ML Project - Coded

Machine Learning - 2 - Project Report

Isha Shukla

17 March 2024

Problem 1

CNBE, a prominent news channel, is gearing up to provide insightful coverage of recent elections, recognizing the importance of data-driven analysis. A comprehensive survey has been conducted, capturing the perspectives of 1525 voters across various demographic and socio-economic factors. This dataset encompasses 9 variables, offering a rich source of information regarding voters' characteristics and preferences.

1.1 Define the problem and perform Exploratory Data Analysis

```
<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
```

```
8
```

Observations:

- Dropped the 'Unnamed: 0' column from the dataset as it is not useful for our study.
- It has 1525 rows and 9 columns.
- It has 2 categorical (object) type and 7 integer data types.
- There is no null value in any column.
- There are 8 duplicated columns. After dropping duplicates, we have 1517 rows and 9 columns.

Data Description

	count	unique	top	freq
vote	1517	2	Labour	1057
gender	1517	2	female	808

	count	mean	std	min	25%	50%	75%	max
age	1517.0	54.241266	15.701741	24.0	41.0	53.0	67.0	93.0
economic.cond.national	1517.0	3.245221	0.881792	1.0	3.0	3.0	4.0	5.0
economic.cond.household	1517.0	3.137772	0.931069	1.0	3.0	3.0	4.0	5.0
Blair	1517.0	3.335531	1.174772	1.0	2.0	4.0	4.0	5.0
Hague	1517.0	2.749506	1.232479	1.0	2.0	2.0	4.0	5.0
Europe	1517.0	6.740277	3.299043	1.0	4.0	6.0	10.0	11.0
political.knowledge	1517.0	1.540541	1.084417	0.0	0.0	2.0	2.0	3.0

```
vote                       -0.857014
age                         0.139800
economic.cond.national     -0.238474
economic.cond.household    -0.144148
Blair                      -0.539514
Hague                       0.146191
Europe                     -0.141891
political.knowledge        -0.422928
dtype: float64
```

Skewness Rules

- Skewness between -0.5 and 0.5: Fairly symmetrical data.
- Skewness between -1 and -0.5, or between 0.5 and 1: Moderately skewed data.
- Skewness less than -1 or greater than 1: Highly skewed data.

Insights:

- The data is fairly symmetrical.
- Blair is higher than -0.5.
- There is not much skewness in the dataset.

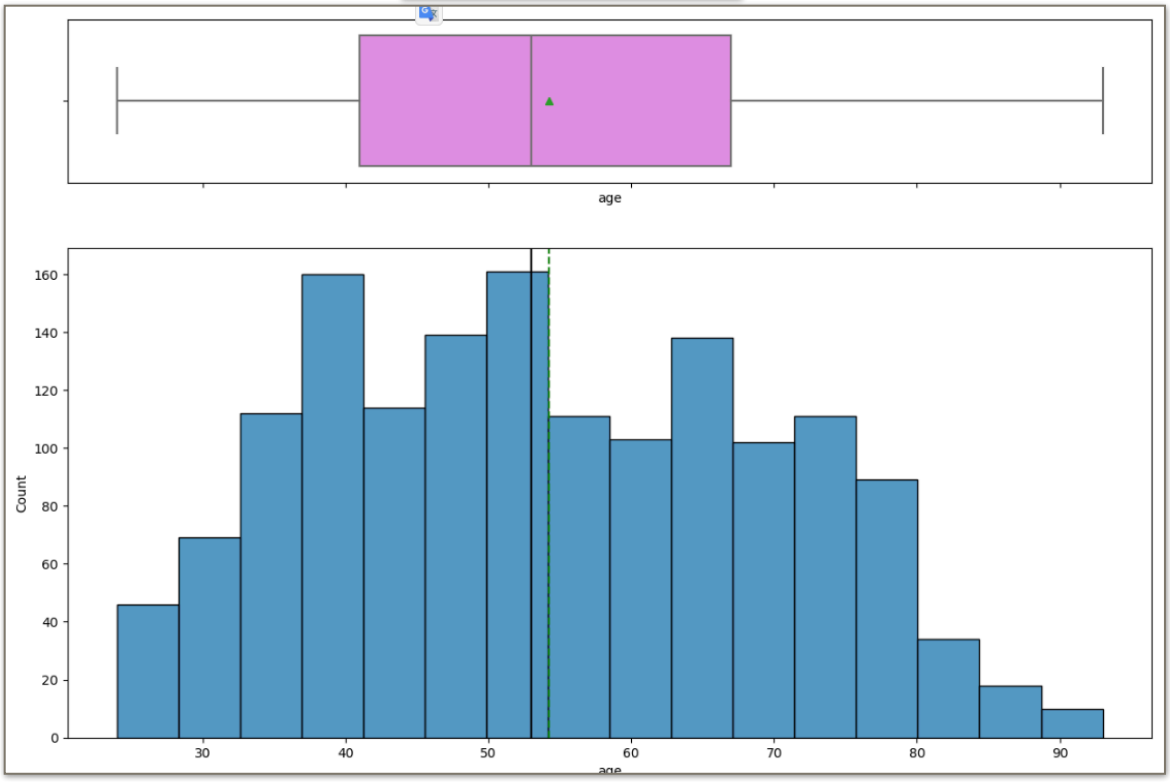
Univariate analysis

```
vote
Labour          1057
Conservative     460
Name: count, dtype: int64
------------------------------------------------
gender
female    808
male      709
Name: count, dtype: int64
------------------------------------------------
```

- There are two types vote:
  - Labour - 1057
  - Conservative - 460
- There are 808 females and 709 males.

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
```



Observation about age

- There is no outlier in the age column.
- The mean age is 54.241266.
- Maximum number of people are aged between 41 and 67.

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
```

```
economic.cond.national
3    604
4    538
2    256
5     82
1     37
Name: count, dtype: int64
<Axes: xlabel='economic.cond.national', ylabel='count'>
```

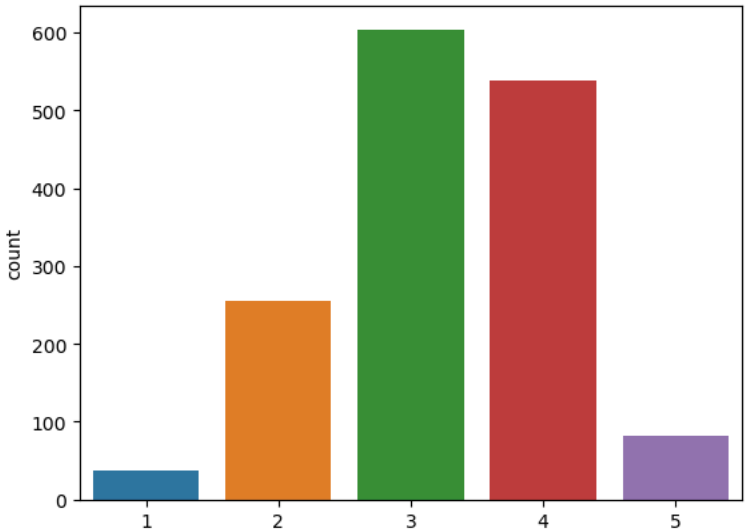


Fig: Count Plot for economic.cond.national

Observation

- 3 has the highest value.
- The mean is 3.245221.
- 1 has the lowest value 37.

