#Prepared By

Jitendra Kumar Nayak

Date:05/12/2021

#Business Report

Machine Learning

Table of Contents

List of Figures

List of Tables

Problem 1

Executive Summary

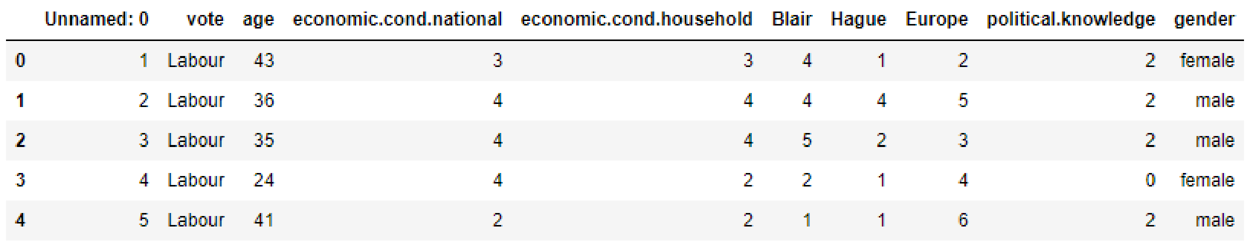
One of the leading news channels CNBE wants to analyze recent elections. This survey was conducted on 1525 voters with 9 variables. You have to build a model, to predict which party a voter will vote for on the basis of the given information, to create an exit poll that will help in predicting overall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

Introduction

The objective is to analyse the data, prepare it, and build various classification models to predict the voter's choice of party. Insights from the exploratory data analysis, and from the models, have been provided at the end.

1.1 Read the dataset. Do the descriptive statistics and do the null value condition check. Write an inference on it.
{Read the dataset. Describe the data briefly. Interpret the inferences for each. Initial steps like head() .info(), Data Types, etc . Null value check, Summary stats, Skewness must be discussed.}

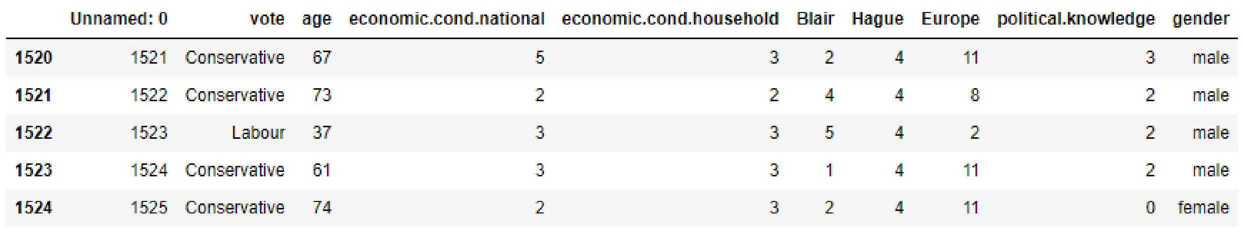
Let us have a look at first and last few records of data:



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

Table 1: Sample of Data(Problem 1) - 1

(above table generated with .head() function of PANDAS library in Python)



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

Table 2: Sample of Data(Problem 1) - 2

(above table generated with .tail() function of PANDAS library in Python)

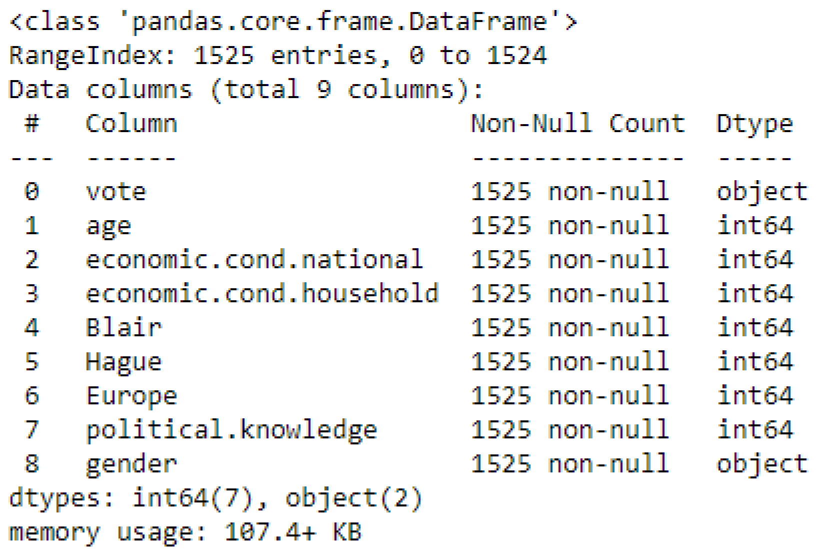
Let us drop the ‘Unnamed: 0’ feature which seems to contain just indices and proceed further.

Data Dictionary:

1. **vote**: Party choice: Conservative or Labour
2. **age**: in years
3. **economic.cond.national**: Assessment of current national economic conditions, 1 to 5.
4. **economic.cond.household**: Assessment of current household economic conditions, 1 to 5.
5. **Blair**: Assessment of the Labour leader, 1 to 5.
6. **Hague**: Assessment of the Conservative leader, 1 to 5.
7. **Europe**: an 11-point scale that measures respondents' attitudes toward European integration. High scores represent ‘Eurosceptic’ sentiment.
8. **political.knowledge**: Knowledge of parties' positions on European integration, 0 to 3.
9. **gender**: female or male.

Basic Exploratory Data Analysis

Let us check the data types of variables and the missing values(if any):



```
<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 3: General Information of Data(Problem1)

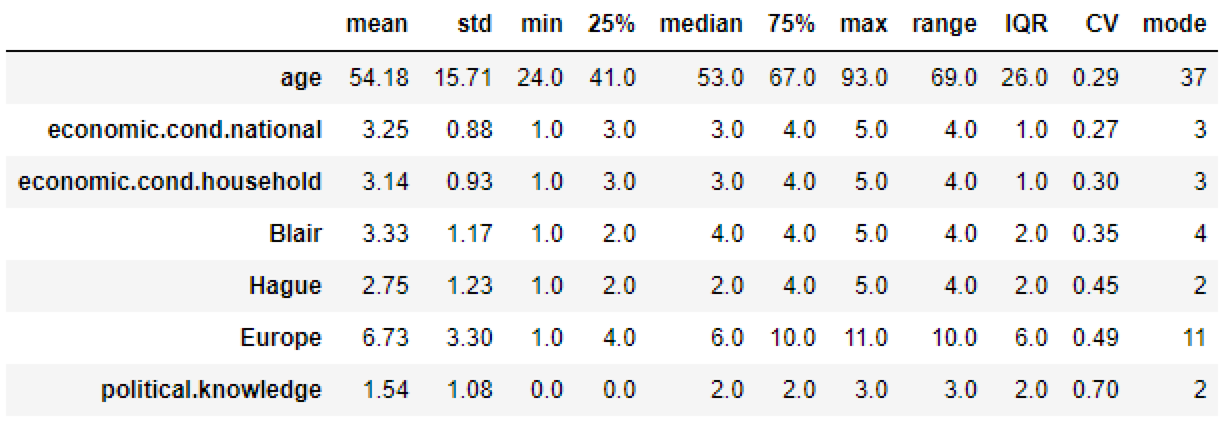
(above table generated by '.info' of 'PANDAS' in Python)

We can see that

- shape of data is (1525x9), after dropping 'Unnamed: 0' feature
- there are total 1525 entries(rows) of voters in data indexed from 0 to 1524
- there are 9 columns in total(vote, age, economic.cond.national, economic.cond.household, Blair, Hague, Europe, political.knowledge, gender)
- except for 'vote' and 'gender', which are of data-type object, all remaining variables are of integer data-type(64 bits)
- the memory usage by data is 107.4+KB
- there is no null value in data set as all columns have 1525 non null objects

'df.isnull().sum()' also confirms the 0 null value count in the data set.

Let us now move to the general description of numeric data(minimum values, maximum values, measures of central tendencies like mean, median, mode etc.):



	mean	std	min	25%	median	75%	max	range	IQR	CV	mode
age	54.18	15.71	24.0	41.0	53.0	67.0	93.0	69.0	26.0	0.29	37
economic.cond.national	3.25	0.88	1.0	3.0	3.0	4.0	5.0	4.0	1.0	0.27	3
economic.cond.household	3.14	0.93	1.0	3.0	3.0	4.0	5.0	4.0	1.0	0.30	3
Blair	3.33	1.17	1.0	2.0	4.0	4.0	5.0	4.0	2.0	0.35	4
Hague	2.75	1.23	1.0	2.0	2.0	4.0	5.0	4.0	2.0	0.45	2
Europe	6.73	3.30	1.0	4.0	6.0	10.0	11.0	10.0	6.0	0.49	11
political.knowledge	1.54	1.08	0.0	0.0	2.0	2.0	3.0	3.0	2.0	0.70	2

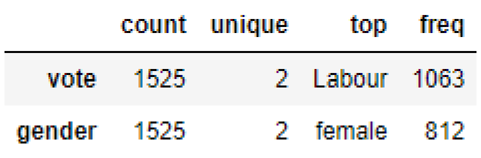
Table 4: General Description of Numeric Data(Problem 1)

(above table generated with .describe() function of PANDAS library in Python)

From above table, we conclude:

- '75%' and 'max' values in the feature 'age' are distant apart, indicating the possible presence of outliers
- rest all values seem to be correct

Let us now move to the general description of 'object' data:



	count	unique	top	freq
vote	1525	2	Labour	1063
gender	1525	2	female	812

Table 5: General Description of 'Object' type data(Problem 1)

(above table generated with .describe() function of PANDAS library in Python)

- The object data seems to be fine.
- Of 1525 total count, 'Labour' class in feature 'vote' has a major frequency of 1063(imbalanced class)
- Of 1525 total count, 'female' class in feature 'gender' has a major frequency of 812(nearly balanced class)

Duplicate Entries:

- There are 8 duplicate entries that have been identified by 'df.duplicated()'.
- These duplicates need to be dropped because they do not add any value to the study, be it associated with different people.
- The shape of data is (1517x9), after dropping the 8 duplicate entries(using '.shape' of PANDAS library in Python).

(Heatmap and Pairplot provided after Univariate Analysis)

Skewness can be observed from:

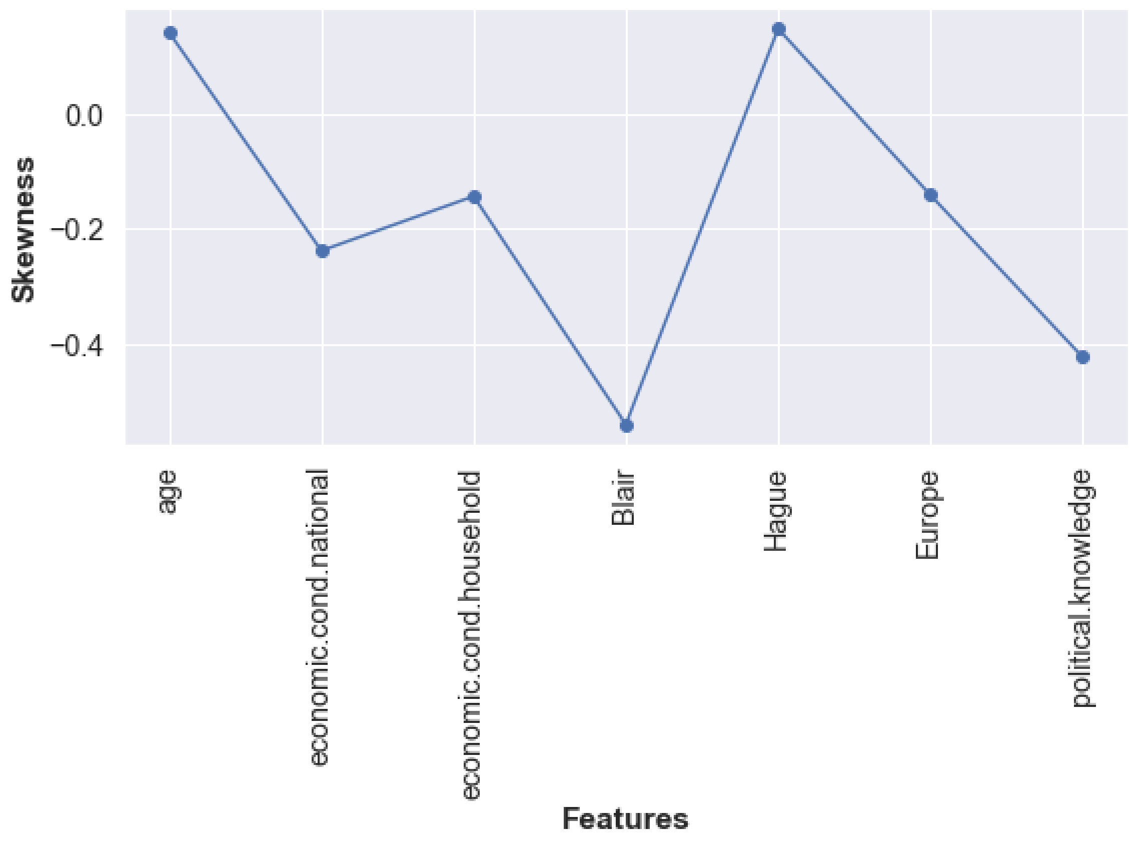
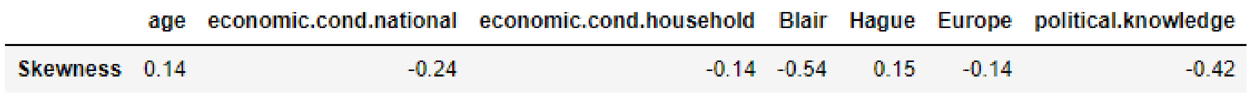


Fig. 1: Skewness of Features

(plot generated using 'skew()' function of 'stats' module of 'scipy' library and 'PANDAS' library of 'PYTHON')



	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge
Skewness	0.14	-0.24	-0.14	-0.54	0.15	-0.14	-0.42

Table 9: Skewness of Features

(table generated using 'skew()' function of 'stats' module of 'scipy' library and 'PANDAS' library of 'PYTHON')

Looking at the above two skewness plots, we can conclude that

- 'age' and 'Hague' are positively skewed or right skewed, that is more weight is on right tail of distribution indicating presence of outliers (if any) on the right side
- due to positive skewness, 'age' and 'Hague': MODE<MEDIAN<MEAN

- features other than ‘age’ and ‘Hague’ are negatively skewed or left skewed, that is more weight is on left tail of distribution indicating presence of outliers (if any) on the left side
- due to negative skewness, for these features: MODE>MEDIAN>MEAN

It should be kept in mind that only ‘age’ is a continuous variable, rest of the numeric features are actually ordinal features.

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

{Perform EDA (Check the null values, Data types, shape, Univariate, bivariate analysis). Also check for outliers (4 pts). Interpret the inferences for each (3 pts) Distribution plots(histogram) or similar plots for the continuous columns. Box plots, Correlation plots. Appropriate plots for categorical variables. Inferences on each plot. Outliers proportion should be discussed, and inferences from above used plots should be there. There is no restriction on how the learner wishes to implement this but the code should be able to represent the correct output and inferences should be logical and correct.}

- ‘df.isnull().sum()’ confirms the 0 null value count in the data set, as earlier.
- Following are the data types of features obtained by ‘.dtypes()’ of PANDAS library in Python:

vote	object
age	int64
economic.cond.national	int64
economic.cond.household	int64
Blair	int64
Hague	int64
Europe	int64
political.knowledge	int64
gender	object

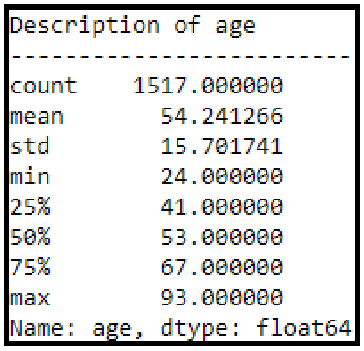
- The shape of data is (1517x9), after dropping the 8 duplicate entries(using ‘.shape’ of PANDAS library in Python)

Let us now go for the Univariate analysis of numeric features in the data set. We will go for general descriptions, distribution plots with kernel density estimates and boxplots for all the features one by one:

(It should be kept in mind that only 'age' is a continuous variable, rest of the numeric features are actually ordinal features)

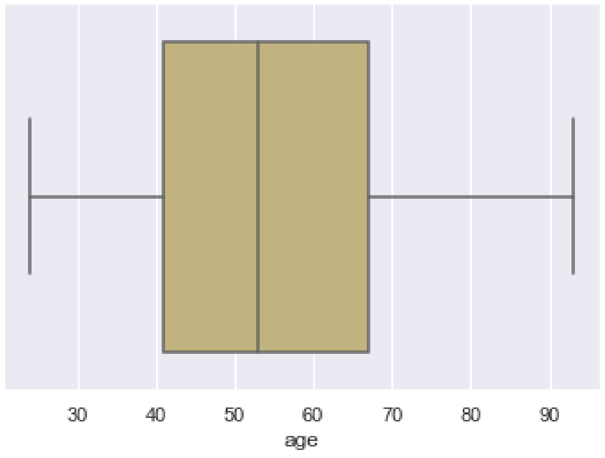
(zoom to at least 125% for better visibility)

<u>**age**</u>



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

```
BoxPlot of age
------------------------------
```



```
Distribution of age
-------------------------------------
```

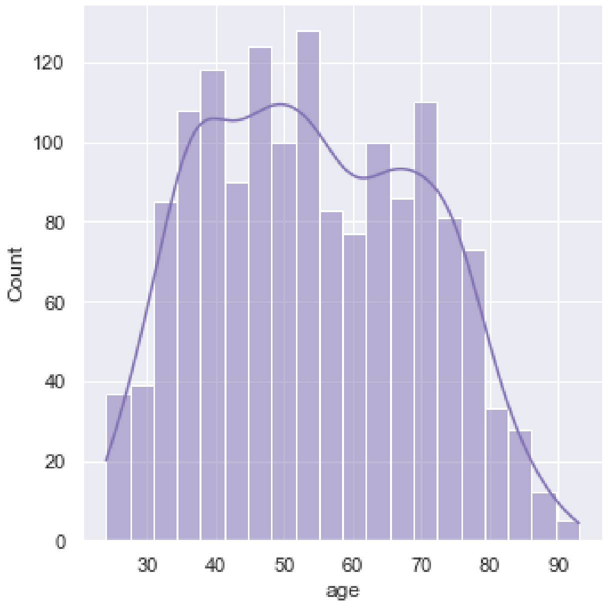
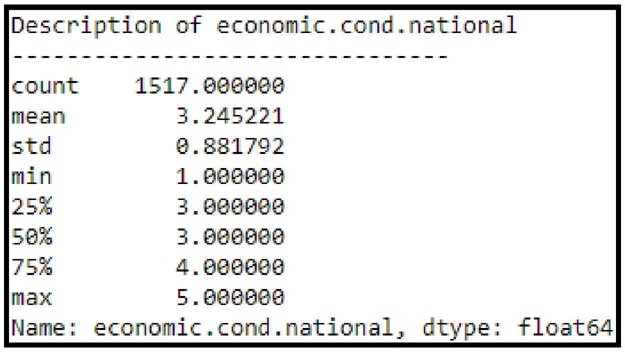


Figure 2: Univariate Analysis of 'age'

(above description generated with '.describe()' function of PANDAS library and plots generated using '.displot()' and '.boxplot()' functions of Seaborn library using Python)

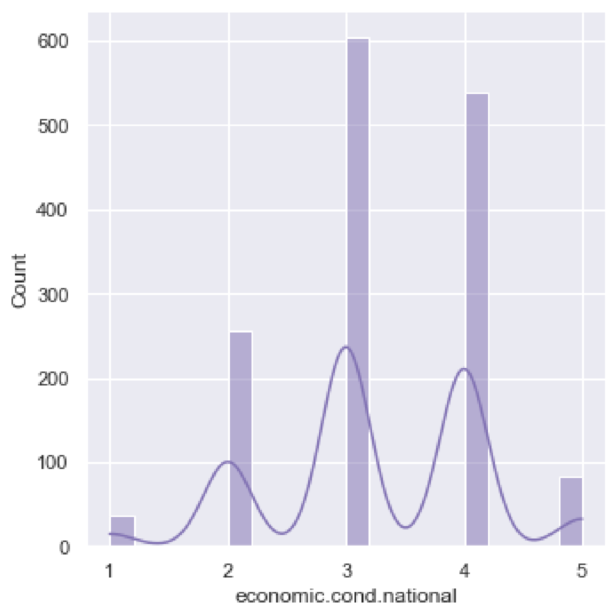
- voters are in the age group 24-93
- 75% of voters are within 67 years of age
- the mean and median ages are very close
- the distribution doesn't resemble a bell curve, hence, not normal
- earlier we saw, 'age' had a skewness of 0.14, but, there are no outliers as we see here
- for right/positive skewness: MEAN>MEDIAN>MODE

<u>**economic.cond.national**</u>



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

```
Distribution of economic.cond.national
-------------------------------------
```



```
BoxPlot of economic.cond.national
------------------------------
```

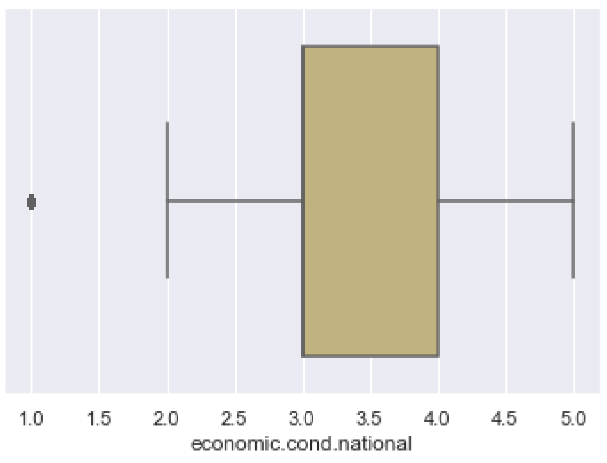
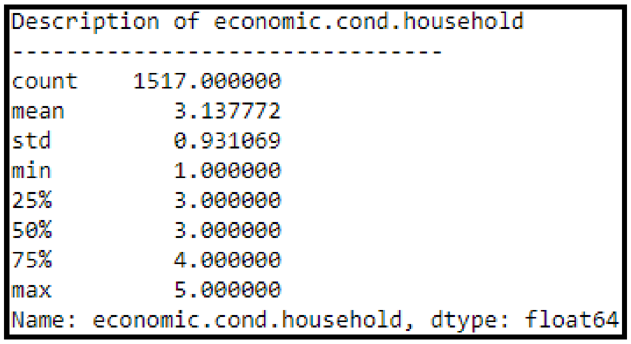


Figure 3: Univariate Analysis of ‘economic.cond.national’

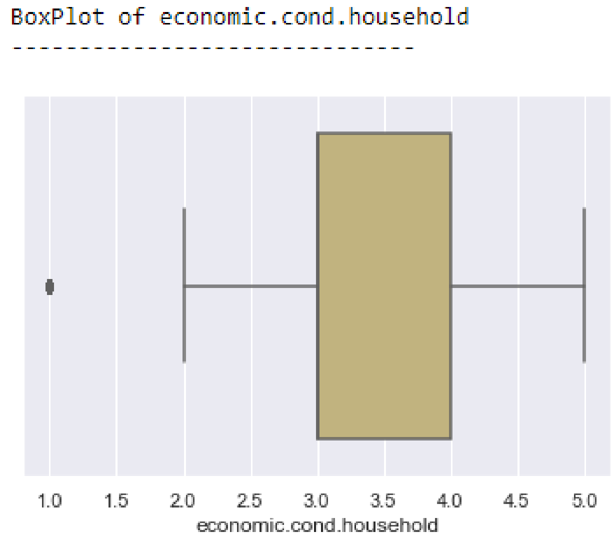
(above description generated with ‘.describe()’ function of PANDAS library and plots generated using ‘.displot()’ and ‘.boxplot()’ functions of Seaborn library using Python)

- (‘economic.cond.national’ is the assessment of current national economic conditions, 1 to 5)
- most of the ratings are ‘3’ or ‘4’(‘3’ being the most frequent)
- the ratings ‘1’ and ‘5’ are rarest ones(‘1’ being the least frequent)
- the mean is more than the median
- from histogram, we confirm the feature to contain categorical information actually, and not continuous numbers
- the distribution doesn't resemble a bell curve, hence, not normal
- earlier we saw, ‘economic.cond.national’ had a skewness of -0.24, and there are outliers on the left side of the boxplot too
- generally, for left/negative skewness: MODE>MEDIAN>MEAN, but here, the mean is more than the median

economic.cond.household



```
Description of economic.cond.household
--------------------------------
count    1517.000000
mean        3.137772
std         0.931069
min         1.000000
25%         3.000000
50%         3.000000
75%         4.000000
max         5.000000
Name: economic.cond.household, dtype: float64
```



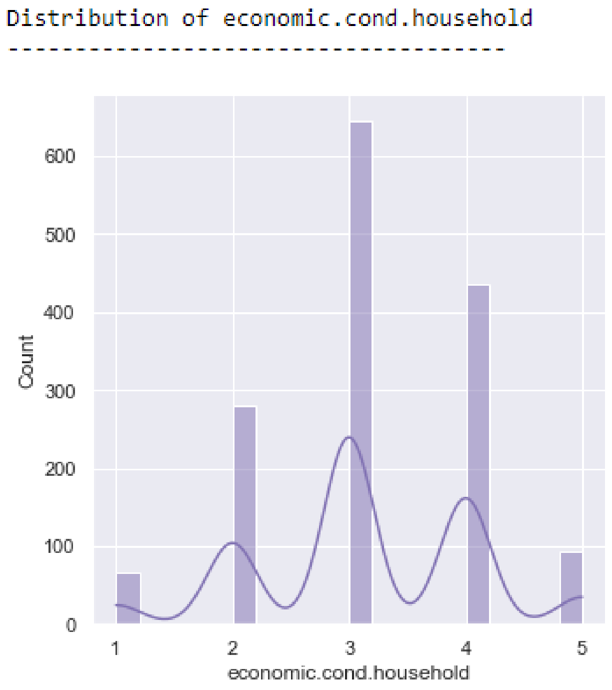
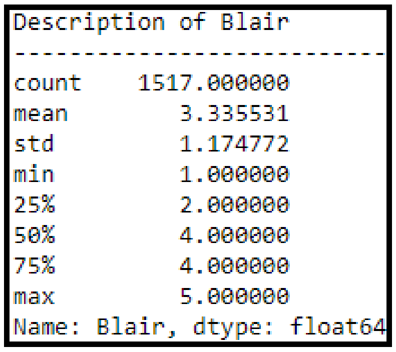


Figure 4: Univariate Analysis of 'economic.cond.household'

(above description generated with '.describe()' function of PANDAS library and plots generated using '.displot()' and '.boxplot()' functions of Seaborn library using Python)

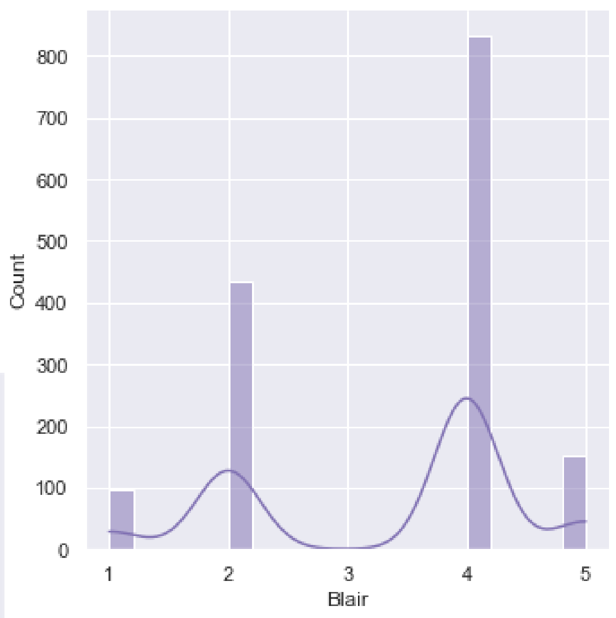
- 'economic.cond.household' is the assessment of current household economic conditions, 1 to 5.
- most of the ratings are '3' or '4'('3' being the most frequent)
- the ratings '1' and '5' are rarest ones('1' being the least frequent)
- the mean is more than the median
- from histogram, we can see the feature to contain categorical information actually, and not continuous numbers
- the distribution doesn't resemble a bell curve, hence, not normal
- earlier we saw, 'economic.cond.household' had a skewness of -0.14, and there are outliers on the left side of the boxplot too
- generally, for left/negative skewness: MODE>MEDIAN>MEAN, but here, the mean is more than the median

<u>Blair</u>



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

```
Distribution of Blair
--------------------------------------
```



```
BoxPlot of Blair
-------------------------------
```

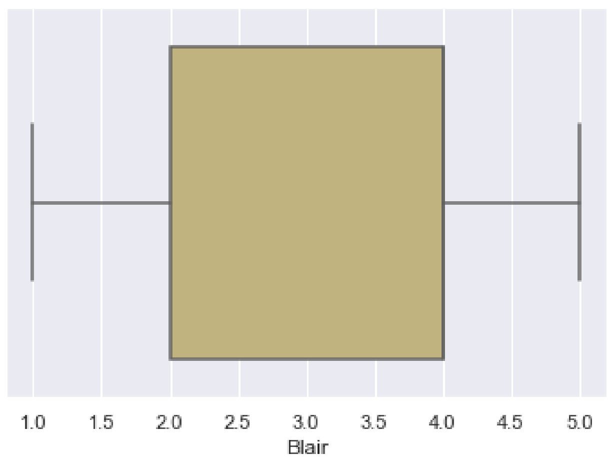
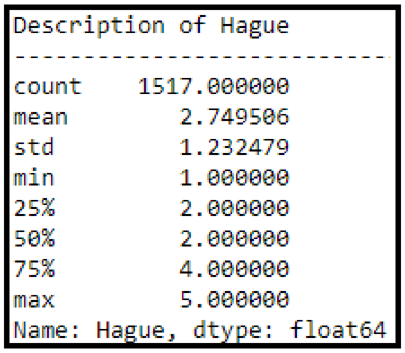


Figure 5: Univariate Analysis of 'Blair'

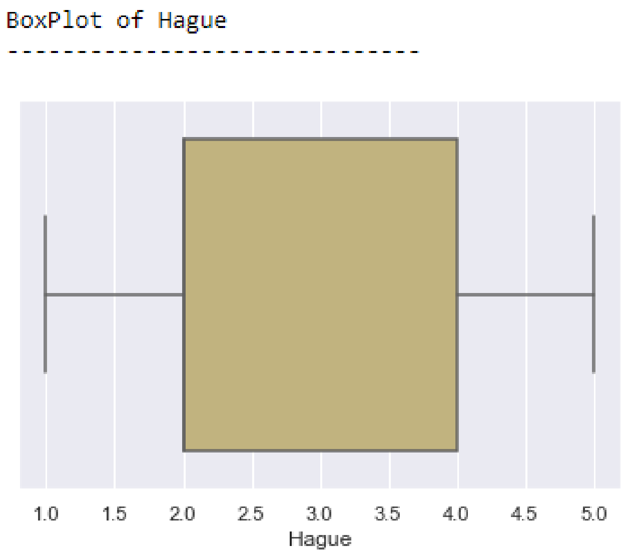
(above description generated with '.describe()' function of PANDAS library and plots generated using '.displot()' and '.boxplot()' functions of Seaborn library using Python)

- (Blair is the assessment of the Labour leader, 1 to 5.)
- '4' is the most frequent rating, indicating voters' affinity to Labour party's leader(s)
- '3' is the least frequent rating
- the mean is lesser than median rating
- from histogram, we can see the feature to contain categorical information actually, and not continuous numbers
- the distribution doesn't resemble a bell curve, hence, not normal
- earlier we saw, 'Blair' had a skewness of -0.54, but there are no outliers
- generally, for left/negative skewness: MODE>MEDIAN>MEAN

<u>Hague</u>



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



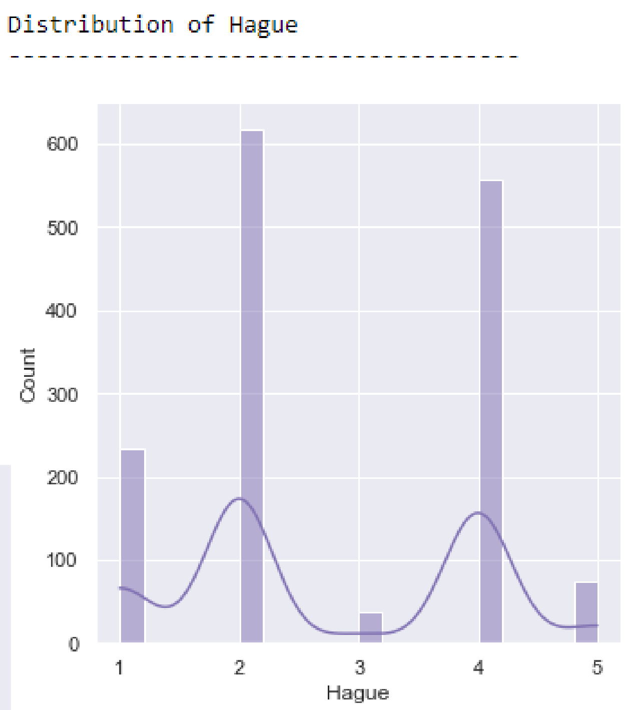
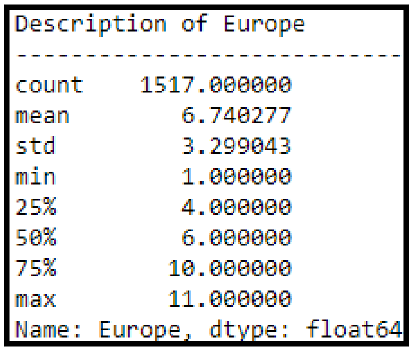


Figure 6: Univariate Analysis of 'Hague'

(above description generated with '.describe()' function of PANDAS library and plots generated using '.displot()' and '.boxplot()' functions of Seaborn library using Python)

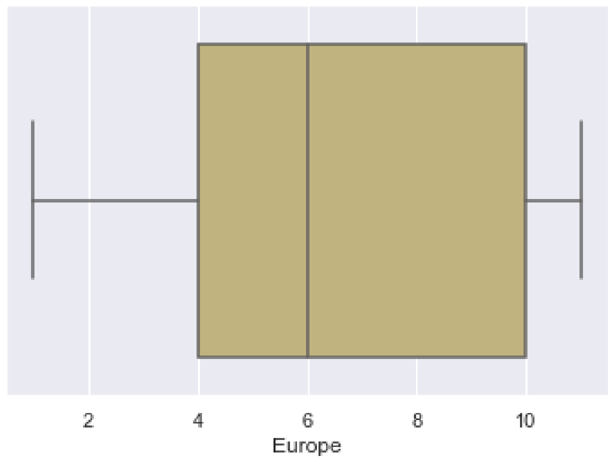
- ('Hague' is the assessment of the Conservative leader, 1 to 5.)
- '2' is the most frequent rating, indicating voters' antipathy to Conservative party's leader(s), though we have significant number of '4' ratings too
- '3' is the least frequent rating
- the mean is more than median rating
- from histogram, we can see the feature to contain categorical information actually, and not continuous numbers
- the distribution doesn't resemble a bell curve, hence, not normal
- earlier we saw, 'Hague' had a skewness of 0.15, but there are no outliers
- generally, for right/positive skewness: MEAN>MEDIAN>MODE

<u>Europe</u>



```
Description of Europe
-----------------------------
count    1517.000000
mean        6.740277
std         3.299043
min         1.000000
25%         4.000000
50%         6.000000
75%        10.000000
max        11.000000
Name: Europe, dtype: float64
```

