national day of inauguration since 178...
1    in washingtons day the task of the people was ...
2    in lincolns day the task of the people was to ...
3    in this day the task of the people is to save ...
4    to us there has come a time in the midst of sw...
Name: speech, dtype: object
```

Stopword removal

```
0    national day inauguration since 1789 people re...
1    washingtons day task people create weld togeth...
2    lincolns day task people preserve nation disru...
3    day task people save nation institutions disru...
4    us come time midst swift happenings pause mome...
Name: speech, dtype: object
```

Word count before and after Stopwords:

	speech	word_count_before	word_count_after
0	national day inauguration since 1789 people re...	20	11
1	washingtons day task people create weld togeth...	16	8
2	lincolns day task people preserve nation disru...	17	8
3	day task people save nation institutions disru...	20	8
4	us come time midst swift happenings pause mome...	53	19
5	lives nations determined count years lifetime ...	51	21
6	men doubt men believe democracy form governmen...	59	26
7	americans know true	9	3
8	eight years ago life republic seemed frozen fa...	38	18
9	later years living years fruitful years people...	42	20

Table 2.2.2 Roosevelt -Wordcount before & after stopword removal

Total word count before and after Stopword removal:

```
word_count_before    1360
word_count_after      627
```

Sample sentence after the removal of stopwords

```
'national day inauguration since 1789 people renewed sense dedication u
nited states'
```

For President John F. Kennedy in 1961:

Creating the speech into a dataframe

	speech
0	Vice President Johnson, Mr. Speaker, Mr. Chief...
1	The world is very different now. For man holds...
2	We dare not forget today that we are the heirs...
3	Let every nation know, whether it wishes us we...
4	This much we pledge -- and more.
5	To those old allies whose cultural and spiritu...
6	To those new States whom we welcome to the ran...
7	To those peoples in the huts and villages acro...
8	To our sister republics south of our border, w...
9	To that world assembly of sovereign states, th...

Table 2.2.3 Kennedy Dataframe

Done the Basic pre-processing such as

Lower-case conversion

```
0    vice president johnson, mr. speaker, mr. chief...
1    the world is very different now. for man holds...
2    we dare not forget today that we are the heirs...
3    let every nation know, whether it wishes us we...
4                   this much we pledge -- and more.
Name: speech, dtype: object
```

Removal of Punctuation

```
0    vice president johnson mr speaker mr chief jus...
1    the world is very different now for man holds ...
2    we dare not forget today that we are the heirs...
3    let every nation know whether it wishes us wel...
4                        this much we pledge  and more
Name: speech, dtype: object
```

Stopword removal

```
0    vice president johnson mr speaker mr chief jus...
1    world different man holds mortal hands power a...
2    dare forget today heirs first revolution let w...
3    let every nation know whether wishes us well i...
4                                          much pledge
Name: speech, dtype: object
```

Word count before and after Stopwords:

	speech	word_count_before	word_count_after
0	vice president johnson mr speaker mr chief jus...	73	46
1	world different man holds mortal hands power a...	68	31
2	dare forget today heirs first revolution let w...	96	46
3	let every nation know whether wishes us well i...	40	25
4	much pledge	7	2
5	old allies whose cultural spiritual origins sh...	52	26
6	new states welcome ranks free pledge word one ...	85	42
7	peoples huts villages across globe struggling ...	73	32
8	sister republics south border offer special pl...	95	50
9	world assembly sovereign states united nations...	67	33

Table 2.2.4 Kennedy-Wordcount before & after removal

Total word count before and after Stopword removal:

```
word_count_before    1390
word_count_after      693
```

Sample sentence after the removal of stopwords

```
'vice president johnson mr speaker mr chief justice president eisenhowe
r vice president nixon president truman reverend clergy fellow citizens
observe today victory party celebration freedom symbolizing end well be
ginning signifying renewal well change sworn almighty god solemn oath f
orebears l prescribed nearly century three quarters ago'
```

For President Richard Nixon in 1973:

Creating the speech into a dataframe

	speech
0	Mr. Vice President, Mr. Speaker, Mr. Chief Jus...
1	When we met here four years ago, America was b...
2	As we meet here today, we stand on the thresho...
3	The central question before us is: How shall w...
4	Let us resolve that this will be what it can b...
5	This past year saw far-reaching results from o...
6	The peace we seek in the world is not the flim...
7	It is important that we understand both the ne...
8	Unless we in America work to preserve the peac...
9	Unless we in America work to preserve freedom,...

Table 2.2.4 Nixon Dataframe

Done the Basic pre-processing such as

Lower-case conversion

```
0    mr. vice president, mr. speaker, mr. chief jus...
1    when we met here four years ago, america was b...
2    as we meet here today, we stand on the thresho...
3    the central question before us is: how shall w...
4    let us resolve that this will be what it can b...
Name: speech, dtype: object
```

Removal of Punctuation