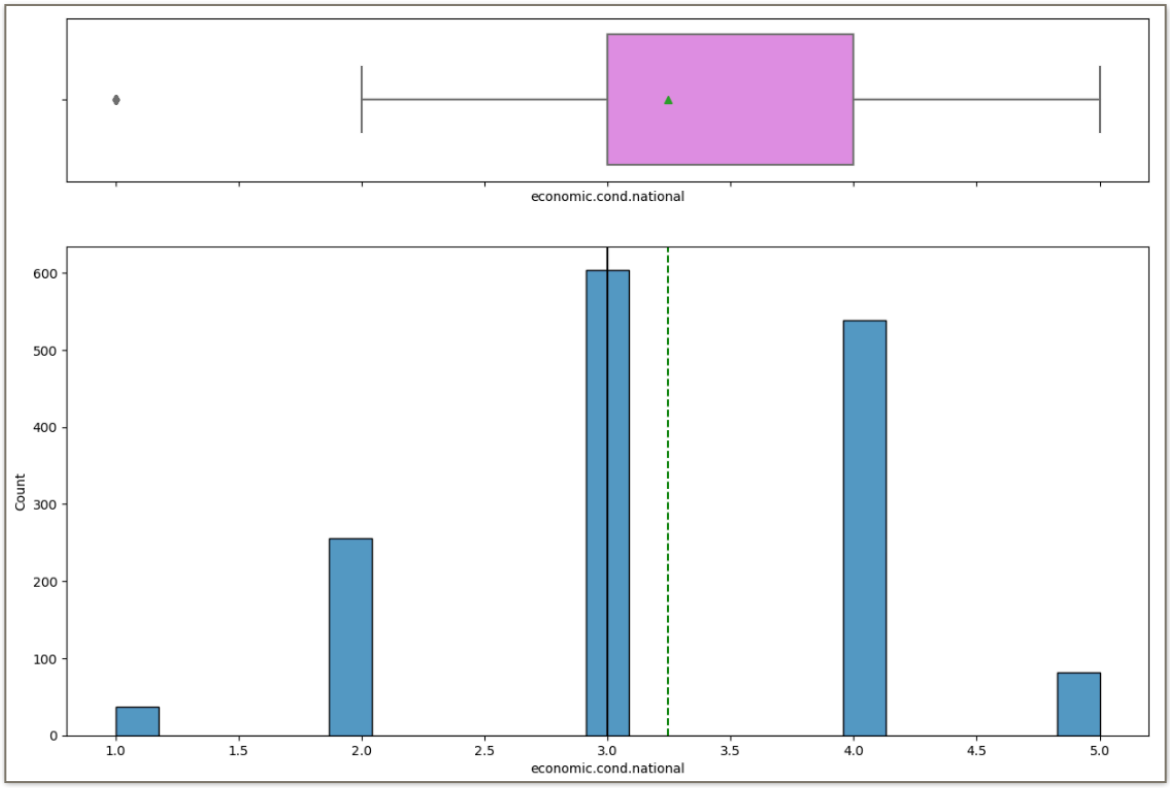


Fig: Boxplot and histogram for economic.cond.national

economic.cond.household

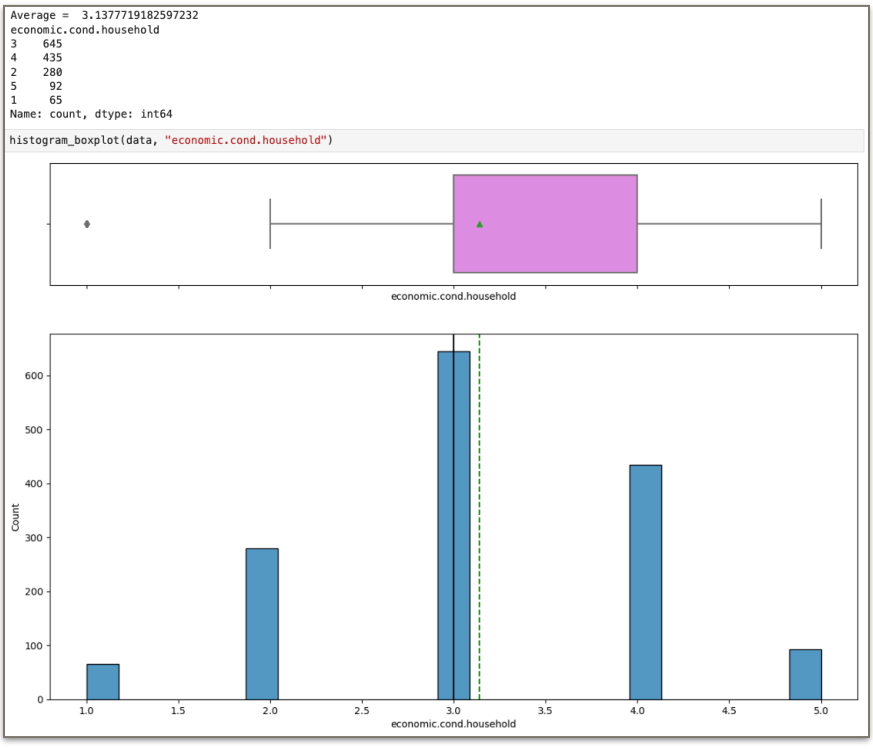


Fig: Boxplot and histogram for economic.cond.household

Observation

- The average economic.cond.household is 3.1377719182597232
- 3 has the highest count of 645.
- 1 has the least count of 65.

Blair

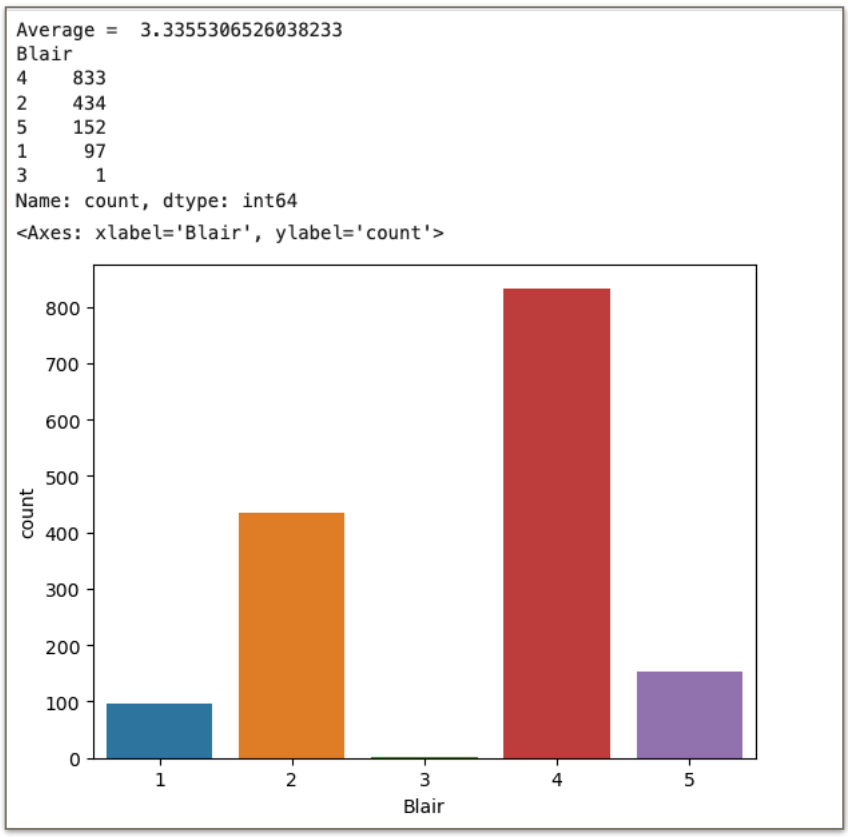


Fig: Count-plot for Blair

Observation

- The average for Blair is 3.3355306526038233
- 4 has highest count of 833.
- 3 has the least count of 1.

Hague

```
Average =  2.7495056031641396
Hague
2    617
4    557
1    233
5     73
3     37
Name: count, dtype: int64
<Axes: xlabel='Hague', ylabel='count'>
```

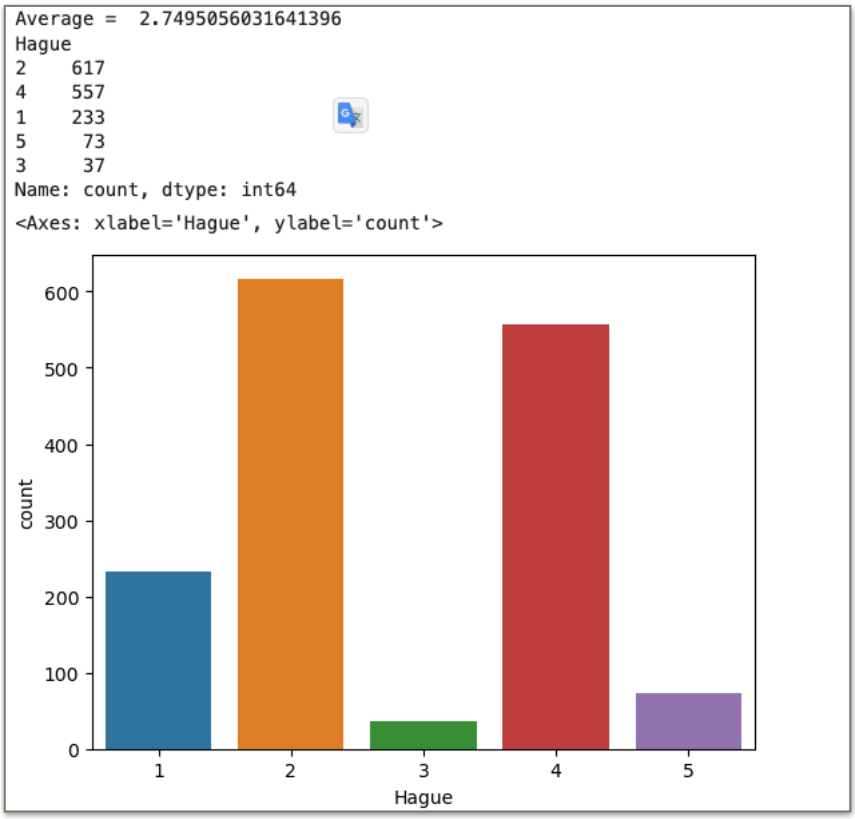


Fig: Count-plot for Hague

Observation

- The average of Hague is 2.7495056031641396
- 2 has highest count of 617.
- 3 has the least count.

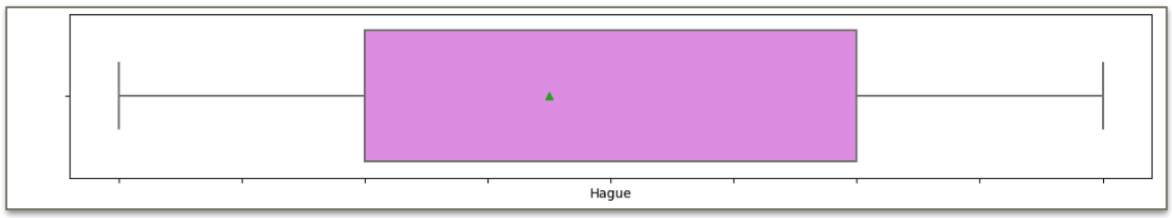


Fig: Boxplot for Hague

Europe

```
Average =  6.7402768622280815
Europe
11    338
6     207
3     128
4     126
5     123
9     111
8     111
1     109
10    101
7      86
2      77
Name: count, dtype: int64
<Axes: xlabel='Europe', ylabel='count'>
```

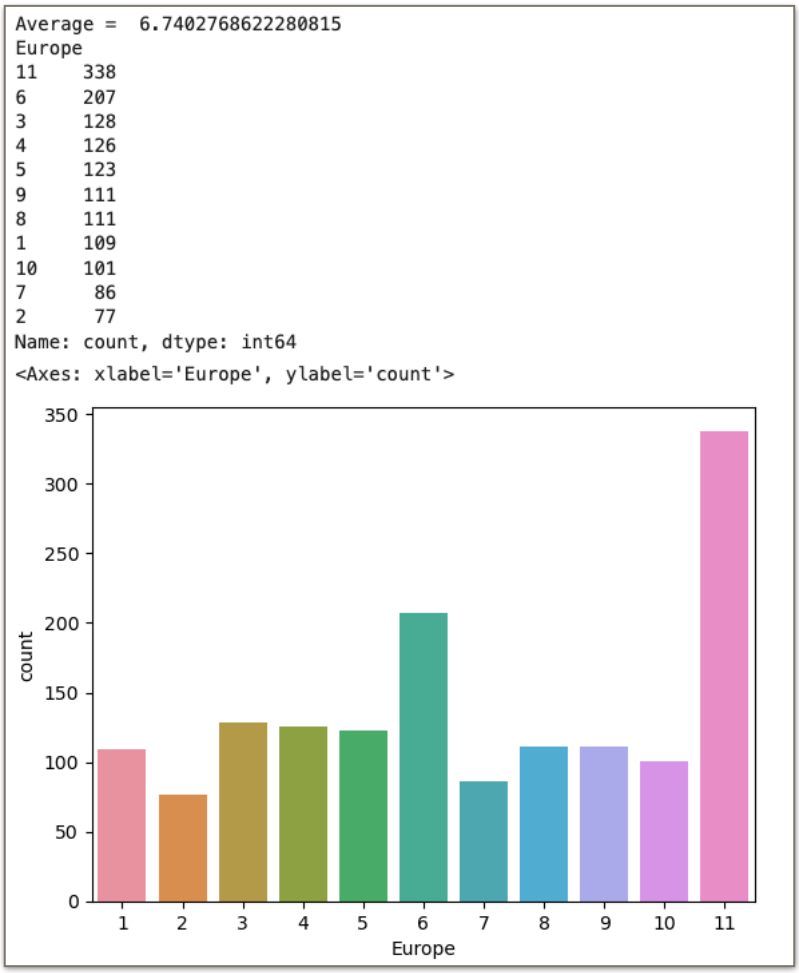


Fig: Count-plot for Europe

Observation

- 11 has the highest count of 338.
- The average of Europe is 6.7402768622280815
- 2 has the least count of 77.

political.knowledge

```
Average =  1.5405405405405406
political.knowledge
2    776
0    454
3    249
1     38
Name: count, dtype: int64
<Axes: xlabel='political.knowledge', ylabel='count'>
```

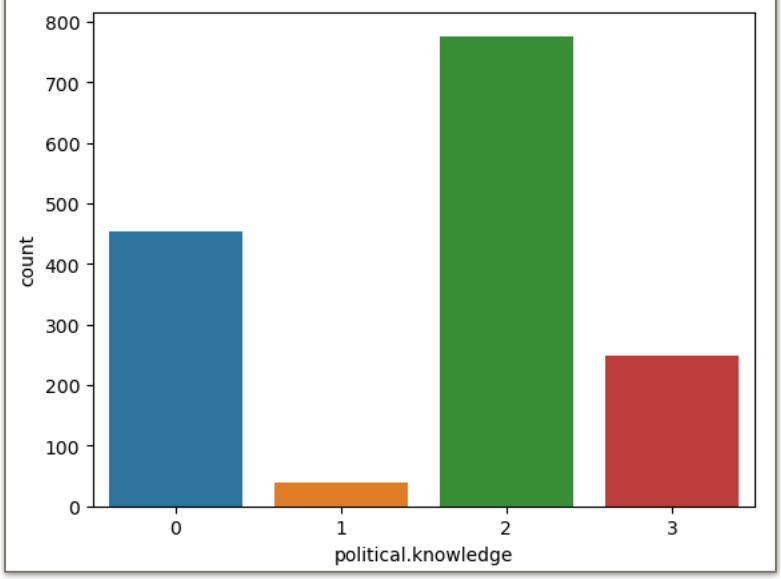


Fig: Count-plot for political.knowledge

Observation

- The average of political.knowledge is 1.5405405405405406
- 2 has the highest count of 776.
- 1 has least count of 38.