```
BoxPlot of Europe
------------------------------
```



```
Distribution of Europe
-------------------------------------
```

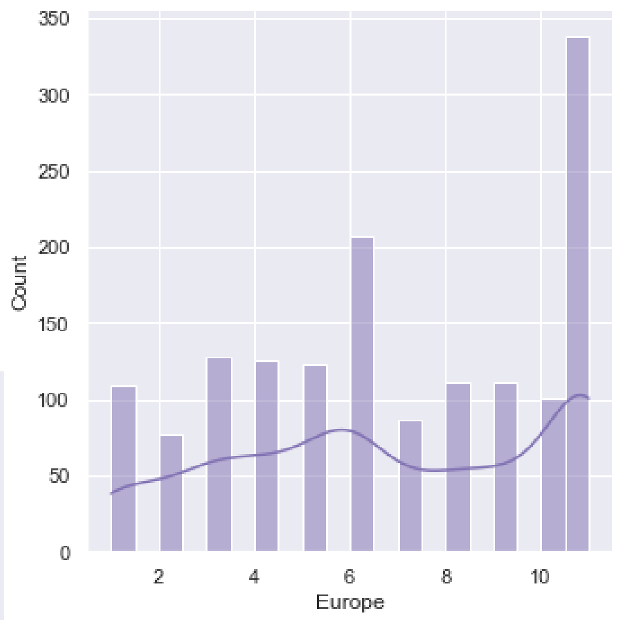
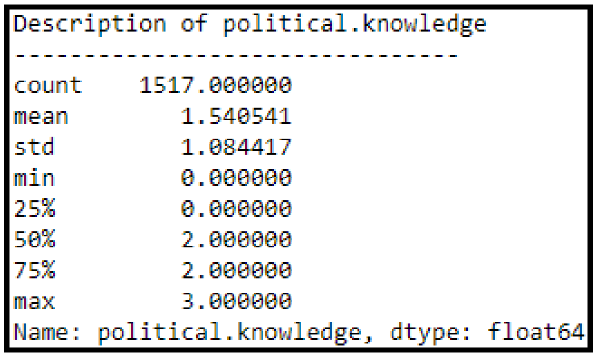


Figure 7: Univariate Analysis of 'Europe'

(above description generated with '.describe()' function of PANDAS library and plots generated using '.displot()' and '.boxplot()' functions of Seaborn library using Python)

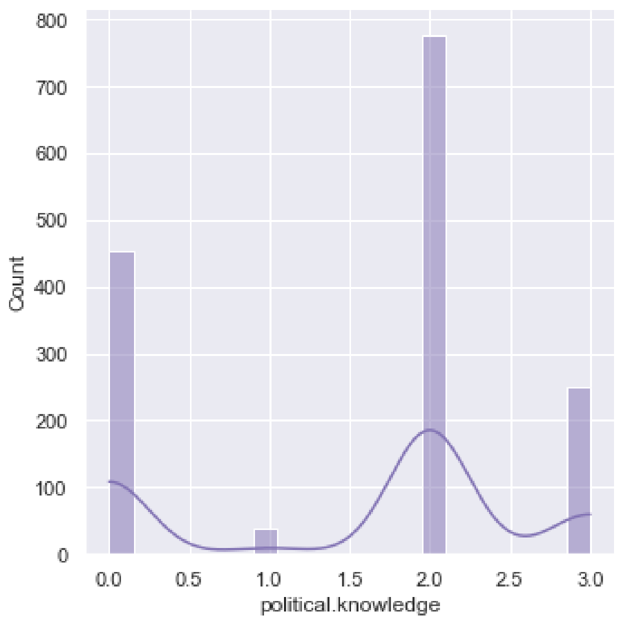
- (Europe is an 11-point scale that measures respondents' attitudes toward European integration. High scores represent 'Eurosceptic' sentiment.)
- '11' is the most frequent rating, indicating voters' 'Eurosceptic' sentiment
- the mean is more than median rating
- from histogram, we can see the feature to contain categorical information actually, and not continuous numbers
- the distribution doesn't resemble a bell curve, hence, not normal
- earlier we saw, 'Europe' had a skewness of -0.14, but there are no outliers
- generally, for left/negative skewness: MODE>MEDIAN>MEAN, but here mean is more than median

<u>political.knowledge</u>



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

```
Distribution of political.knowledge
-------------------------------------
```



```
BoxPlot of political.knowledge
------------------------------
```

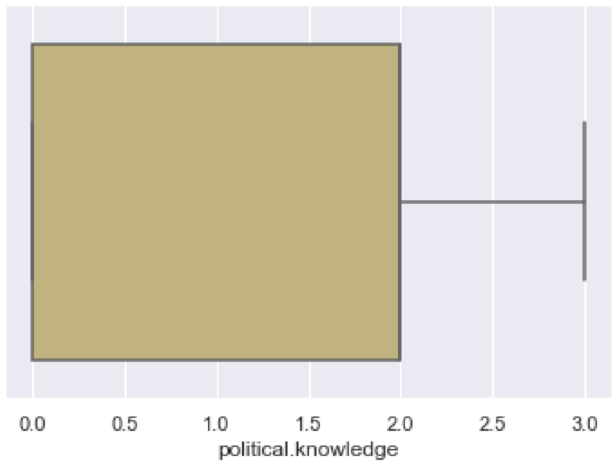


Figure 8: Univariate Analysis of 'political.knowledge'

(above description generated with '.describe()' function of PANDAS library and plots generated using '.displot()' and '.boxplot()' functions of Seaborn library using Python)

- ('political.knowledge' is knowledge of parties' positions on European integration, 0 to 3.)
- '2.0' is the most frequent rating, indicating voters' sound knowledge of parties' positions on European integration
- '1.0' is the least frequent rating
- there is good bunch of '0.0' ratings as well, indicating some voters' absence of knowledge of parties' positions on European integration
- the mean is lesser than median rating
- from histogram, we can see the feature to contain categorical information actually, and not continuous numbers
- the distribution doesn't resemble a bell curve, hence, not normal
- earlier we saw, 'political.knowledge' had a skewness of -0.42, but there are no outliers
- generally, for left/negative skewness: MODE>MEDIAN>MEAN

Let us check the number of outliers in each feature by writing a piece of code using '.quantile()' function of PANDAS library in Python:

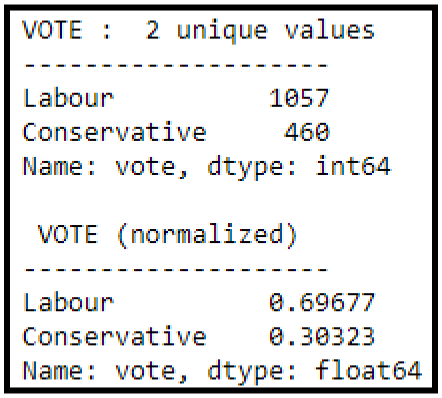
Blair	0
Europe	0
Hague	0
age	0
economic.cond.household	65
economic.cond.national	37
gender	0
political.knowledge	0
vote	0

As we saw in the boxplots, only ‘economic.cond.household’ and ‘economic.cond.national’ have outliers in them.

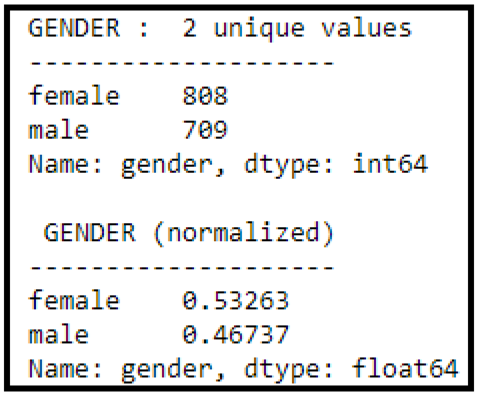
There are 1517 rows after dropping the 8 duplicate entries, so the outlier proportion can be calculated as:

- ‘economic.cond.household’: (no. of outliers)/(no. of entries) = 65/1517 = 0.043
- ‘economic.cond.national’: (no. of outliers)/(no. of entries) = 37/1517 = 0.024

Let us now go for the Univariate analysis of ‘object’ data features in the data set. We will check the unique values, count of unique values and proportion(i.e, normalized values) of unique values in each feature:



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

 VOTE (normalized)
--------------------
Labour           0.69677
Conservative     0.30323
Name: vote, dtype: float64
```



```
GENDER :  2 unique values
--------------------
female    808
male      709
Name: gender, dtype: int64

 GENDER (normalized)
--------------------
female    0.53263
male      0.46737
Name: gender, dtype: float64
```

Tables 7: Univariate Analysis of Categorical Data(Problem 1)

(above tables generated with ‘.value_counts()’ function of PANDAS library using Python)

Before drawing inferences, let us visualise above results in form of countplots:

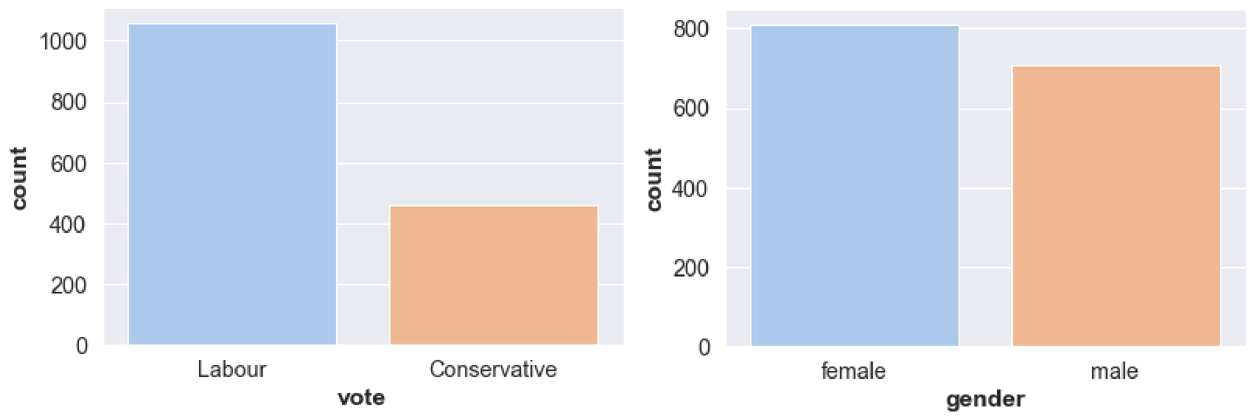


Figure 9: Count plots of Categorical Data(Problem 1)

(above plots generated using '.countplot()' function of Seaborn library in Python)

We can see:

- from the description, all the classes in the features seem to be correct without any anomalies
- feature 'vote' has a high class imbalance of 'Labour':'Conservative' to be roughly around 7:3
- feature 'gender' has a relatively lower class imbalance of 'female':'male' to be roughly around 53:47

Let us go for bivariate and multivariate analysis now.

To check the correlations amongst numerical features, let us have a look at the Pearson's Correlation coefficients in the form of a heatmap:

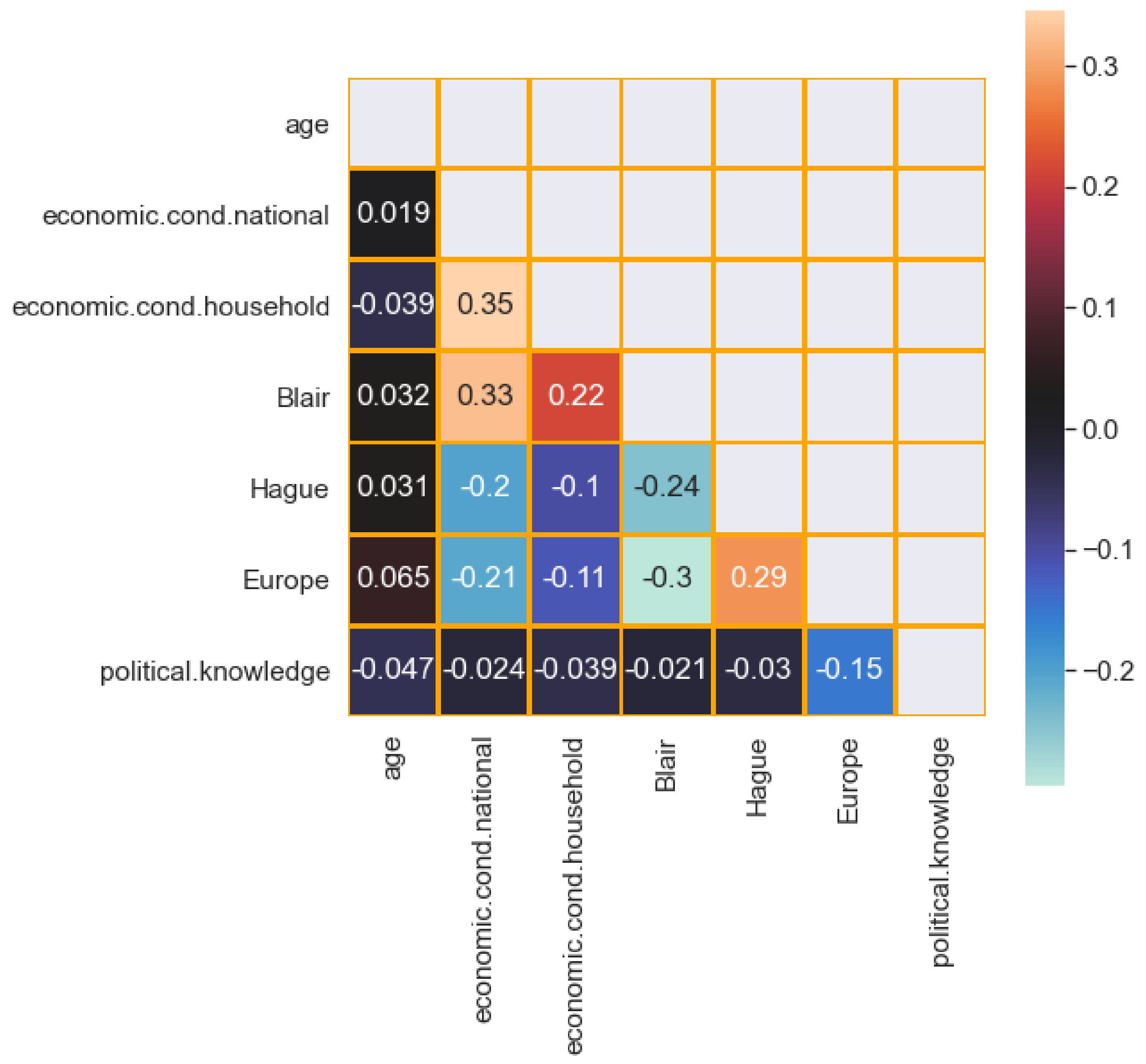


Figure 10: Correlation Heatmap(Problem 1)

(above plot generated using '.heatmap()' function of Seaborn library in Python)

- Pearson's Correlation coefficients near to 1 or -1 are highly positively correlated and highly negatively correlated respectively. Correlation values near to 0 are not or minimally correlated to each other.
- there are not very strong correlations amongst the features
- the highest positive correlation in data is 0.35, which is amongst 'economic.cond.household' and 'economic.cond.national'
- the highest negative correlation in data is -0.24, which is amongst 'Hague' and 'Blair'

Now, let us view the pairplot of the numeric features to confirm our observations of correlation heatmap:

(zoom to at least 150% for better visibility)

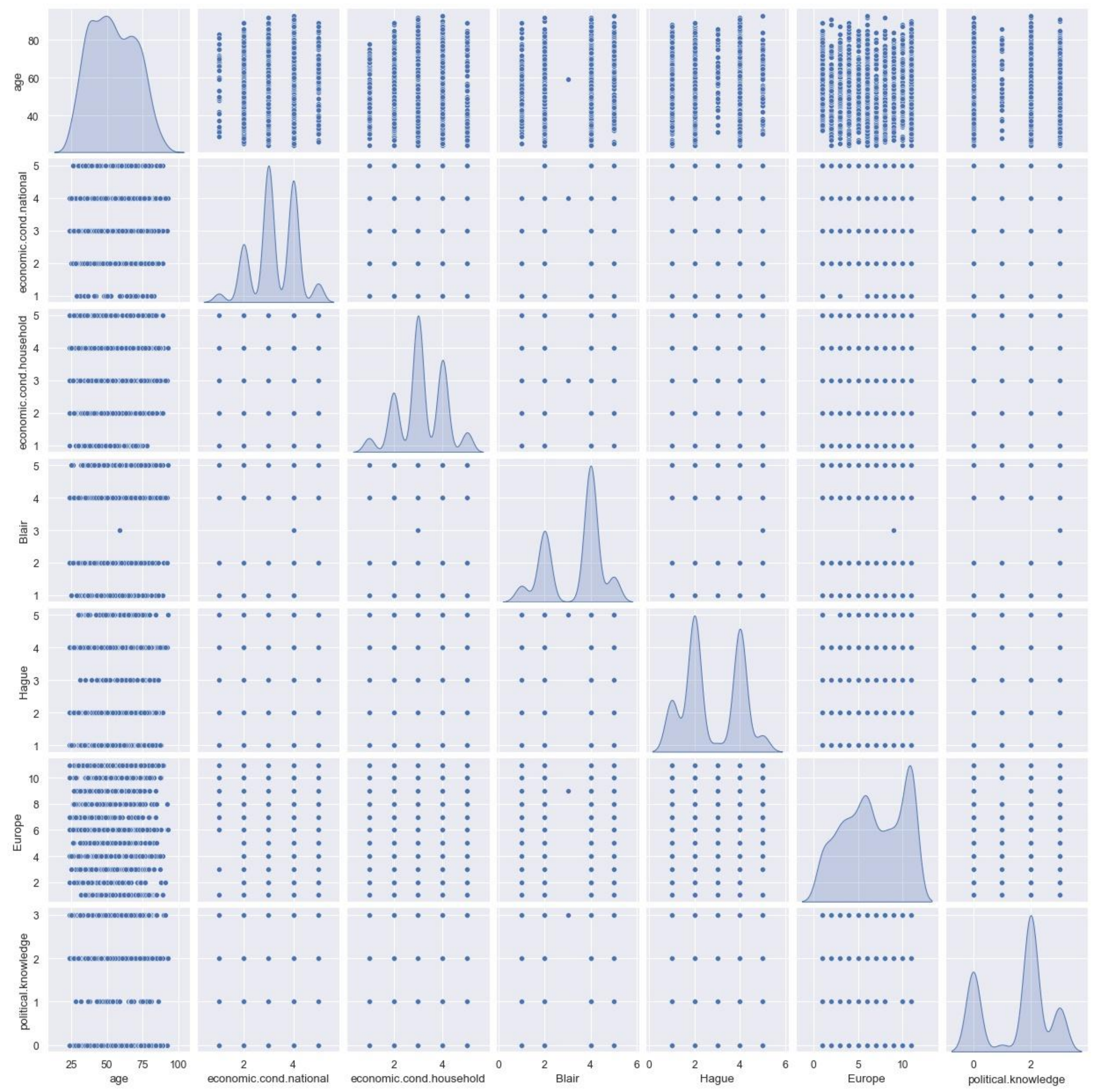


Figure 11: Pairplot(Problem 1)

(above figure generated with '.pairplot()' function of Seaborn library in Python)

Pairplot shows the relationship between the numeric variables in the form of scatterplot and the distribution of the variables in the form of kernel density estimates.

The pattern of clouds of points confirm the weak relationships amongst the features, as observed from the correlation heatmap.

Let us visualise how the target feature 'vote' is distributed over other categorical variable, 'gender':

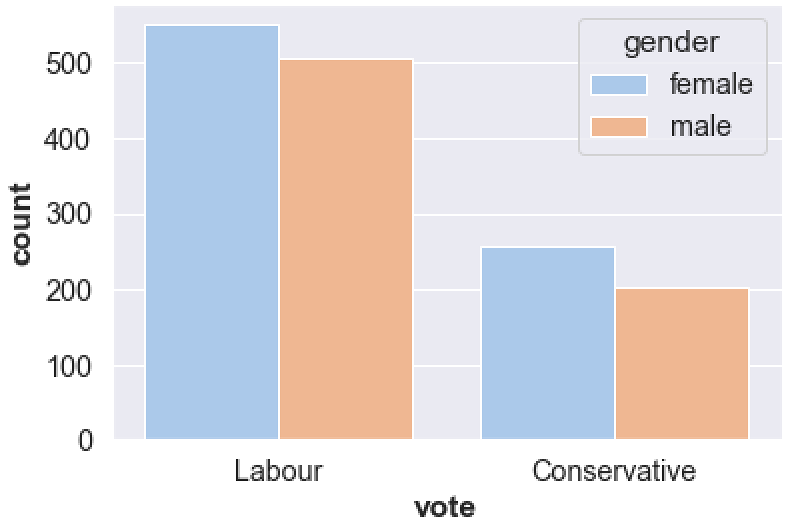


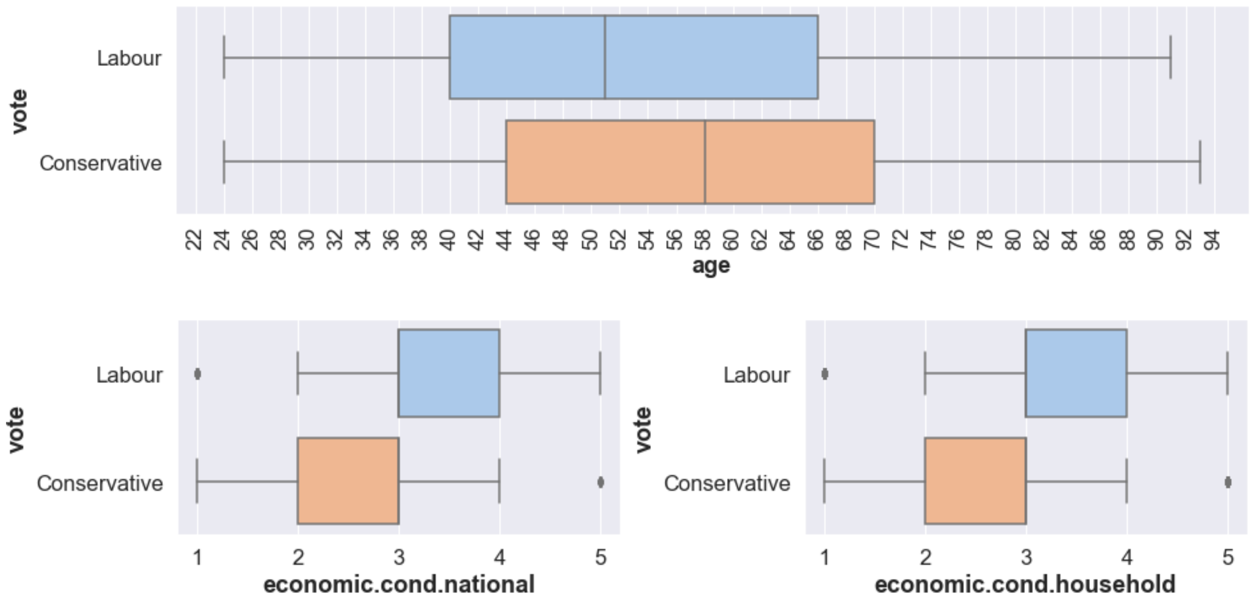
Figure 12: Distribution of Target over 'foreign'(Problem 2)

(above figure generated with '.countplot()' function of Seaborn library in Python)

We can see:

- for both the parties, number of female voters is slightly higher than male voters

Let us now check how the target 'vote' is distributed over the numeric features of the data:



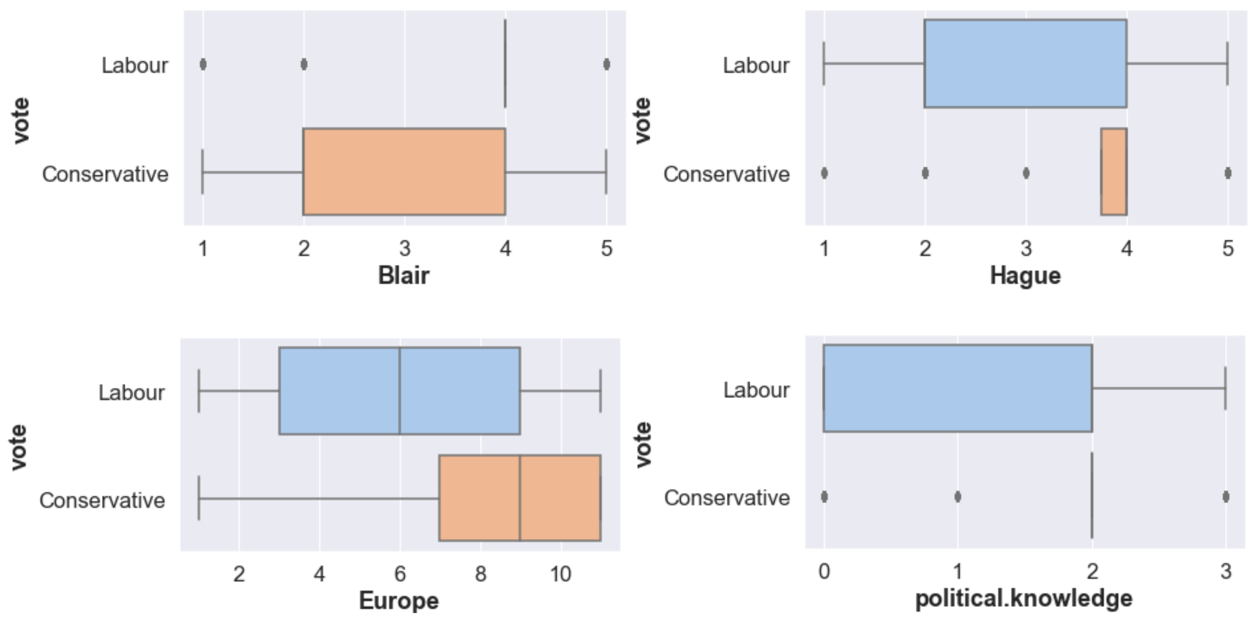


Figure 13: Distribution of Target over Numerical features(Problem 2)

(above figure generated with '.boxplot()' function of Seaborn library in Python)

We can see:

- the Labour party voters are marginally younger
- the current national and household economic conditions of Labour party voters is better
- the voters of Conservative party have more Eurosceptic sentiment
- the voters of Conservative party have sound political knowledge of parties' positions on European integration, while the voters of the Labour party seem to have varied knowledge.

Below we can see the target feature 'vote' versus feature 'age' hued over feature 'gender':

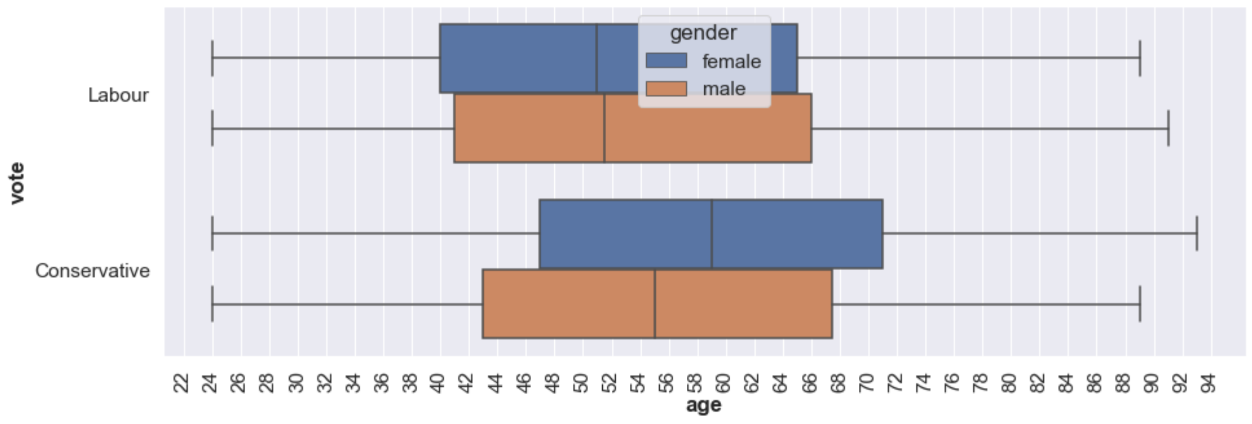


Figure 14: Distribution of 'vote' versus 'age' hued over 'gender'(Problem 2)

(above figure generated with '.boxplot()' function of Seaborn library in Python)

We can see:

- the female voters of Labour party to be marginally younger
- the male voters of both are nearly of the same age

Now, let us view the pairplot of the numeric features hued over target feature 'vote':

(zoom to at least 150% for better visibility)

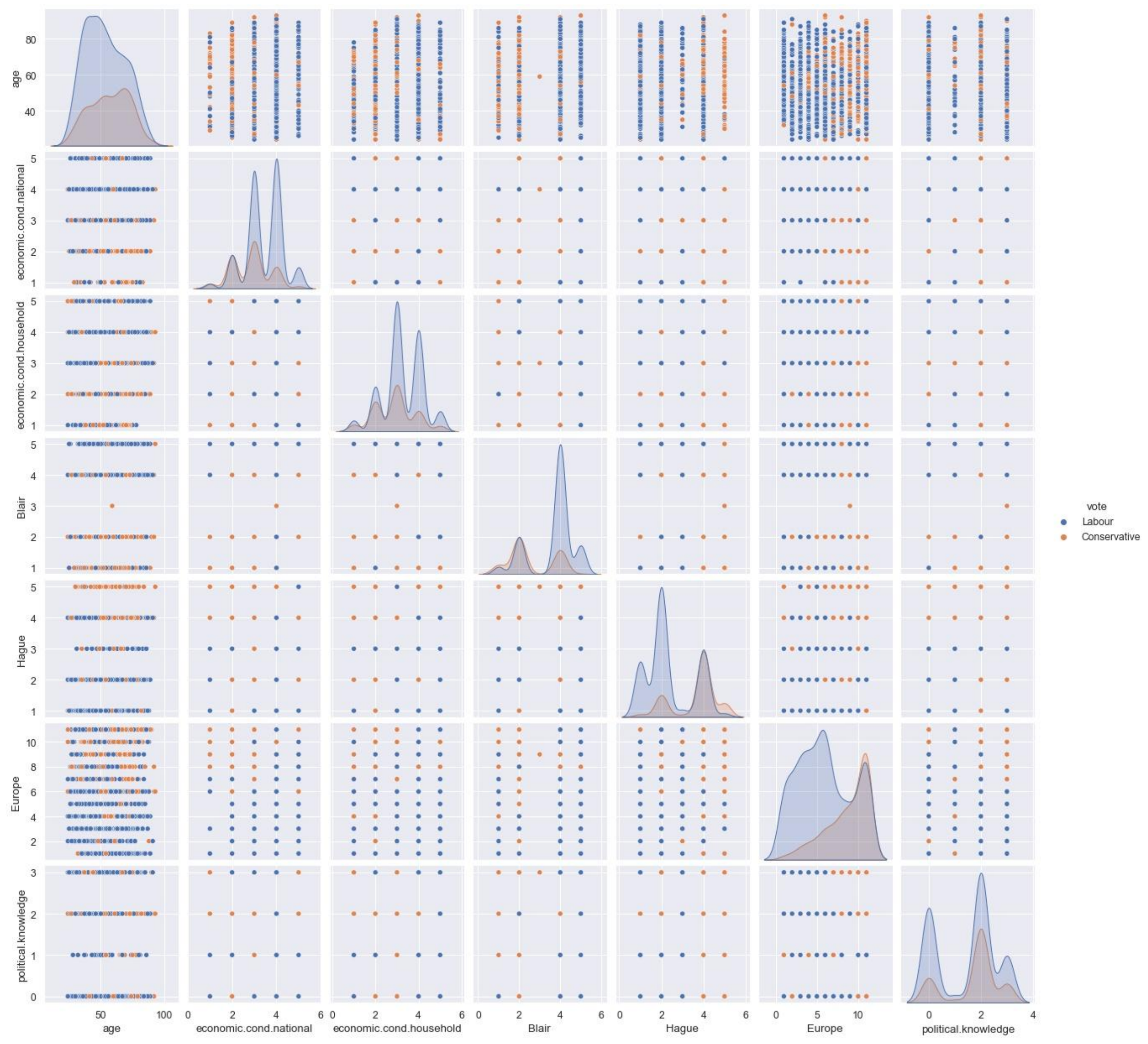


Figure 15: Pairplot hued over feature 'vote'

(above figure generated with '.pairplot()' function of Seaborn library in Python)

Now, let us view the pairplot of the numeric features hued over the feature 'gender':

(zoom to at least 150% for better visibility)

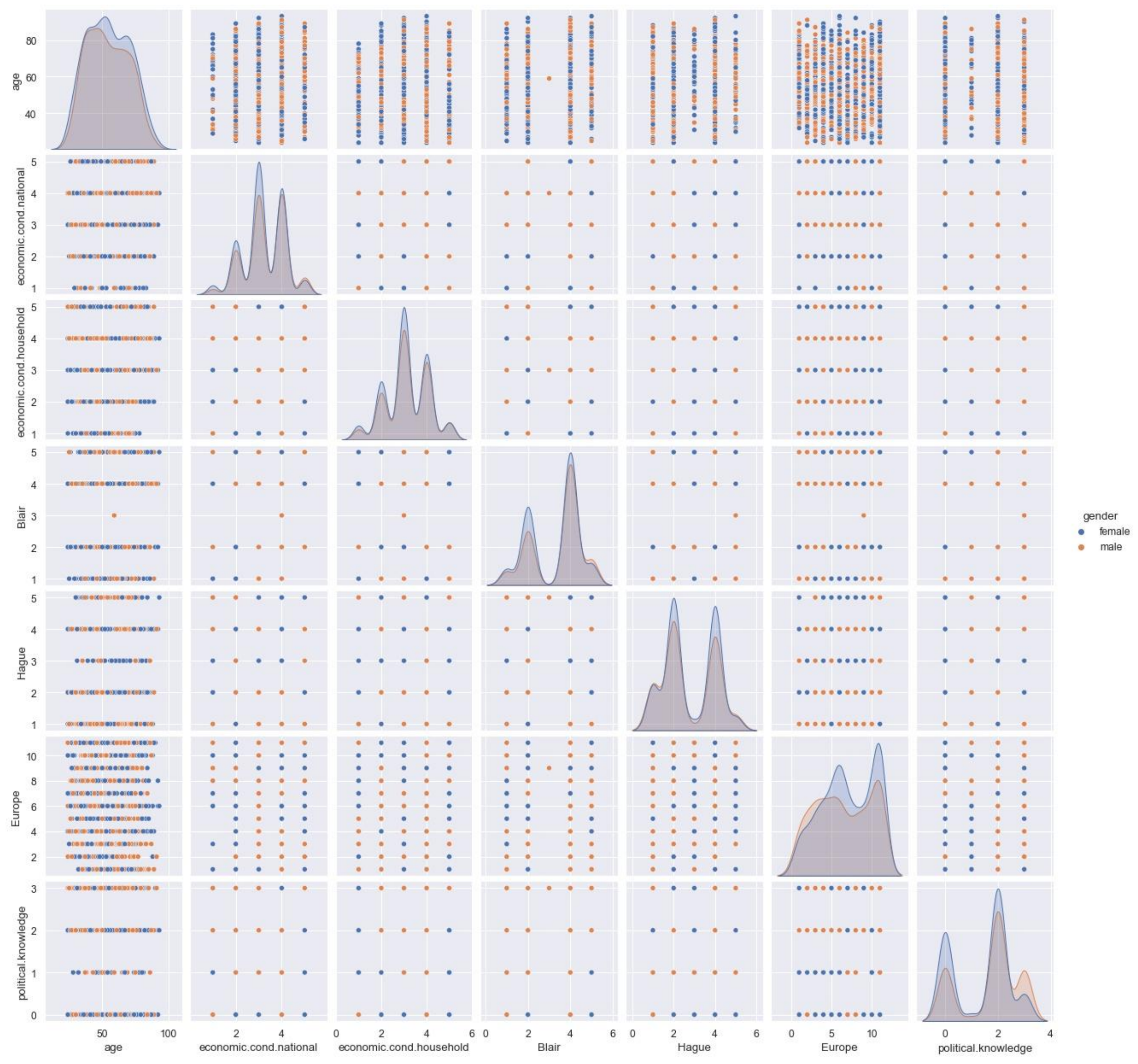


Figure 16: Pairplot hued over feature 'gender'

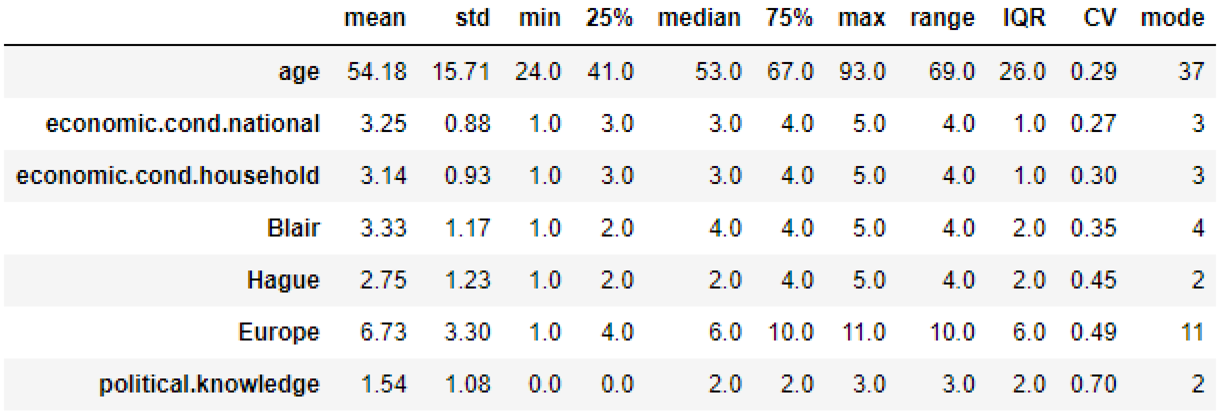
(above figure generated with '.pairplot()' function of Seaborn library in Python)

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).