```
0    mr vice president mr speaker mr chief justice ...
1    when we met here four years ago america was bl...
2    as we meet here today we stand on the threshol...
3    the central question before us is how shall we...
4    let us resolve that this will be what it can b...
Name: speech, dtype: object
```

Stopword removal

```
0    mr vice president mr speaker mr chief justice ...
1    met four years ago america bleak spirit depres...
2       meet today stand threshold new era peace world
3    central question us shall use peace let us res...
4    let us resolve become time great responsibilit...
Name: speech, dtype: object
```

Word count before and after Stopwords:

	speech	word_count_before	word_count_after
0	mr vice president mr speaker mr chief justice ...	25	19
1	met four years ago america bleak spirit depres...	27	16
2	meet today stand threshold new era peace world	19	8
3	central question us shall use peace let us res...	51	24
4	let us resolve become time great responsibilit...	38	17
5	past year saw farreaching results new policies...	81	43
6	peace seek world flimsy peace merely interlude...	29	12
7	important understand necessity limitations ame...	19	8
8	unless america work preserve peace peace	14	6
9	unless america work preserve freedom freedom	13	6

Table 2.2.5 Nixon-Wordcount before & after stopword removal

Total word count before and after Stopword removal:

```
word_count_before    1819
word_count_after      833
```

Sample sentence after the removal of stopwords

```
'mr vice president mr speaker mr chief justice senator cook mrs eisenho
wer fellow citizens great good country share together'
```

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)

Answer:

Top 3 common words in speech

For President Franklin D. Roosevelt in 1941:

```
nation       11
know         10
democracy     9
spirit        9
dtype: int64
```

For President John F. Kennedy in 1961:

```
let      16
us       12
sides     8
world     8
dtype: int64
```

For President Richard Nixon in 1973:

```
us        26
let       22
peace     19
world     16
dtype: int64
```

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

Answer:

Word Cloud for President Roosevelt After Stop Word Removal

Figure 2.4.1 Roosevelt- Word Cloud

Word Cloud for President Kennedy After Stop Word Removal

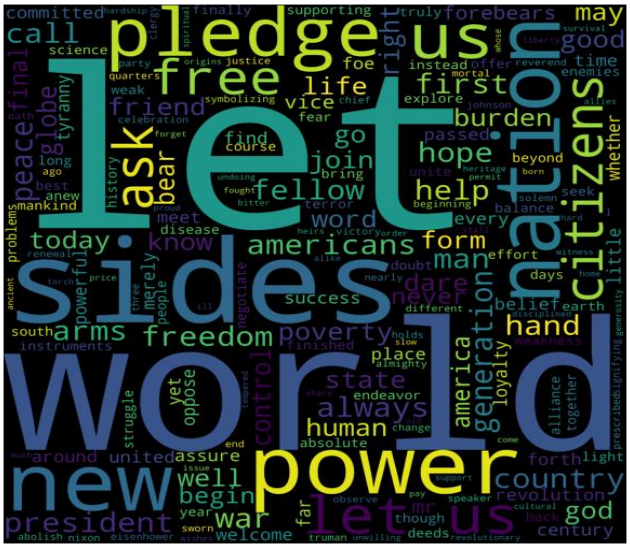


Figure 2.4.2 Kennedy- Word Cloud

Word Cloud for President Nixon After Stop Word Removal

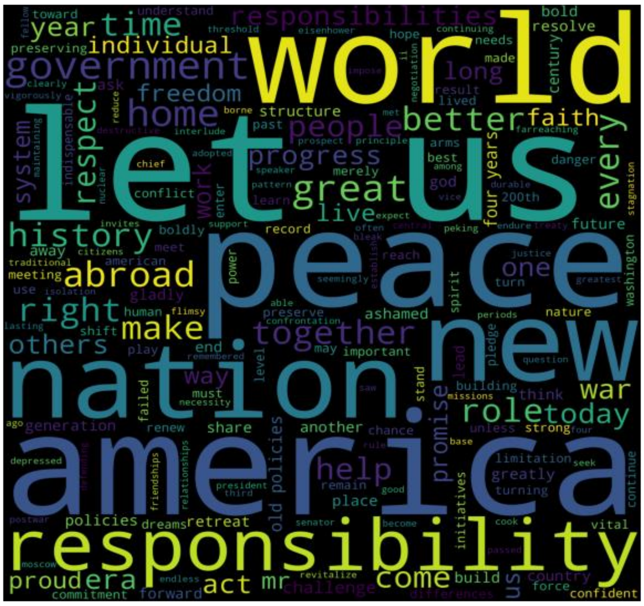


Figure 2.4.3 Nixon- Word Cloud

--END OF REPORT--