Bivariate Analysis

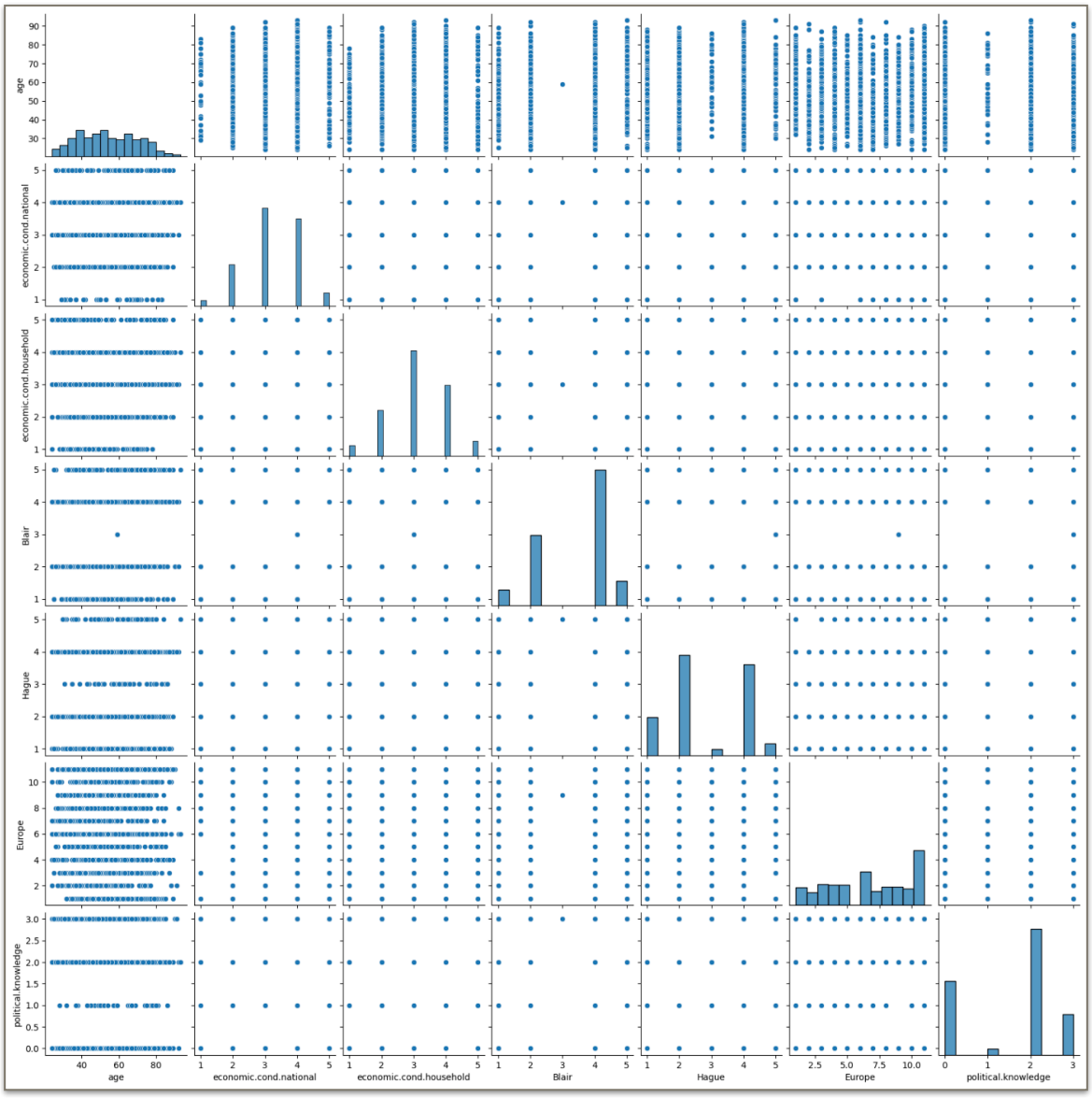


Fig: Pair-plot for the dataset

Observation

- From the histogram, we can see that, the 'Blair','Europe' and 'political.knowledge' variables are slightly left skewed.
- From the scatter plots, we can see that, there is mostly no correlation between the variables.
- All variables seems to normally distributed.
- Using a heatmap to visualise the correlation matrix is an effective way to identify multicollinearity. In the heatmap, you can easily identify pairs of variables that have high correlations, indicating potential multicollinearity issues.

	vote	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge
vote	1.000000	-0.109274	0.302280	0.174688	0.426606	-0.468186	-0.384612	-0.111589
age	-0.109274	1.000000	0.018687	-0.038868	0.032084	0.031144	0.064562	-0.046598
economic.cond.national	0.302280	0.018687	1.000000	0.347687	0.326141	-0.200790	-0.209150	-0.023510
onomic.cond.household	0.174688	-0.038868	0.347687	1.000000	0.215822	-0.100392	-0.112897	-0.038528
Blair	0.426606	0.032084	0.326141	0.215822	1.000000	-0.243508	-0.295944	-0.021299
Hague	-0.468186	0.031144	-0.200790	-0.100392	-0.243508	1.000000	0.285738	-0.029906
Europe	-0.384612	0.064562	-0.209150	-0.112897	-0.295944	0.285738	1.000000	-0.151197
political.knowledge	-0.111589	-0.046598	-0.023510	-0.038528	-0.021299	-0.029906	-0.151197	1.000000

Fig: Correlation between the variables

Observation

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

- Moderate positive correlation exists between economic.cond.national' and 'economic.cond.household'.
- 'Blair' shows moderate positive correlation with 'economic.cond.national' and 'economic.cond.household'.
- Moderate positive correlation is observed between 'Europe' and 'Hague'.
- Moderate negative correlation is found between 'Hague' and 'economic.cond.national', as well as 'Blair'.
- 'Europe' exhibits moderate negative correlation with 'economic.cond.national' and ‘Blair'.

Analysis Blair vs Age and Hague vs Age

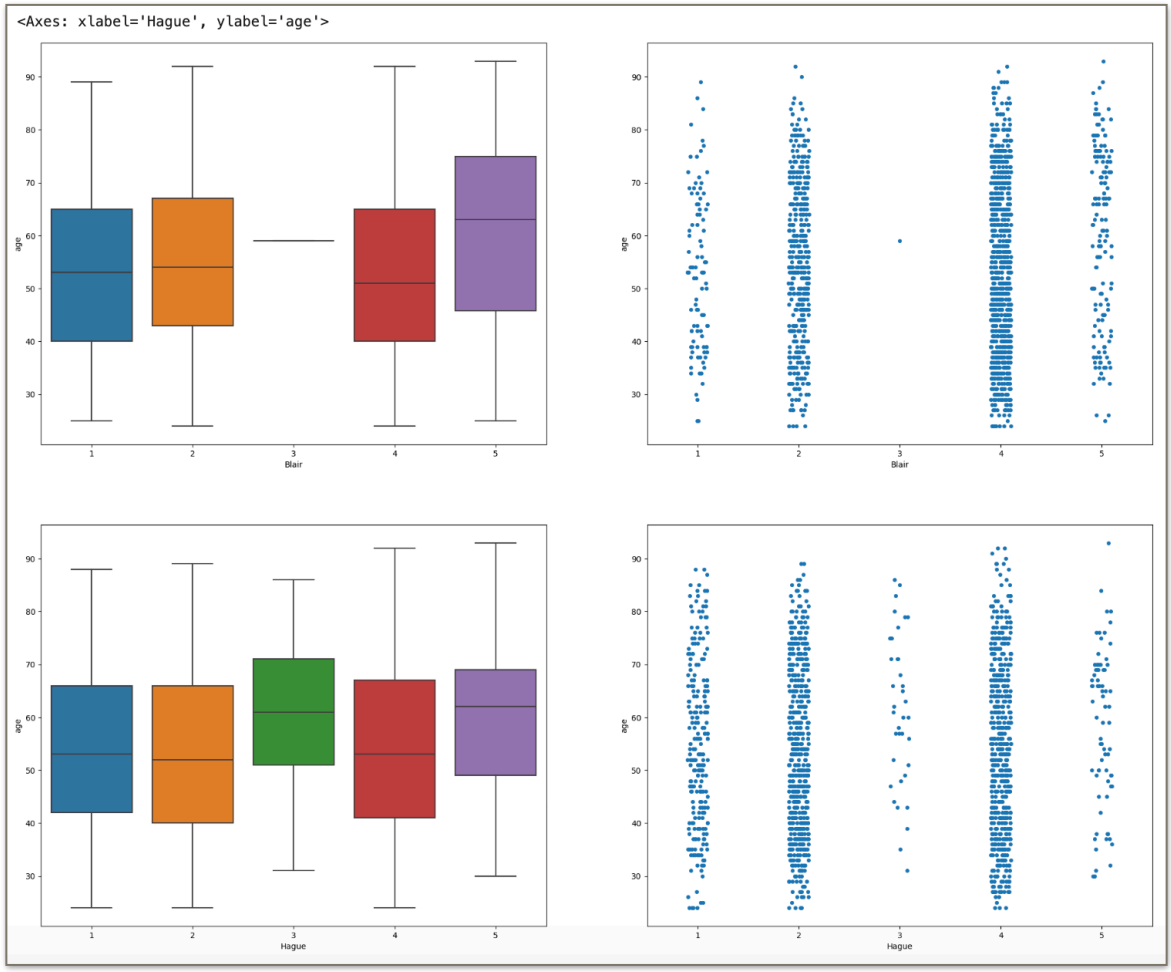


Fig: Box-plot and Scatter-plot for Blair vs Age and Hague vs Age

Observation

- The age distribution for Blair appears more spread out.
- The age distribution for Hague seems more concentrated.
- People above 40 years thinks Blair is doing a good job.
- Hague has slightly more concentration of neutral points than that of Blair for people above 50 years of age.