{Encode the data (having string values) for Modelling. Is Scaling necessary here or not?(2 pts), Data Split: Split the data into train and test (70:30) (2 pts). The learner is expected to check

and comment about the difference in scale of different features on the bases of appropriate measure for example std dev, variance, etc. Should justify whether there is a necessity for scaling. Object data should be converted into categorical/numerical data to fit in the models. (pd.categorical().codes(), pd.get_dummies(drop_first=True)) Data split, ratio defined for the split, train-test split should be discussed.}

- Let us now move to the general description of numeric data to inspect standard deviations, coefficients of variation and ranges:



	mean	std	min	25%	median	75%	max	range	IQR	CV	mode
age	54.18	15.71	24.0	41.0	53.0	67.0	93.0	69.0	26.0	0.29	37
economic.cond.national	3.25	0.88	1.0	3.0	3.0	4.0	5.0	4.0	1.0	0.27	3
economic.cond.household	3.14	0.93	1.0	3.0	3.0	4.0	5.0	4.0	1.0	0.30	3
Blair	3.33	1.17	1.0	2.0	4.0	4.0	5.0	4.0	2.0	0.35	4
Hague	2.75	1.23	1.0	2.0	2.0	4.0	5.0	4.0	2.0	0.45	2
Europe	6.73	3.30	1.0	4.0	6.0	10.0	11.0	10.0	6.0	0.49	11
political.knowledge	1.54	1.08	0.0	0.0	2.0	2.0	3.0	3.0	2.0	0.70	2

Table 8: General Description of Numeric Data(Problem 1)

(above table generated with .describe() function of PANDAS library in Python)

From above table, we conclude:

- 'age' is in a very different scale than rest of the features, which is obvious as 'age' is the only continuous column, rest of the numeric features are ordinal ratings.
- the coefficients of variation are quite high for 'Hague', 'Europe' and 'political.knowledge'.

This calls for feature scaling. We need to perform Feature Scaling when we are dealing with:

- Distance-based algorithms (KNN, K-means, SVM) as these are very sensitive to the range of the data points.
- Gradient Descent Based algorithms (Linear and Logistic Regression, Neural Network). Here, accuracy is not much affected, but convergence becomes faster

Feature Scaling is not mandatory when dealing with Tree-based algorithms.

❖ LDA needs the data with the same variance but it is very difficult to make ordinal data as normally distributed data.
❖ The performance of logistic regression does not improve with data scaling, the reason is that, if there are predictor variables with large ranges that affect the target variable, a regression algorithm will make the corresponding coefficients small so that they do not

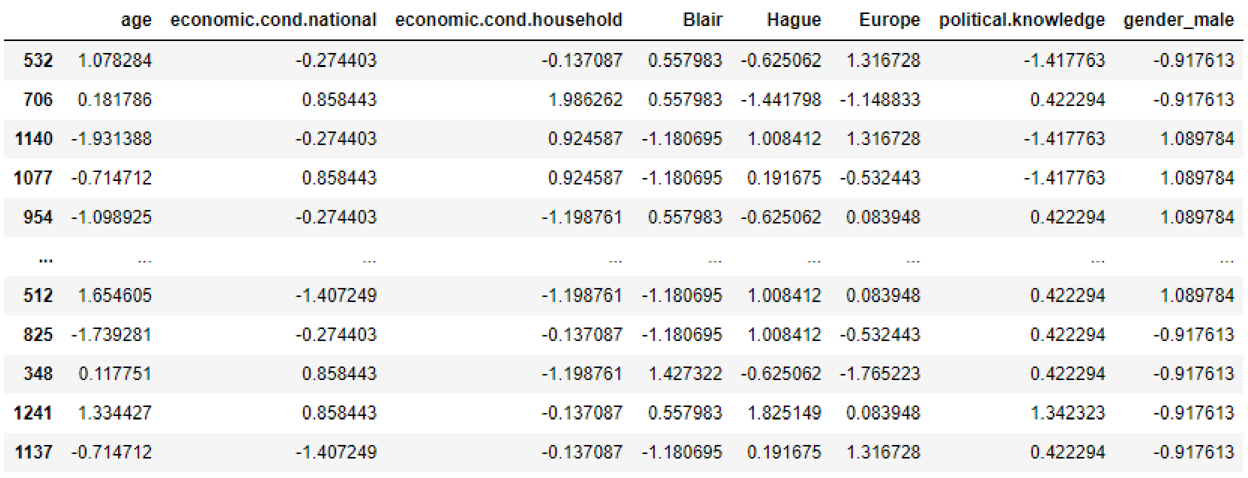
affect predictions so much. Nonetheless, the convergence time is much less for scaled data if the data set is large.

❖ As Naive Bayes algorithm is based on probability not on distance or gradient descent, so it doesn't require feature scaling.
❖ As KNN algorithm is distance based, it requires scaling of features.
❖ As random forest, bagging and boosting are based on Classification & Regression Trees (CART), they do not require scaling of data.

★ Before scaling, let us encode the feature 'gender' using 'pd.get_dummies()' with 'drop_first=True'. The feature is now 'gender_male' with 0 denoting female and 1 denoting male.
★ The ordinal features need not be encoded as they already have an order in them, with a higher number indicating higher weightage.
★ The target feature need not be encoded, but for probability threshold optimisation, let us keep a copy of target encoded dataset with feature name 'vote' being replaced by 'vote_Labour' where 1 indicates class 'Labour' and 0 indicates class 'Conservative'.
★ We separate the predictor and target variables.
★ 'train_test_split' from 'sklearn.model_selection' is being used to split the data into training and testing sets:
    ○ 'test_size=.30' is being provided to split the data with samples in the ratio of 7:3 for train and test sets respectively.
    ○ 'random_state=1' is being provided for consistency.
    ○ 'stratify=y' is being given to have the train and test set target feature have classes in the same proportion as in unsplit data.
    ○ shape of train set: 1061 rows × 8 columns
    ○ shape of test set: 456 rows × 8 columns

For feature scaling:

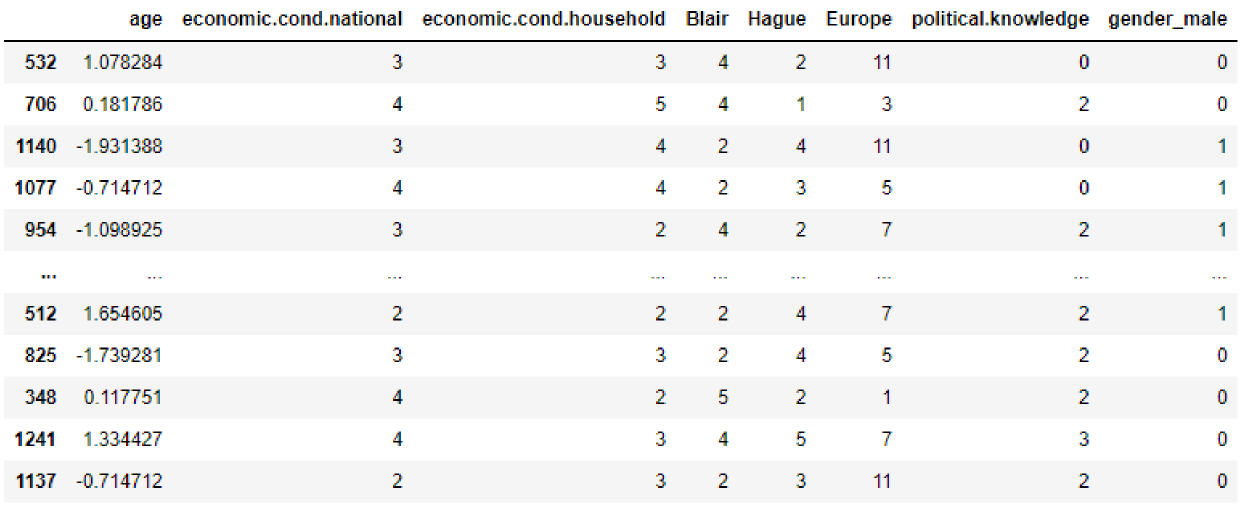
- we will do both standardisation and normalisation separately, and keep separate copies.
- for each scaling technique,we again make 2 sets:
    - 1st set with only 'age' feature scaled as it is the only continuous variable.
    - 2nd set with all numeric features scaled.
- 'StandardScaler' from 'sklearn.preprocessing' is being used for data standardisation.
- 'MinMaxScaler()' from 'sklearn.preprocessing' is being used for data normalisation.
- For both scaling techniques, train set is fit and transformed into the scaling model, and test set is transformed only(using the same train fitted model with train set's parameters)



	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	gender_male
532	1.078284	-0.274403	-0.137087	0.557983	-0.625062	1.316728	-1.417763	-0.917613
706	0.181786	0.858443	1.986262	0.557983	-1.441798	-1.148833	0.422294	-0.917613
1140	-1.931388	-0.274403	0.924587	-1.180695	1.008412	1.316728	-1.417763	1.089784
1077	-0.714712	0.858443	0.924587	-1.180695	0.191675	-0.532443	-1.417763	1.089784
954	-1.098925	-0.274403	-1.198761	0.557983	-0.625062	0.083948	0.422294	1.089784
...	...	...	...	...	...	...	...	...
512	1.654605	-1.407249	-1.198761	-1.180695	1.008412	0.083948	0.422294	1.089784
825	-1.739281	-0.274403	-0.137087	-1.180695	1.008412	-0.532443	0.422294	-0.917613
348	0.117751	0.858443	-1.198761	1.427322	-0.625062	-1.765223	0.422294	-0.917613
1241	1.334427	0.858443	-0.137087	0.557983	1.825149	0.083948	1.342323	-0.917613
1137	-0.714712	-1.407249	-0.137087	-1.180695	0.191675	1.316728	0.422294	-0.917613

Table 9: Sample of Train data with all features standardised

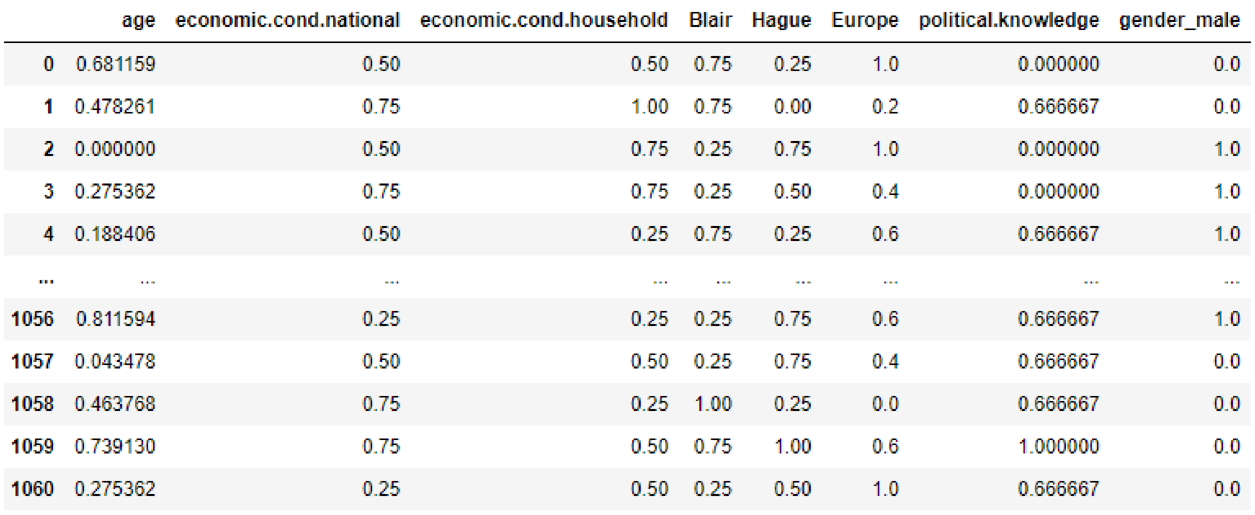
(table generated with PANDAS library in Python)



	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	gender_male
532	1.078284	3	3	4	2	11	0	0
706	0.181786	4	5	4	1	3	2	0
1140	-1.931388	3	4	2	4	11	0	1
1077	-0.714712	4	4	2	3	5	0	1
954	-1.098925	3	2	4	2	7	2	1
...	...	...	...	...	...	...	...	...
512	1.654605	2	2	2	4	7	2	1
825	-1.739281	3	3	2	4	5	2	0
348	0.117751	4	2	5	2	1	2	0
1241	1.334427	4	3	4	5	7	3	0
1137	-0.714712	2	3	2	3	11	2	0

Table 10: Sample of Train data with 'age' feature standardised

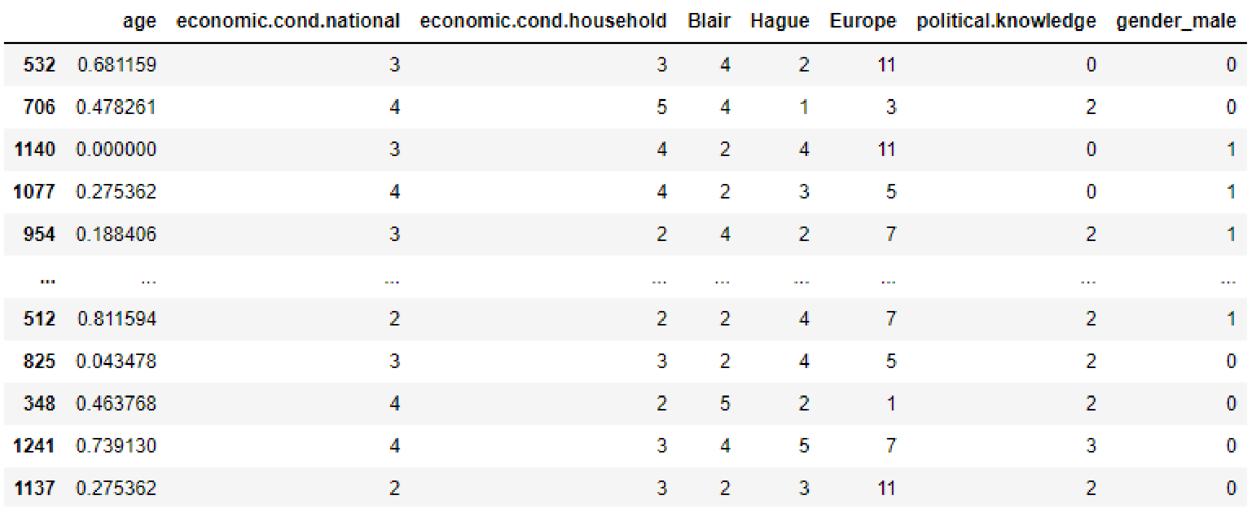
(table generated with PANDAS library in Python)



	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	gender_male
0	0.681159	0.50	0.50	0.75	0.25	1.0	0.000000	0.0
1	0.478261	0.75	1.00	0.75	0.00	0.2	0.666667	0.0
2	0.000000	0.50	0.75	0.25	0.75	1.0	0.000000	1.0
3	0.275362	0.75	0.75	0.25	0.50	0.4	0.000000	1.0
4	0.188406	0.50	0.25	0.75	0.25	0.6	0.666667	1.0
...	...	...	...	...	...	...	...	...
1056	0.811594	0.25	0.25	0.25	0.75	0.6	0.666667	1.0
1057	0.043478	0.50	0.50	0.25	0.75	0.4	0.666667	0.0
1058	0.463768	0.75	0.25	1.00	0.25	0.0	0.666667	0.0
1059	0.739130	0.75	0.50	0.75	1.00	0.6	1.000000	0.0
1060	0.275362	0.25	0.50	0.25	0.50	1.0	0.666667	0.0

Table 11: Sample of Train data with all features normalised

(table generated with PANDAS library in Python)

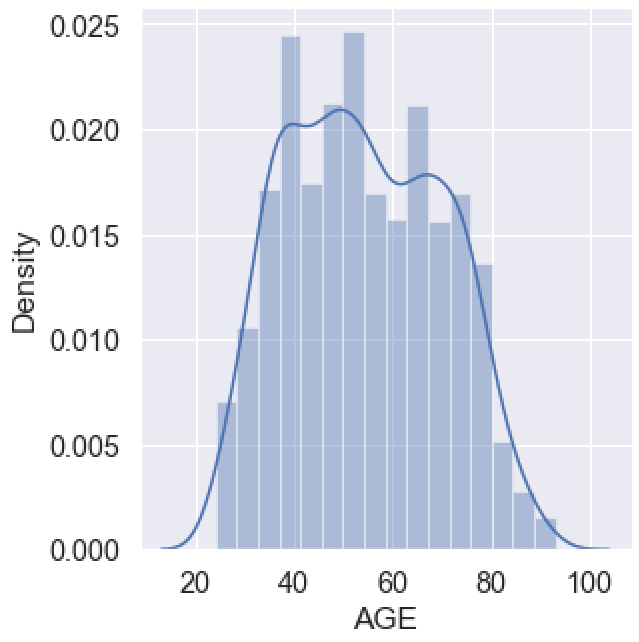


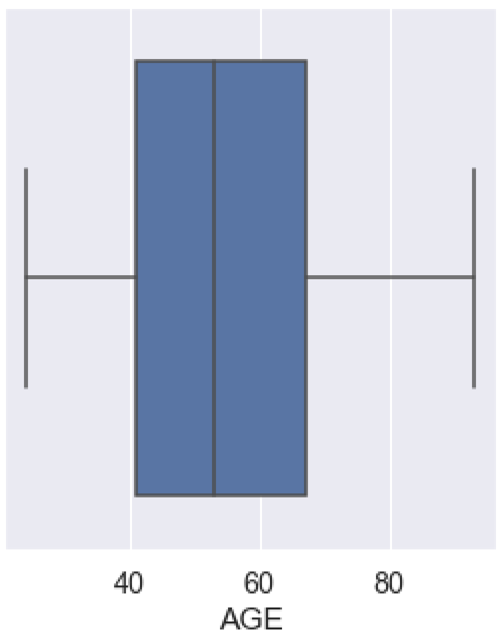
	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	gender_male
532	0.681159	3	3	4	2	11	0	0
706	0.478261	4	5	4	1	3	2	0
1140	0.000000	3	4	2	4	11	0	1
1077	0.275362	4	4	2	3	5	0	1
954	0.188406	3	2	4	2	7	2	1
...	...	...	...	...	...	...	...	...
512	0.811594	2	2	2	4	7	2	1
825	0.043478	3	3	2	4	5	2	0
348	0.463768	4	2	5	2	1	2	0
1241	0.739130	4	3	4	5	7	3	0
1137	0.275362	2	3	2	3	11	2	0

Table 12: Sample of Train data with 'age' feature normalised

(table generated with PANDAS library in Python)

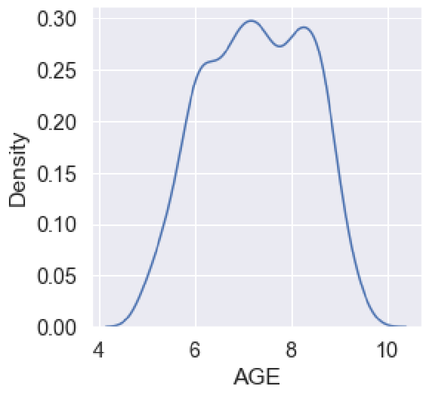
Let us have a look at the 'age' feature:

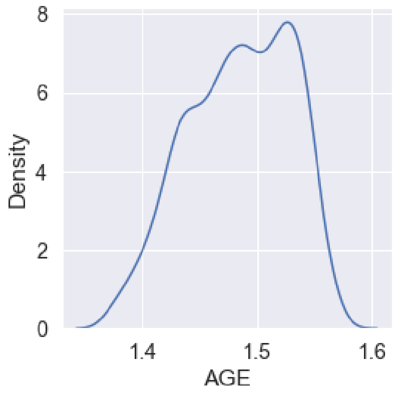


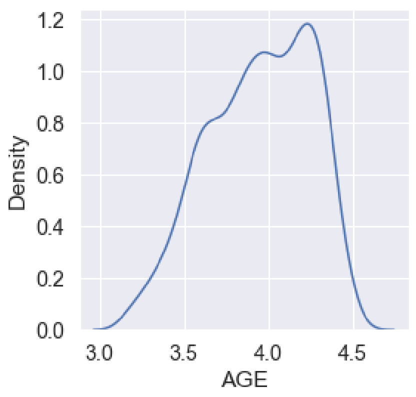


Skewness: 0.140 Kurtosis: -0.944

Let us apply a few transformations to the continuous feature 'age':







Square Root transformation **10th Root transformation** **Log transformation**

Square Root transformation	10th Root transformation	Log transformation
Skewness: -0.096	Skewness: -0.296	Skewness: -0.348
Kurtosis: -0.927	Kurtosis: -0.789	Kurtosis: -0.735

- We see the magnitude of skewness has gone down marginally only for square root transformation. A reduction in magnitude of kurtosis is also indicating the flatness of the feature has reduced too.
- We keep a copy of the data frame with square root transformation applied to the 'age' feature.

- We will be using it for logistic regression and k nearest neighbours algorithms. Even though these algorithms do not assume normally distributed independent variables, their accuracies will get altered.
- We won't be using it for tree based algorithms as such transformations don't affect tree based algorithms.
- LDA and Naive Bayes algorithm also won't get affected by such feature transformation as they both calculate probabilities for classification(LDA also uses Bayes' theorem to estimate class probabilities).

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

{Apply Logistic Regression and LDA (Linear Discriminant Analysis) (2 pts). Interpret the inferences of both models (2 pts). Successful implementation of each model. Logical reason behind the selection of different values for the parameters involved in each model. Calculate Train and Test Accuracies for each model. Comment on the validness of models (over fitting or under fitting)}

Logistic Regression

- 'LogisticRegression' from 'sklearn.linear_model' is being used to build the models.
- The parameters being passed to 'LogisticRegression' are:
    - solver='lbfgs' : Stands for Limited-memory Broyden–Fletcher–Goldfarb–Shanno. It approximates the second derivative matrix updates with gradient evaluations. It stores only the last few updates, so it saves memory. It isn't super fast with large data sets. L-BFGS takes you closer to optimal than gradient descent although per iteration cost is huge.
    - 'max_iter=10000' : 10000 iterations seems to be enough for lbfgs solver and 1061 samples in train set
    - 'penalty='l2'' : adding a lasso regularisation penalty as lasso is found to perform better than ridge or elastic net (Elastic-Net regularization is only supported by the 'saga' solver, which in turn does not support lasso penalty).
- Following are the train and test accuracies for different logistic regression models:

	Train Accuracy	**Test Accuracy**
Model 1 (unscaled data)	0.8285	0.8553
Model 2 ('age' column standardised only)	0.8285	0.8553
Model 3 (all features standardised)	0.8275	0.8553
Model 4 ('age' column normalised data)	0.8294	0.8531

Model 5 (all features normalised)	0.8266	0.8509

- The models aren’t in overfit or underfit situations as the test accuracies are within the 10% boundary of train accuracies.
- A grid search operation was carried out using ‘GridSearchCV’ of ‘sklearn.model_selection’. Following was the grid of parameters:

'penalty':['none', 'l2', 'elasticnet', 'l1'],

'solver':['newton-cg', 'lbfgs', 'sag', 'saga', 'liblinear'],

'tol':[0.0001,0.00001,0.001]}

- ‘GridSearchCV’ also took the following parameters:
    - ❖ cv = 3, indicating a 3 fold cross-validation
    - ❖ n_jobs=-1, indicating all the cores to be engaged for parallel processing
    - ❖ scoring='f1', indicating the model to chosen from grid search based on the f1-score

(Note that not the entire grid be passed in one go as ‘newton-cg’ solver supports only ‘l2’ and ‘none’ penalties, and only ‘saga’ solver supports ‘elasticnet’. So, the grid search needs to done in more than one go)

- ‘f1-score’ is being used as the ‘scoring policy’ in the grid search operation.
- Following were the best parameters from the grid search operation:

  'penalty': 'l2', 'solver': 'sag', 'tol': 0.0001
- Following are the train and test accuracies for different logistic regression models after grid search:

	Train Accuracy	**Test Accuracy**
Model 1 (unscaled data)	0.8256	0.8575
Model 2 (only ‘age’ column standardised)	0.8285	0.8553
Model 3 (all features standardised)	0.8275	0.8553
Model 4 (only ‘age’ column normalised)	0.8294	0.8531
Model 5 (all features normalised)	0.8266	0.8509

- The models aren’t in overfit or underfit situations as the test accuracies are within the 10% boundary of train accuracies.
- Only the accuracies for unscaled data have changed when compared with the results before grid search.
- The 1st model before grid search seems to be the best. Following is the model intercept and coefficients for the same:

  Intercept: 3.3269

  Coefficient for 'age' is -0.0133.

  Coefficient for 'economic.cond.national' is 0.3469.

  Coefficient for 'economic.cond.household' is 0.0346.

  Coefficient for 'Blair' is 0.5491.

  Coefficient for 'Hague' is -0.8554.

  Coefficient for 'Europe' is -0.2079.

  Coefficient for 'political.knowledge' is -0.4032.

  Coefficient for 'gender_male' is 0.0496.

  - We see the feature ‘Hague’ has the highest influence on predictions(also has highest negative coefficient).
  - ‘Blair’ has the highest positive coefficient.
  - ‘political.knowledge’ also has a high negative coefficient.

<u>Linear Discriminant Analysis</u>

- ‘LinearDiscriminantAnalysis’ from ‘sklearn.discriminant_analysis’ is being used to build the models.
- The parameters being passed to ‘LinearDiscriminantAnalysis’ are:
  - solver='svd' : Stands for Singular Value Decomposition. SVD gives you much better insight on the accuracy of the solution and nicely handles singular and near-singular cases.
  - ‘tol=0.0001’ : tolerance of 0.0001(default) seems to be good for data

- Following are the train and test accuracies for different LDA models:

	Train Accuracy	**Test Accuracy**

Model 1 (unscaled data)	0.8228	0.8531
Model 2 (only ‘age’ column standardised)	0.8228	0.8531
Model 3 (all features standardised)	0.8228	0.8531
Model 4 (only ‘age’ column normalised)	0.8228	0.8531
Model 5 (all features normalised)	0.8228	0.8531

- The models aren’t in overfit or underfit situations as the test accuracies are within the 10% boundary of train accuracies.
- We see the same results for unscaled and all scaled data, because Linear Discriminant Analysis (LDA) finds it’s coefficients using the variation between the classes(uses Bayes’ Theorem to estimate probabilities), so the scaling doesn’t matter either.
- A grid search operation was carried out using ‘GridSearchCV’ of ‘sklearn.model_selection’. Following was the grid of parameters:

'solver':['svd','eigen','lsqr']

- ‘GridSearchCV’ also took the following parameters:
    - ❖ cv = 3, indicating a 3 fold cross-validation
    - ❖ n_jobs=-1, indicating all the cores to be engaged for parallel processing
    - ❖ scoring='f1', indicating the model to chosen from grid search based on the f1-score
- ‘f1-score’ is being used as the ‘scoring policy’ in the grid search operation.
- Following were the best parameters from the grid search operation:

    'solver': 'svd'

(Not going for the modelling again as we had used svd in our original models before grid search also)

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

{Apply KNN Model and Naïve Bayes Model (2pts). Interpret the inferences of each model (2 pts). Successful implementation of each model. Logical reason behind the selection of different values for the parameters involved in each model. Calculate Train and Test Accuracies for each model. Comment on the validness of models (over fitting or under fitting)}

<u>KNN Model</u>

- ‘KNeighborsClassifier’ from ‘sklearn.neighbors’ is being used to build the models.
- The parameters being passed to ‘KNeighborsClassifier’ are:

    - ‘n_neighbors=5’ : given the shape of the train set, k=5 seems to be a good trade off between speed and accuracy. Nonetheless, we have optimised the k-value below too
    - weights='uniform' : assigning a uniform weight rather than inverse-distance, as we are just looking at 5 nearest neighbours .
    - algorithm='auto' : 'auto' will attempt to decide the most appropriate algorithm amongst kd-tree, ball-tree and brute-force search.
- Following are the train and test accuracies for different knn models:

	Train Accuracy	**Test Accuracy**
Model 1 (unscaled data)	0.8492	0.8070
Model 2 (‘age’ column standardised only)	0.8615	0.8509
Model 3 (all features standardised)	0.8615	0.8377
Model 4 (‘age’ column normalised data)	0.8633	0.8465
Model 5 (all features normalised)	0.8680	0.8443

- The models aren’t in overfit or underfit situations as the test accuracies are within the 10% boundary of train accuracies.
- The performance metrics of the model with only ‘age’ column standardised is better than the rest.
- A grid search operation was carried out using ‘GridSearchCV’ of ‘sklearn.model_selection’. Following was the grid of parameters:

'weights':['uniform','distance'],

'leaf_size':[10,15,20,30,50,100,200,400,500]

- ‘GridSearchCV’ also took the following parameters:
    - ❖ cv = 3, indicating a 3 fold cross-validation
    - ❖ n_jobs=-1, indicating all the cores to be engaged for parallel processing
    - ❖ scoring='f1', indicating the model to chosen from grid search based on the f1-score
- ‘f1-score’ is being used as the ‘scoring policy’ in the grid search operation.
- Following were the best parameters from the grid search operation:

  'leaf_size': 10, 'weights': 'uniform'
- Instead of passing ‘n_neighbors’ as a parameter in the grid search, let us try to plot the misclassification error and visualise the best k value being chosen.

    - A custom function is being written that finds the accuracies for the k values 1 to 30.
    - Misclassification error can be obtained by subtracting the accuracy score from 1:

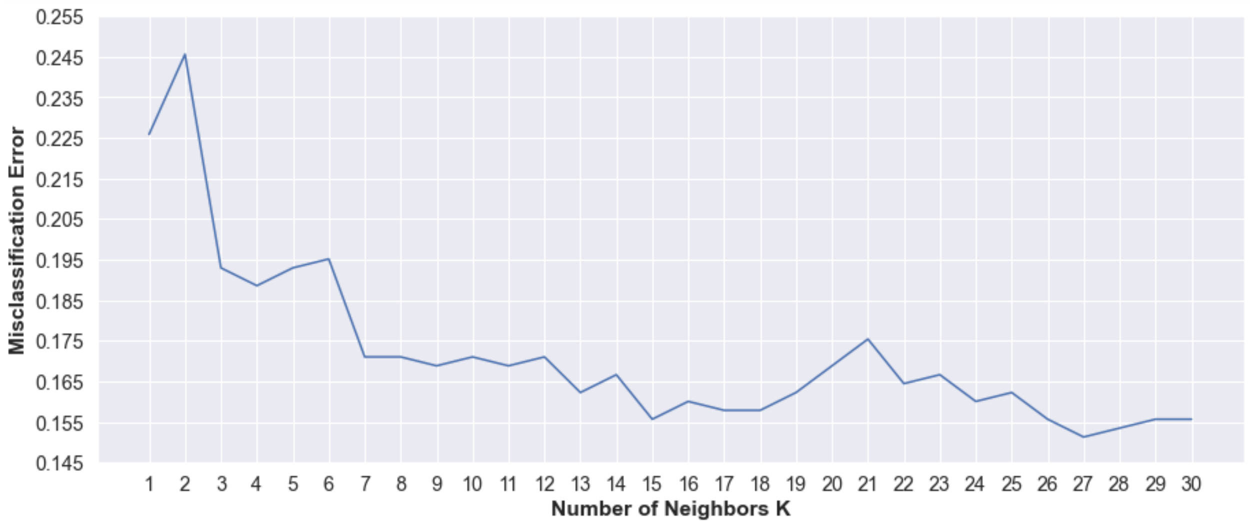


Figure 17: Optimal k-value Identification

(plot generated using matplotlib in Python)

    - k=27 is giving the least misclassification error.
- Following are the train and test accuracies for different knn models after grid search and k=27:

	Train Accuracy	**Test Accuracy**
Model 1 (unscaled data)	0.8058	0.8465
Model 2 (only ‘age’ column standardised)	0.8351	0.8509
Model 3 (all features standardised)	0.8322	0.8443
Model 4 (only ‘age’ column normalised)	0.8275	0.8421
Model 5 (all features normalised)	0.8285	0.8421

- The models aren’t in overfit or underfit situations as the test accuracies are within the 10% boundary of train accuracies.
- The performance metrics of the models have gone down as compared with previous ones.

Naïve Bayes Model

- 'GaussianNB' from 'sklearn.naive_bayes' is being used to build the models.
- No priors are being passed as default, and would be calculated from the data.
- Following are the train and test accuracies for different Naïve Bayes models:

	Train Accuracy	**Test Accuracy**
Model 1 (unscaled data)	0.8200	0.8575
Model 2 ('age' column standardised only)	0.8200	0.8575
Model 3 (all features standardised)	0.8200	0.8575
Model 4 ('age' column normalised data)	0.8200	0.8575
Model 5 (all features normalised)	0.8200	0.8575

- The models aren't in overfit or underfit situations as the test accuracies are within the 10% boundary of train accuracies.
- We see the same results for unscaled and all scaled data, because Naïve Bayes uses probability for classification, and not distance or any gradient descent approach.

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

{Model Tuning (4 pts) , Bagging (1.5 pts) and Boosting (1.5 pts). Apply grid search on each model (include all models) and make models on best_params. Define a logic behind choosing particular values for different hyper-parameters for grid search. Compare and comment on performances of all. Comment on feature importance if applicable. Successful implementation of both algorithms along with inferences and comments on the model performances.}

Logistic Regression

1) **Grid Search**

- A grid search operation was carried out using 'GridSearchCV' of 'sklearn.model_selection'. Following was the grid of parameters:

'penalty':['none', 'l2', 'elasticnet', 'l1'],

'solver':['newton-cg', 'lbfgs', 'sag', 'saga', 'liblinear'],

'tol':[0.0001,0.00001,0.001]}

- 'GridSearchCV' also took the following parameters:

- ❖ cv = 3, indicating a 3 fold cross-validation
- ❖ n_jobs=-1, indicating all the cores to be engaged for parallel processing
- ❖ scoring='f1', indicating the model to chosen from grid search based on the f1-score

(Note that not the entire grid be passed in one go as 'newton-cg' solver supports only 'l2' and 'none' penalties, and only 'saga' solver supports 'elasticnet'. So, the grid search needs to done in more than one go)

- 'f1-score' is being used as the 'scoring policy' in the grid search operation.
- Following were the best parameters from the grid search operation:

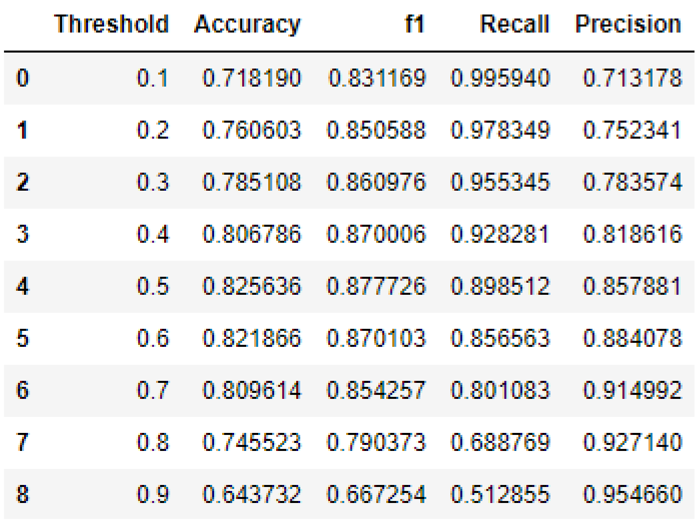
  'penalty': 'l2', 'solver': 'sag', 'tol': 0.0001
- Following are the train and test accuracies for different logistic regression models after grid search:

	Train Accuracy	**Test Accuracy**
Model 1 (unscaled data)	0.8256	0.8575
Model 2 (only 'age' column standardised)	0.8285	0.8553
Model 3 (all features standardised)	0.8275	0.8553
Model 4 (only 'age' column normalised)	0.8294	0.8531
Model 5 (all features normalised)	0.8266	0.8509

- The models aren't in overfit or underfit situations as the test accuracies are within the 10% boundary of train accuracies.
- Only the accuracies for unscaled data have changed.