Heatmap

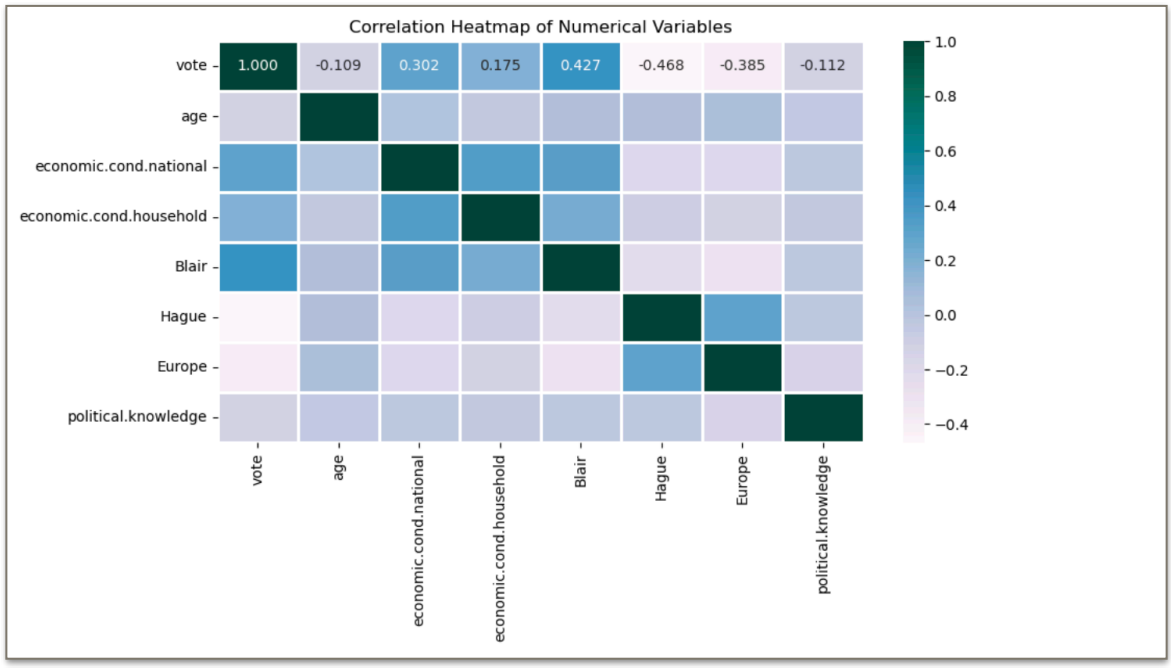


Fig: Heatmap between the variables

Observation

- This negative correlation suggests that as age increases, the likelihood of voting decreases slightly (although the correlation is weak).
- This negative correlation indicates that older individuals tend to have lower “Blair” scores.
- The correlation coefficient between “Hague” and “age” is approximately -0.385.
- Moderate positive correlation exists between economic.cond.national' and 'economic.cond.household'.
- 'Blair' shows moderate positive correlation with 'economic.cond.national' and 'economic.cond.household'.
- Moderate positive correlation is observed between 'Europe' and 'Hague'.
- Moderate negative correlation is found between 'Hague' and 'economic.cond.national', as well as 'Blair'.
- 'Europe' exhibits moderate negative correlation with 'economic.cond.national' and ‘Blair’.

Stacked Plot

```
vote      0     1   All
gender
All     460  1057  1517
female  257   551   808
male    203   506   709
```

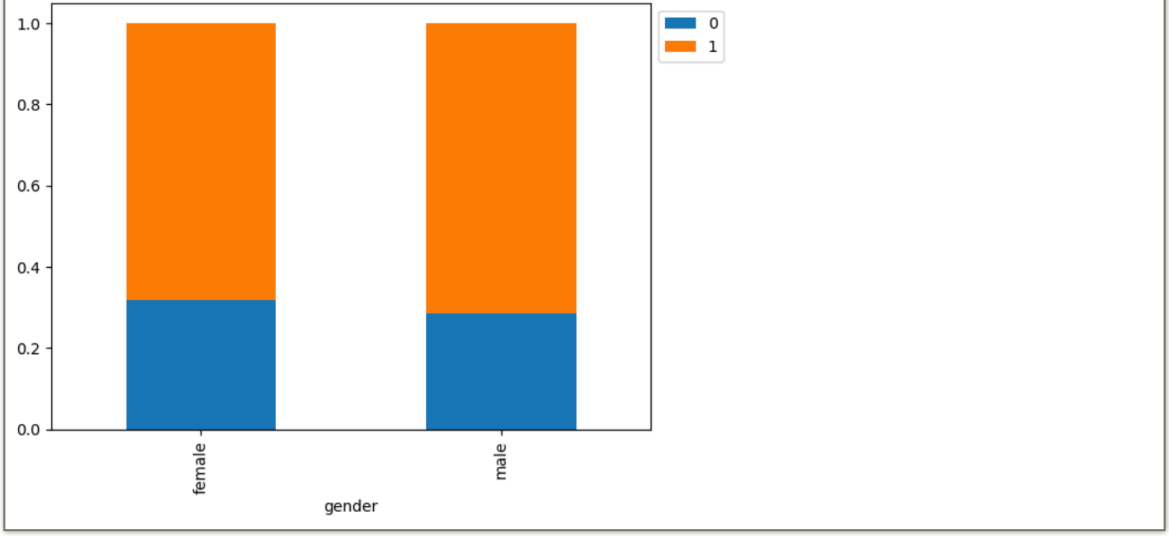


Fig: The category counts and plot a stacked bar chart

Observation

- The plot represents voting results categorised by gender.
- There are two bars: one labeled “female” and the other “male”.
- Each bar is divided into two colors:
  - Blue: Represents do not votes labeled as “0”.
  - Orange: Represents votes labeled as “1”.
- Notably, both genders have a higher count of votes labeled as “1”.
- There were 257 female votes labeled as “0.”
- A total of 551 female votes were labeled as “1.”
- Among males, there were 203 votes labeled as “0.”
- A significant 506 male votes were labeled as “1.”

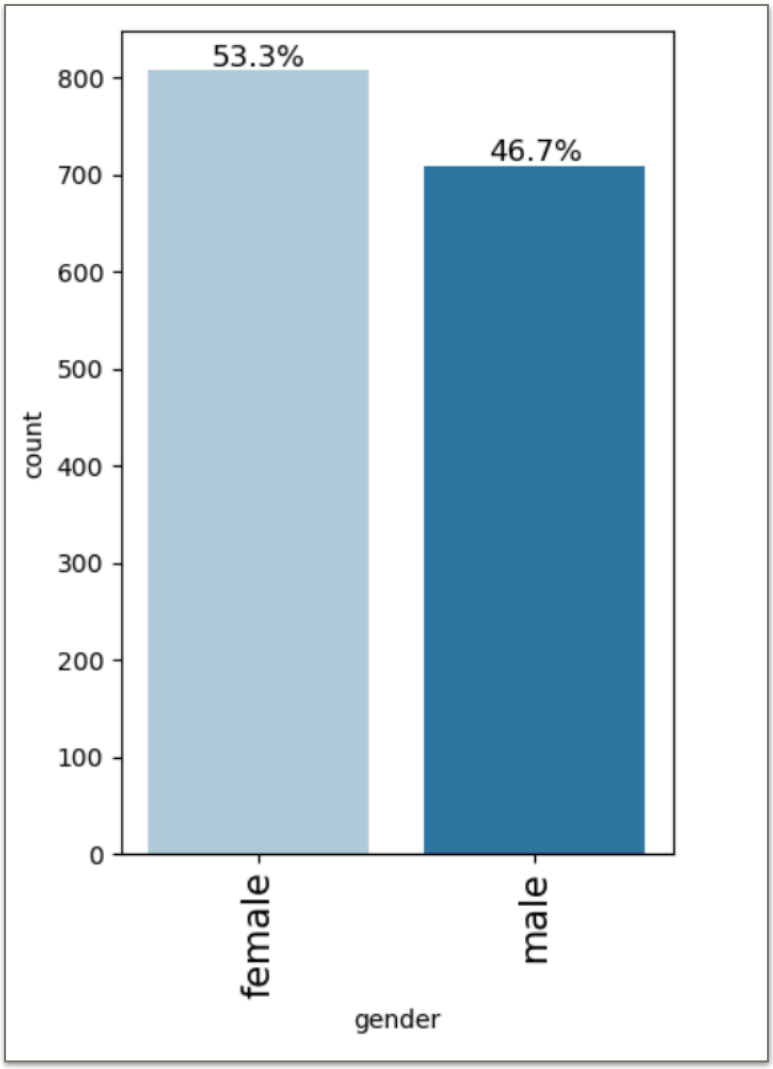


Fig: Bar Plot with Percentage at the top.

Observation

- Percentages are indicated above each bar:
  - Female: 53.3%
  - Male: 46.7%
- Overall, it appears that the female count is slightly higher than the male count in the given data.

Distribution Plot And Boxplot

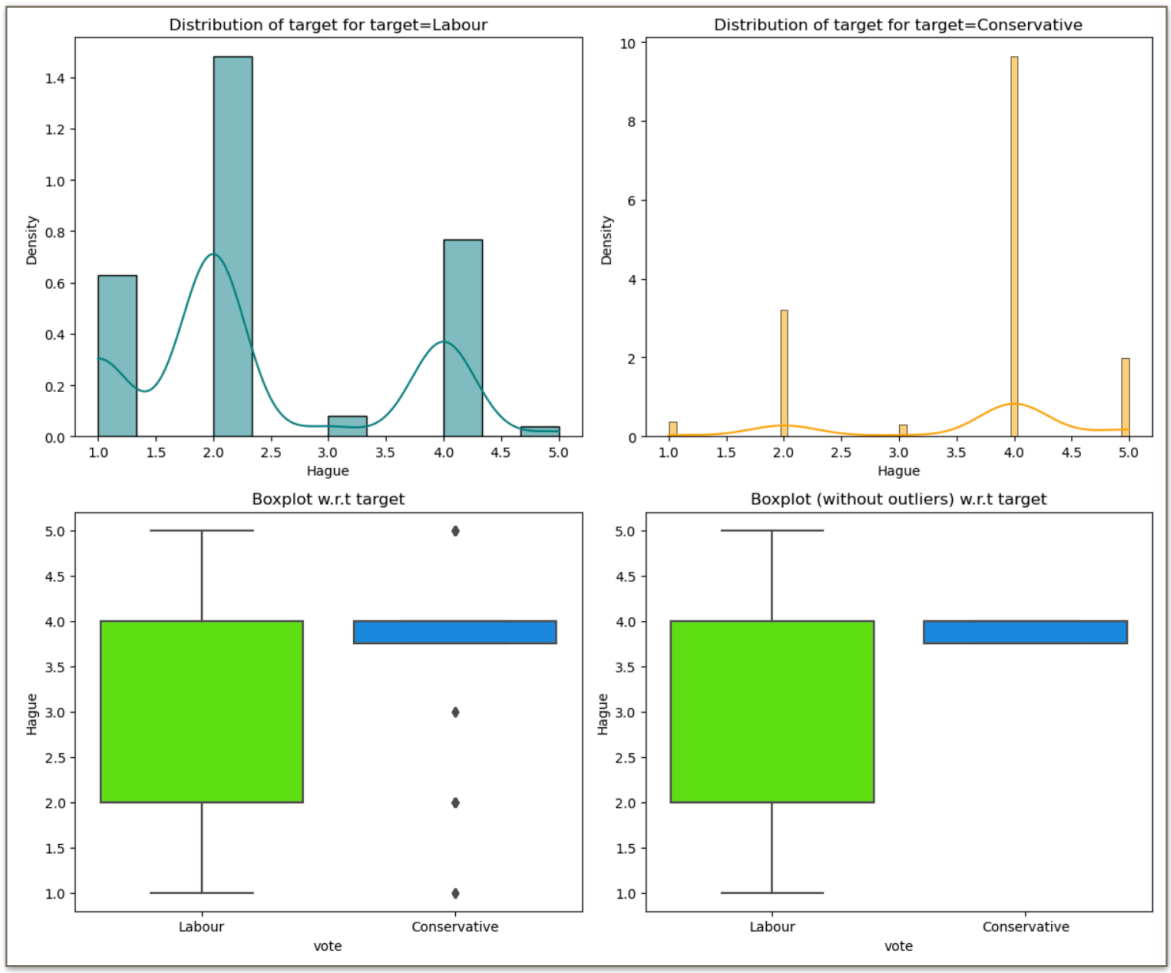


Fig: Distribution Plot Hague vs Vote

Observation

- For the Labour party, there is a prominent peak around 2.0 Hague.
- For the Conservative party, the peak is around 4.0 Hague.
- The bottom right plot is a cleaner version of the box-plot without outliers.
- The Labour party has a wider spread of data compared to the Conservative party.
- The central tendency (median) for Labour is higher than that for Conservative.

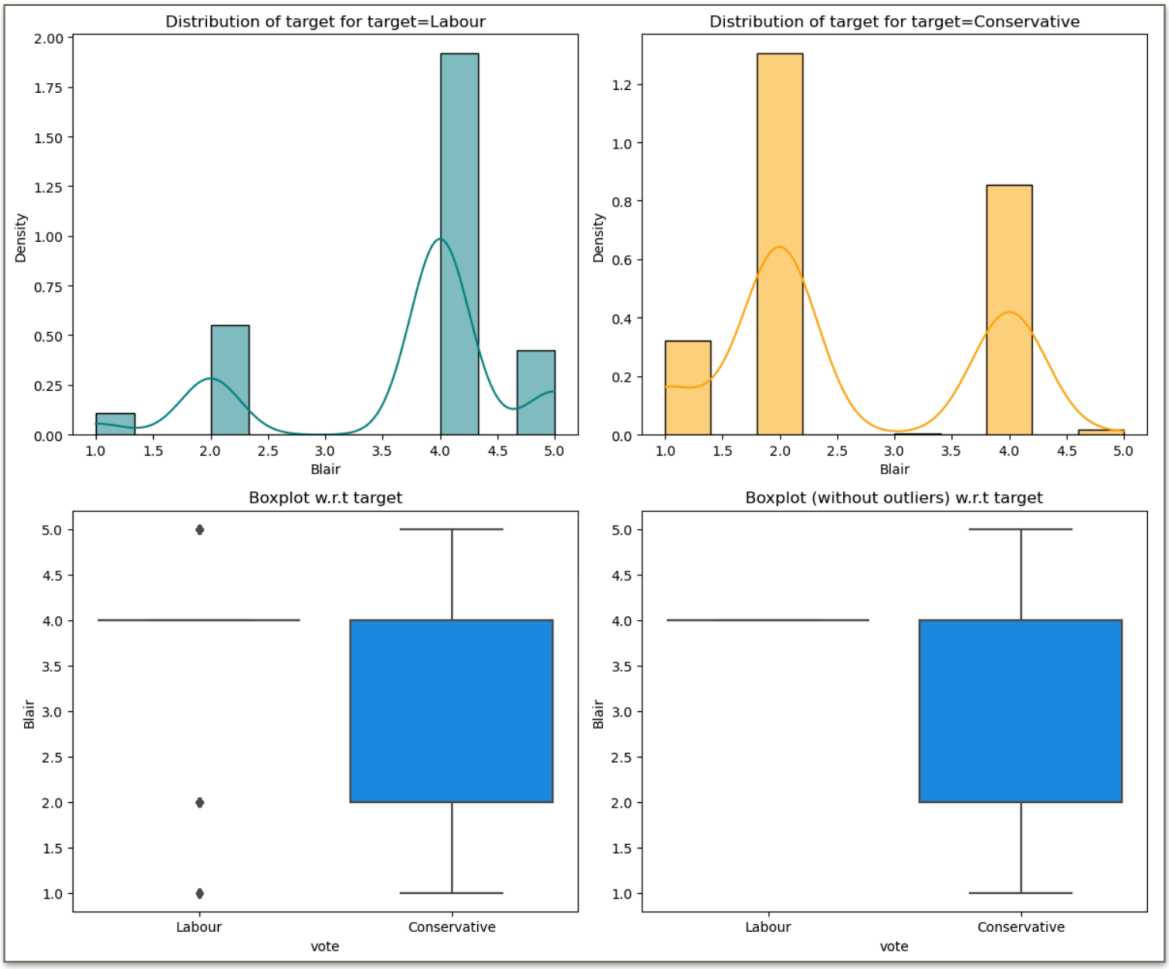


Fig: Distribution Plot and Box-plot Blair vs Vote

Observation

- For the Labour party, there is a prominent peak around 4.0 Blair.
- For the Conservative party, the peak is around 2.0 Hague.
- The bottom right plot is a cleaner version of the box-plot without outliers.
- The Conservative party has a wider spread of data compared to the Labour party.
- The density values in the “Conservative” distribution plot are slightly lower than those in the “Labour” plot.

1.2 Data Pre-processing

Outlier Treatment/Check

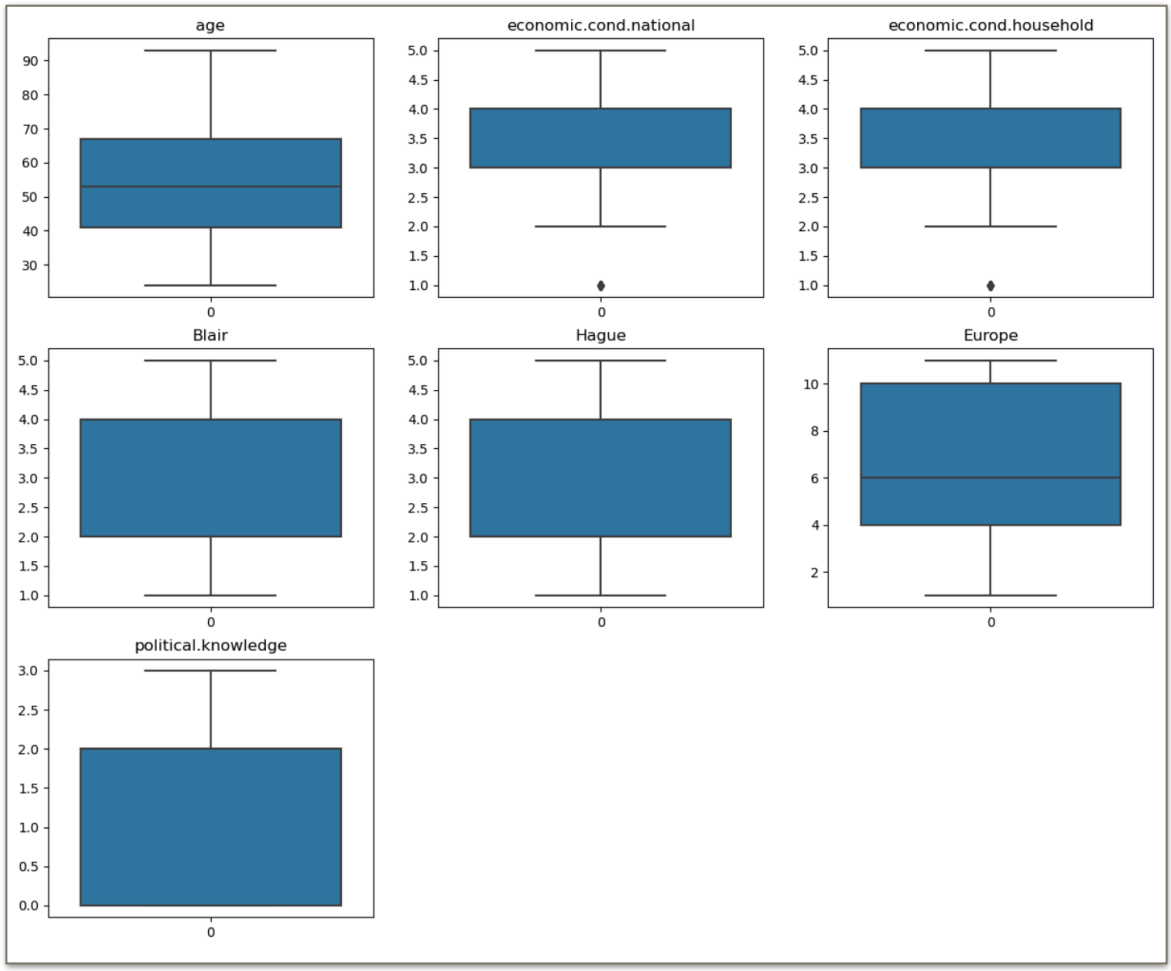


Fig: Box-plot

Observations

- In the first row of plots, "age" has a wider interquartile range (IQR) compared to “economic.cond.national” and “economic.cond.household.” Both economic condition plots exhibit outliers.
- The plot for "political knowledge" at the bottom displays no outliers and a smaller IQR compared to the other variables.
- In the second row, "Blair," "Hague," and "Europe" have similar IQRs but different median values.

Train - Test Split

Target Variable Encoding:

- The target variable "vote" has been encoded to binary labels: 1 for "Labour" and 0 for other values.

Feature Engineering:

- The feature set X is prepared by dropping the "vote" column from the dataset.