2) <u>Probability Threshold Optimisation</u>

- Same grid search parameters' results obtained in model 1 were used as the data set.
- Model was built using 'LogisticRegression' of 'sklearn.linear_model'.
- Custom cut-offs from 0.1 to 0.9 with step increment of 0.1 were tried for probability threshold optimisation.
- The results were compared using the following table:



	Threshold	Accuracy	f1	Recall	Precision
0	0.1	0.718190	0.831169	0.995940	0.713178
1	0.2	0.760603	0.850588	0.978349	0.752341
2	0.3	0.785108	0.860976	0.955345	0.783574
3	0.4	0.806786	0.870006	0.928281	0.818616
4	0.5	0.825636	0.877726	0.898512	0.857881
5	0.6	0.821866	0.870103	0.856563	0.884078
6	0.7	0.809614	0.854257	0.801083	0.914992
7	0.8	0.745523	0.790373	0.688769	0.927140
8	0.9	0.643732	0.667254	0.512855	0.954660

Table 13: Metrics for different thresholds for LR model

(table generated with 'sklearn' and 'statsmodels' libraries of Python)

Threshold of 0.5 gives the best f1-score as well as accuracy, so no point in changing the probability threshold.

3) <u>SMOTE</u>

- Same grid search parameters' results obtained in model 1 were used as the data set.
- Over-sampling of the minority class was done using 'SMOTE' of 'imblearn.over_sampling'.
- Train and test shapes before over-sampling:

  Train:(1061, 8) Test:(456, 8)

- Train and test shapes after over-sampling:

  Train:(1478, 8) Test:(456, 8)

- Train and test accuracies before over-sampling:

  Train: 0.8256 Test:0.8575

- Train and test accuracies after over-sampling:

  Train: 0.8187 Test: 0.8311

- We see a drop in test and train accuracies after applying SMOTE oversampling.

4) <u>Adding polynomial features</u>

- Same grid search parameters' results obtained in model 1 were used for the model.
- 'PolynomialFeatures' from 'sklearn.preprocessing' was used to create new polynomial features with a maximum degree of 3. Degrees beyond 4 might result in making the model overfit, and give weird shapes to the distribution of features.

- In this case, 4th degree polynomials were also tried which resulted in an over-fit scenario, so not included in the report.
- Model was built using ‘LogisticRegression’ of ‘sklearn.linear_model’.
- Model was fit on the train set.
- Predictions were made both on the train and test sets, and performance metrics were calculated.
- Following were the coefficients found:

7.77734290e-04, 5.37056091e-03, 1.73646988e-03,

8.55298823e-04, 2.73266092e-03, 1.92880703e-03,

1.66633474e-03, 6.87953222e-04, 1.47375696e-05,

-6.44216435e-03, -1.46366663e-02, 2.53590957e-02,

5.39426238e-03, -3.35390116e-03, 9.51031377e-03,

2.83207185e-04, 2.22426466e-03, 6.57501174e-03,

4.09809564e-03, 2.74764691e-03, 1.57340793e-03,

-7.82593984e-05, 4.08092801e-03, 1.84769680e-03,

-3.27442458e-03, 1.81347209e-04, -3.38914213e-04,

5.90828727e-03, 7.78116062e-03, 1.88235062e-03,

2.40473437e-04, 3.07234217e-03, 1.61000524e-03,

-5.92259586e-04, -3.38468322e-03, -1.62530953e-03,

-2.22981222e-04, 4.93025427e-03, -1.83718712e-03,

-1.93735115e-03, 6.90954167e-04, -1.20949510e-03,

1.35800612e-03, 4.56586962e-04, -1.38328298e-03,

1.11537381e-03, 2.58717040e-04, 2.61455737e-03,

-5.03241645e-03, -1.42944626e-03, 2.86000891e-03,

-2.02165848e-03, 1.23514831e-04, -1.46137397e-03,

1.70769956e-03, -2.81374674e-04, -2.41105283e-04,

5.98995163e-03, 1.57087768e-02, 5.77707293e-03,

-4.40528800e-03, 7.00014493e-04, -1.05494655e-03,

```
 1.46329212e-02,  1.37121531e-02,  4.95944201e-03,
-6.61442295e-04,  5.54576131e-03,  5.03566181e-03,
-1.10260482e-03, -7.86950323e-03, -5.56550953e-03,
-1.05419119e-03,  2.65458702e-03,  2.00921596e-03,
 2.10404020e-03, -2.87558919e-03, -9.13721675e-03,
-1.32654993e-03, -2.62639204e-03, -1.68249050e-02,
-6.40871647e-03, -1.50896288e-03,  5.30917162e-03,
 2.55732733e-03, -1.06187862e-03, -1.48949866e-02,
-2.09985848e-03, -1.76114031e-04, -4.96167026e-03,
-1.15187388e-02, -2.04533902e-03, -7.55826772e-03
```

- Train and test accuracies before adding polynomial features:

  Train: 0.8256 Test:0.8575

- Train and test accuracies after adding polynomial features:

  Train: 0.8501 Test: 0.8355

- We see a drop in test accuracy after adding polynomial features, but train accuracy has increased.

5) <u>Adding polynomial features with square root transformation on ‘age’</u>

- Till now, we saw the model with added polynomial features was giving the best results. Let us use this model but use a dataset with square root transformation applied on the ‘age’ feature.
- Model was built using ‘LogisticRegression’ of ‘sklearn.linear_model’.
- Model was fit on the train set.
- Predictions were made both on the train and test sets, and performance metrics were calculated.
- Following were the coefficients found:

```
 0.0209863 ,  0.05090154,  0.02475174, -0.00046843,  0.04524289,
 0.04753032,  0.0365651 ,  0.01736925, -0.00809543,  0.00109595,
-0.09050773,  0.10990887,  0.03244794, -0.03609412,  0.06690611,
-0.04636001, -0.04551952,  0.03306253,  0.03380933, -0.0146695 ,
 0.01324734, -0.01202545, -0.00442537,  0.00725335, -0.1312183 ,
-0.01231167, -0.01567224,  0.09497599,  0.11262857,  0.01370137,
-0.00434515,  0.07897743,  0.0334583 , -0.02150944, -0.09423512,
-0.03986927, -0.00299806,  0.02782832, -0.04360683, -0.04996006,
 0.0253182 , -0.02359459,  0.02614861,  0.01547691,  0.01700574,
 0.00613105,  0.02188655,  0.00415111, -0.01965241, -0.01449358,
 0.04768037, -0.017073  , -0.01164873, -0.04840904,  0.0346514 ,
 0.00897931,  0.00874325,  0.08915226,  0.10929955, -0.00304325,
-0.00942554, -0.03965673,  0.01210791,  0.07124918, -0.01318484,
-0.01022924, -0.06855711, -0.00558862,  0.08361595,  0.041348  ,
-0.01034357, -0.01346397, -0.00864014, -0.12195279,  0.01492754,
 0.01470319, -0.1372792 ,  0.01643773, -0.03374247,  0.02077465,
-0.00555492,  0.03501526, -0.00184772, -0.00298451, -0.00369262,
 0.05759415, -0.04119476,  0.04470419,  0.02285401,  0.01736787,
-0.07207608, -0.02479687, -0.0570009
```

- Train and test accuracies before adding polynomial features:

  Train: 0.8256   Test:0.8575

- Train and test accuracies after adding polynomial features without square root transformation on 'age' feature:

  Train: 0.8501   Test: 0.8355

- Train and test accuracies after adding polynomial features with square root transformation on 'age' feature:

  Train: 0.8473   Test: 0.8114

- We see a drop in train and test accuracies after square root transformation on the feature 'age'.

So, for logistic regression we can conclude that the model with added polynomial features without square root transformation on 'age' is giving the best results.

<u>Linear Discriminant Analysis</u>

1) <u>Grid Search</u>

- A grid search operation was carried out using ‘GridSearchCV’ of ‘sklearn.model_selection’. Following was the grid of parameters:

'solver':['svd','eigen','lsqr']

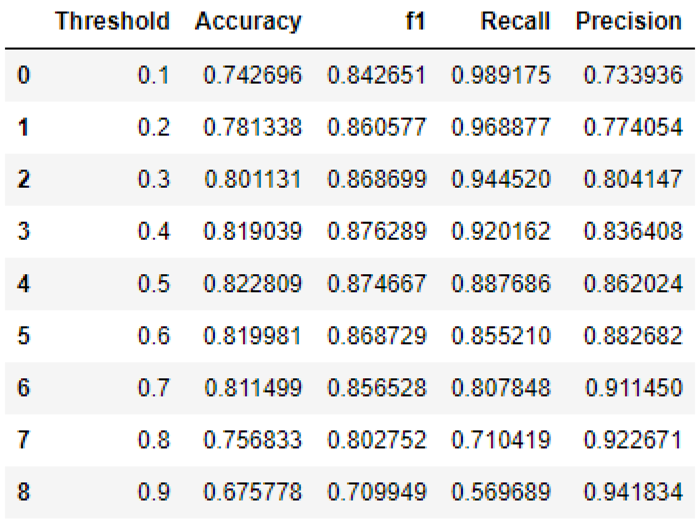
- ‘GridSearchCV’ also took the following parameters:
  - ❖ cv = 3, indicating a 3 fold cross-validation
  - ❖ n_jobs=-1, indicating all the cores to be engaged for parallel processing
  - ❖ scoring='f1', indicating the model to chosen from grid search based on the f1-score
- ‘f1-score’ is being used as the ‘scoring policy’ in the grid search operation.
- Following were the best parameters from the grid search operation:

'solver': 'svd'

(Not going for the modelling again as we had used svd in our original models before grid search also)

2) <u>Probability Threshold Optimisation</u>

- Same grid search parameters’ results obtained in model 1 were used as the data set.
- Custom cut-offs from 0.1 to 0.9 with step increment of 0.1 were tried for probability threshold optimisation.
- The results were compared using the following table:



	Threshold	Accuracy	f1	Recall	Precision
0	0.1	0.742696	0.842651	0.989175	0.733936
1	0.2	0.781338	0.860577	0.968877	0.774054
2	0.3	0.801131	0.868699	0.944520	0.804147
3	0.4	0.819039	0.876289	0.920162	0.836408
4	0.5	0.822809	0.874667	0.887686	0.862024
5	0.6	0.819981	0.868729	0.855210	0.882682
6	0.7	0.811499	0.856528	0.807848	0.911450
7	0.8	0.756833	0.802752	0.710419	0.922671
8	0.9	0.675778	0.709949	0.569689	0.941834

Table 14: Metrics for different thresholds for LDA model

(table generated with ‘sklearn’ and ‘statsmodels’ libraries of Python)

- Threshold of 0.4 gives the best f1-score, while threshold 0.5 gives best accuracy.
- Train and test accuracies before threshold optimisation:

Train: 0.8228 Test: 0.8531

- Train and test accuracies after threshold optimisation:

  Train: 0.8190 Test: 0.8487

- We see a drop in train and test accuracies after optimising the f1 score, also there is no significant rise in f1-score.

3) <u>SMOTE</u>

- Same grid search parameters' results obtained in model 1 were used as the data set.
- Over-sampling of the minority class was done using 'SMOTE' of 'imblearn.over_sampling'.
- Train and test shapes before over-sampling:

  Train:(1061, 8) Test:(456, 8)

- Train and test shapes after over-sampling:

  Train:(1478, 8) Test:(456, 8)

- Train and test accuracies before over-sampling:

  Train: 0.8228 Test: 0.8531

- Train and test accuracies after over-sampling:

  Train: 0.8160 Test: 0.8333

- We see a drop in test and train accuracies after applying SMOTE oversampling.

4) <u>Adding polynomial features</u>

- Same grid search parameters' results obtained in model 1 were used for the model.
- 'PolynomialFeatures' from 'sklearn.preprocessing' was used to create new polynomial features with a maximum degree of 3. Degrees beyond 4 might result in making the model overfit, and give weird shapes to the distribution of features.
- In this case, 4th degree polynomials were also tried which resulted in an over-fit scenario, so not included in the report.
- Model was fit on the train set.
- Predictions were made both on the train and test sets, and performance metrics were calculated.
- Train and test accuracies before adding polynomial features:

  Train: 0.8228 Test: 0.8531

- Train and test accuracies after adding polynomial features:

  Train: 0.8501 Test: 0.8487

- We see a drop in test accuracy after adding polynomial features, but train accuracy has increased.

So, for LDA we can conclude that the model with added polynomial features is giving the best results.

<u>KNN Model</u>

1) **<u>Grid Search</u>**

- A grid search operation was carried out using ‘GridSearchCV’ of ‘sklearn.model_selection’. Following was the grid of parameters:

'weights':['uniform','distance'],

'leaf_size':[10,15,20,30,50,100,200,400,500]

- ‘GridSearchCV’ also took the following parameters:
    - ❖ cv = 3, indicating a 3 fold cross-validation
    - ❖ n_jobs=-1, indicating all the cores to be engaged for parallel processing
    - ❖ scoring='f1', indicating the model to chosen from grid search based on the f1-score
- ‘f1-score’ is being used as the ‘scoring policy’ in the grid search operation.
- Following were the best parameters from the grid search operation:

  'leaf_size': 10, 'weights': 'uniform'
- Instead of passing ‘n_neighbors’ as a parameter in the grid search, let us try to plot the misclassification error and visualise the best k value being chosen.
    - ○ A custom function is being written that finds the accuracies for the k values 1 to 30.
    - ○ Misclassification error can be obtained by subtracting the accuracy score from 1:

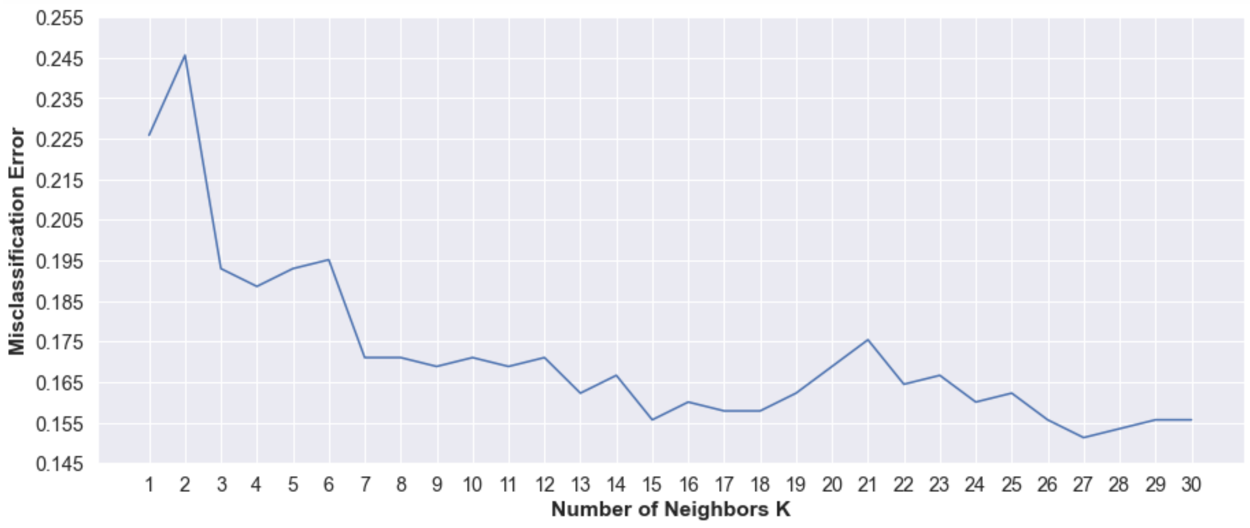


Figure 17: Optimal k-value Identification

(plot generated using matplotlib in Python)

 - k=27 is giving the least misclassification error.

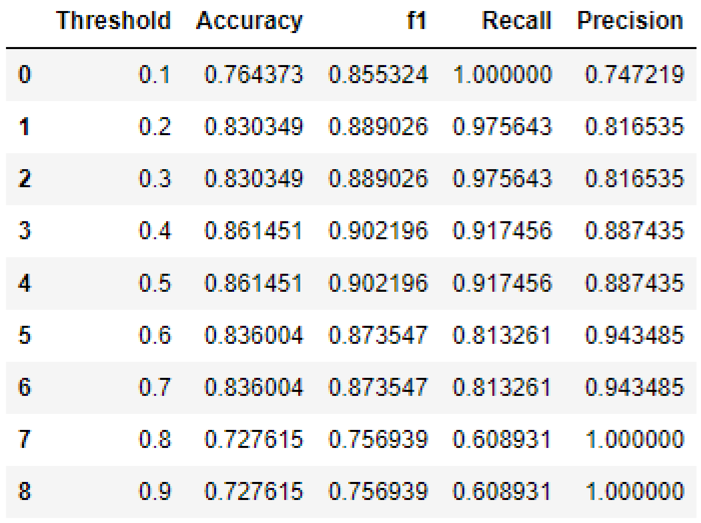
- Following are the train and test accuracies for different knn models after grid search and k=27:

	Train Accuracy	**Test Accuracy**
Model 1 (unscaled data)	0.8058	0.8465
Model 2 (only ‘age’ column standardised)	0.8351	0.8509
Model 3 (all features standardised)	0.8322	0.8443
Model 4 (only ‘age’ column normalised)	0.8275	0.8421
Model 5 (all features normalised)	0.8285	0.8421

- The models aren’t in overfit or underfit situations as the test accuracies are within the 10% boundary of train accuracies.

2) <u>Probability Threshold Optimisation</u>

- Model 1 before the grid search operation with ‘age’ column standardised is being used which gave best results until yet.
- Custom cut-offs from 0.1 to 0.9 with step increment of 0.1 were tried for probability threshold optimisation.
- The results were compared using the following table:



	Threshold	Accuracy	f1	Recall	Precision
0	0.1	0.764373	0.855324	1.000000	0.747219
1	0.2	0.830349	0.889026	0.975643	0.816535
2	0.3	0.830349	0.889026	0.975643	0.816535
3	0.4	0.861451	0.902196	0.917456	0.887435
4	0.5	0.861451	0.902196	0.917456	0.887435
5	0.6	0.836004	0.873547	0.813261	0.943485
6	0.7	0.836004	0.873547	0.813261	0.943485
7	0.8	0.727615	0.756939	0.608931	1.000000
8	0.9	0.727615	0.756939	0.608931	1.000000

Table 15: Metrics for different thresholds for KNN model

(table generated with 'sklearn' and 'statsmodels' libraries of Python)

Thresholds of 0.5 and 0.4 give the best f1-score as well as accuracy(exactly the same), so no point in changing the probability threshold.

3) <u>SMOTE</u>

- Model 1 before the grid search operation with 'age' column standardised is being used which gave best results until yet.
- Over-sampling of the minority class was done using 'SMOTE' of 'imblearn.over_sampling'.
- Train and test shapes before over-sampling:

  Train:(1061, 8) Test:(456, 8)

- Train and test shapes after over-sampling:

  Train:(1478, 8) Test:(456, 8)

- Train and test accuracies before over-sampling:

  Train: 0.8615 Test:0.8509

- Train and test accuracies after over-sampling:

  Train: 0.8897 Test: 0.8333

- We see a drop in test accuracy but increase in train accuracy after applying SMOTE oversampling.

4) <u>Adding polynomial features</u>

- 'PolynomialFeatures' from 'sklearn.preprocessing' was used to create new polynomial features with a maximum degree of 3. Degrees beyond 4 might result in making the model overfit, and give weird shapes to the distribution of features.
- In this case, 4th degree polynomials were also tried which resulted in an over-fit scenario, so not included in the report.
- Model was fit on the train set.
- Predictions were made both on the train and test sets, and performance metrics were calculated.
- Train and test accuracies before adding polynomial features:

  Train: 0.8615 Test:0.8509

- Train and test accuracies after adding polynomial features:

  Train: 0.8624 Test: 0.8246

- We see a drop in test accuracy after adding polynomial features, but train accuracy has increased.

5) **<u>Original model with square root transformation on ‘age’ feature</u>**

- Till now, we saw the original model(before grid search) with ‘age’ feature standardised, was giving the best results. Let us use this model but use a dataset with square root transformation applied on the ‘age’ feature.
- Model was fit on the train set.
- Predictions were made both on the train and test sets, and performance metrics were calculated.
- Train and test accuracies before square root transformation on ‘age’ feature:

  Train: 0.8615 Test:0.8509

- Train and test accuracies after square root transformation on ‘age’ feature:

  Train: 0.8558 Test:0.8487

- We see a reduction in train and test accuracies after square root transformation on ‘age’ feature.

So, for KNN we can conclude that the original model(before grid search) with ‘age’ feature standardised, was the best.

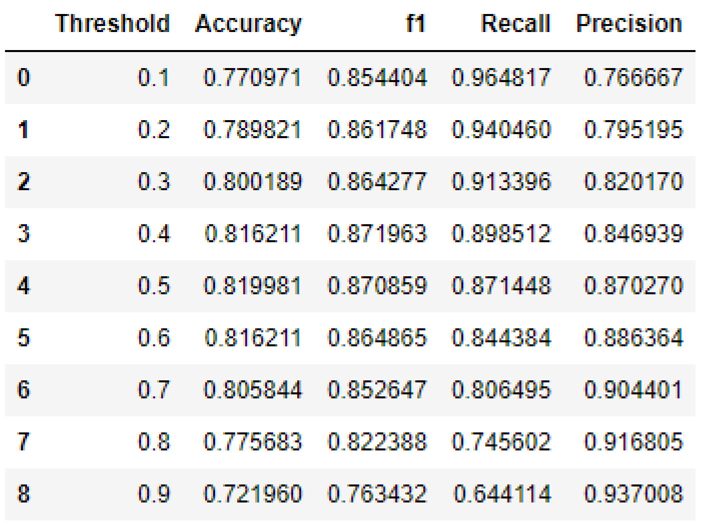
<u>Naïve Bayes Model</u>

1) **<u>Grid Search</u>**

- No hyper-parameter to tune.

2) **<u>Probability Threshold Optimisation</u>**

- Custom cut-offs from 0.1 to 0.9 with step increment of 0.1 were tried for probability threshold optimisation.
- The results were compared using the following table:



	Threshold	Accuracy	f1	Recall	Precision
0	0.1	0.770971	0.854404	0.964817	0.766667
1	0.2	0.789821	0.861748	0.940460	0.795195
2	0.3	0.800189	0.864277	0.913396	0.820170
3	0.4	0.816211	0.871963	0.898512	0.846939
4	0.5	0.819981	0.870859	0.871448	0.870270
5	0.6	0.816211	0.864865	0.844384	0.886364
6	0.7	0.805844	0.852647	0.806495	0.904401
7	0.8	0.775683	0.822388	0.745602	0.916805
8	0.9	0.721960	0.763432	0.644114	0.937008

Table 16: Metrics for different thresholds for Naïve Bayes model

(table generated with 'sklearn' and 'statsmodels' libraries of Python)

- Threshold of 0.4 gives the best f1-score, while threshold 0.5 gives best accuracy.
- Train and test accuracies before threshold optimisation:

  Train: 0.8200 Test: 0.8575

- Train and test accuracies after threshold optimisation:

  Train: 0.8162 Test: 0.8575

- We see a drop in train and test accuracies after optimising the f1 score, also there is no significant rise in f1-score.

3) <u>SMOTE</u>

- Over-sampling of the minority class was done using 'SMOTE' of 'imblearn.over_sampling'.
- Train and test shapes before over-sampling:

  Train:(1061, 8) Test:(456, 8)

- Train and test shapes after over-sampling:

  Train:(1478, 8) Test:(456, 8)

- Train and test accuracies before over-sampling:

  Train: 0.8200 Test: 0.8575

- Train and test accuracies after over-sampling:

  Train: 0.8106 Test: 0.8224

We see a drop in test and train accuracies after applying SMOTE oversampling.

4) **Adding polynomial features**

- 'PolynomialFeatures' from 'sklearn.preprocessing' was used to create new polynomial features with a maximum degree of 4. Degrees beyond 4 might result in making the model overfit, and give weird shapes to the distribution of features.
- In this case, maximum of 3rd degree polynomials were also tried which resulted in an over-fit scenario, so not included in the report.
- Model was fit on the train set.
- Predictions were made both on the train and test sets, and performance metrics were calculated.
- Train and test accuracies before adding polynomial features:

  Train: 0.8200 Test: 0.8575

- Train and test accuracies after adding polynomial features:

  Train: 0.8172 Test: 0.8114

- We see a drop in train and test accuracy after adding polynomial features.

So, for Naïve Bayes we can conclude that the original model had the best performance.

Random Forest(Bagging)

1) **General Model (without pruning and other regularisations):**

- As Random Forest is a tree based algorithm, feature scaling isn't required.
- 'RandomForestClassifier' is being used from 'sklearn.ensemble' to build the models.
- The only parameter being passed is 'random_state=1' for consistency.
- No other parameter is being passed. Let us see if random forest is suitable for this problem(an overfit is expected, which will be taken care of in the grid search)
- Train Accuracy: 1.0 Test Accuracy: 0.8399

2) **Grid Search**

- A grid search operation was carried out using 'GridSearchCV' of 'sklearn.model_selection'. Following was the grid of parameters:

'n_estimators':[100, 50, 150, 200, 300, 500],

'criterion':['gini', 'entropy'],

'max_depth':[5, 6, 7, 8, 9],

'min_samples_split':[2, 5, 10, 20],

'min_samples_leaf':[1, 3, 5, 7, 10],

'max_features':[2, 3, 4, 5, 6],

'bootstrap':[True, False],

'oob_score':[False, True],

'n_jobs':[-1],

'warm_start':[False, True],

'ccp_alpha':[0.0, 0.2, 0.4]

- 'GridSearchCV' also took the following parameters:
  - ❖ cv = 3, indicating a 3 fold cross-validation
  - ❖ n_jobs=-1, indicating all the cores to be engaged for parallel processing
  - ❖ scoring='f1', indicating the model to chosen from grid search based on the f1-score
- 'f1-score' is being used as the 'scoring policy' in the grid search operation.
- Following were the best parameters from the grid search operation:

  'bootstrap': True, 'ccp_alpha': 0.0, 'criterion': 'gini', 'max_depth': 5, 'max_features': 2, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100, 'oob_score': False, 'warm_start': False

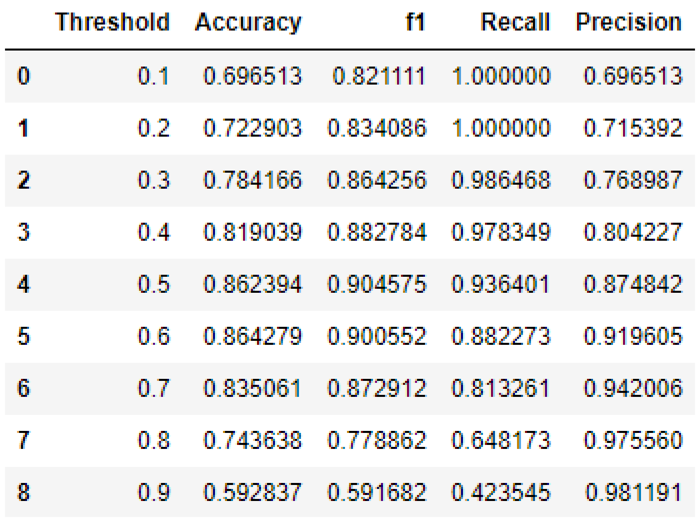
- Accuracy scores from best estimator of grid search:

  Train Accuracy: 0.8624 Test Accuracy: 0.8662

We see the overfit has reduced after the grid search, and the train and test accuracies to be nearly equal.

3) <u>Probability Threshold Optimisation</u>

- The model with parameters after grid search is being used here.
- Custom cut-offs from 0.1 to 0.9 with step increment of 0.1 were tried for probability threshold optimisation.
- The results were compared using the following table:



	Threshold	Accuracy	f1	Recall	Precision
0	0.1	0.696513	0.821111	1.000000	0.696513
1	0.2	0.722903	0.834086	1.000000	0.715392
2	0.3	0.784166	0.864256	0.986468	0.768987
3	0.4	0.819039	0.882784	0.978349	0.804227
4	0.5	0.862394	0.904575	0.936401	0.874842
5	0.6	0.864279	0.900552	0.882273	0.919605
6	0.7	0.835061	0.872912	0.813261	0.942006
7	0.8	0.743638	0.778862	0.648173	0.975560
8	0.9	0.592837	0.591682	0.423545	0.981191

Table 17: Metrics for different thresholds for RF model

(table generated with 'sklearn' and 'statsmodels' libraries of Python)

Threshold of 0.5 gives the best f1-score, whereas threshold of 0.6 gives the best accuracy. As the changes are very marginal and default 0.5 threshold has the best f1-score, so no point in changing the probability threshold.

4) **SMOTE**

- The model with parameters after grid search is being used here.
- Over-sampling of the minority class was done using 'SMOTE' of 'imblearn.over_sampling'.
- Train and test shapes before over-sampling:

  Train:(1061, 8) Test:(456, 8)

- Train and test shapes after over-sampling:

  Train:(1478, 8) Test:(456, 8)

- Train and test accuracies before over-sampling:

  Train: 0.8624 Test: 0.8662

- Train and test accuracies after over-sampling:

  Train: 0.8572 Test: 0.8443

We see a drop in test and train accuracies after applying SMOTE oversampling.

6) **Adding polynomial features**

- The model with parameters after grid search is being used here.

- 'PolynomialFeatures' from 'sklearn.preprocessing' was used to create new polynomial features with a maximum degree of 3. Degrees beyond 4 might result in making the model overfit, and give weird shapes to the distribution of features.
- In this case, 4th degree polynomials were also tried which resulted in an over-fit scenario, so not included in the report.
- Model was fit on the train set.
- Predictions were made both on the train and test sets, and performance metrics were calculated.
- Train and test accuracies before adding polynomial features:

  Train: 0.8624 Test: 0.8662

- Train and test accuracies after adding polynomial features:

  Train: 0.8633 Test: 0.8333

We see a drop in test accuracy after adding polynomial features, but train accuracy has increased, marginally though.

So, for random forest, we conclude that the model after grid search is giving the best results.

<u>Ada Boost</u>

1) **<u>General Model (without pruning and other regularisations):</u>**

- As AdaBoost is a tree based algorithm, feature scaling isn't required.
- 'AdaBoostClassifier' is being used from 'sklearn.ensemble' to build the models.
- The only parameter being passed is 'random_state=1' for consistency.
- No base estimator is being provided as 'AdaBoostClassifier' takes a simple 1-depth decision tree as base estimator by default. And boosting algorithms compute faster with simpler base models.
- Train Accuracy: 0.8398 Test Accuracy: 0.8355
- The model is right-fit.

2) **<u>Grid Search</u>**

- A grid search operation was carried out using 'GridSearchCV' of 'sklearn.model_selection'. Following was the grid of parameters:

'n_estimators':[30,40,50,100,150,200]

'learning_rate':[1.0 ,0.1, 0.01, 0.5]

'algorithm':['SAMME.R', 'SAMME']

- 'GridSearchCV' also took the following parameters:
  - ❖ cv = 3, indicating a 3 fold cross-validation

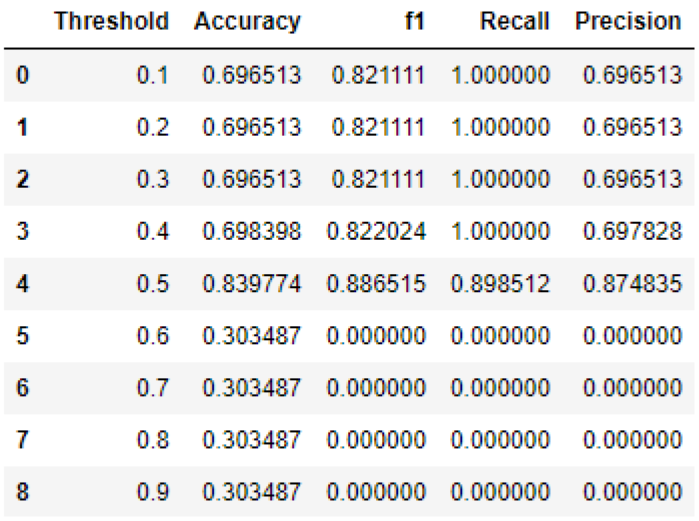
    - ❖ n_jobs=-1, indicating all the cores to be engaged for parallel processing
    - ❖ scoring='f1', indicating the model to chosen from grid search based on the f1-score
- ‘f1-score’ is being used as the ‘scoring policy’ in the grid search operation.
- Following were the best parameters from the grid search operation:

  'algorithm': 'SAMME.R', 'learning_rate': 1.0, 'n_estimators': 30
- Accuracy scores from best estimator of grid search:

  Train Accuracy: 0.8303 Test Accuracy: 0.8377

We see comparable results with the previous model, with marginal changes.

3) <u>Probability Threshold Optimisation</u>
- The first model (before grid search) is being used here.
- Custom cut-offs from 0.1 to 0.9 with step increment of 0.1 were tried for probability threshold optimisation.
- The results were compared using the following table:



	Threshold	Accuracy	f1	Recall	Precision
0	0.1	0.696513	0.821111	1.000000	0.696513
1	0.2	0.696513	0.821111	1.000000	0.696513
2	0.3	0.696513	0.821111	1.000000	0.696513
3	0.4	0.698398	0.822024	1.000000	0.697828
4	0.5	0.839774	0.886515	0.898512	0.874835
5	0.6	0.303487	0.000000	0.000000	0.000000
6	0.7	0.303487	0.000000	0.000000	0.000000
7	0.8	0.303487	0.000000	0.000000	0.000000
8	0.9	0.303487	0.000000	0.000000	0.000000

Table 18: Metrics for different thresholds for Adaboost model

(table generated with ‘sklearn’ and ‘statsmodels’ libraries of Python)

Threshold of 0.5 gives the best f1-score as well as the best accuracy, so no point in changing the probability threshold.

4) <u>SMOTE</u>
- The first model (before grid search) is being used here.
- Over-sampling of the minority class was done using ‘SMOTE’ of ‘imblearn.over_sampling’.
- Train and test shapes before over-sampling:

Train:(1061, 8)   Test:(456, 8)

- Train and test shapes after over-sampling:

  Train:(1478, 8)   Test:(456, 8)

- Train and test accuracies before over-sampling:

  Train: 0.8398   Test: 0.8355

- Train and test accuracies after over-sampling:

  Train: 0.8369   Test: 0.8399

We see comparable results with the previous model, with marginal changes, after applying SMOTE oversampling.

5) <u>Adding polynomial features</u>

- The first model (before grid search) is being used here.
- 'PolynomialFeatures' from 'sklearn.preprocessing' was used to create new polynomial features with a maximum degree of 3. Degrees beyond 4 might result in making the model overfit, and give weird shapes to the distribution of features.
- In this case, 3rd and 4th degree polynomials were also tried which resulted in a more over-fit scenario, so not included in the report.
- Model was fit on the train set.
- Predictions were made both on the train and test sets, and performance metrics were calculated.
- Train and test accuracies before adding polynomial features:

  Train: 0.8398   Test: 0.8355

- Train and test accuracies after adding polynomial features:

  Train: 0.8549   Test: 0.8202

We see an overfit scenario after adding polynomial features.

So, for Adaboost, we conclude that the first model (before grid search) is giving the best results.

<u>Gradient Boost</u>

1) <u>General Model (without pruning and other regularisations):</u>

- As gradient-boost is a tree based algorithm, feature scaling isn't required.
- 'GradientBoostingClassifier' is being used from 'sklearn.ensemble' to build the models.
- The only parameter being passed is 'random_state=1' for consistency. For rest parameters, we go with the default and would tune them in grid search operation.
- Train Accuracy: 0.8860   Test Accuracy: 0.8421
- The model is overfit.