Categorical Variable Encoding:

- Categorical variables in the feature set X are one-hot encoded using pd.get_dummies(), creating dummy variables for each category.

Data Splitting:

- The dataset is split into training and testing sets using train_test_split().
- The split ratio is set to 70:30 for training and testing respectively.
- Stratification based on the target variable "vote" is applied to maintain class distribution in both sets.

```
Shape of Training set :  (1061, 9)
Shape of test set :  (456, 9)
Percentage of classes in training set:
vote
1    0.696513
0    0.303487
Name: proportion, dtype: float64
Percentage of classes in test set:
vote
1    0.697368
0    0.302632
Name: proportion, dtype: float64
```

Fig: Train-Test Split Shape

Observation

Training Set Shape:

- The training set consists of 1061 samples and 9 features.

Test Set Shape:

- The test set comprises 456 samples and 9 features.

Class Distribution in Training Set:

- The training set is imbalanced, with approximately 69.65% of samples belonging to class 1 (e.g., "Labour") and around 30.35% belonging to class 0.

Class Distribution in Test Set:

- Similarly, the test set also exhibits class imbalance, with about 69.74% of samples belonging to class 1 and approximately 30.26% belonging to class 0.

Encoding

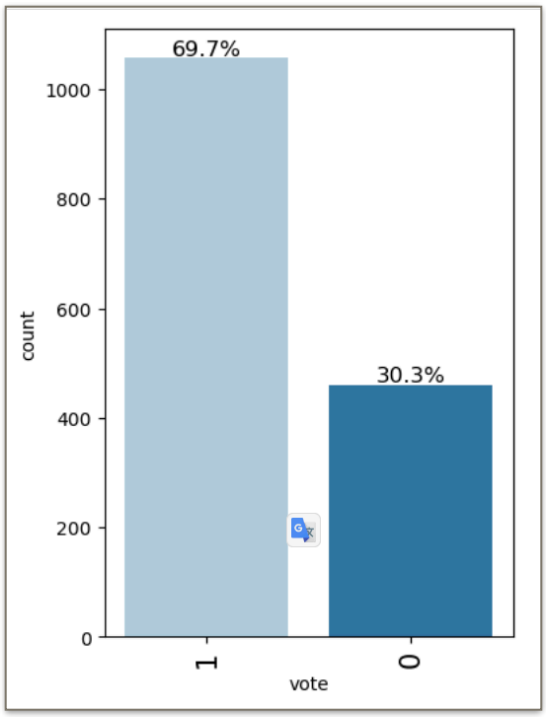


Fig: Bar-plot for Categorical Variable Encoding(vote)

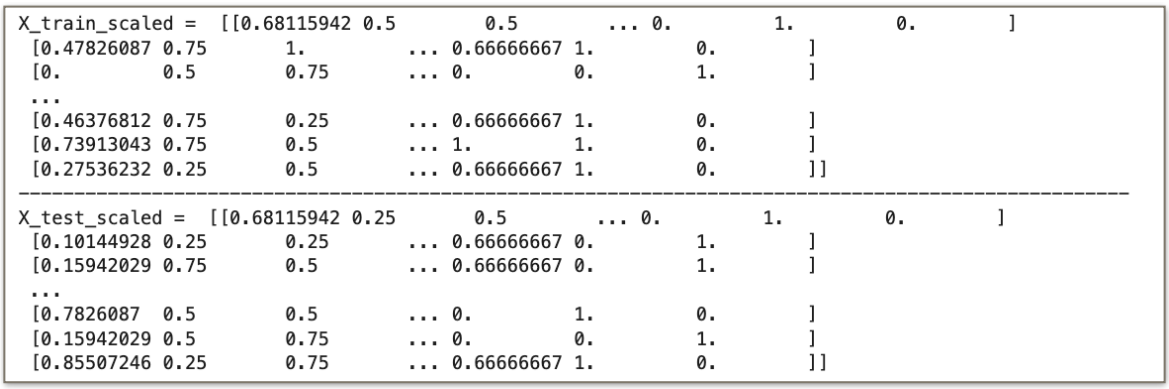
- The apply() function is used to apply a lambda function to each element of the 'vote' column, converting 'Labour' to 1 and 'Conservative' to 0.
- The feature set X is prepared by dropping the 'vote' column from the DataFrame.
- One-hot encoding is then applied to the categorical features in X using pd.get_dummies().

- Ensure to replace 'Labour' and 'Conservative' with the actual class labels in your dataset. This code will create dummy variables for each category in the categorical features, with 'Labour' encoded as 1 and 'Conservative' encoded as 0.

Scaling

Scaling is a preprocessing technique used in machine learning to standardise the range of features or variables in the dataset. It involves transforming the values of features into a specified range or distribution to make them more comparable and suitable for modelling.

The main goal of scaling is to ensure that features with different scales or units have equal weight during model training. When features have different scales, some machine learning algorithms might give more importance to features with larger scales, leading to biased or inaccurate results. Scaling helps to mitigate this issue by bringing all features to a similar scale.



```
X_train_scaled =  [[0.68115942 0.5        0.5        ... 0.         1.         0.        ]
 [0.47826087 0.75       1.         ... 0.66666667 1.         0.        ]
 [0.         0.5        0.75       ... 0.         0.         1.        ]
 ...
 [0.46376812 0.75       0.25       ... 0.66666667 1.         0.        ]
 [0.73913043 0.75       0.5        ... 1.         1.         0.        ]
 [0.27536232 0.25       0.5        ... 0.66666667 1.         0.        ]]
--------------------------------------------------------------------------------------------------------------
X_test_scaled =  [[0.68115942 0.25       0.5        ... 0.         1.         0.        ]
 [0.10144928 0.25       0.25       ... 0.66666667 0.         1.        ]
 [0.15942029 0.75       0.5        ... 0.66666667 0.         1.        ]
 ...
 [0.7826087  0.5        0.5        ... 0.         1.         0.        ]
 [0.15942029 0.5        0.75       ... 0.         0.         1.        ]
 [0.85507246 0.25       0.75       ... 0.66666667 1.         0.        ]]
```

Fig: Scaled dataset

For scaling the data we did:

- We import the MinMaxScaler from sklearn.preprocessing.
- We initialize the MinMaxScaler object.
- We fit the scaler to the training data (X_train) and simultaneously transform it.
- We transform the test data (X_test) using the same scaler that was fitted to the training data. It's important to use the same scaler for both training and testing data to ensure consistency.

After scaling, the features in both the training and testing sets will have a mean of 0 and a standard deviation of 1, which helps in standardizing the range of the features and makes them comparable.

1.3 Model Building

Bagging - Model Building and Hyper-parameter Tuning

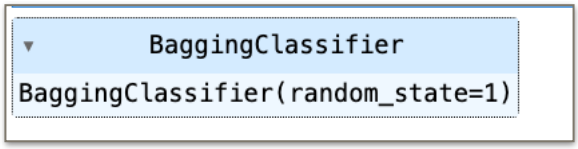
Bagging, also known as bootstrap aggregation, is the ensemble learning method that is commonly used to reduce variance within a noisy data set. It works by training each base model on a random subset of the training data, with replacement (bootstrapping), and then aggregating their predictions to make the final prediction. Bagging is primarily used for reducing variance and improving the stability and robustness of machine learning models.

Initialization:

- A BaggingClassifier model is created.
- The random state parameter is set to 1. This parameter ensures that the random number generation used for bootstrapping and selecting features will be reproducible. Setting it to a specific value (in this case, 1) allows for consistent results across different runs of the code.

Training:

- The BaggingClassifier model is trained on the training data (X_train and y_train) using the fit() method. During training, the model learns patterns in the data and constructs an ensemble of base classifiers, typically decision trees, each trained on a random subset of the training data. The aggregation of predictions from these base classifiers forms the final prediction of the BaggingClassifier.



Checking model performance on training set

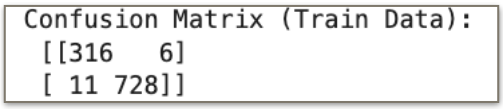


Fig: Confusion Matrix of Train Data

In this case, the confusion matrix you've provided is a 2x2 matrix representing the counts of true positive (TP), false positive (FP), true negative (TN), and false negative (FN) predictions made by the model on the training data. Here's a breakdown:

- True Positives (TP): 316
- False Positives (FP): 6

- False Negatives (FN): 11
- True Negatives (TN): 728

	Accuracy	Recall	Precision	F1
0	0.983977	0.985115	0.991826	0.988459

Fig: Accuracy, Recall, Precision and F1 values for Train Data

Accuracy = 98.4%
Recall = 98.5%
Precision = 99.2%
F1-score = 98.8%

Checking model performance on test set

```
Confusion Matrix (Test Data):
 [[101  37]
 [ 34 284]]
```

The confusion matrix provided appears to be for the test data. As with the training data confusion matrix, this is a 2x2 matrix representing the counts of true positive (TP), false positive (FP), true negative (TN), and false negative (FN) predictions made by the model on the test data. Here's the breakdown:

- True Positives (TP): 101
- False Positives (FP): 37
- False Negatives (FN): 34
- True Negatives (TN): 284

	Accuracy	Recall	Precision	F1
0	0.844298	0.893082	0.884735	0.888889

Accuracy =84.4%
Recall = 89.3%
Precision = 88.5%
F1-score = 88.9%

Hyperparameter Tuning - Bagging Classifier

Hyperparameter tuning for a Bagging Classifier involves optimizing the parameters that control the ensemble of base classifiers, such as the number of base estimators, the sampling strategy, and the parameters of the base estimator itself.

Bagging Classifier:

- A Bagging Classifier is an ensemble meta-estimator that fits base classifiers on random subsets of the original dataset and then aggregates their individual predictions by voting (for classification) or averaging (for regression) to form a final prediction. It helps improve the stability and accuracy of the model.

GridSearchCV:

- GridSearchCV is a method provided by scikit-learn for hyperparameter tuning. It systematically searches through a predefined grid of hyperparameters, performing cross-validation at each point, and selects the combination of hyperparameters that maximizes the performance metric specified. It exhaustively tries all combinations of hyperparameters.

F1 Score:

- The F1 score is a metric used to evaluate the performance of a classification model. It considers both the precision and recall of the model to compute a single score. It is the harmonic mean of precision and recall, with values ranging from 0 to 1, where a higher value indicates better model performance.

Parameter Grid:

- The parameter grid is a dictionary specifying the hyperparameters and their corresponding values that will be searched during the hyperparameter tuning process. In this context, it defines the set of values for n_estimators (number of base estimators) and max_samples (the proportion of samples to draw from the original dataset when training each base estimator).

Cross-Validation:

- Cross-validation is a resampling technique used to evaluate machine learning models on a limited dataset. It partitions the dataset into multiple subsets (folds), trains the model on a combination of these subsets, and evaluates its performance on the remaining subset. This helps to assess how the model generalizes to unseen data.

Best Estimator:

- After hyperparameter tuning, the best estimator is the model with the optimal combination of hyperparameters found during the search process. It encapsulates the trained model with the best hyperparameters, ready to be used for making predictions on new data.

From this code:

- **Parameter Grid**: Specifies hyperparameters like n_estimators=100 and max_samples for tuning.
- **Scoring Metric**: Utilizes F1-score to evaluate model performance during grid search, with options for other metrics like accuracy or precision.
- **Cross-Validation**: Employs 5-fold cross-validation (cv=5) to robustly evaluate model performance on multiple subsets of the data.
- **Best Estimator**: Identifies the optimal model configuration using grid_obj.best_estimator_ after the grid search.
- **Fitting the Model**: Trains the best model on the entire training dataset using fit(X_train, y_train) for final deployment.

```
BaggingClassifier
BaggingClassifier(max_samples=0.5, n_estimators=100, random_state=1)
```

Fig: Bagging Classifier

Checking model performance on training set

```
Confusion Matrix (Train Data - Tuned Estimator):
 [[284  38]
 [ 20 719]]
```

- True Positives (TP): 284
- False Positives (FP): 38
- False Negatives (FN): 20
- True Negatives (TN): 719

	Accuracy	Recall	Precision	F1
0	0.945335	0.972936	0.949802	0.96123

Accuracy =94.5%
Recall = 97.3%
Precision = 94.9%

F1-score = 96.1%

Checking model performance on test set

```
Confusion Matrix (Test Data - Tuned Estimator):
 [[ 96  42]
 [ 28 290]]
```

- True Positives (TP): 96
- False Positives (FP): 42
- False Negatives (FN): 28
- True Negatives (TN): 290

	Accuracy	Recall	Precision	F1
0	0.846491	0.91195	0.873494	0.892308

Accuracy = 85.6%
Recall = 91.2%
Precision = 97.3%
F1-score = 89%

Random Forest

```
▾                    RandomForestClassifier
RandomForestClassifier(class_weight='balanced', random_state=1)
```

Fig: Random Forest Classifier

Checking model performance on training set

```
Confusion Matrix (Train Data):
 [[322   0]
 [  0 739]]
```

- True Positives (TP): 322
- False Positives (FP): 0
- False Negatives (FN): 0
- True Negatives (TN): 739

	Accuracy	Recall	Precision	F1
0	1.0	1.0	1.0	1.0

Accuracy = 100%
Recall = 100%
Precision = 100%
F1-score = 100%

Checking model performance on test set

```
Confusion Matrix (Test Data):
 [[ 93  45]
 [ 27 291]]
```

- True Positives (TP): 93
- False Positives (FP): 45
- False Negatives (FN): 27
- True Negatives (TN): 291

	Accuracy	Recall	Precision	F1
0	0.842105	0.915094	0.866071	0.889908

Accuracy = 84.2%
Recall = 91.5%
Precision = 86.6%
F1-score = 88.9%

Hyperparameter Tuning - Random Forest

```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=10, oob_score=True, random_state=1)
```

Classifier Selection: RandomForestClassifier is chosen for modeling.
Parameter Grid Definition: Two hyperparameters (n_estimators and max_depth) are defined for tuning.
Scoring Metric: F1-score is selected as the metric to evaluate model performance.

Grid Search Execution: GridSearchCV is used to explore parameter combinations using 5-fold cross-validation with parallel processing.
Best Estimator Identification: The best model configuration is determined using grid_obj.best_estimator_.
Model Fitting: The best model is trained on the entire training data for final deployment.

```
Confusion Matrix (Train Data - Tuned Estimator):
 [[302  20]
 [ 16 723]]
```

- True Positives (TP): 302
- False Positives (FP): 20
- False Negatives (FN): 16
- True Negatives (TN): 723

	Accuracy	Recall	Precision	F1
0	0.96607	0.978349	0.973082	0.975709

Accuracy = 96.6%
Recall = 97.8%
Precision = 97.3%
F1-score = 97.6%

Checking model performance on test set

```
Confusion Matrix (Test Data - Tuned Estimator):
 [[ 93  45]
 [ 26 292]]
```

- True Positives (TP): 93
- False Positives (FP): 45
- False Negatives (FN): 26
- True Negatives(TN): 292

	Accuracy	Recall	Precision	F1
0	0.844298	0.918239	0.866469	0.891603

Accuracy = 84.4%
Recall = 91.8%

Precision = 86.6%
F1-score = 89.2%

Boosting - Model Building and Hyperparameter Tuning

AdaBoost Classifier

AdaBoost, short for Adaptive Boosting, is a popular ensemble learning method used for classification tasks.

AdaBoost has several advantages, including its ability to handle noisy data and its resistance to overfitting. However, it can be sensitive to outliers and noisy data points, which may negatively affect its performance. Overall, AdaBoost is a powerful algorithm for classification tasks, especially when combined with weak learners that have low bias and high variance.

```
AdaBoostClassifier
AdaBoostClassifier(random_state=1)
```

- We use AdaBoostClassifier from the sklearn.ensemble module to create an instance of the AdaBoost Classifier.
- The parameter random_state is set to 1 to ensure reproducibility of results.
- We then fit the AdaBoost Classifier to the training data (X_train and y_train) using the fit() method, allowing the classifier to learn from the data and adapt to the underlying patterns.

Checking model performance on training set

```
Confusion Matrix (Train Data):
 [[227  95]
 [ 75 664]]
```

- True Positives (TP): 227
- False Positives (FP): 95
- False Negatives (FN): 75
- True Negatives(TN): 664

	Accuracy	Recall	Precision	F1
0	0.839774	0.898512	0.874835	0.886515

Accuracy = 83.9%
Recall = 89.8%
Precision = 86.6%

F1-score = 88.7%

Checking model performance on test set

```
Confusion Matrix (Test Data):
 [[ 93  45]
 [ 30 288]]
```

- True Positives (TP): 93
- False Positives (FP): 45
- False Negatives (FN): 30
- True Negatives(TN): 288

	Accuracy	Recall	Precision	F1
0	0.835526	0.90566	0.864865	0.884793

Accuracy = 83.5%
Recall = 90.6%
Precision = 86.5%
F1-score = 88.5%

Hyperparameter Tuning - AdaBoost Classifier

- **Classifier Selection**: AdaBoostClassifier is chosen as the classifier, initialized with default parameters and a random state of 1.
- **Parameter Grid Definition**: A parameter grid parameters is defined, specifying the hyperparameters (n_estimators and learning_rate) to be tuned along with their respective values.
- **Scoring Metric**: F1-score is selected as the scoring metric to evaluate the performance of different parameter combinations during the grid search.
- **Grid Search Execution**: GridSearchCV is utilized to perform hyperparameter tuning. It systematically explores the parameter grid using 5-fold cross-validation and searches for the combination of hyperparameters that maximizes the F1-score.
- **Best Estimator Identification**: After completing the grid search, the best estimator with the optimal combination of hyperparameters is identified using grid_obj.best_estimator_.
- **Model Fitting**: The best model is then trained on the entire training dataset for final deployment.

```
▾                    AdaBoostClassifier
AdaBoostClassifier(learning_rate=0.1, n_estimators=200, random_state=1)
```

Checking model performance on training set

```
Confusion Matrix (Train Data - Tuned Estimator):
 [[212 110]
 [ 67 672]]
```

- True Positives (TP): 212
- False Positives (FP): 110
- False Negatives (FN): 67
- True Negatives(TN): 672

	Accuracy	Recall	Precision	F1
0	0.833176	0.909337	0.859335	0.883629

Accuracy = 83.3%
Recall = 90.9%
Precision = 85.9%
F1-score = 88.4%

Checking model performance on test set

```
Confusion Matrix (Test Data - Tuned Estimator):
 [[ 91  47]
 [ 23 295]]
```

- True Positives (TP): 91
- False Positives (FP): 47
- False Negatives (FN): 23
- True Negatives(TN): 295

	Accuracy	Recall	Precision	F1
0	0.846491	0.927673	0.862573	0.893939

Accuracy = 84.6%
Recall = 92.8%
Precision = 86.3%
F1-score = 89.4%

1.4 Model Performance evaluation

Model Performance Comparison and Final Model Selection

Training performance comparison:

	Bagging Classifier	Tuned Bagging Classifier	Random Forest	Tuned Random Forest	Adaboost Classifier	Tuned Adaboost Classifier
Accuracy	0.983977	0.945335	1.0	0.966070	0.839774	0.833176
Recall	0.985115	0.972936	1.0	0.978349	0.898512	0.909337
Precision	0.991826	0.949802	1.0	0.973082	0.874835	0.859335
F1	0.988459	0.961230	1.0	0.975709	0.886515	0.883629

Testing performance comparison:

	Bagging Classifier	Tuned Bagging Classifier	Random Forest	Tuned Random Forest	Adaboost Classifier	Tuned Adaboost Classifier
Accuracy	0.844298	0.846491	0.842105	0.844298	0.835526	0.846491
Recall	0.893082	0.911950	0.915094	0.918239	0.905660	0.927673
Precision	0.884735	0.873494	0.866071	0.866469	0.864865	0.862573
F1	0.888889	0.892308	0.889908	0.891603	0.884793	0.893939

Analysis

Bagging Classifier:

- **Training Performance:** Achieves high scores across all metrics, indicating robust performance on the training data.
- **Testing Performance:** Shows good performance on the testing data with decent accuracy, recall, precision, and F1-score.
- **Generalization:** Demonstrates good generalization ability, as seen by its performance on unseen data.
- After using bagging, the model is over-fitted. The values are high. But the difference between the train and test accuracy is high.

Tuned Bagging Classifier:

- **Training Performance:** Achieves slightly lower scores compared to Bagging Classifier, indicating potential improvement opportunities.
- **Testing Performance:** Shows similar performance to the Bagging Classifier on the testing data, with a slight improvement in some metrics.
- **Hyperparameter Tuning:** May benefit from further hyperparameter tuning to enhance performance.
- The tuning of the model has help the model recover from over- fitting.

Random Forest:

- **Training Performance:** Achieves perfect scores across all metrics on the training data, suggesting potential overfitting.
- **Testing Performance:** Shows slightly lower performance compared to the training data, indicating some degree of overfitting.
- **Overfitting:** Likely overfits the training data due to its high complexity, leading to reduced performance on unseen data.

Tuned Random Forest:

- **Training Performance:** Achieves slightly lower scores compared to the Random Forest, indicating some regularization.
- **Testing Performance:** Shows similar performance to the Random Forest on the testing data, with some improvement in recall and F1-score.
- **Hyperparameter Tuning:** Helps mitigate overfitting and improves generalization ability compared to the default Random Forest.

AdaBoost Classifier:

- **Training Performance:** Shows lower scores compared to Bagging Classifier and Random Forest, indicating weaker performance on the training data.
- **Testing Performance:** Performs decently on the testing data but shows lower scores compared to other classifiers.
- **Weak Learners:** Relies on weak learners, which may limit its predictive power compared to stronger base learners used in other classifiers.