2) <u>Grid Search</u>

- A grid search operation was carried out using ‘GridSearchCV’ of ‘sklearn.model_selection’. Following was the grid of parameters:

'n_estimators':[30,40,50,100,150,200]

'learning_rate':[1.0 ,0.1, 0.01, 0.5]

'loss':['deviance', 'exponential'],

'subsample':[1.0, 0.1, 0.01, 0.5],

'criterion':['friedman_mse', 'squared_error', 'mse', 'mae'],

'min_samples_split':[2,4, 6, 10],

'min_samples_leaf':[1, 3, 5],

'min_weight_fraction_leaf':[0.0, 0.2, 0.5],

'max_depth':[3, 4, 6, 10],

'validation_fraction':[0.1, 0.2 ,0.3],

'tol':[0.01, 0.001, 0.0001],

'ccp_alpha':[0.0, 0.2, 0.4]}

- ‘GridSearchCV’ also took the following parameters:
    - ❖ cv = 3, indicating a 3 fold cross-validation
    - ❖ n_jobs=-1, indicating all the cores to be engaged for parallel processing
    - ❖ scoring='f1', indicating the model to chosen from grid search based on the f1-score
- ‘f1-score’ is being used as the ‘scoring policy’ in the grid search operation.
- Following were the best parameters from the grid search operation:

  'ccp_alpha': 0.0, 'criterion': 'friedman_mse', 'learning_rate': 1.0, 'loss': 'deviance', 'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 30, 'subsample': 1.0, 'tol': 0.01, 'validation_fraction': 0.1

- Accuracy scores from best estimator of grid search:

  Train Accuracy: 0.8860 Test Accuracy: 0.8421

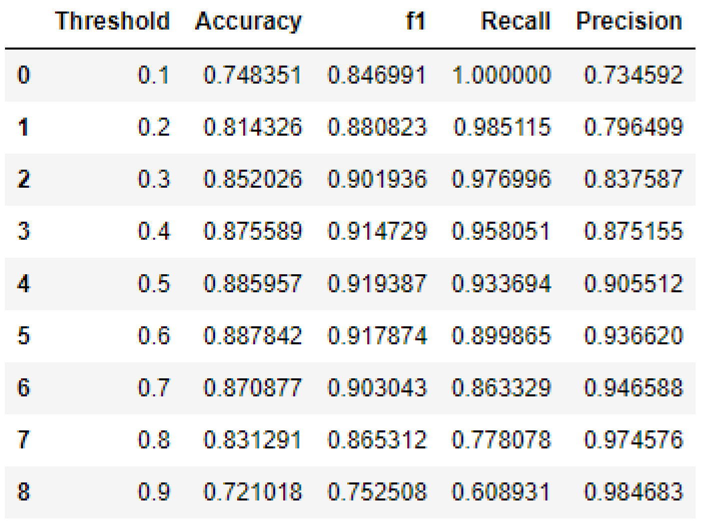
- Accuracy scores before grid search:

Train Accuracy: 0.8860   Test Accuracy: 0.8421

- We see the default parameters were already optimised for the classifier, hence no change in train and test accuracies.

3) <u>Probability Threshold Optimisation</u>

- Custom cut-offs from 0.1 to 0.9 with step increment of 0.1 were tried for probability threshold optimisation.
- The results were compared using the following table:



	Threshold	Accuracy	f1	Recall	Precision
0	0.1	0.748351	0.846991	1.000000	0.734592
1	0.2	0.814326	0.880823	0.985115	0.796499
2	0.3	0.852026	0.901936	0.976996	0.837587
3	0.4	0.875589	0.914729	0.958051	0.875155
4	0.5	0.885957	0.919387	0.933694	0.905512
5	0.6	0.887842	0.917874	0.899865	0.936620
6	0.7	0.870877	0.903043	0.863329	0.946588
7	0.8	0.831291	0.865312	0.778078	0.974576
8	0.9	0.721018	0.752508	0.608931	0.984683

Table 19: Metrics for different thresholds for gradient-boost model

(table generated with 'sklearn' and 'statsmodels' libraries of Python)

Though threshold of 0.6 gives marginally better accuracy than the threshold of 0.5, that gives the best f1-score, so no point in changing the probability threshold.

4) <u>SMOTE</u>

- Over-sampling of the minority class was done using 'SMOTE' of 'imblearn.over_sampling'.
- Train and test shapes before over-sampling:

  Train:(1061, 8)   Test:(456, 8)

- Train and test shapes after over-sampling:

  Train:(1478, 8)   Test:(456, 8)

- Train and test accuracies before over-sampling:

  Train: 0.8860   Test: 0.8421

- Train and test accuracies after over-sampling:

Train: 0.8836 Test: 0.8158

We see a more overfit scenario, after applying SMOTE oversampling.

5) **<u>Adding polynomial features</u>**

- ‘PolynomialFeatures’ from ‘sklearn.preprocessing’ was used to create new polynomial features with a maximum degree of 3. Degrees beyond 4 might result in making the model overfit, and give weird shapes to the distribution of features.
- In this case, 3rd and 4th degree polynomials were also tried which resulted in a more over-fit scenario, so not included in the report.
- Model was fit on the train set.
- Predictions were made both on the train and test sets, and performance metrics were calculated.
- Train and test accuracies before adding polynomial features:

  Train: 0.8860 Test: 0.8421

- Train and test accuracies after adding polynomial features:

  Train: 0.9189 Test: 0.8311

We see a more overfit scenario after adding polynomial features.

So, for gradient-boost, we conclude that the first model (before grid search) is giving the best results.

<u>Bagging(‘BaggingClassifier’ with Decision Tree)</u>

1) **<u>General Model (without pruning and other regularisations):</u>**

- As bagging is a tree based algorithm, feature scaling isn’t required.
- ‘DecisionTreeClassifier’ is being used from ‘sklearn.tree’ to build the base models, ‘BaggingClassifier’ is being used from ‘sklearn.ensemble’ to build the bagging models.
- We will take two decision trees as base models, feed them to ‘BaggingClassifier’ separately:
  - CART1 : simple decision tree with no pruning or regularisation parameter.
  - CART2 : decision tree after grid search operation.
- A grid search operation was carried out for base models using ‘GridSearchCV’ of ‘sklearn.model_selection’. Following was the grid of parameters:

'criterion':['gini', 'entropy'],

'splitter':['random', 'best'],

'max_depth':[7, 4, 5 ,6, 8, 9, 10],

'min_samples_split':[2, 3, 4, 5, 10],

'min_samples_leaf':[7, 4, 5, 6, 8, 9, 10, 11, 12, 13],

'max_features':[6, 5, 7, 7]

- 'GridSearchCV' also took the following parameters:
    - ❖ cv = 3, indicating a 3 fold cross-validation
    - ❖ n_jobs=-1, indicating all the cores to be engaged for parallel processing
    - ❖ scoring='f1', indicating the model to chosen from grid search based on the f1-score
- 'f1-score' is being used as the 'scoring policy' in the grid search operation.
- Following were the best parameters from the grid search operation:

  'criterion': 'gini', 'max_depth': 7, 'max_features': 6, 'min_samples_leaf': 7, 'min_samples_split': 2, 'splitter': 'random'
- Base Models Accuracies:

CART1:	Train Accuracy: 1.0	Test Accuracy: 0.7785
CART2:	Train Accuracy: 0.8407	Test Accuracy: 0.8377

We can see the CART1 base model is overfit and the CART2 base model is right fit.

- Base models were fed into the 'BaggingClassifier' one by one with all other parameters as default.
- Bagging Models Accuracies:

with CART1:	Train Accuracy: 0.9840	Test Accuracy: 0.8421
with CART2:	Train Accuracy: 0.8549	Test Accuracy: 0.8618

We can see the bagging model with CART1 base model is overfit and the one with the CART2 base model is right fit.

2) <u>Grid Search</u>

- A grid search operation was carried out for both bagging models separately using 'GridSearchCV' of 'sklearn.model_selection'. Following was the grid of parameters:

'n_estimators ': [200,10,20,50,5,6,7,8,9,30,100,400,500,250,300,350,100,150],

'max_samples' : [0.3,1.0,0.7,0.5],

'max_features' : [1.0,0.5,0.7,0.3],

'bootstrap' : [True,False],

'bootstrap_features' : [False,True],

'oob_score' : [False,True],

'warm_start' : [False,True]

- 'GridSearchCV' also took the following parameters:
    - ❖ cv = 3, indicating a 3 fold cross-validation
    - ❖ n_jobs=-1, indicating all the cores to be engaged for parallel processing
    - ❖ scoring='f1', indicating the model to chosen from grid search based on the f1-score
- 'f1-score' is being used as the 'scoring policy' in the grid search operation.
- Following were the best parameters from the grid search operation:

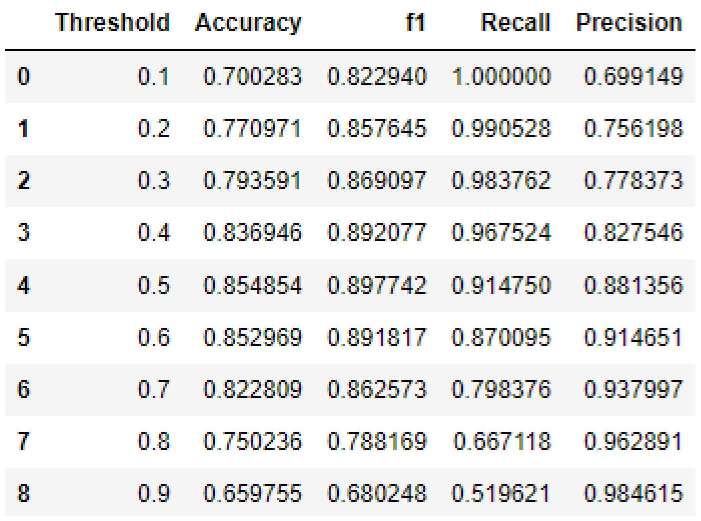
  'bootstrap': True, 'bootstrap_features': False, 'max_features': 1.0, 'max_samples': 0.3, 'n_estimators': 200, 'oob_score': False, 'warm_start': False
- Bagging Models Accuracies after grid search:

with CART1:	Train Accuracy: 0.9123	Test Accuracy: 0.8487
with CART2:	Train Accuracy: 0.8549	Test Accuracy: 0.8575

We can see the bagging model with CART1 base model is overfit and the one with the CART2 base model is right fit. These models are less overfit compared to former models(without grid search).

3) <u>Probability Threshold Optimisation</u>

- The model with CART2 as base model after grid search is being used here.
- Custom cut-offs from 0.1 to 0.9 with step increment of 0.1 were tried for probability threshold optimisation.
- The results were compared using the following table:



	Threshold	Accuracy	f1	Recall	Precision
0	0.1	0.700283	0.822940	1.000000	0.699149
1	0.2	0.770971	0.857645	0.990528	0.756198
2	0.3	0.793591	0.869097	0.983762	0.778373
3	0.4	0.836946	0.892077	0.967524	0.827546
4	0.5	0.854854	0.897742	0.914750	0.881356
5	0.6	0.852969	0.891817	0.870095	0.914651
6	0.7	0.822809	0.862573	0.798376	0.937997
7	0.8	0.750236	0.788169	0.667118	0.962891
8	0.9	0.659755	0.680248	0.519621	0.984615

Table 20: Metrics for different thresholds for bagging model(decision trees)

(table generated with 'sklearn' and 'statsmodels' libraries of Python)

Threshold 0.5 gives the best f1-score as well as accuracy, so no point in changing the probability threshold.

4) <u>SMOTE</u>

- The model with CART2 as base model after grid search is being used here.
- Over-sampling of the minority class was done using 'SMOTE' of 'imblearn.over_sampling'.
- Train and test shapes before over-sampling:

  Train:(1061, 8)   Test:(456, 8)

- Train and test shapes after over-sampling:

  Train:(1478, 8)   Test:(456, 8)

- Train and test accuracies before over-sampling:

  Train: 0.8549   Test: 0.8575

- Train and test accuracies after over-sampling:

  Train: 0.8525   Test: 0.8399

We see an overfit scenario, after applying SMOTE oversampling.

5) <u>Adding polynomial features</u>

- The model with CART2 as base model after grid search is being used here.
- 'PolynomialFeatures' from 'sklearn.preprocessing' was used to create new polynomial features with a maximum degree of 3. Degrees beyond 4 might result in making the model overfit, and give weird shapes to the distribution of features.
- In this case, 4th degree polynomials were also tried which resulted in comparable results with a model with a maximum of 3rd degree polynomials, so not included in the report.
- Model was fit on the train set.
- Predictions were made both on the train and test sets, and performance metrics were calculated.
- Train and test accuracies before adding polynomial features:

  Train: 0.8549   Test: 0.8575

- Train and test accuracies after adding polynomial features:

  Train: 0.8483   Test: 0.8268

We see a marginal overfit scenario after adding polynomial features, with reduction in train and test accuracies.

So, for bagging with decision trees, we conclude that the model with CART2 as base model after grid search, is giving the best results.

<u>Bagging('BaggingClassifier' with Random Forest)</u>

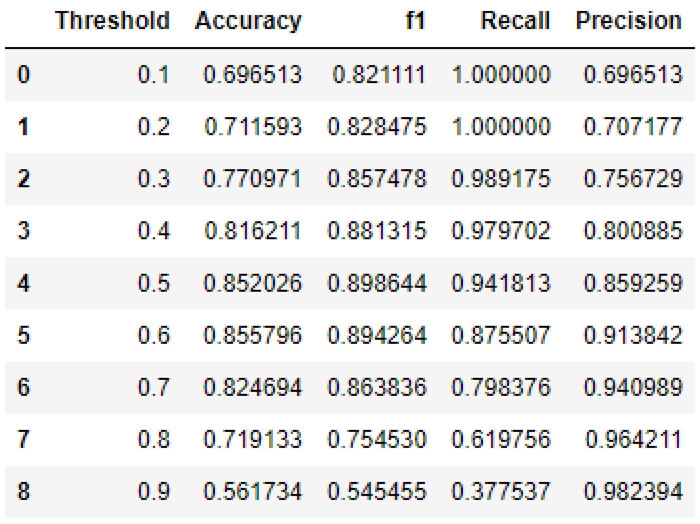
1) **<u>Grid Search</u>**

- We will use the same parameters for 'BaggingClassifier' as found in the previous grid search operation, which gave us the right fit scenario.
- For the base model, we will use random forest with parameters same as found in the grid search that gave the right fit during random forest grid search operation.
- Bagging Model Accuracies:

Train Accuracy: 0.8520 Test Accuracy: 0.8530

The model is right fit, but the accuracies have gone down as compared with the corresponding random forest and bagging with decision trees models.

2) **<u>Probability Threshold Optimisation</u>**
3) Custom cut-offs from 0.1 to 0.9 with step increment of 0.1 were tried for probability threshold optimisation.
4) The results were compared using the following table:



	Threshold	Accuracy	f1	Recall	Precision
0	0.1	0.696513	0.821111	1.000000	0.696513
1	0.2	0.711593	0.828475	1.000000	0.707177
2	0.3	0.770971	0.857478	0.989175	0.756729
3	0.4	0.816211	0.881315	0.979702	0.800885
4	0.5	0.852026	0.898644	0.941813	0.859259
5	0.6	0.855796	0.894264	0.875507	0.913842
6	0.7	0.824694	0.863836	0.798376	0.940989
7	0.8	0.719133	0.754530	0.619756	0.964211
8	0.9	0.561734	0.545455	0.377537	0.982394

Table 21: Metrics for different thresholds for bagging model(random forest)

(table generated with 'sklearn' and 'statsmodels' libraries of Python)

Threshold of 0.5 gives the best f1-score, whereas threshold of 0.6 gives the best accuracy. As the changes are very marginal and default 0.5 threshold has the best f1-score, so no point in changing the probability threshold.

3) <u>SMOTE</u>

- Over-sampling of the minority class was done using 'SMOTE' of 'imblearn.over_sampling'.
- Train and test shapes before over-sampling:

Train:(1061, 8) Test:(456, 8)

- Train and test shapes after over-sampling:

  Train:(1478, 8) Test:(456, 8)

- Train and test accuracies before over-sampling:

  Train: 0.8520 Test: 0.8530

- Train and test accuracies after over-sampling:

  Train: 0.8539 Test: 0.8399

We see a marginal drop in test accuracy after applying SMOTE oversampling. Above seems to be a marginal shift towards overfit from rightfit.

4) <u>Adding polynomial features</u>

- 'PolynomialFeatures' from 'sklearn.preprocessing' was used to create new polynomial features with a maximum degree of 3. Degrees beyond 4 might result in making the model overfit, and give weird shapes to the distribution of features.
- In this case, 4th degree polynomials were also tried which resulted in an over-fit scenario, so not included in the report.
- Model was fit on the train set.
- Predictions were made both on the train and test sets, and performance metrics were calculated.
- Train and test accuracies before adding polynomial features:

  Train: 0.8520 Test: 0.8530

- Train and test accuracies after adding polynomial features:

  Train: 0.8558 Test: 0.8333

We see a drop in test accuracy after adding polynomial features, but train accuracy has increased, marginally though.

So, for bagging with random forest, we conclude that the model after grid search is giving the best results, with the base model as the random forest found after grid search.

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.

{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, classification report (4 pts) Final Model - Compare and comment on all models on the basis of the performance metrics in a structured tabular manner. Describe which model is best/optimized, After comparison which model suits the best for the problem in hand on the basis of different measures. Comment on the final model.(3 pts)}

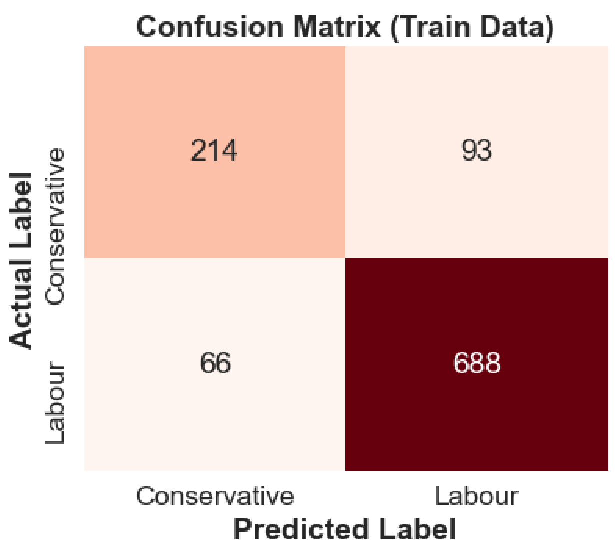
Let us compare the performance metrics of the best models of various algorithms in a tabular format:

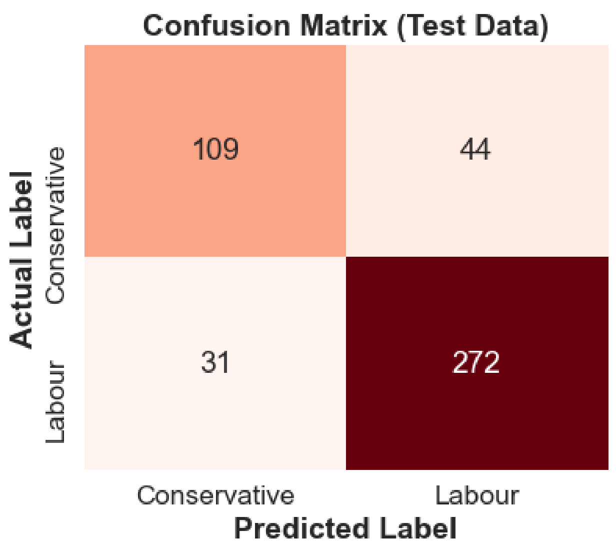
		Accuracy	AUC	Precision		Recall		F1-score	
				Conser vative	Labour	Conser vative	Labour	Conser vative	Labour
Logistic Regression	Train	0.85	0.910	0.76	0.88	0.70	0.91	0.73	0.90
	Test	0.84	0.884	0.78	0.86	0.71	0.90	0.74	0.88
LDA	Train	0.85	0.917	0.76	0.89	0.71	0.91	0.73	0.90
	Test	0.85	0.898	0.80	0.87	0.73	0.91	0.76	0.89
KNN	Train	0.86	0.931	0.79	0.89	0.73	0.92	0.76	0.90
	Test	0.85	0.878	0.77	0.88	0.72	0.91	0.75	0.89
Naïve Bayes	Train	0.82	0.873	0.70	0.87	0.70	0.87	0.70	0.87
	Test	0.86	0.912	0.79	0.88	0.72	0.92	0.75	0.90
Random Forest	Train	0.86	0.926	0.83	0.87	0.69	0.94	0.75	0.90
	Test	0.87	0.920	0.85	0.87	0.67	0.95	0.75	0.91
Ada Boost	Train	0.85	0.937	0.78	0.88	0.70	0.92	0.74	0.90
	Test	0.82	0.892	0.76	0.85	0.68	0.89	0.72	0.87
Gradient Boost	Train	0.92	0.970	0.90	0.93	0.81	0.96	0.85	0.94
	Test	0.83	0.894	0.79	0.85	0.68	0.91	0.73	0.88
Bagging with Decision Trees	Train	0.85	0.921	0.82	0.86	0.61	0.95	0.70	0.90
	Test	0.83	0.901	0.86	0.82	0.58	0.95	0.69	0.88
Bagging with Random Forest	Train	0.86	0.927	0.87	0.85	0.59	0.96	0.70	0.90
	Test	0.83	0.898	0.89	0.82	0.58	0.96	0.70	0.88

Table 22: Metrics Comparison

(values imported from Jupyter Notebook)

Logistic Regression



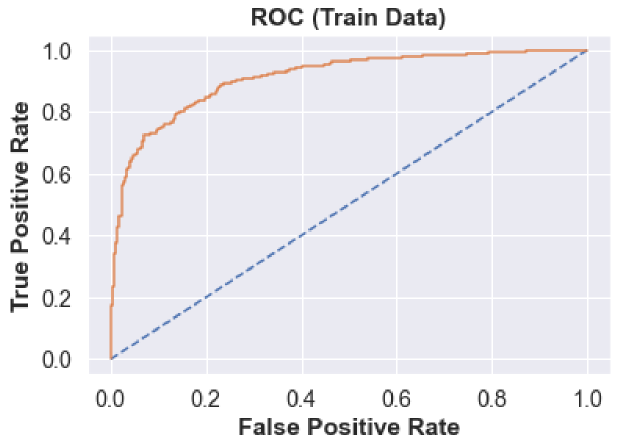


Training Classification Report:

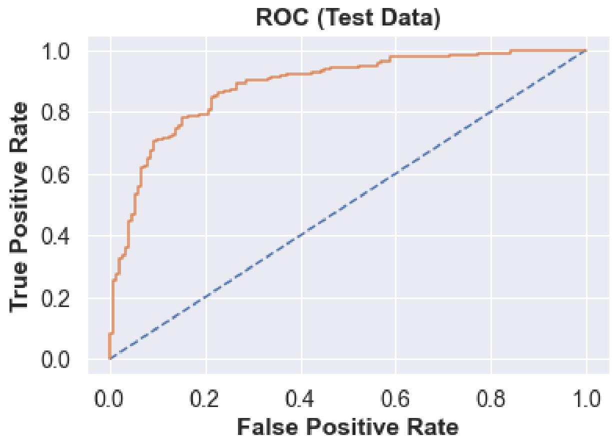
```
              precisionrecallf1-scoresupport

Conservative       0.76   0.70     0.73    307
      Labour       0.88   0.91     0.90    754

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

Testing Classification Report:

```
              precisionrecallf1-scoresupport

Conservative       0.78   0.71     0.74    153
      Labour       0.86   0.90     0.88    303

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

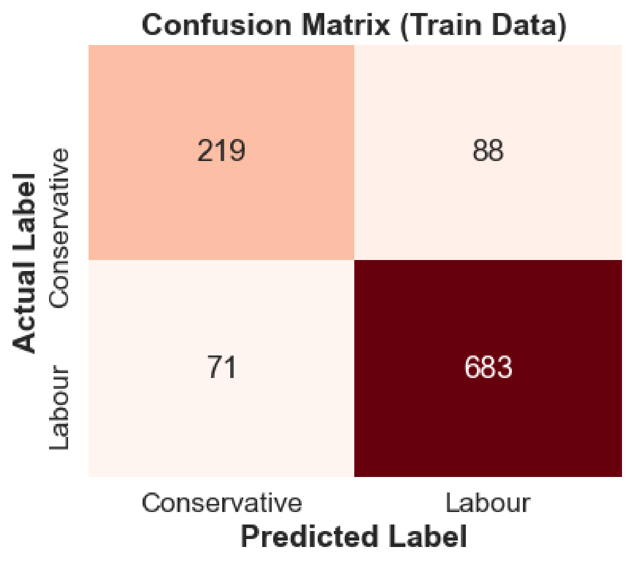


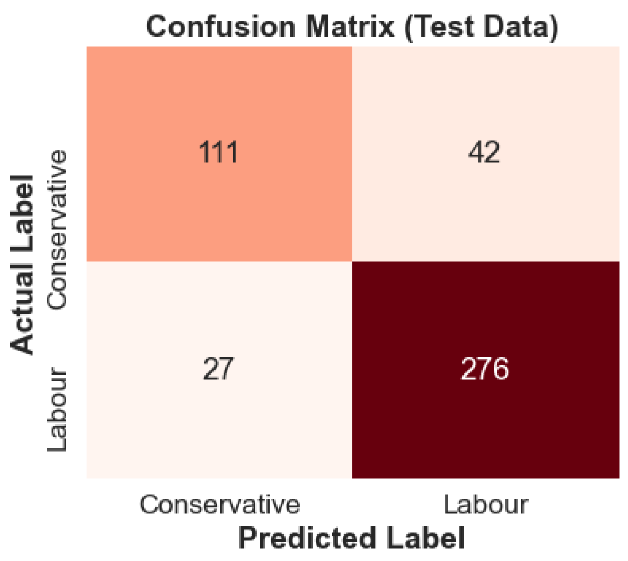
AUC: 0.910



AUC: 0.884

Linear Discriminant Analysis



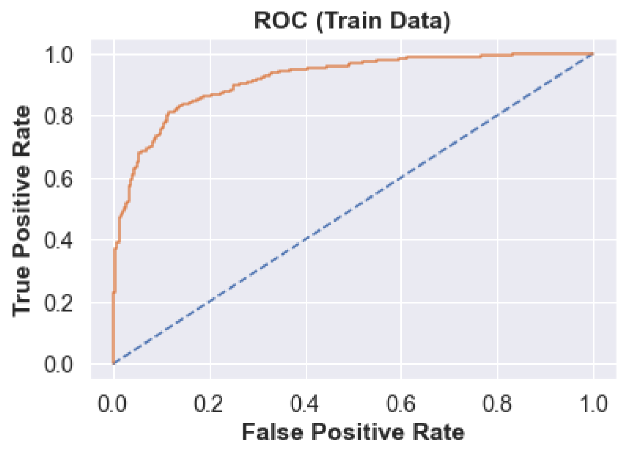


Training Classification Report:

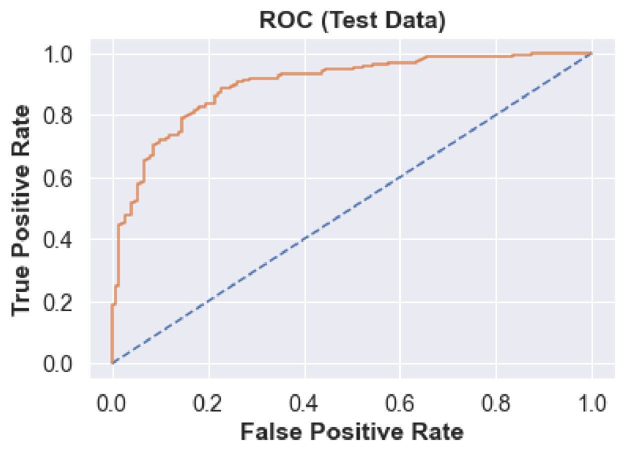
```
              precisionrecallf1-scoresupport

Conservative       0.76   0.71      0.73     307
      Labour       0.89   0.91      0.90     754

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

Testing Classification Report:

```
              precisionrecallf1-scoresupport

Conservative       0.80   0.73      0.76     153
      Labour       0.87   0.91      0.89     303

    accuracy                        0.85     456
   macro avg       0.84   0.82      0.83     456
weighted avg       0.85   0.85      0.85     456
```

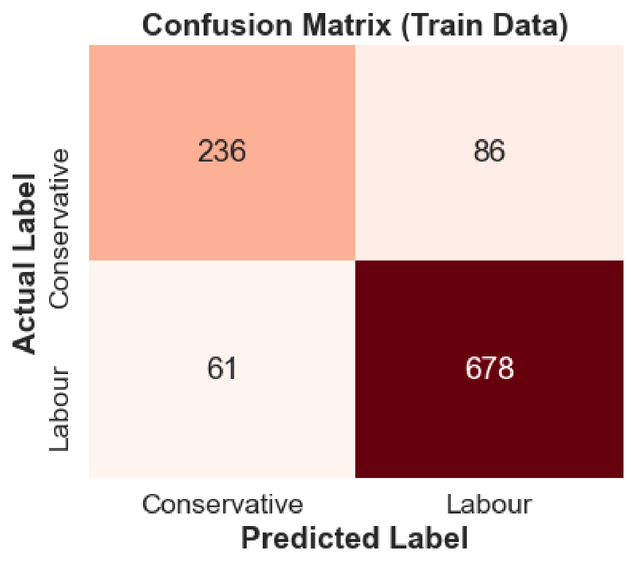


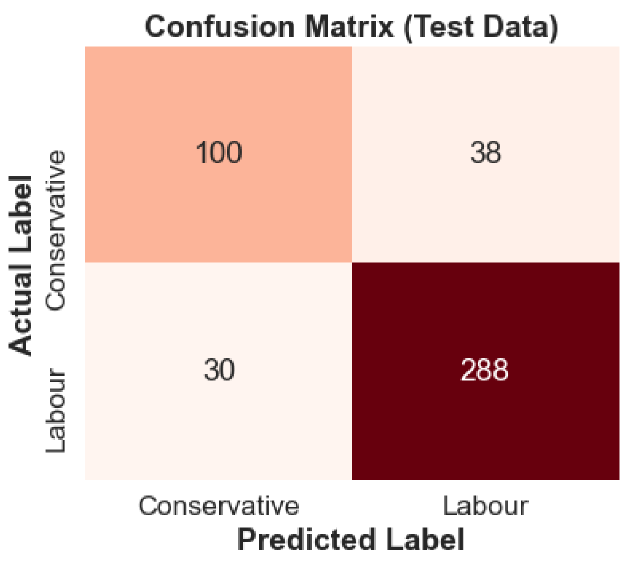
AUC: 0.917



AUC: 0.898

K Nearest Neighbours



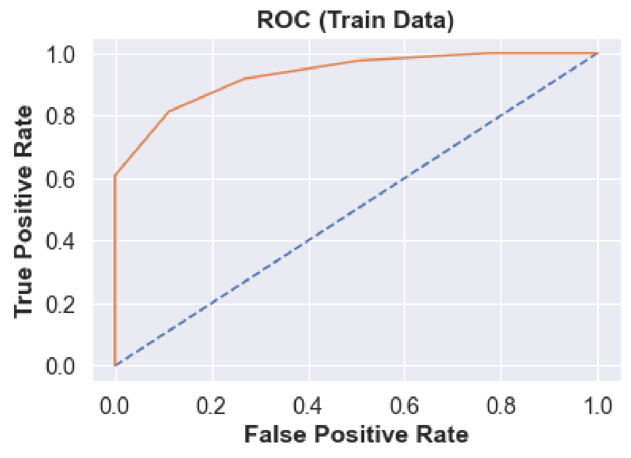


Training Classification Report:

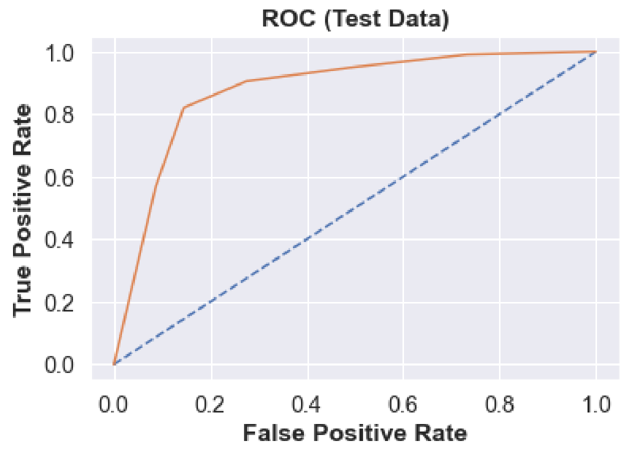
```
              precisionrecallf1-scoresupport

Conservative       0.79   0.73     0.76     322
      Labour       0.89   0.92     0.90     739

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

Testing Classification Report:

```
              precisionrecallf1-scoresupport

Conservative       0.77   0.72     0.75     138
      Labour       0.88   0.91     0.89     318

    accuracy                       0.85     456
   macro avg       0.83   0.82     0.82     456
weighted avg       0.85   0.85     0.85     456
```

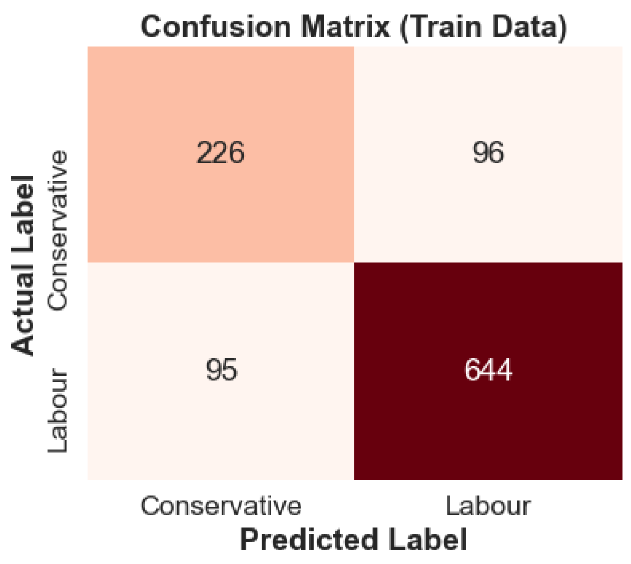


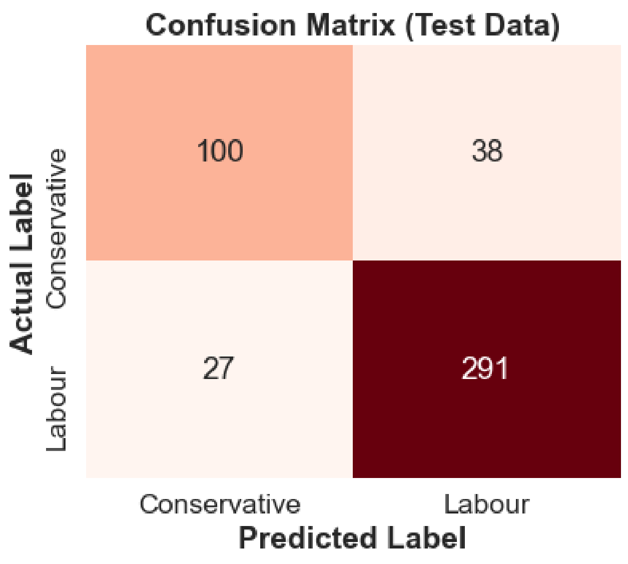
AUC: 0.931



AUC: 0.878

Naïve Bayes



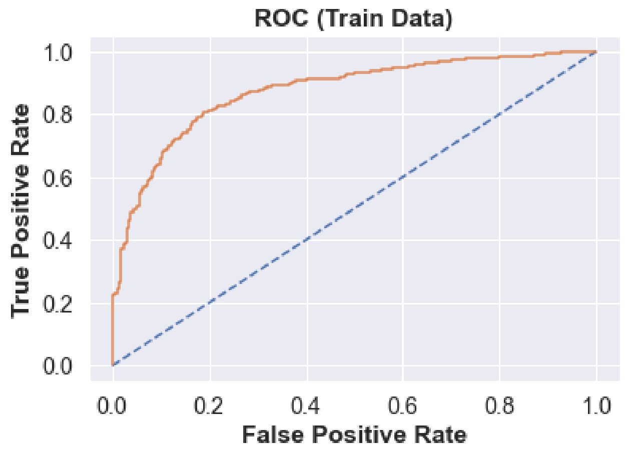


Training Classification Report:

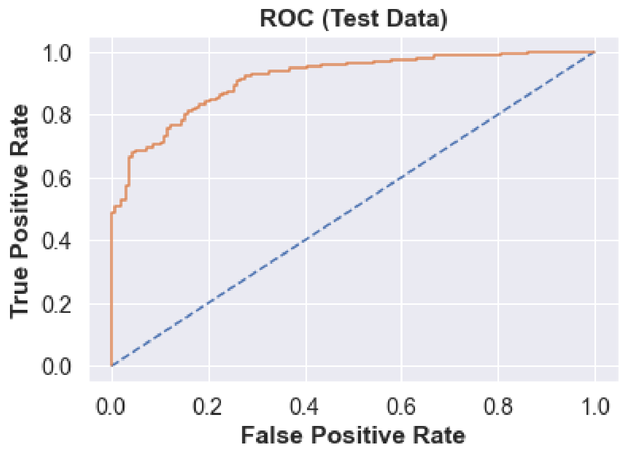
Testing Classification Report:

```
              precisionrecallf1-scoresupport

Conservative       0.70   0.70      0.70     322
      Labour       0.87   0.87      0.87     739

    accuracy                        0.82    1061
   macro avg       0.79   0.79      0.79    1061
weighted avg       0.82   0.82      0.82    1061
```

```
              precisionrecallf1-scoresupport

Conservative       0.79   0.72      0.75     138
      Labour       0.88   0.92      0.90     318

    accuracy                        0.86     456
   macro avg       0.84   0.82      0.83     456
weighted avg       0.86   0.86      0.86     456
```

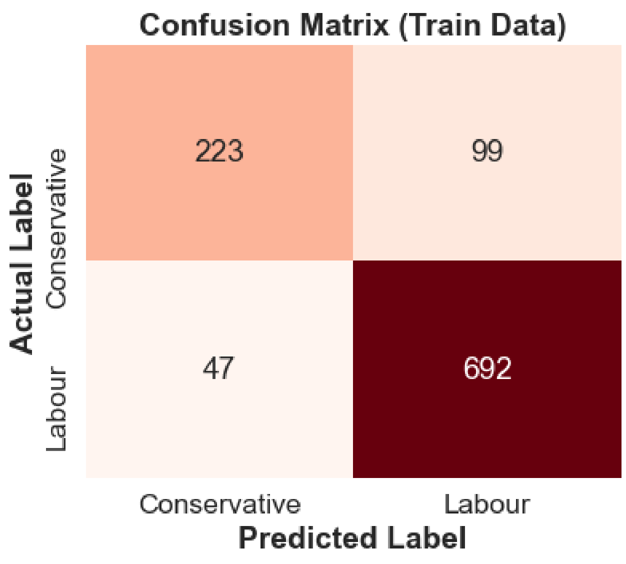


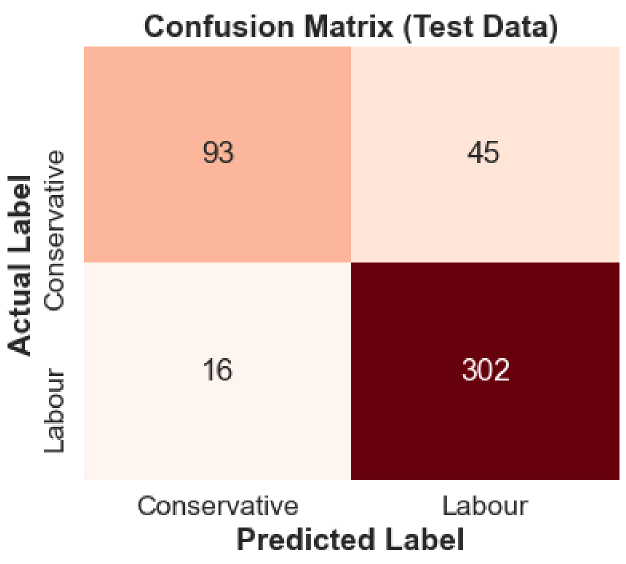
AUC: 0.873



AUC: 0.912

Random Forest



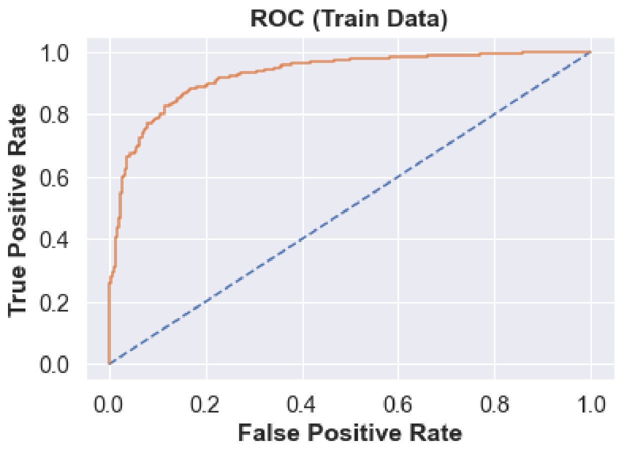


Training Classification Report:

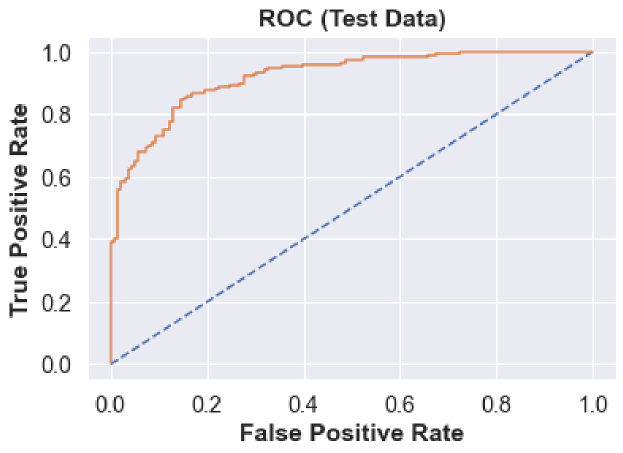
```
              precisionrecallf1-scoresupport

Conservative       0.83  0.69     0.75     322
      Labour       0.87  0.94     0.90     739

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

Testing Classification Report:

```
              precisionrecallf1-scoresupport

Conservative       0.85  0.67     0.75     138
      Labour       0.87  0.95     0.91     318

    accuracy                      0.87     456
   macro avg       0.86  0.81     0.83     456
weighted avg       0.87  0.87     0.86     456
```

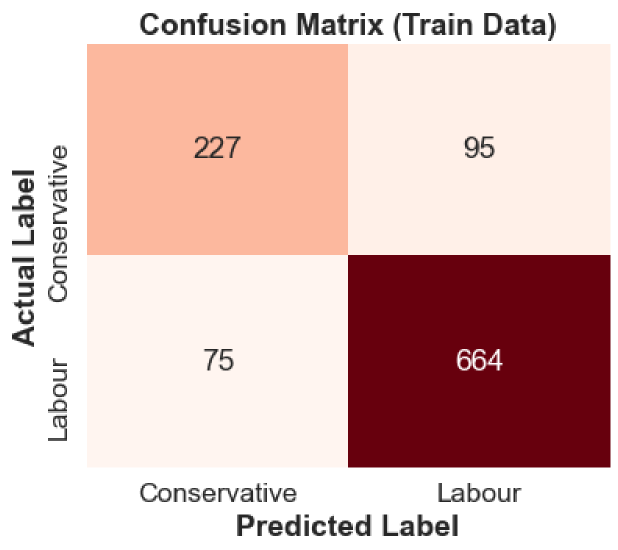


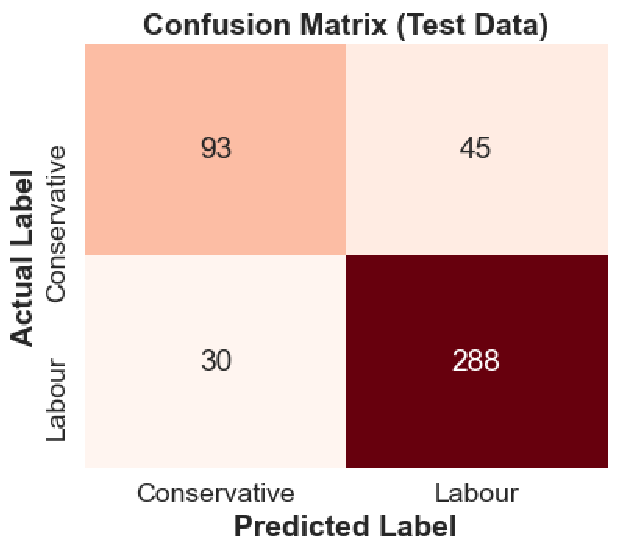
AUC: 0.926



AUC: 0.920

Ada Boost



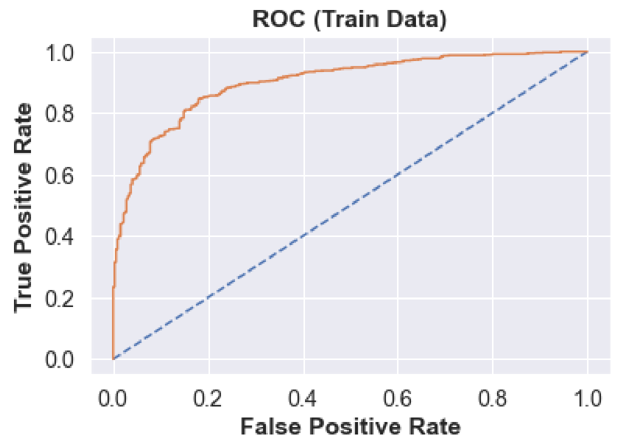


Training Classification Report:

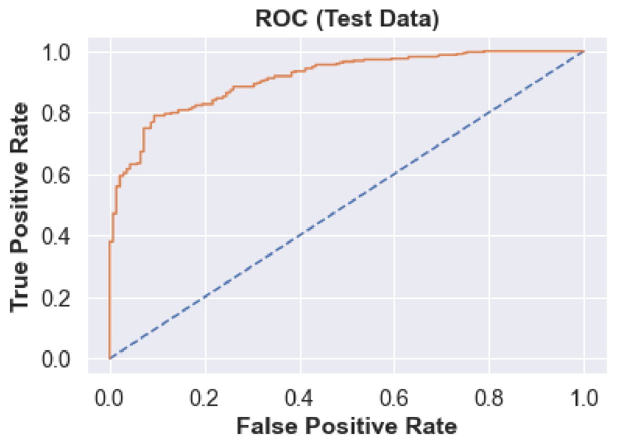
```
              precisionrecallf1-scoresupport

Conservative      0.75  0.70     0.73     322
      Labour      0.87  0.90     0.89     739

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

Testing Classification Report:

```
              precisionrecallf1-scoresupport

Conservative      0.76  0.67     0.71     138
      Labour      0.86  0.91     0.88     318

    accuracy                     0.84     456
   macro avg      0.81  0.79     0.80     456
weighted avg      0.83  0.84     0.83     456
```

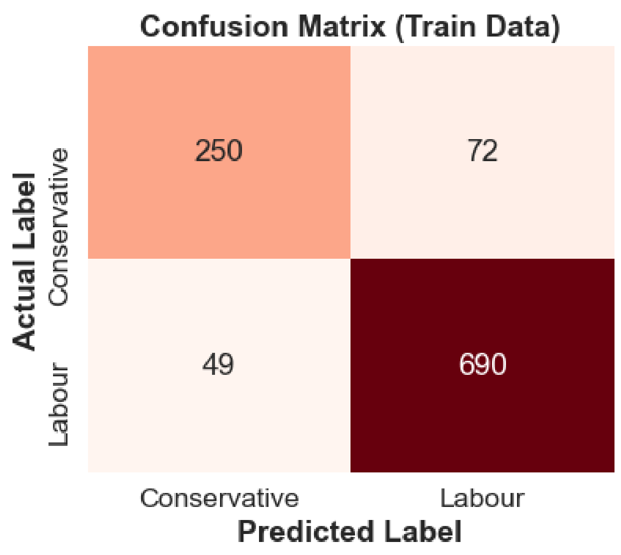


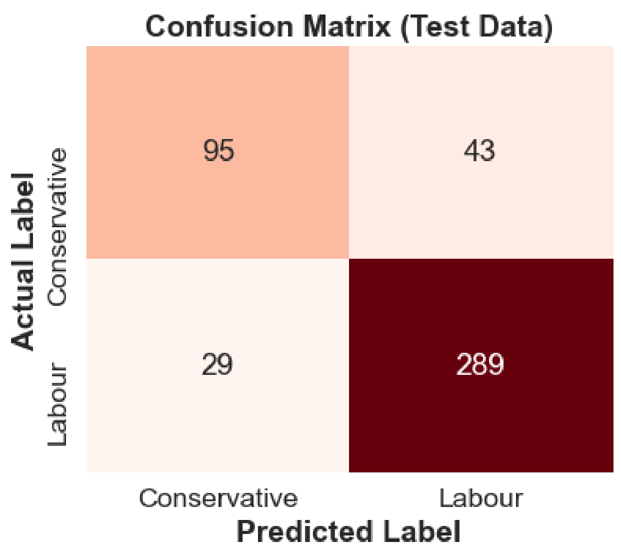
AUC: 0.900



AUC: 0.910

Gradient Boost



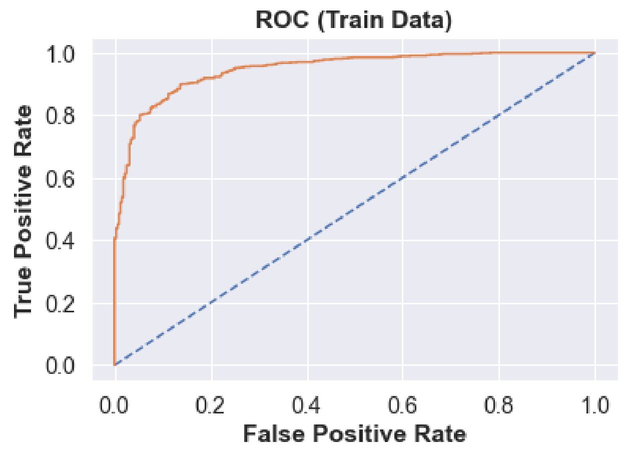


Training Classification Report:

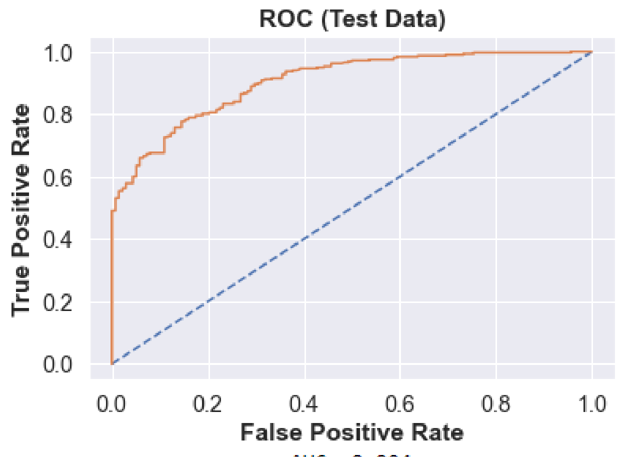
```
             precisionrecallf1-scoresupport
Conservative      0.84   0.78     0.81     322
      Labour      0.91   0.93     0.92     739

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

Testing Classification Report:

```
             precisionrecallf1-scoresupport
Conservative      0.77   0.69     0.73     138
      Labour      0.87   0.91     0.89     318

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

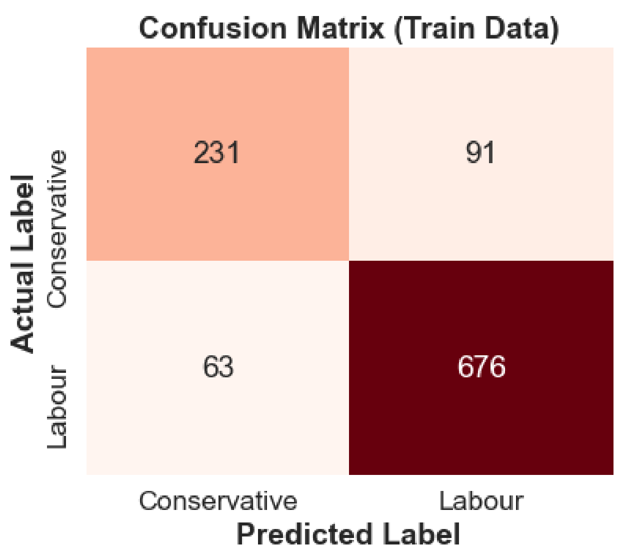


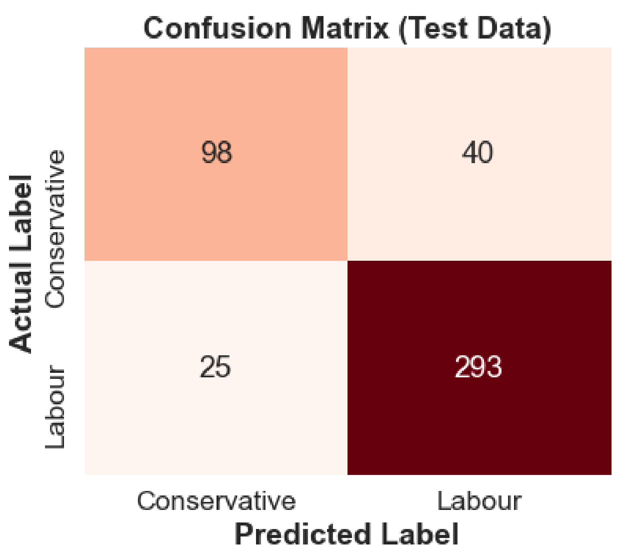
AUC: 0.947



AUC: 0.904

Bagging with Decision Trees



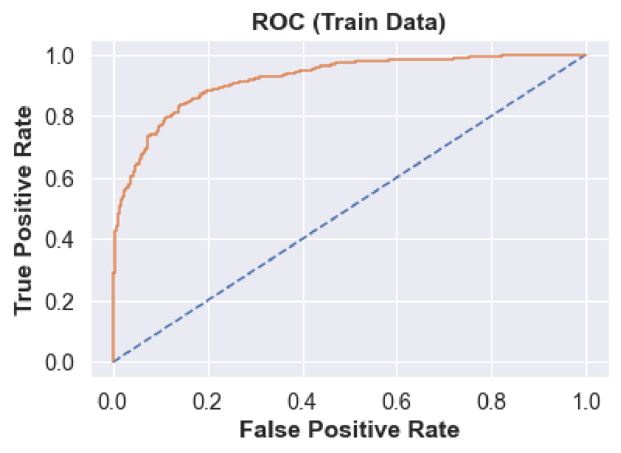


Training Classification Report:

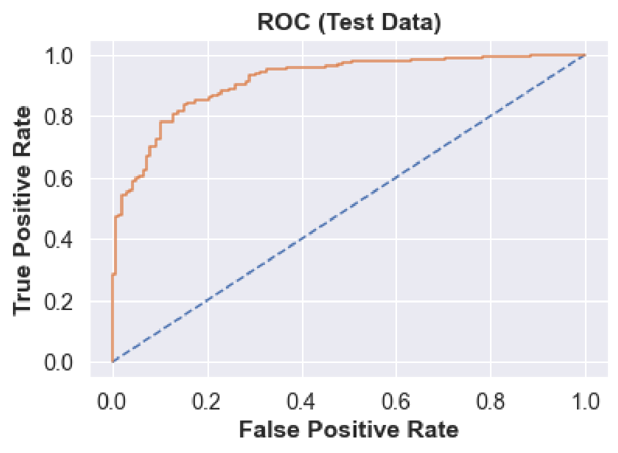
```
              precisionrecallf1-scoresupport

Conservative      0.79    0.72     0.75     322
      Labour      0.88    0.91     0.90     739

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

Testing Classification Report:

```
              precisionrecallf1-scoresupport

Conservative      0.80    0.71     0.75     138
      Labour      0.88    0.92     0.90     318

    accuracy                       0.86     456
   macro avg      0.84    0.82     0.83     456
weighted avg      0.85    0.86     0.86     456
```

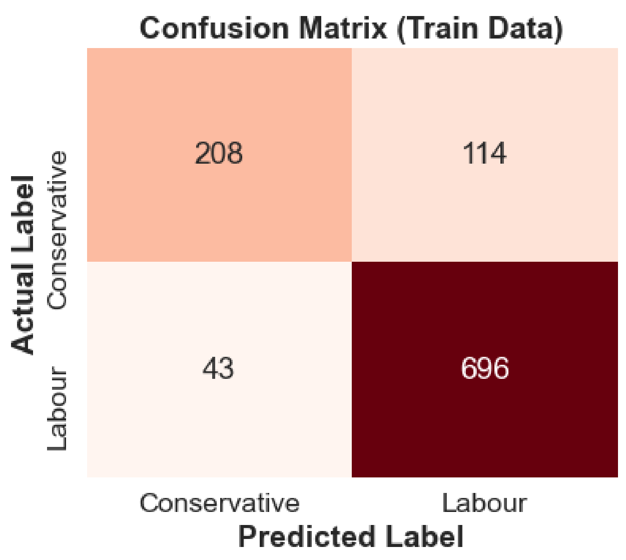


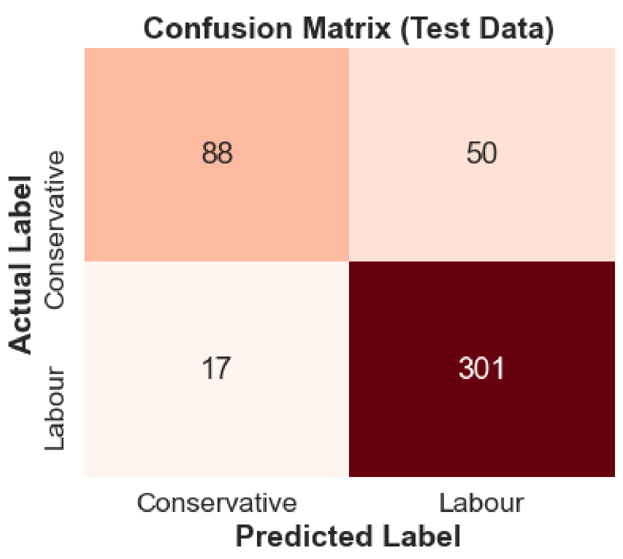
AUC: 0.920



AUC: 0.914

Bagging with Random Forests



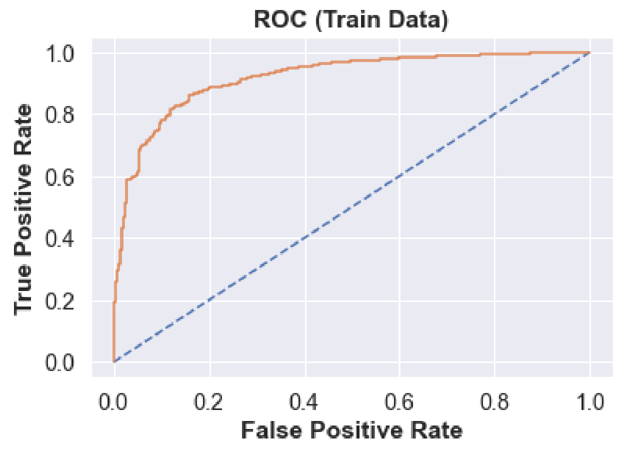


<u>**Training Classification Report:**</u>

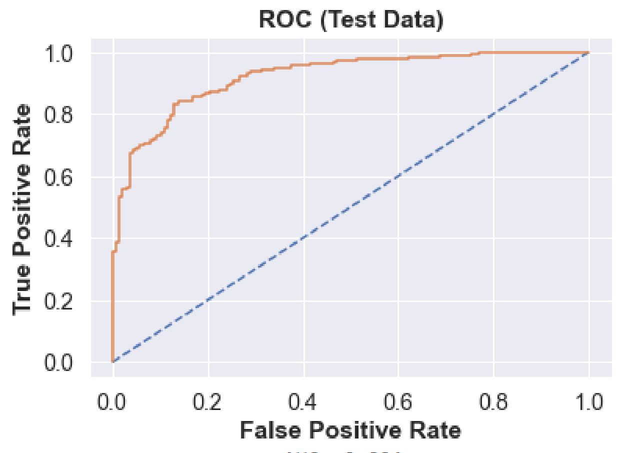
```
                precisionrecallf1-scoresupport

Conservative      0.83   0.65      0.73      322
      Labour      0.86   0.94      0.90      739

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

<u>**Testing Classification Report:**</u>

```
                precisionrecallf1-scoresupport

Conservative      0.84   0.64      0.72      138
      Labour      0.86   0.95      0.90      318

    accuracy                       0.85      456
   macro avg      0.85   0.79      0.81      456
weighted avg      0.85   0.85      0.85      456
```



AUC: 0.918



AUC: 0.921

- Accuracy speaks about the proportion of correct mapping made by the prediction model. Even more so, a test can have a high accuracy but actually perform worse than a test with a lower accuracy.
- Whilst accuracy score is fine for balanced classes,it can be very misleading for unbalanced classes.

- AUC is **scale-invariant**. It measures how well predictions are ranked, rather than their absolute values.
- AUC is **classification-threshold-invariant**. It measures the quality of the model's predictions irrespective of what classification threshold is chosen.
- Recall tells what proportion of actual positives was identified correctly.
- A model that produces no false negatives has a recall of 1.0.
- Precision tells what proportion of positive identifications was actually correct.
- A model that produces no false positives has a precision of 1.0.
- It is difficult to compare two models with different Precision and Recall. So to make them comparable, we use F1-Score.

From the table we see the models Random Forest, Naïve Bayes and LDA are giving similar accuracies:

Random Forest:	Train Accuracy: 0.86	Test Accuracy: 0.87
Naïve Bayes:	Train Accuracy: 0.86	Test Accuracy: 0.85
LDA:	Train Accuracy: 0.85	Test Accuracy: 0.85

But, we can go ahead with the random forest model, because of following advantages of random forests over decision trees:

- Randomness and voting mechanisms
- Trees are unpruned(unless specified)
- Trees are diverse
- Handling overfitting

If we want our model to generalise better on unseen data, we can go ahead with bagging model with random forests as base models that have more randomness in terms of trees than a simple random forest(though the test accuracy is slightly lower than random forest, it is expected to generalise better on unseen data):

Bagging with Random Forest:	Train Accuracy: 0.86	Test Accuracy: 0.83

1.8 Based on these predictions, what are the insights?

{ Based on your analysis and working on the business problem, detail out appropriate insights and recommendations to help the management solve the business objective. There should be at least 3-4 Recommendations and insights in total. Recommendations should be easily understandable and business specific, students should not give any technical suggestions. Full marks should only be allotted if the recommendations are correct and business specific.}

Feature importance is calculated as the decrease in node impurity weighted by the probability of reaching that node. The node probability can be calculated by the number of samples that

reach the node, divided by the total number of samples. The higher the value the more important the feature.
Let us have a look at the feature importances of Random Forest model:

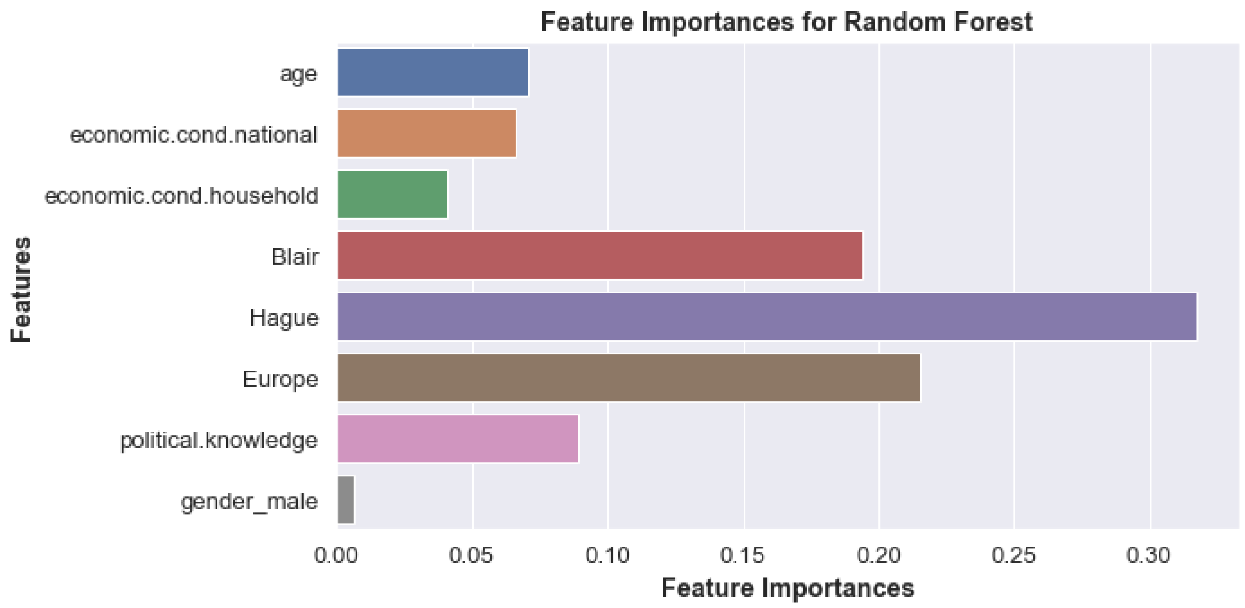


Figure 18: Feature Importances (Random Forest)

(plot generated using Python)

- We see the feature ‘Hague’ has the highest feature importance of all, making it the most significant predictor in the random forest model. ‘Hague’ has the assessment of the Conservative Party leader William Hague, from 1 to 5.
- Followed by ‘Hague’, the next significant predictors in the random forest model are ‘Europe’ and ‘Blair’. ‘Europe’ is an 11-point scale that measures respondents' attitudes toward European integration. High scores represent ‘Eurosceptic’ sentiment. ‘Blair’ has the assessment of the Labour Party leader Tony Blair, from 1 to 5.
- ‘gender_male’ has a very low feature importance indicating the gender of voter did not help in predicting the choice of party.

For the logistic regression models, the first model before grid search seems to be the best for interpreting the model coefficients. Following is the model intercept and coefficients for the same:

Intercept: 3.3269

Coefficient for 'age' is -0.0133.

Coefficient for 'economic.cond.national' is 0.3469.

Coefficient for 'economic.cond.household' is 0.0346.

Coefficient for 'Blair' is 0.5491.

Coefficient for 'Hague' is -0.8554.

Coefficient for 'Europe' is -0.2079.

Coefficient for 'political.knowledge' is -0.4032.

Coefficient for 'gender_male' is 0.0496.

From the logistic regression coefficients and intercept we can write the discriminant score, DS as:

DS = 3.33 + (age)(-0.01) + (economic.cond.national)(0.35) + (economic.cond.household)(0.03) + (Blair)(0.55) + (Hague)(-0.86) + (Europe)(-0.21) + (political.knowledge)(-0.40) + (gender_male)(0.04)

From the coefficients, we can conclude about the important factors:

- We see the feature ‘Hague’ has the highest influence on predictions(has the highest magnitude of coefficient, also has the highest negative coefficient of -0.86). ‘Hague’ has the assessment of the Conservative Party leader William Hague, from 1 to 5. A negative coefficient indicates, if the rating goes up by 1, the DS goes down by 0.86.
- ‘Blair’ has the highest positive coefficient, and second highest magnitude of coefficient, making it the most significant predictor after ‘Hague’. ‘Blair’ has the assessment of the Labour Party leader Tony Blair, from 1 to 5. A positive coefficient of 0.55 indicates, if the rating goes up by 1, the DS goes up by 0.55.
- ‘political.knowledge’, ‘economic.cond.national’ and ‘Europe’ have significant contributions in predicting the winning party too.
- Features ‘age’ and ‘gender_male’ have very low coefficients indicating age and gender of the voter weren’t of much significance in prediction as far as linear discriminant analysis is concerned.

Let us also have a look at the feature importances of Ada Boost and Gradient Boost:

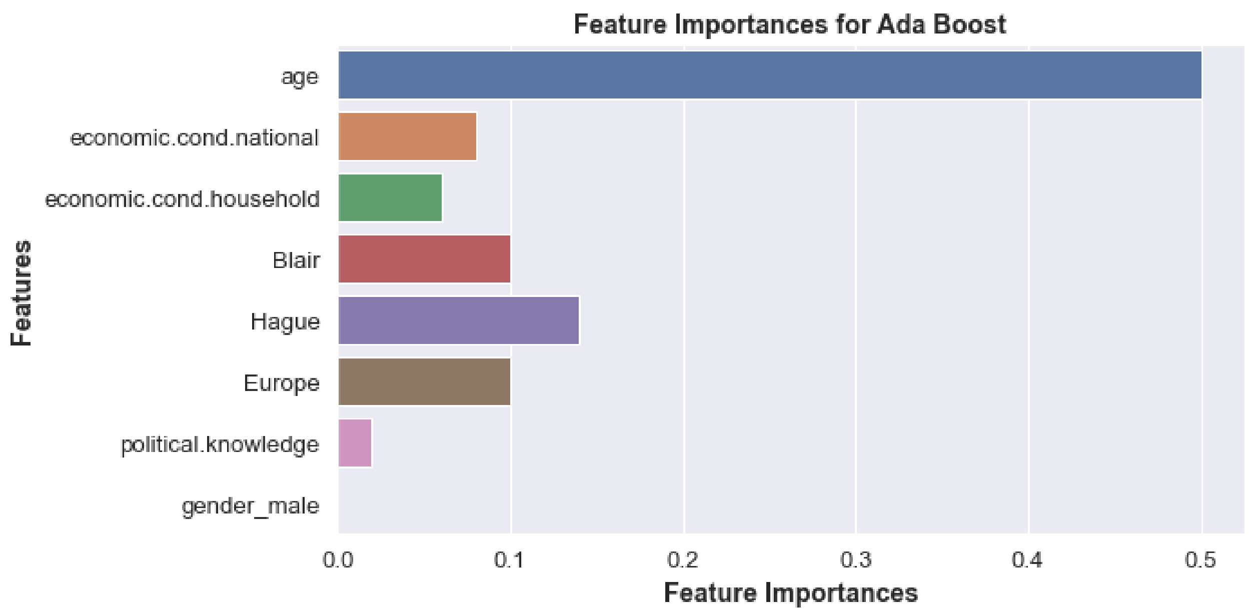


Figure 19: Feature Importances (Ada Boost)

(plot generated using Python)

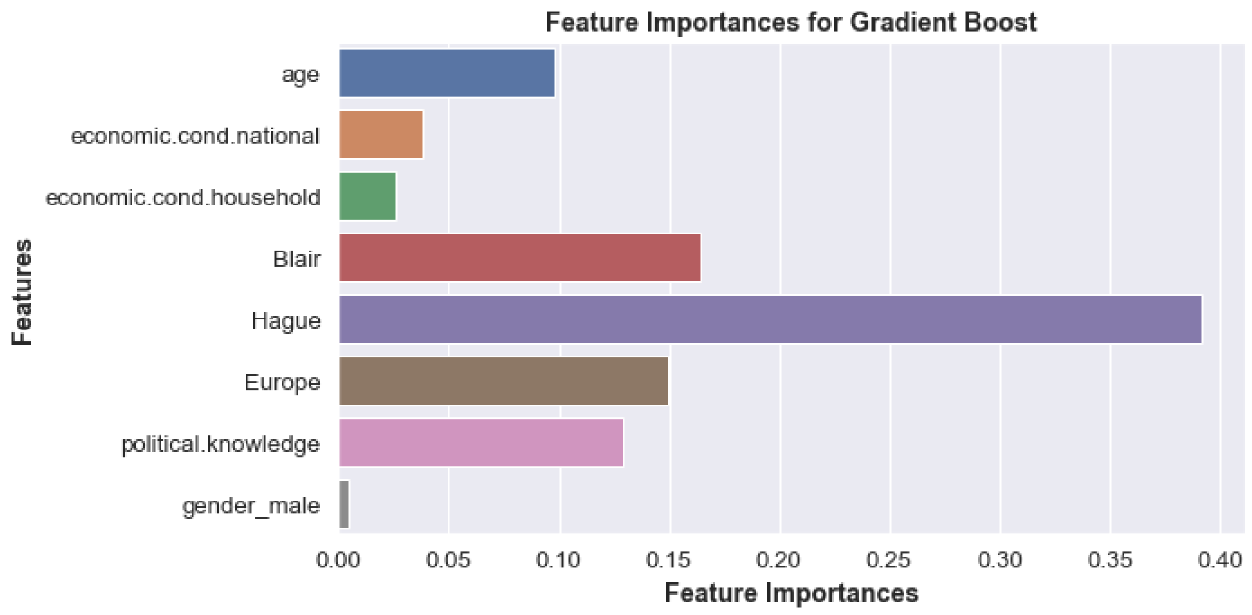
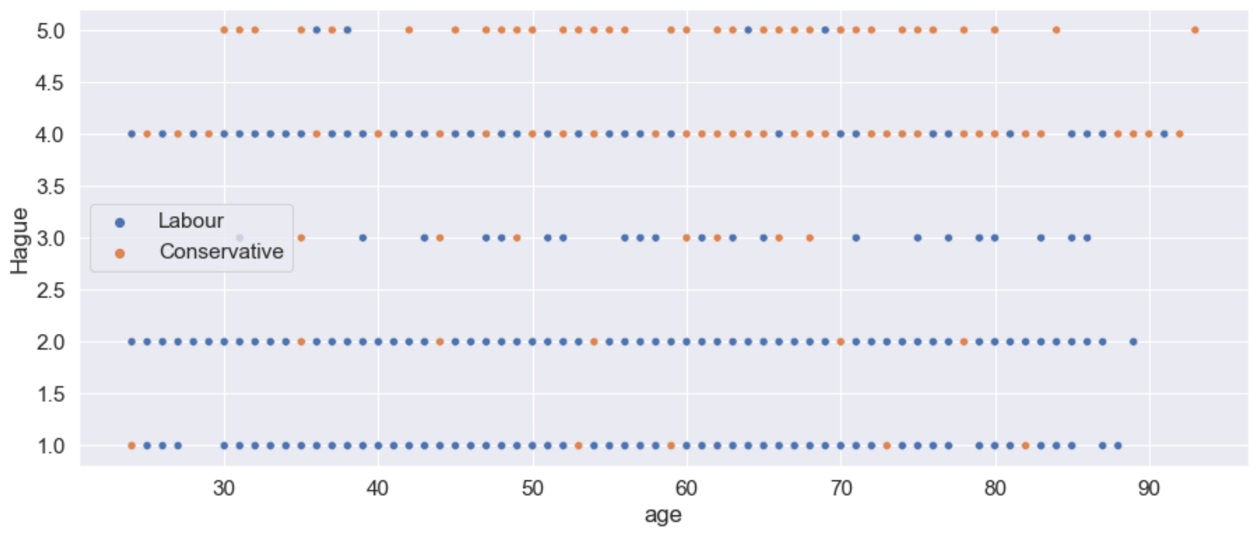


Figure 20: Feature Importances (Gradient Boost)

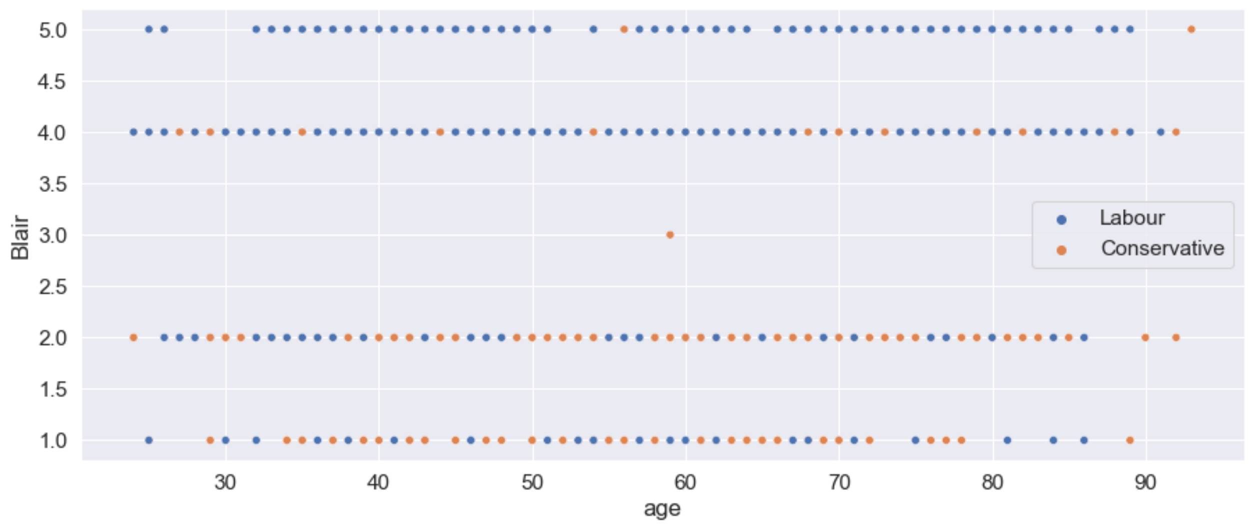
(plot generated using Python)

Let us have a look at a scatter plot of ‘Hague’ vs ‘age’ hued over ‘vote’:



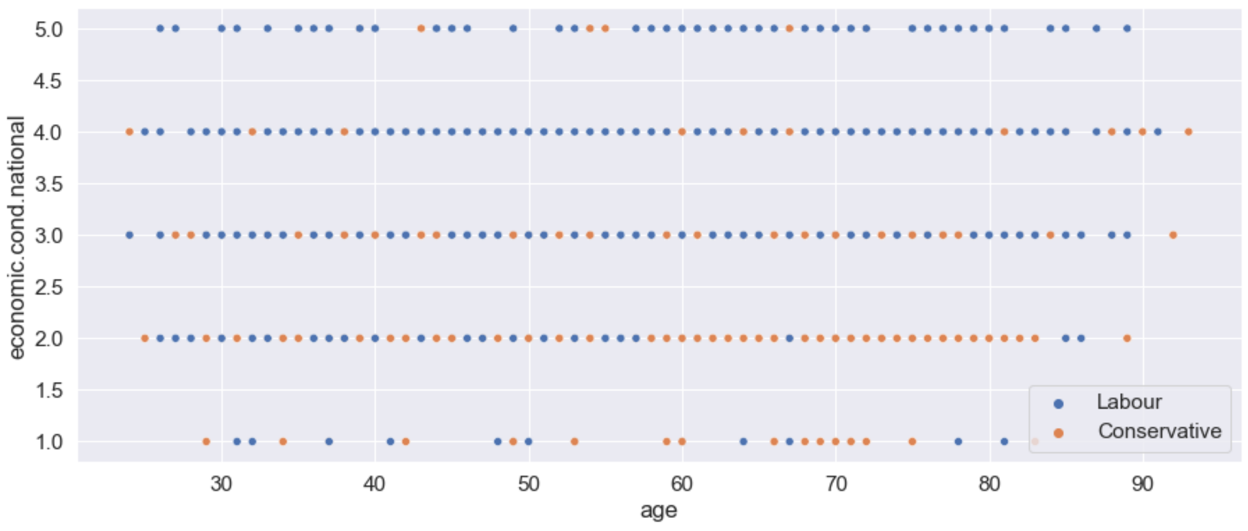
- We see significant number of people who have rated William Hague(Conservative Party) with 3 or 4, have actually voted for Tony Blair(Labour Party)

Let us have a look at a scatter plot of ‘Blair’ vs ‘age’ hued over ‘vote’:



- We see few people who have rated Tony Blair(Labour Party) with 1 or 1, have actually voted for him, maybe because of dislike for William Hague(Conservative Party).

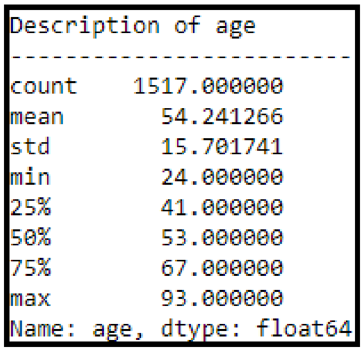
Let us have a look at a scatter plot of ‘economic.cond.national’ vs ‘age’ hued over ‘vote’:



- The voters in favour Conservative party can be seen with an assessment score of 1 or 2 for current national economic conditions.

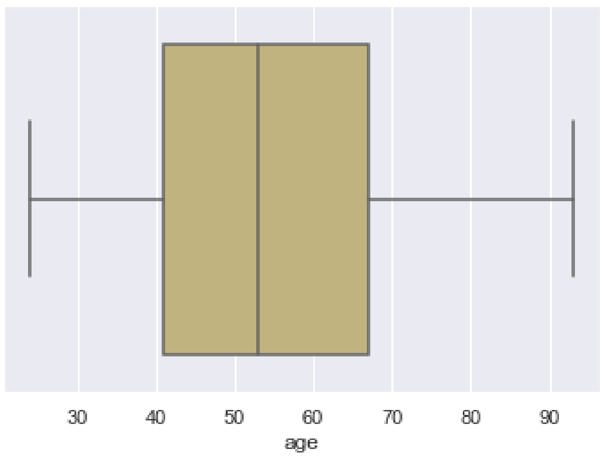
<u>EDA Insights</u>

<u>age</u>



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

```
BoxPlot of age
------------------------------
```



```
Distribution of age
-------------------------------------
```

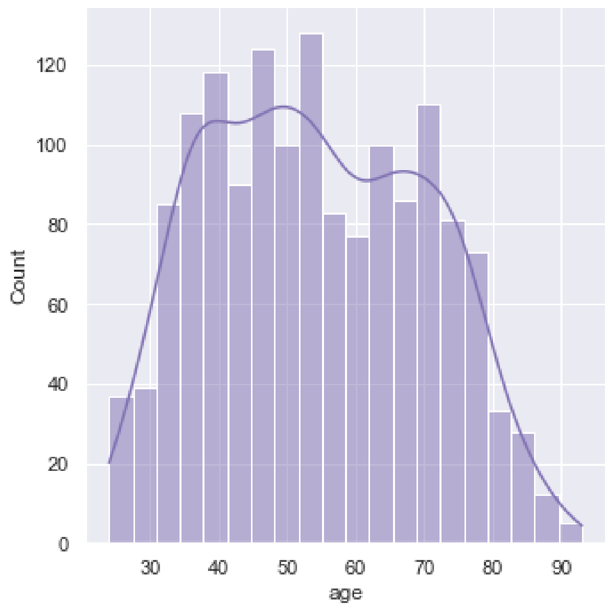
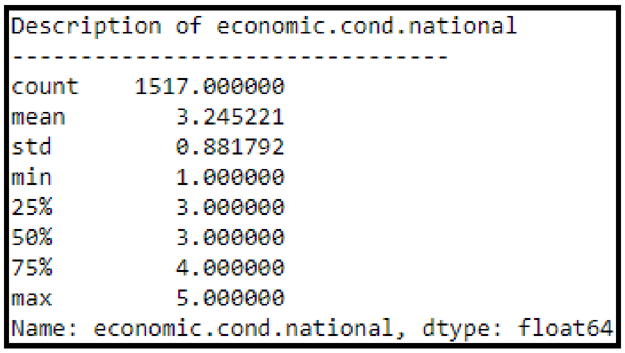


Figure 1: Univariate Analysis of 'age'

(above description generated with '.describe()' function of PANDAS library and plots generated using '.displot()' and '.boxplot()' functions of Seaborn library using Python)

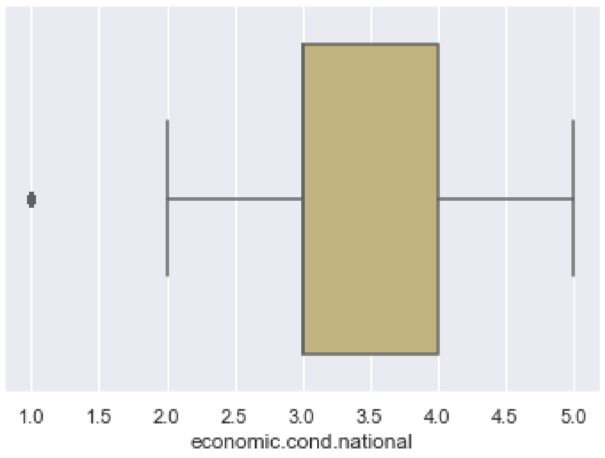
- voters are in the age group 24-93
- 75% of voters are within 67 years of age
- the mean and median ages are very close
- the distribution doesn't resemble a bell curve, hence, not normal
- earlier we saw, 'age' had a skewness of 0.14, but, there are no outliers as we see here
- for right/positive skewness: MEAN>MEDIAN>MODE

<u>economic.cond.national</u>



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

```
BoxPlot of economic.cond.national
------------------------------
```



```
Distribution of economic.cond.national
-------------------------------------
```

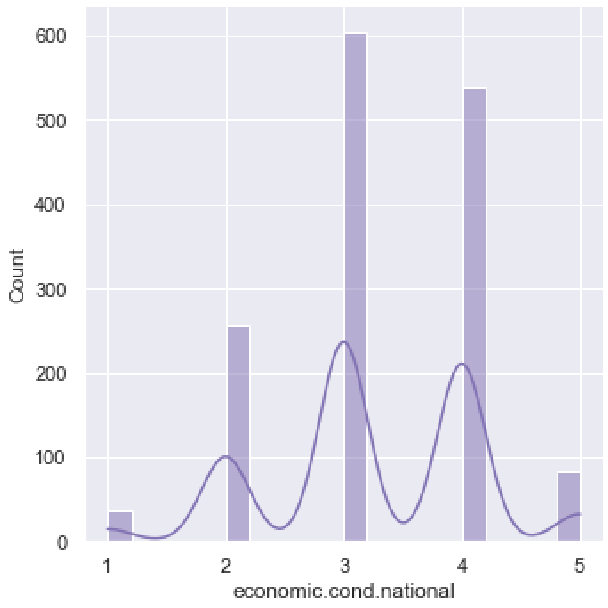


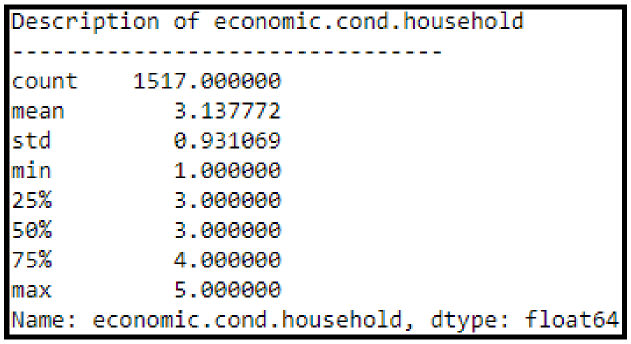
Figure 1: Univariate Analysis of 'economic.cond.national'

(above description generated with '.describe()' function of PANDAS library and plots generated using '.displot()' and '.boxplot()' functions of Seaborn library using Python)

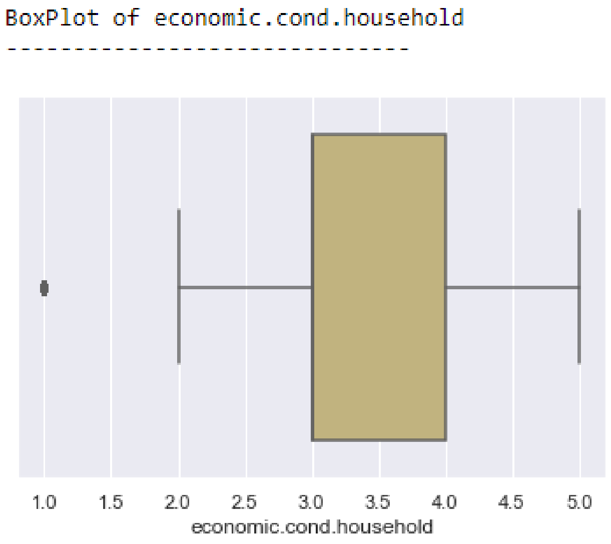
- ('economic.cond.national' is the assessment of current national economic conditions, 1 to 5)

- most of the ratings are ‘3’ or ‘4’(‘3’ being the most frequent)
- the ratings ‘1’ and ‘5’ are rarest ones(‘1’ being the least frequent)
- the mean is more than the median
- from histogram, we confirm the feature to contain categorical information actually, and not continuous numbers
- the distribution doesn't resemble a bell curve, hence, not normal
- earlier we saw, ‘economic.cond.national’ had a skewness of -0.24, and there are outliers on the left side of the boxplot too
- generally, for left/negative skewness: MODE>MEDIAN>MEAN, but here, the mean is more than the median

economic.cond.household



```
Description of economic.cond.household
--------------------------------
count    1517.000000
mean        3.137772
std         0.931069
min         1.000000
25%         3.000000
50%         3.000000
75%         4.000000
max         5.000000
Name: economic.cond.household, dtype: float64
```



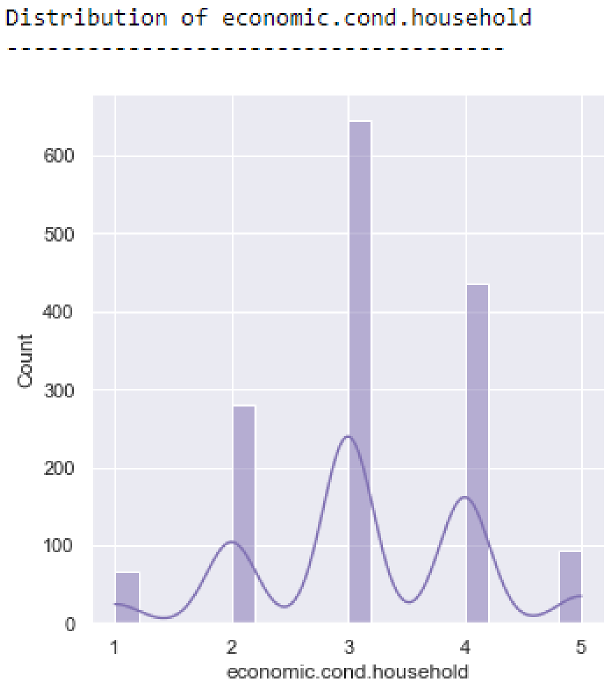


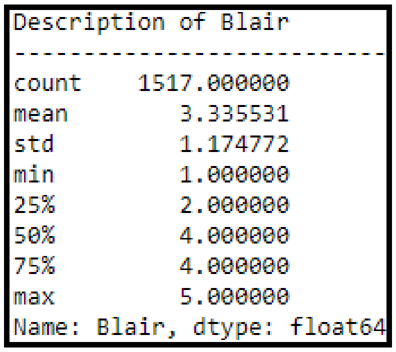
Figure 1: Univariate Analysis of ‘economic.cond.household’

(above description generated with ‘.describe()’ function of PANDAS library and plots generated using ‘.displot()’ and ‘.boxplot()’ functions of Seaborn library using Python)

- ‘economic.cond.household’ is the assessment of current household economic conditions, 1 to 5.
- most of the ratings are ‘3’ or ‘4’(‘3’ being the most frequent)

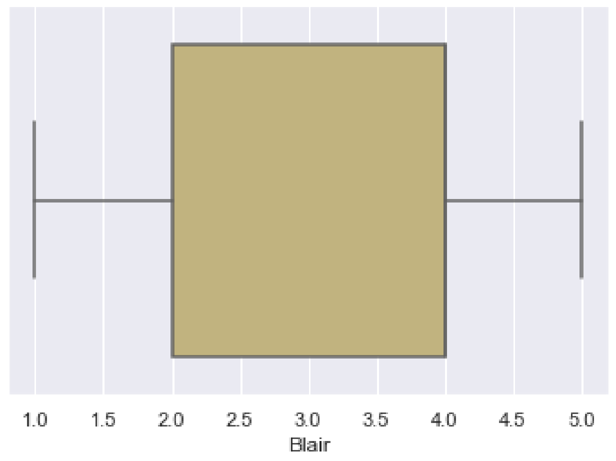
- the ratings ‘1’ and ‘5’ are rarest ones(‘1’ being the least frequent)
- the mean is more than the median
- from histogram, we can see the feature to contain categorical information actually, and not continuous numbers
- the distribution doesn't resemble a bell curve, hence, not normal
- earlier we saw, ‘economic.cond.household’ had a skewness of -0.14, and there are outliers on the left side of the boxplot too
- generally, for left/negative skewness: MODE>MEDIAN>MEAN, but here, the mean is more than the median

Blair



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

```
BoxPlot of Blair
-----------------------------
```



```
Distribution of Blair
-------------------------------------
```

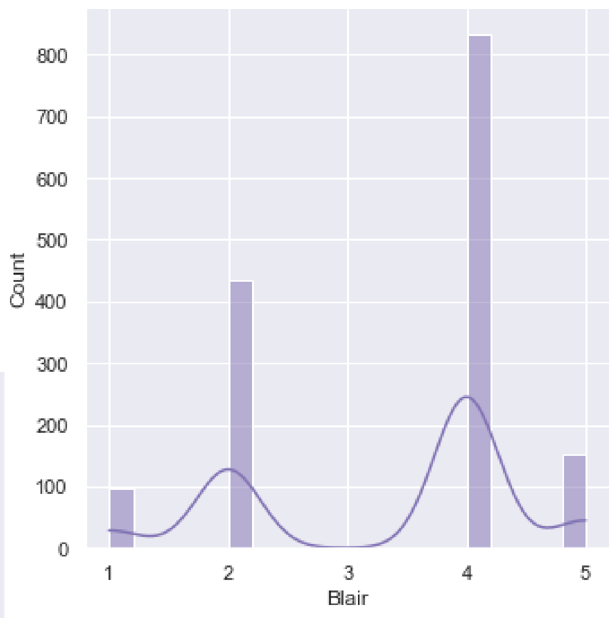


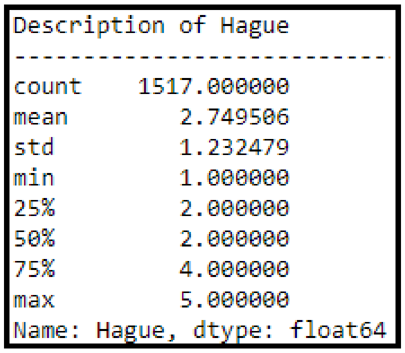
Figure 1: Univariate Analysis of ‘Blair’

(above description generated with ‘.describe()’ function of PANDAS library and plots generated using ‘.displot()’ and ‘.boxplot()’ functions of Seaborn library using Python)

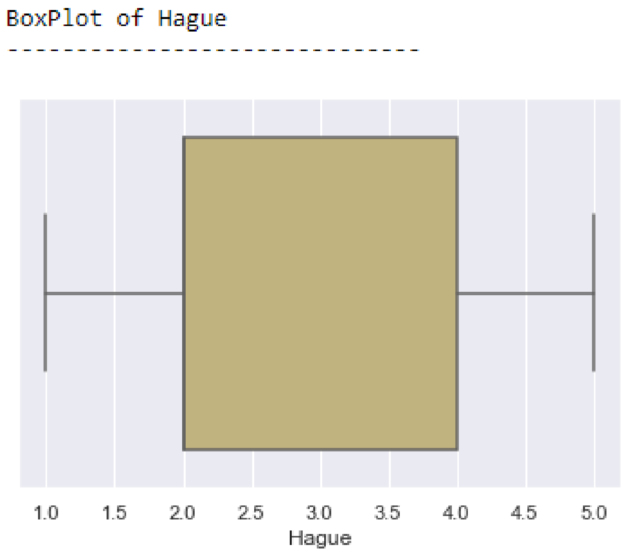
- (Blair is the assessment of the Labour leader, 1 to 5.)
- ‘4’ is the most frequent rating, indicating voters’ affinity to Labour party’s leader(s)
- ‘3’ is the least frequent rating
- the mean is lesser than median rating

- from histogram, we can see the feature to contain categorical information actually, and not continuous numbers
- the distribution doesn't resemble a bell curve, hence, not normal
- earlier we saw, 'Blair' had a skewness of -0.54, but there are no outliers
- generally, for left/negative skewness: MODE>MEDIAN>MEAN

Hague



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



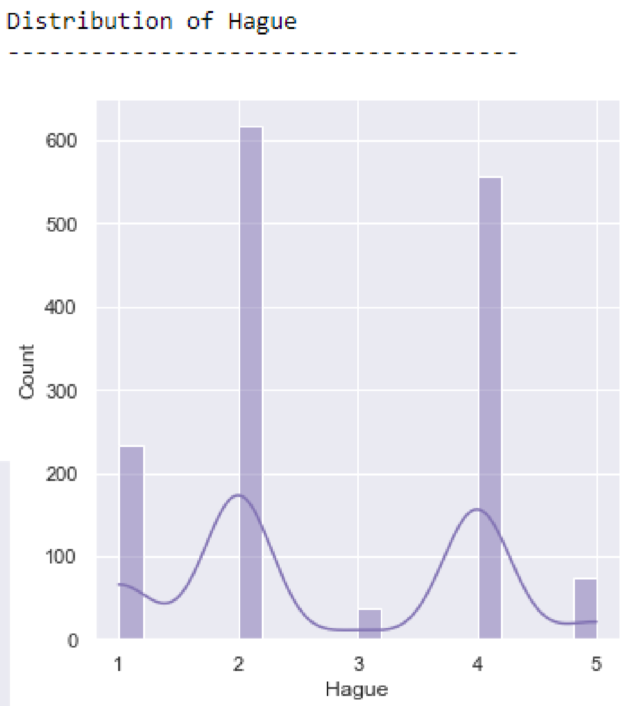


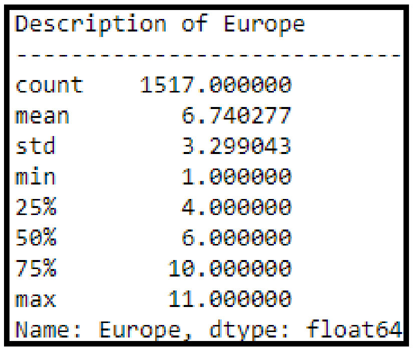
Figure 1: Univariate Analysis of 'Hague'

(above description generated with '.describe()' function of PANDAS library and plots generated using '.displot()' and '.boxplot()' functions of Seaborn library using Python)

- ('Hague' is the assessment of the Conservative leader, 1 to 5.)
- '2' is the most frequent rating, indicating voters' antipathy to Conservative party's leader(s), though we have significant number of '4' ratings too
- '3' is the least frequent rating
- the mean is more than median rating
- from histogram, we can see the feature to contain categorical information actually, and not continuous numbers
- the distribution doesn't resemble a bell curve, hence, not normal

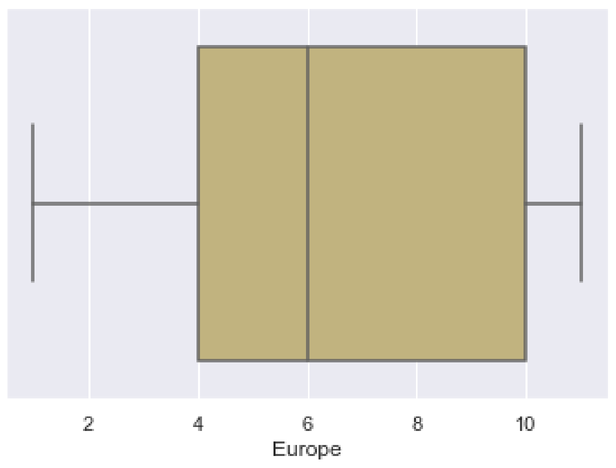
- earlier we saw, ‘Hague’ had a skewness of 0.15, but there are no outliers
- generally, for right/positive skewness: MEAN>MEDIAN>MODE

Europe



```
Description of Europe
----------------------------
count    1517.000000
mean        6.740277
std         3.299043
min         1.000000
25%         4.000000
50%         6.000000
75%        10.000000
max        11.000000
Name: Europe, dtype: float64
```

```
BoxPlot of Europe
------------------------------
```



```
Distribution of Europe
-------------------------------------
```

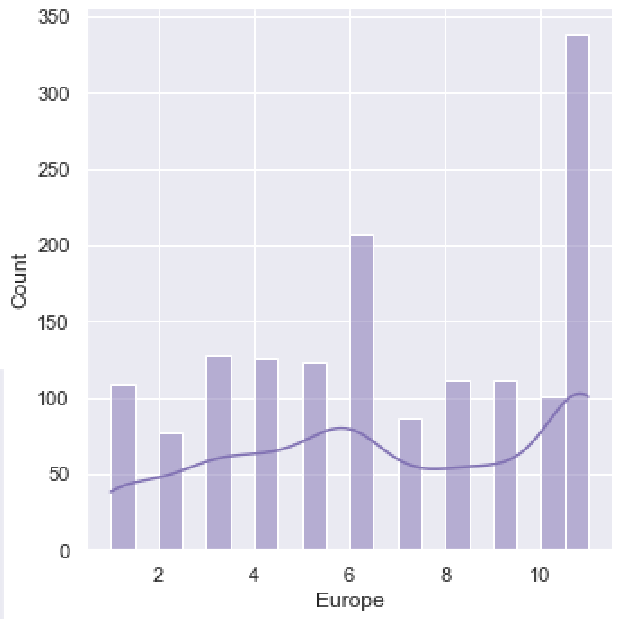
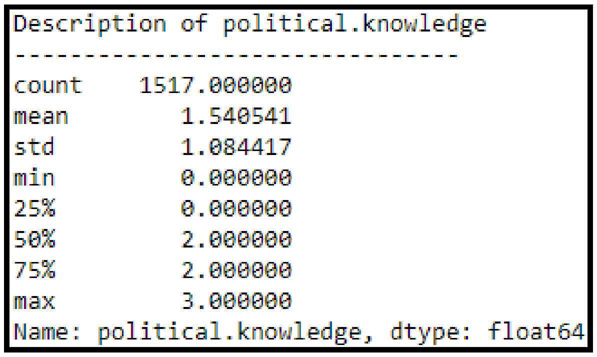


Figure 1: Univariate Analysis of ‘Europe’

(above description generated with ‘.describe()’ function of PANDAS library and plots generated using ‘.displot()’ and ‘.boxplot()’ functions of Seaborn library using Python)

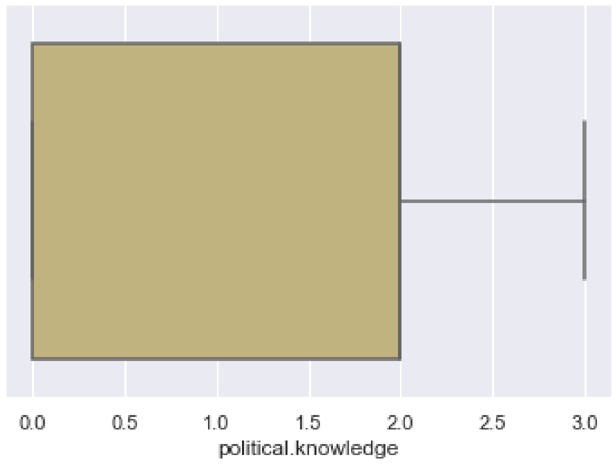
- (Europe is an 11-point scale that measures respondents' attitudes toward European integration. High scores represent ‘Eurosceptic’ sentiment.)
- ‘11’ is the most frequent rating, indicating voters’ ‘Eurosceptic’ sentiment
- the mean is more than median rating
- from histogram, we can see the feature to contain categorical information actually, and not continuous numbers
- the distribution doesn't resemble a bell curve, hence, not normal
- earlier we saw, ‘Europe’ had a skewness of -0.14, but there are no outliers
- generally, for left/negative skewness: MODE>MEDIAN>MEAN, but here mean is more than median

political.knowledge



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

```
BoxPlot of political.knowledge
------------------------------
```



```
Distribution of political.knowledge
-------------------------------------
```

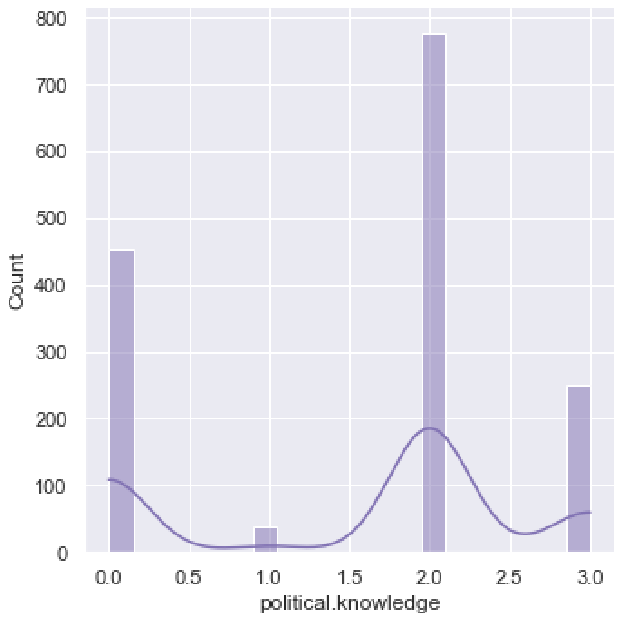
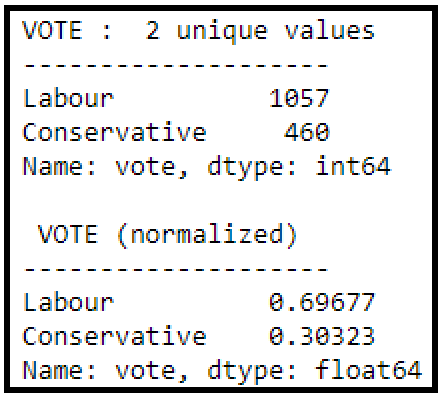


Figure 1: Univariate Analysis of 'political.knowledge'

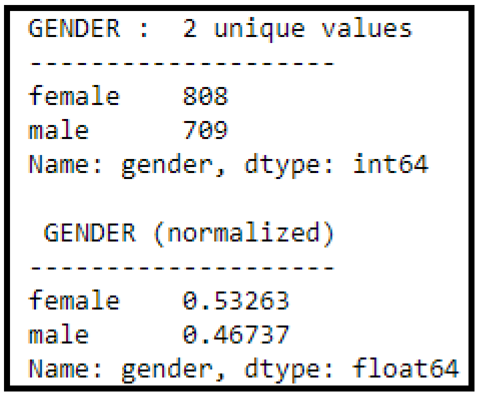
(above description generated with '.describe()' function of PANDAS library and plots generated using '.displot()' and '.boxplot()' functions of Seaborn library using Python)

- ('political.knowledge' is knowledge of parties' positions on European integration, 0 to 3.)
- '2.0' is the most frequent rating, indicating voters' sound knowledge of parties' positions on European integration
- '1.0' is the least frequent rating
- there is good bunch of '0.0' ratings as well, indicating some voters' absence of knowledge of parties' positions on European integration
- the mean is lesser than median rating
- from histogram, we can see the feature to contain categorical information actually, and not continuous numbers
- the distribution doesn't resemble a bell curve, hence, not normal
- earlier we saw, 'political.knowledge' had a skewness of -0.42, but there are no outliers
- generally, for left/negative skewness: MODE>MEDIAN>MEAN

Let us now go for the Univariate analysis of 'object' data features in the data set. We will check the unique values, count of unique values and proportion(i.e, normalized values) of unique values in each feature:



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

 VOTE (normalized)
--------------------
Labour           0.69677
Conservative     0.30323
Name: vote, dtype: float64
```



```
GENDER :  2 unique values
--------------------
female    808
male      709
Name: gender, dtype: int64

 GENDER (normalized)
--------------------
female    0.53263
male      0.46737
Name: gender, dtype: float64
```

Tables 6: Univariate Analysis of Categorical Data(Problem 1)

(above tables generated with '.value_counts()' function of PANDAS library using Python)

Before drawing inferences, let us visualise above results in form of countplots:

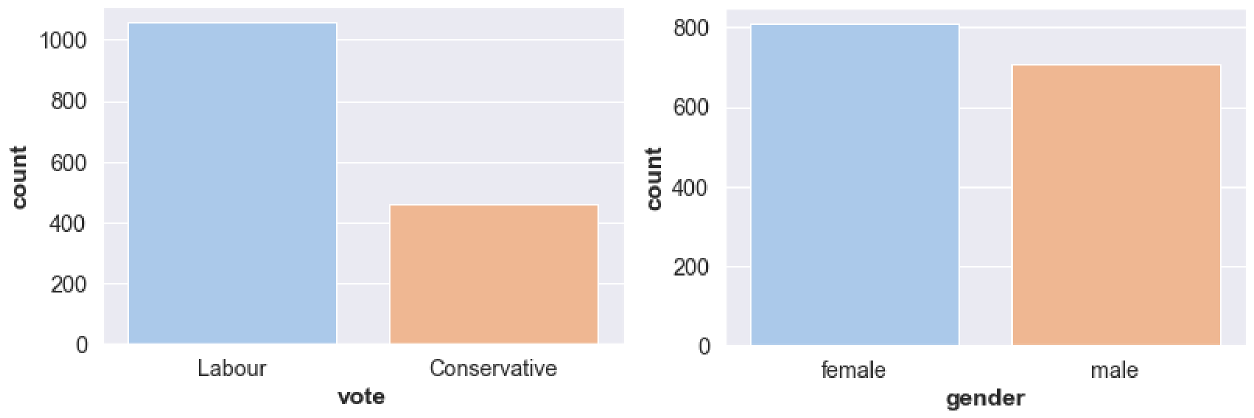


Figure 2: Count plots of Categorical Data(Problem 1)

(above plots generated using '.countplot()' function of Seaborn library in Python)

We can see:

- from the description, all the classes in the features seem to be correct without any anomalies
- feature 'vote' has a high class imbalance of 'Labour':'Conservative' to be roughly around 7:3
- feature 'gender' has a relatively lower class imbalance of 'female':'male' to be roughly around 53:47

Let us visualise how the target feature ‘vote’ is distributed over other categorical variable, ‘gender’:

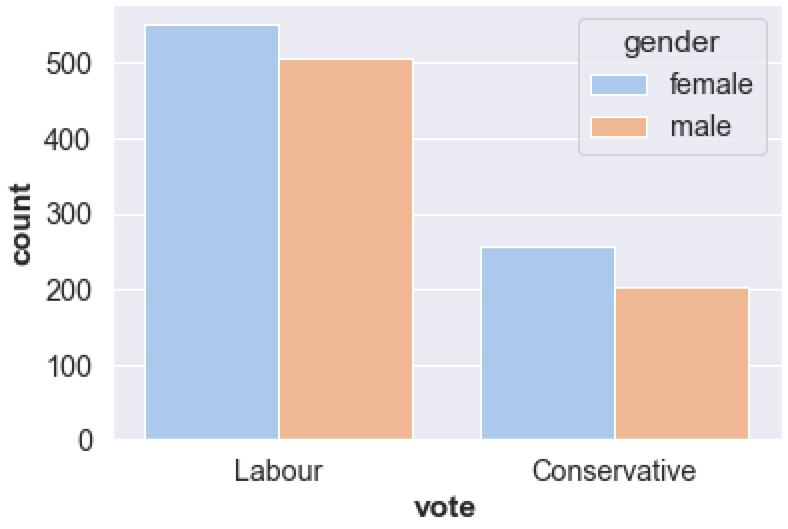


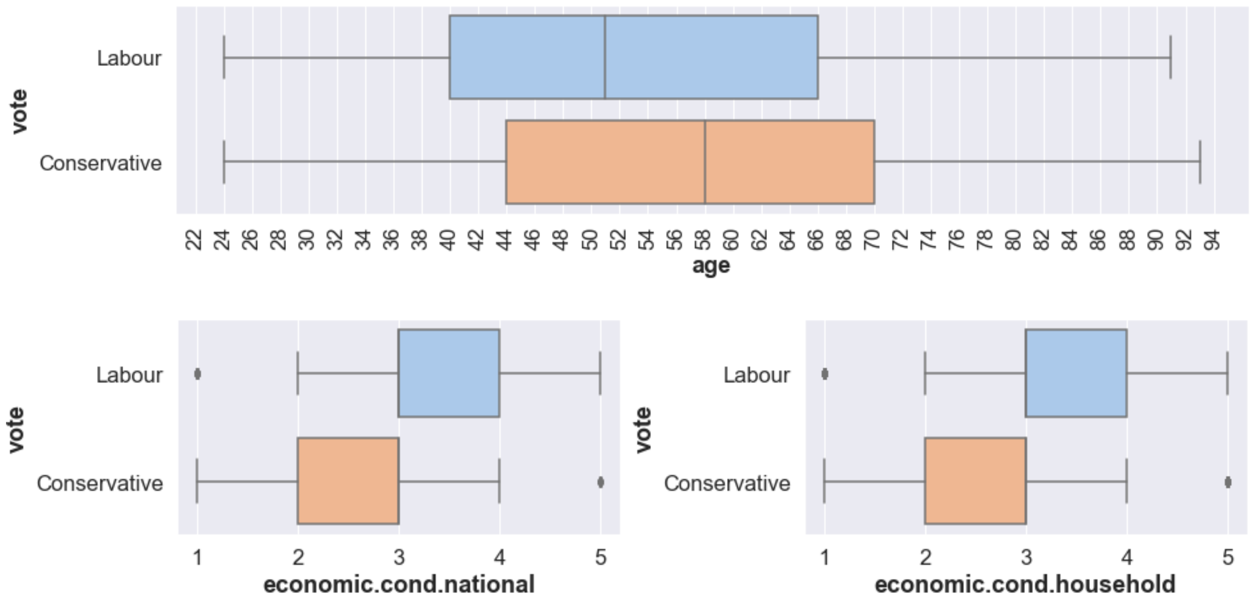
Figure 19: Distribution of Target over ‘foreign’(Problem 2)

(above figure generated with ‘.countplot()’ function of Seaborn library in Python)

We can see:

- for both the parties, number of female voters is slightly higher than male voters

Let us now check how the target ‘vote’ is distributed over the numeric features of the data:



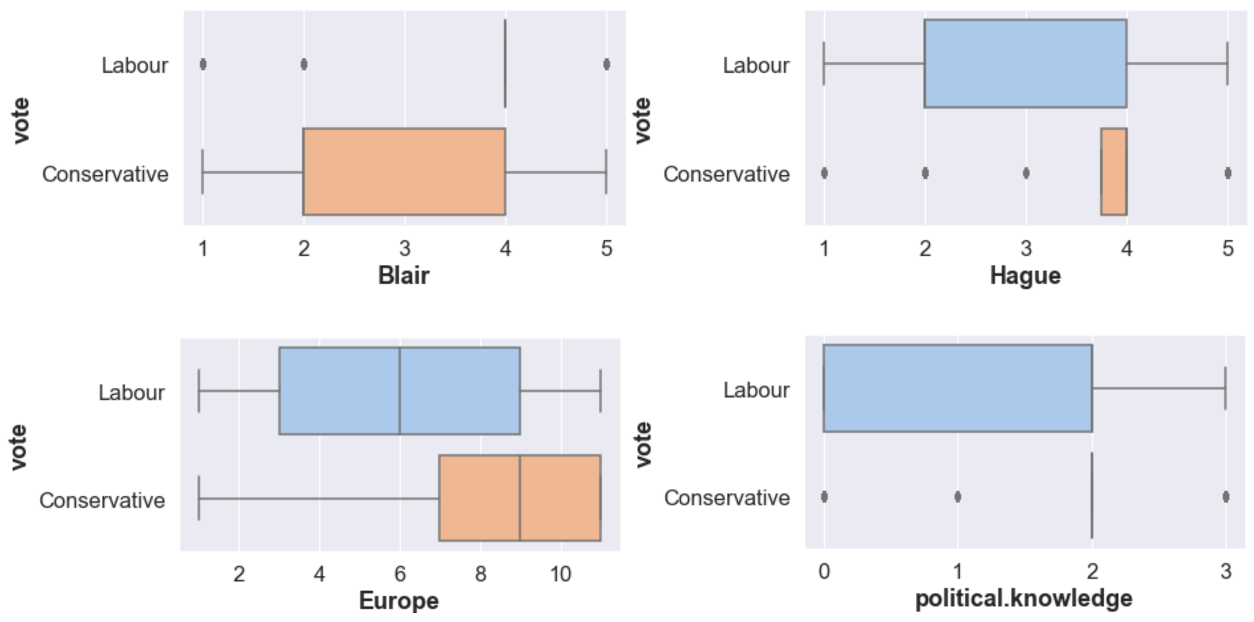


Figure 20: Distribution of Target over Numerical features(Problem 2)

(above figure generated with '.boxplot()' function of Seaborn library in Python)

We can see:

- the Labour party voters are marginally younger
- the current national and household economic conditions of Labour party voters is better
- the voters of Conservative party have more Eurosceptic sentiment
- the voters of Conservative party have sound political knowledge of parties' positions on European integration, while the voters of the Labour party seem to have varied knowledge.

Below we can see the target feature 'vote' versus feature 'age' hued over feature 'gender':

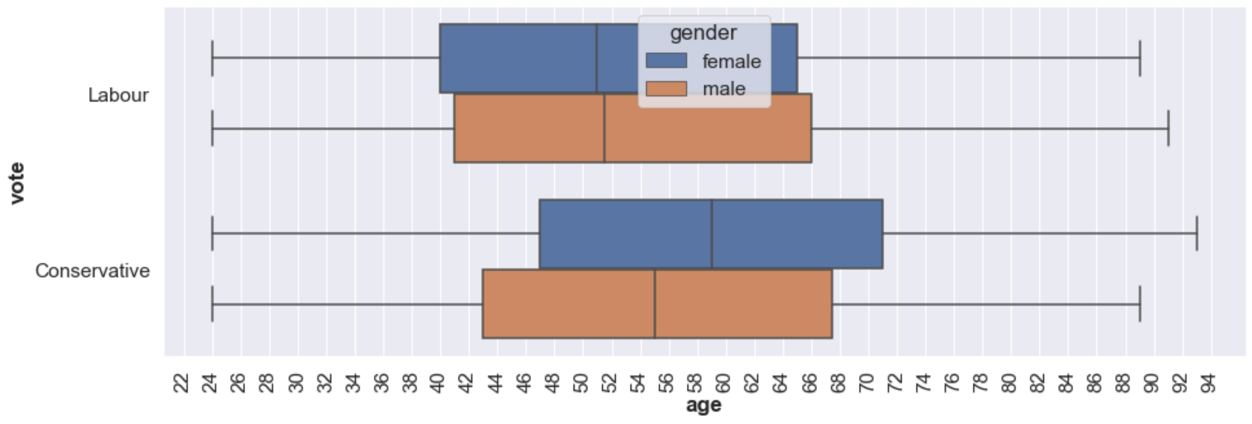


Figure 20: Distribution of 'vote' versus 'age' hued over 'gender'(Problem 2)

(above figure generated with '.boxplot()' function of Seaborn library in Python)

We can see:

- the female voters of Labour party to be marginally younger
- the male voters of both are nearly of the same age

Problem 2

Executive Summary

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

Introduction

The objective is to use text analytics to go through the above pieces of text. NLTK will be used too, which is a toolkit built for working with NLP in Python. It provides us various text processing libraries with a lot of test datasets.

2.1) Find the number of characters, words and sentences for the mentioned documents. (Hint: use .words(), .raw(), .sent() for extracting counts)

- The number of words can be found by the '.words()' function of the nltk library in Python.
  The number of words in the speeches of
  - ❖ President Franklin D. Roosevelt were found to be 1536
  - ❖ President John F. Kennedy were found to be 1546
  - ❖ President Richard Nixon were found to be 2028
- The number of sentences can be found by the '.sents()' function of the nltk library in Python.
  The number of sentences in the speeches of
  - ❖ President Franklin D. Roosevelt were found to be 68
  - ❖ President John F. Kennedy were found to be 52
  - ❖ President Richard Nixon were found to be 69
- The number of characters can be found by the length of the '.split()' function of the nltk library in Python.
  The number of characters **with spaces** in the speeches of

- ❖ President Franklin D. Roosevelt were found to be 7571
- ❖ President John F. Kennedy were found to be 7618
- ❖ President Richard Nixon were found to be 9991

The number of characters **without spaces** in the speeches of

- ❖ President Franklin D. Roosevelt were found to be 6174
- ❖ President John F. Kennedy were found to be 6202
- ❖ President Richard Nixon were found to be 8122

2.2) Remove all the stopwords from the three speeches. Show the word count before and after the removal of stopwords. Show a sample sentence after the removal of stopwords.

Other libraries that would be used along with nltk are numpy, pandas, string and random. Other libraries would be imported down the line if need arises.

- text first needs to be converted to lowercase using '.lower()' function of python
- we will be using 'stopwords.words('english')' from 'nltk.corpus' for getting the nltk's general English stopwords to be removed.
- 'punctuation' from the 'string' library will be used, which contains the list of punctuations to be removed.
- 'PorterStemmer' from 'nltk.stem' would be used for stemming the text.

<u>President Franklin D. Roosevelt</u>

- Initial number of words in President Franklin D. Roosevelt's raw speech: 1536
- Number of words after stopwords and punctuations removal: 657
- For sample, '.join()' is used to merge the first 50 strings after stop words and punctuations removal:

  national day inauguration since 1789 people renewed sense dedication united states washington day task people create weld together nation lincoln day task people preserve nation disruption within day task people save nation institutions disruption without us come time midst swift happenings pause moment take stock -- recall place history rediscover

- number of words after stemming: 657
- For sample, '.join()' is used to merge the first 50 strings after stemming:

  nation day inaugur sinc 1789 peopl renew sens dedic unit state washington day task peopl creat weld togeth nation lincoln day task peopl preserv nation disrupt within day task peopl save nation institut disrupt without us come time midst swift happen paus moment take stock -- recal place histori rediscov

- .FreqDist() reveals ‘--’ string to be occurring most frequently:
  FreqDist({'--': 25, 'nation': 17, 'know': 10, 'peopl': 9, 'spirit': 9, 'life': 9, 'democraci': 9, 'us': 8, 'america': 8, 'live': 7, ...})

Let us remove these 25 ‘--’ strings and analyse the speech again:

- Initial number of words in President Franklin D. Roosevelt’s raw speech: 1536
- Number of words after stopwords and punctuations removal: 632
- For sample, ‘.join()’ is used to merge the first 50 strings after stop words and punctuations removal:

  national day inauguration since 1789 people renewed sense dedication united states washington day task people create weld together nation lincoln day task people preserve nation disruption within day task people save nation institutions disruption without us come time midst swift happenings pause moment take stock recall place history rediscover may

- number of words after stemming: 632
- For sample, ‘.join()’ is used to merge the first 50 strings after stemming:

  nation day inaugur sinc 1789 peopl renew sens dedic unit state washington day task peopl creat weld togeth nation lincoln day task peopl preserv nation disrupt within day task peopl save nation institut disrupt without us come time midst swift happen paus moment take stock recal place histori rediscov may

- .FreqDist() reveals most frequently occurring strings:
  FreqDist({'nation': 17, 'know': 10, 'peopl': 9, 'spirit': 9, 'life': 9, 'democraci': 9, 'us': 8, 'america': 8, 'live': 7, 'year': 7, ...})

President John F. Kennedy

- Initial number of words in President John F. Kennedy’s raw speech: 1546
- Number of words after stopwords and punctuations removal: 722
- For sample, ‘.join()’ is used to merge the first 50 strings after stop words and punctuations removal:

  vice president johnson mr speaker mr chief justice president eisenhower vice president nixon president truman reverend clergy fellow citizens observe today victory party celebration freedom -- symbolizing end well beginning -- signifying renewal well change sworn almighty god solemn oath forebears l prescribed nearly century three quarters ago world different

- number of words after stemming: 722

- For sample, '.join()' is used to merge the first 50 strings after stemming:

  vice presid johnson mr speaker mr chief justic presid eisenhow vice presid nixon presid truman reverend clergi fellow citizen observ today victori parti celebr freedom -- symbol end well begin -- signifi renew well chang sworn almighti god solemn oath forebear l prescrib nearli centuri three quarter ago world differ

- .FreqDist() reveals '--' string to be occurring most frequently:
  FreqDist({'--': 25, 'let': 16, 'us': 12, 'power': 9, 'world': 8, 'nation': 8, 'side': 8, 'new': 7, 'pledg': 7, 'ask': 6, ...})

Let us remove these 25 '--' strings and analyse the speech again:

- Initial number of words in President John F. Kennedy's raw speech: 1546
- Number of words after stopwords and punctuations removal: 697
- For sample, '.join()' is used to merge the first 50 strings after stop words and punctuations removal:

  vice president johnson mr speaker mr chief justice president eisenhower vice president nixon president truman reverend clergy fellow citizens observe today victory party celebration freedom symbolizing end well beginning signifying renewal well change sworn almighty god solemn oath forebears l prescribed nearly century three quarters ago world different man holds

- number of words after stemming: 697
- For sample, '.join()' is used to merge the first 50 strings after stemming:

  vice presid johnson mr speaker mr chief justic presid eisenhow vice presid nixon presid truman reverend clergi fellow citizen observ today victori parti celebr freedom symbol end well begin signifi renew well chang sworn almighti god solemn oath forebear l prescrib nearli centuri three quarter ago world differ man hold

- .FreqDist() reveals most frequently occurring strings:
  FreqDist({'let': 16, 'us': 12, 'power': 9, 'world': 8, 'nation': 8, 'side': 8, 'new': 7, 'pledg': 7, 'ask': 6, 'citizen': 5, ...})

<u>President Richard Nixon</u>

- Initial number of words in President Richard Nixon's raw speech: 2028
- Number of words after stopwords and punctuations removal: 853

- For sample, ‘.join()’ is used to merge the first 50 strings after stop words and punctuations removal:

  mr vice president mr speaker mr chief justice senator cook mrs eisenhower fellow citizens great good country share together met four years ago america bleak spirit depressed prospect seemingly endless war abroad destructive conflict home meet today stand threshold new era peace world central question us shall use peace let

- number of words after stemming: 853
- For sample, ‘.join()’ is used to merge the first 50 strings after stemming:

  mr vice presid mr speaker mr chief justic senat cook mr eisenhow fellow citizen great good countri share togeth met four year ago america bleak spirit depress prospect seemingli endless war abroad destruct conflict home meet today stand threshold new era peac world central question us shall use peac let

- .FreqDist() reveals most frequently occurring strings:
  FreqDist({'us': 26, 'let': 22, 'america': 21, 'peac': 19, 'world': 18, 'respons': 17, '--': 17, 'new': 15, 'nation': 15, 'govern': 10, ...})

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)

<u>President Franklin D. Roosevelt</u>

- .FreqDist() reveals most frequently occurring strings:
  FreqDist({'--': 25, 'nation': 17, 'know': 10, 'peopl': 9, 'spirit': 9, 'life': 9, 'democraci': 9, 'us': 8, 'america': 8, 'live': 7, ...})

- Let us remove these 25 ‘--’ strings and get the frequency distribution of strings again:

  FreqDist({'nation': 17, 'know': 10, 'peopl': 9, 'spirit': 9, 'life': 9, 'democraci': 9, 'us': 8, 'america': 8, 'live': 7, 'year': 7, ...})

Hence, the top 3 words in President Franklin D. Roosevelt’s speech are:

- ☐ ‘nation’ (17 times)
- ☐ ‘know’ (10 times)
- ☐ ‘peopl’ (9 times) [‘spirit’, ‘life’ and 'democraci' also occur 9 times, but the string ‘peopl’ comes first in the speech, then ‘spirit’, followed by ‘life’, and finally 'democraci'.]

<u>President John F. Kennedy</u>

- .FreqDist() reveals most frequently occurring strings:

FreqDist({'--': 25, 'let': 16, 'us': 12, 'power': 9, 'world': 8, 'nation': 8, 'side': 8, 'new': 7, 'pledg': 7, 'ask': 6, ...})

- Let us remove these 25 ‘--’ strings and get the frequency distribution of strings again:

  FreqDist({'let': 16, 'us': 12, 'power': 9, 'world': 8, 'nation': 8, 'side': 8, 'new': 7, 'pledg': 7, 'ask': 6, 'citizen': 5, ...})

Hence, the top 3 words in President John F. Kennedy’s speech are:

- ☐ ‘let’ (16 times)
- ☐ ‘us’ (12 times)
- ☐ ‘power’ (9 times)

<u>President Richard Nixon</u>

- .FreqDist() reveals most frequently occurring strings:
  FreqDist({'us': 26, 'let': 22, 'america': 21, 'peac': 19, 'world': 18, 'respons': 17, '--': 17, 'new': 15, 'nation': 15, 'govern': 10, ...})

Hence, the top 3 words in President Richard Nixon’s speech are:

- ☐ ‘us’ (26 times)
- ☐ ‘let’ (22 times)
- ☐ ‘america’ (21 times)

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

- we will be using ‘stopwords.words('english')’ from ‘nltk.corpus’ for getting the nltk’s general English stopwords to be removed.
- ‘punctuation’ from the ‘string’ library will be used, which contains the list of punctuations to be removed.
- ‘WordCloud’ and ‘matplotlib’ libraries are being used for plotting the word cloud.
- Following parameters were passed for each word cloud:

  width = 3000 (width of the canvas)

  height = 3000 (height of the canvas)

  background_color ='black' (background color of canvas)

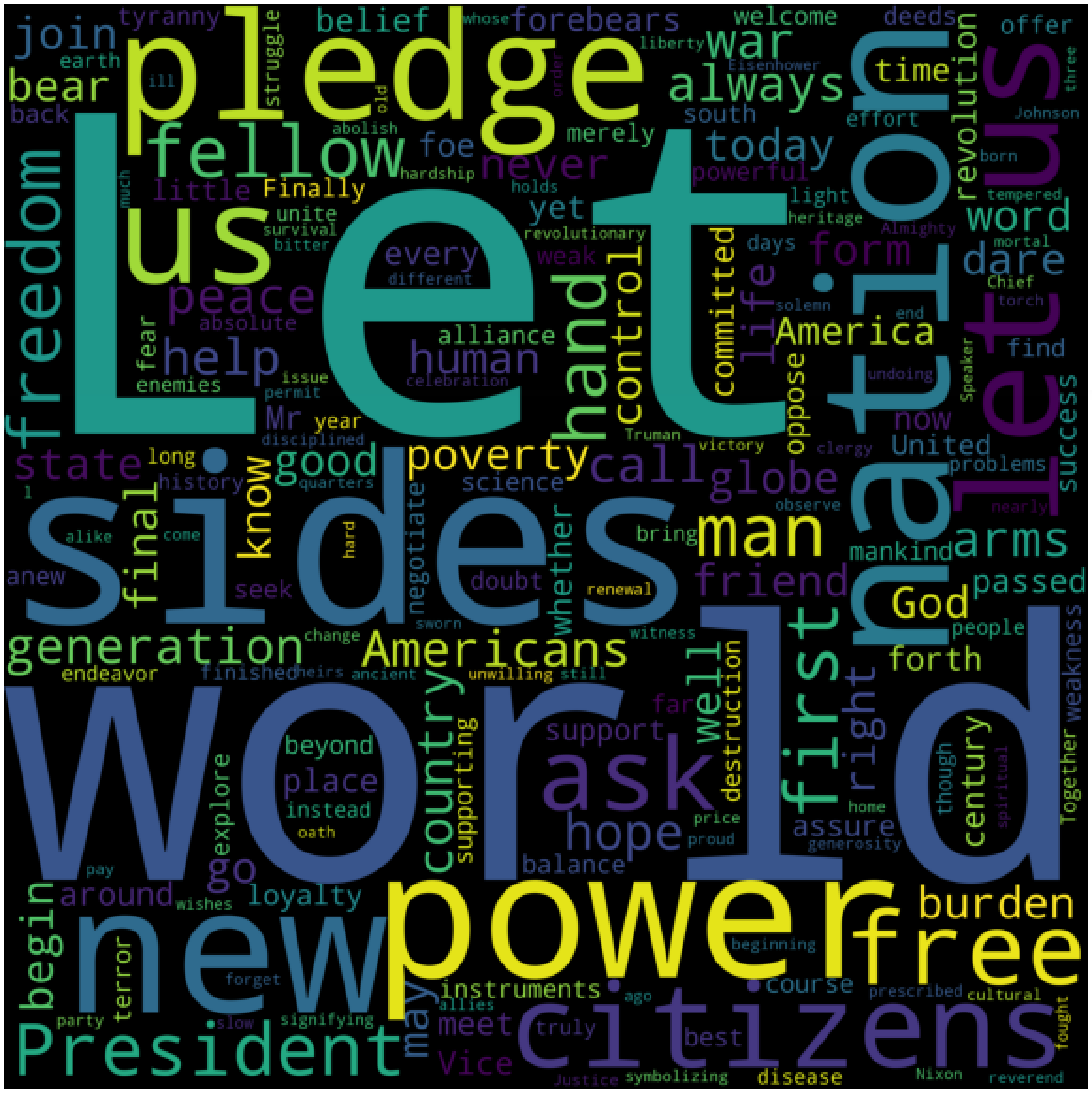
  min_font_size = 10 (exclude words that have a font lesser than 10)

  random_state=100 (for consistent output)

President Franklin D. Roosevelt

nation
spirit
know
people
life
democracy
America
freedom
year
mind
human
men
new
body
live
Government
person
hope
sense
surging
within
alone
continent
little
task
destiny
Republic
first
old
sacred
future
create
world
upon
day
understand
Constitution
speaks us
every
faith
time
enterprise
still
form
must
will
seem
without
American
Washington
something
free
together
measure
word
three
thing
land
history
United States
true
came
measured
preservation
ideal
doubt
stock
Lincoln
inauguration
enduring
happenings
unexplained
evil
naught
believe
fire
greater
individual
risk
carried
pause
branches
determined
institutions
tide
man
mystical
function
dire
score
limited
away
remains
better
elections
us
dedication
enough
present
even
way
preserve
slavery
revive
proved
home
ebbing
kind
seemed
less
fate
save
ten
majority
may
fullness
reason
wholly
taken
built
early
vital
take
coordinate
moment
national
wave
swift
real
peril
many
find
common
brought
lifetime
forward
Bill
purpose
predictions
acted
enlists
expression
inaction
renewed
maintained
fatalistic
quickly
inviolate
living
framework
successfully
Prophets
place
experience
artificial
count
undertaken
Rights
downfall
disruption
security
written
freely
put
midst
rediscover
fact
recall
action
die
grow
continue
States
ago
lines
shock
later
fruitful
come
tyranny
understanding
terror
frame
initiative
boldly
material
seen
survived
frozen
go
joined
weld
crisis
structures
decisively
become
sacred
Eight
day
dying
women

President John F. Kennedy

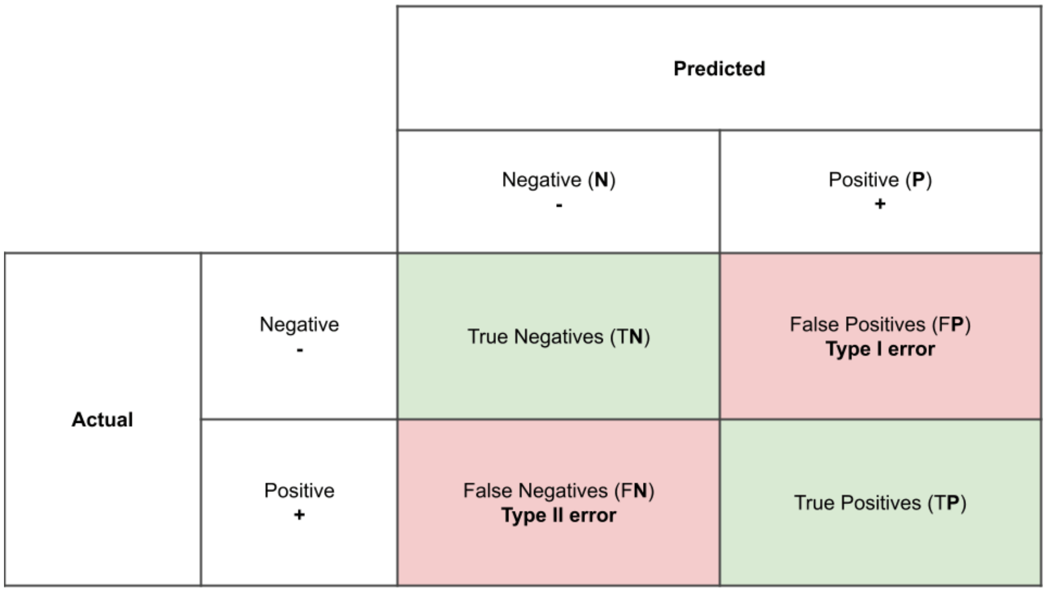


President Richard Nixon

Let us
world
peace
America
nation
new
responsibility
government
history
great
abroad
responsibilities
role
every
people
time
home
together
make
progress
promise
proud
faith
era
respect
others
year
freedom
help
better
right
today
come
individual
four years
live
war
way
challenge
one
God
old policies
act
work
system
us
long
ashamed
structure
away
forward
end
gladly
differences
initiatives
chance
another
building
turn
danger
Unless
continue
confident
commitment
American
Mr
dreams
policies
pledge
century
build
nature
turning
lead
failed
generation
greatly
place
share
resolve
important
retreat
learn
lived
toward
conflict
limitation
bold
third
friendships
understand
vigorously
traditional
missions
support
treaty
merely
revitalize
preserving
reach
200th
play
ask
power
greatest
country
past
human
often
nuclear
force
reduce
principle
ago
meet
negotiation
destructive
vital
isolation
confrontation
Senator
level
durable
spirit
think
arms
able
seemingly
Speaker
pattern
remain
defending
expect
indispensable
Washington
future
enter
base
clearly
good
boldly
periods
best
become
strong
preserve
seek
reaching
needs
continuing
threshold
endure
establish
may
citizens
maintaining
relationships
renew
hope
stand
meeting
Eisenhower
depressed
question
endless
interlude
II
remembered
made
postwar
Cook
Moscow
record
lasting
result
shift
fellow
Peking
borne
Chief
among
impose
use
met
invites
President
four
must
far
Justice
necessity
central
saw
Vice
prospect
adopted
bleak
rule
flimsy
stagnation

Appendix

- Classification accuracy is the ratio of correct predictions to total predictions made.
- A confusion matrix is a tabular summary of the number of correct and incorrect predictions made by a classifier. It can be used to evaluate the performance of a classification model through the calculation of performance metrics like accuracy, precision, recall, and F1-score.

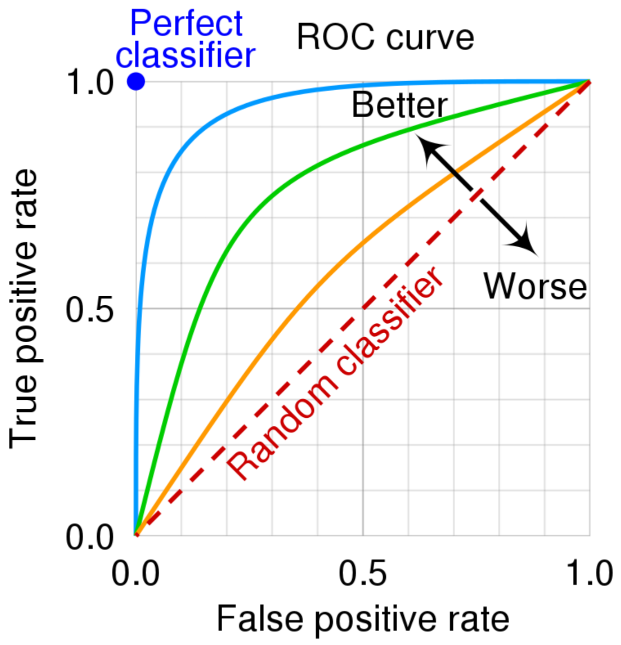


		Predicted	
		Negative (**N**) -	Positive (**P**) +
Actual	Negative -	True Negatives (T**N**)	False Positives (F**P**) **Type I error**
	Positive +	False Negatives (F**N**) **Type II error**	True Positives (T**P**)

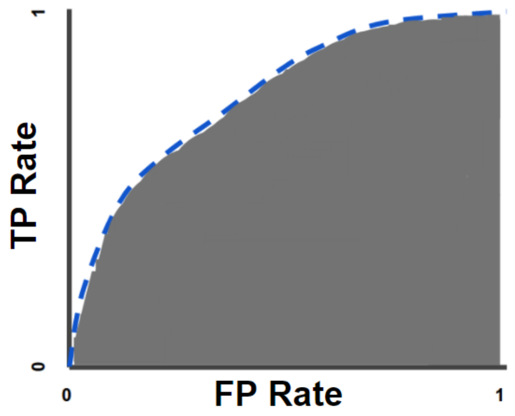
- An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:
    - ❖ True Positive Rate(TPR)
    - ❖ False Positive Rate(FPR)

$$TPR = \frac{TP}{TP + FN} \qquad FPR = \frac{FP}{FP + TN}$$

- An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. The following figure shows few ROC curves:

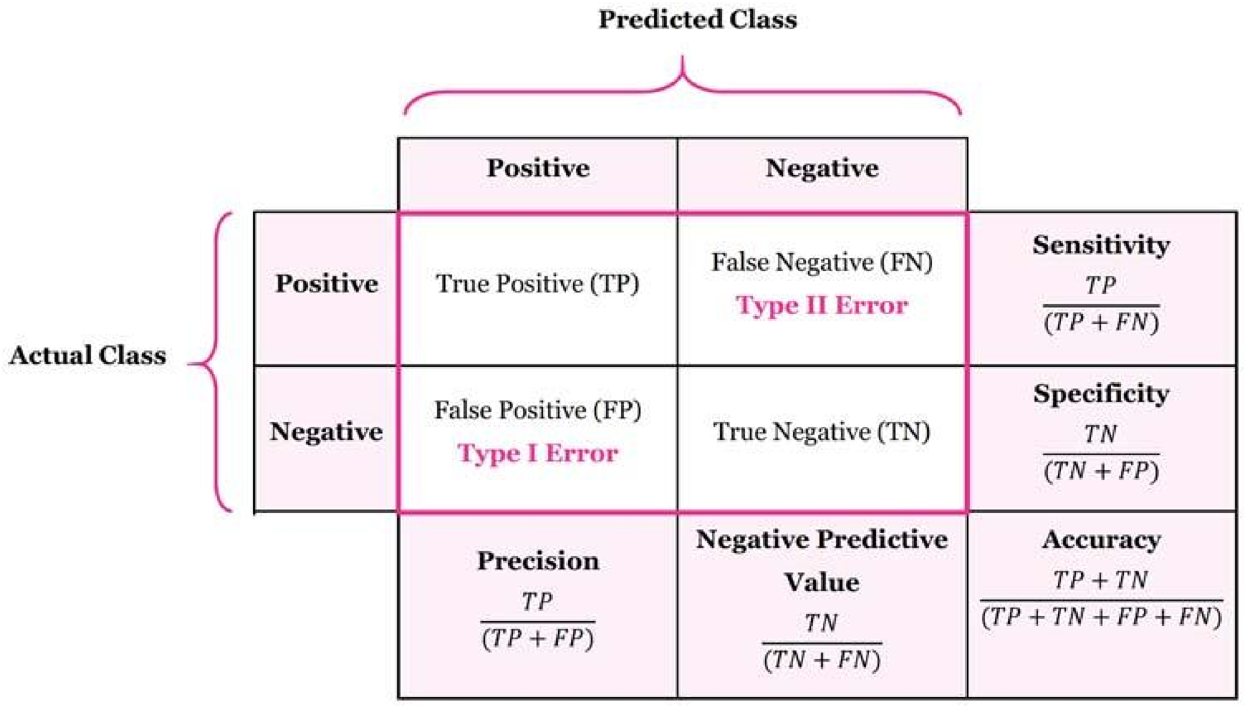


- AUC stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1). The grey shaded portion in the below ROC curve represents the AUC for it:



- A classification report is a performance evaluation metric in machine learning. It is used to show the precision, recall, F1 Score, and support of any trained classification model.It provides a better understanding of the overall performance of our trained model. To understand the classification report of a machine learning model, following are the metrics displayed in the report:
    - ❖ **Precision:** Precision is defined as the ratio of true positives to the sum of true and false positives. A model that produces no false positives has a precision of 1.0.

- ❖ **Recall(or Sensitivity):** Recall is defined as the ratio of true positives to the sum of true positives and false negatives. A model that produces no false negatives has a recall of 1.0.
- ❖ **F1-score:** The F1 is the weighted harmonic mean of precision and recall. The closer the value of the F1 score is to 1.0, the better the expected performance of the model is.
- ❖ **Support:** Support is the number of actual occurrences of the class in the dataset. It doesn't vary between models, it just diagnoses the performance evaluation process.



		Predicted Class		
		Positive	**Negative**	
Actual Class	**Positive**	True Positive (TP)	False Negative (FN) **Type II Error**	**Sensitivity** $\frac{TP}{(TP+FN)}$
	Negative	False Positive (FP) **Type I Error**	True Negative (TN)	**Specificity** $\frac{TN}{(TN+FP)}$
		Precision $\frac{TP}{(TP+FP)}$	**Negative Predictive Value** $\frac{TN}{(TN+FN)}$	**Accuracy** $\frac{TP+TN}{(TP+TN+FP+FN)}$

References

❖ The logos for ‘Great Learning’, ‘Great Lakes’ and ‘UT Austin Texas McCombs’ School of Business’ have been taken from ‘https://olympus.greatlearning.in/’

❖ Author is grateful to creators of:
   - ➢ https://towardsdatascience.com/
   - ➢ https://medium.com
   - ➢ https://www.geeksforgeeks.org/
   - ➢ https://stackoverflow.com
   - ➢ https://www.analyticsvidhya.com
   - ➢ https://www.google.com