Tuned AdaBoost Classifier:

- **Training Performance:** Achieves slightly improved scores compared to the AdaBoost Classifier on the training data.
- **Testing Performance:** Shows improved performance compared to the AdaBoost Classifier on the testing data, with higher F1-score and recall.
- **Hyperparameter Tuning:** Helps enhance performance and improves generalization ability compared to the default AdaBoost Classifier.
- The model is a good model. There is no over-fitting. There is improvement from the regular model.

K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a straightforward algorithm used for classification and regression tasks. It determines the class or value of a data

point based on the majority vote or average of its K nearest neighbours, respectively. The choice of K and distance metric significantly impacts its performance, and it may suffer from scalability issues with large datasets.

- **Classifier Initialization**: KNeighborsClassifier(n_neighbors=5) initializes a KNN classifier with 5 neighbors.
- **Model Training**: knn.fit(X_train, y_train) trains the KNN classifier on the training data (X_train features and y_train labels).
- **Prediction Generation**: pred_label = knn.predict(X_test) generates predictions for the test data (X_test) using the trained KNN model.
- **Model Evaluation**: knn.score(X_test, y_test) evaluates the model's performance on the test data, calculating the accuracy score.

Checking model performance on training set

```
Train =  0.8416588124410933
[[223  99]
 [ 69 670]]
              precision    recall  f1-score   support

           0       0.76      0.69      0.73       322
           1       0.87      0.91      0.89       739

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

- **Accuracy on Training Data:** The accuracy of the KNN model on the training data is approximately 84.17%.
- **Confusion Matrix:**
- True Positives (TP): 223
- True Negatives (TN): 670
- False Positives (FP): 99
- False Negatives (FN): 69
- The model seems to have relatively higher performance in predicting class 1 (positive class) compared to class 0 (negative class).

Classification Report:

- **Precision:** The model's precision for class 0 (negative class) is 0.76, and for class 1 (positive class) is 0.87.
- **Recall:** The model's recall for class 0 is 0.69, and for class 1 is 0.91.
- F1-score: The model's F1-score for class 0 is 0.73, and for class 1 is 0.89.

- **Support:** The number of occurrences of each class in the training data (class 0: 322, class 1: 739).

Observations:

- The model achieves a relatively high accuracy on the training data.
- Precision, recall, and F1-score are higher for class 1 compared to class 0, indicating better performance in predicting positive instances.
- The model's performance seems reasonable, but it may benefit from further optimization or exploration of different algorithms or hyperparameters.

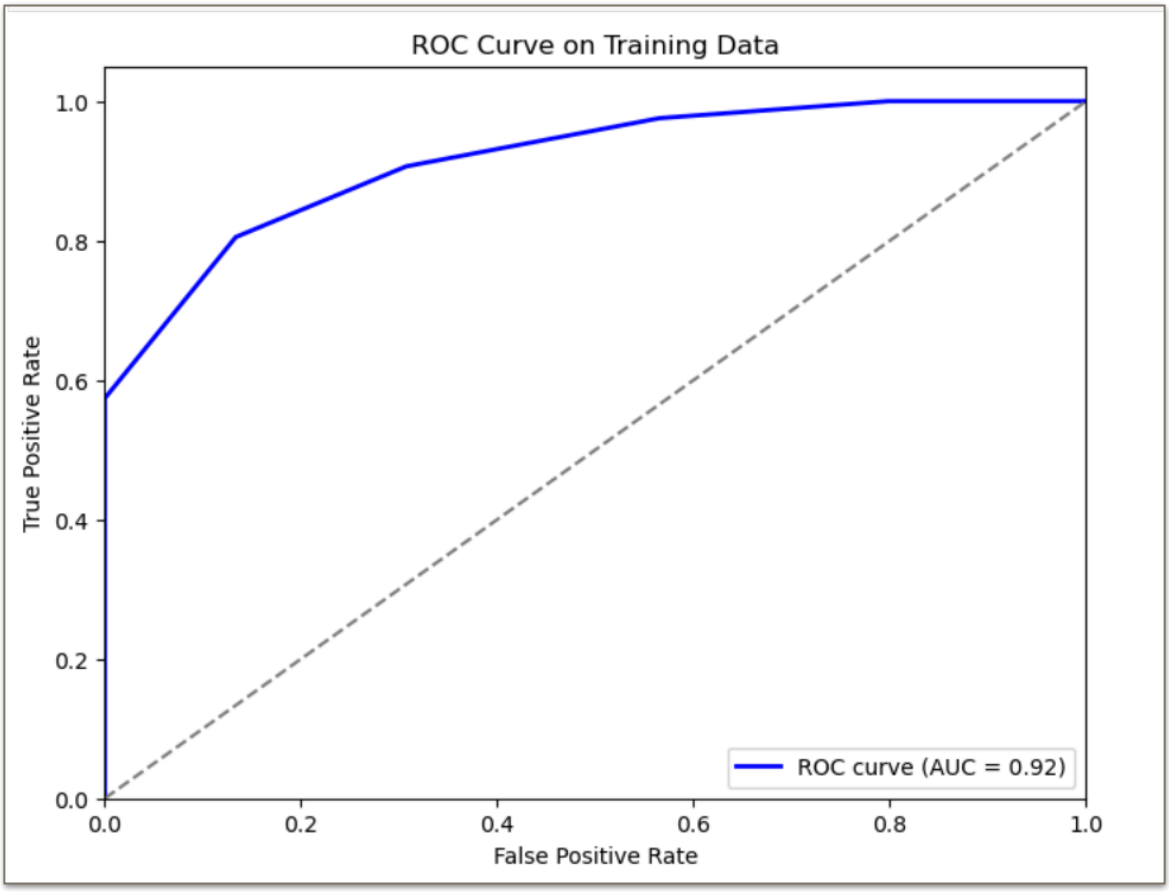


Fig: ROC Curve on Training Data

Checking model performance on testing set

```
Test =  0.8026315789473685
[[ 86  52]
 [ 38 280]]
              precision    recall  f1-score   support

           0       0.69      0.62      0.66       138
           1       0.84      0.88      0.86       318

    accuracy                           0.80       456
   macro avg       0.77      0.75      0.76       456
weighted avg       0.80      0.80      0.80       456
```

Accuracy on Test Data: The accuracy of the KNN model on the test data is approximately 80.26%.

Confusion Matrix:
- True Positives (TP): 86
- True Negatives (TN): 280
- False Positives (FP): 52
- False Negatives (FN): 38

- The model seems to have relatively higher performance in predicting class 1 (positive class) compared to class 0 (negative class).

Classification Report:
- **Precision:** The model's precision for class 0 (negative class) is 0.69, and for class 1 (positive class) is 0.84.
- **Recall:** The model's recall for class 0 is 0.62, and for class 1 is 0.88.
- **F1-score:** The model's F1-score for class 0 is 0.66, and for class 1 is 0.86.
- **Support:** The number of occurrences of each class in the test data (class 0: 138, class 1: 318).

Observations:
- The model achieves a reasonable accuracy on the test data, indicating decent generalization ability.
- Precision, recall, and F1-score are higher for class 1 compared to class 0, indicating better performance in predicting positive instances.
- The model's performance on the test data is consistent with its performance on the training data, suggesting robustness.

Overall, the KNN model demonstrates good performance on the test data, but further analysis and fine-tuning may be needed to improve its performance further or explore alternative algorithms.

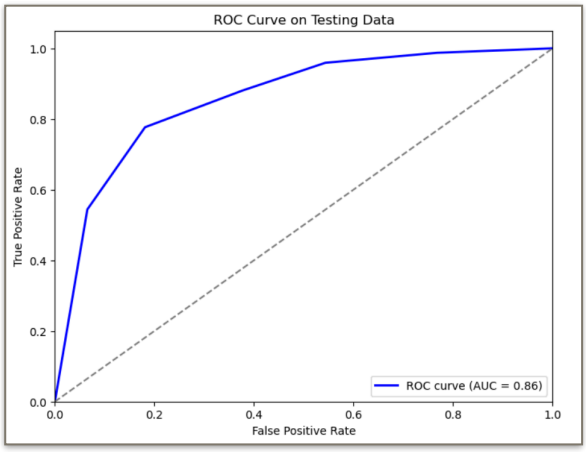


Fig: ROC Curve on Testing Data

Naive Bayes

Naive Bayes is a probabilistic classification algorithm based on Bayes' theorem with an assumption of independence between features.
A Gaussian Naive Bayes (NB) classifier and fits it to the training data (X_train and y_train). This process involves training the NB model on the features (X_train) and corresponding labels (y_train), enabling the model to learn the underlying patterns and relationships in the data to make predictions.

```
[[226  95]
 [ 96 644]]
              precision    recall  f1-score   support

           0       0.70      0.70      0.70       321
           1       0.87      0.87      0.87       740

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

Based on the provided performance metrics for the Gaussian Naive Bayes (NB) training model:

Confusion Matrix:

- True Positives (TP): 226
- True Negatives (TN): 644
- False Positives (FP): 95
- False Negatives (FN): 96

Classification Report:

- **Precision:** The precision for class 0 (negative class) is 0.70, and for class 1 (positive class) is 0.87.

- **Recall:** The recall for class 0 is 0.70, and for class 1 is 0.87.
- **F1-score:** The F1-score for class 0 is 0.70, and for class 1 is 0.87.
- **Support:** The number of occurrences of each class in the dataset (class 0: 321, class 1: 740).

Observation:

- The model achieves an accuracy of approximately 82%, indicating its ability to correctly classify instances.
- Precision, recall, and F1-score are relatively balanced between the two classes, suggesting no significant bias towards any class.
- The model shows consistent performance across different evaluation metrics, indicating its robustness.

Based on the provided performance metrics for the Gaussian Naive Bayes (NB) model on the test dataset:

```
[[102  28]
 [ 36 290]]
              precision    recall  f1-score   support

           0       0.74      0.78      0.76       130
           1       0.91      0.89      0.90       326

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

Confusion Matrix:

- True Positives (TP): 102
- True Negatives (TN): 290
- False Positives (FP): 28
- False Negatives (FN): 36

Classification Report:

- **Precision:** The precision for class 0 (negative class) is 0.74, and for class 1 (positive class) is 0.91.
- **Recall:** The recall for class 0 is 0.78, and for class 1 is 0.89.
- **F1-score:** The F1-score for class 0 is 0.76, and for class 1 is 0.90.
- **Support:** The number of occurrences of each class in the dataset (class 0: 130, class 1: 326).

Observation:

- The model achieves an accuracy of approximately 86% on the test dataset, indicating its ability to correctly classify instances.
- Precision, recall, and F1-score are relatively high for both classes, suggesting the model's effectiveness in predicting both positive and negative instances.
- The model shows consistent performance across different evaluation metrics, indicating its reliability in real-world applications.

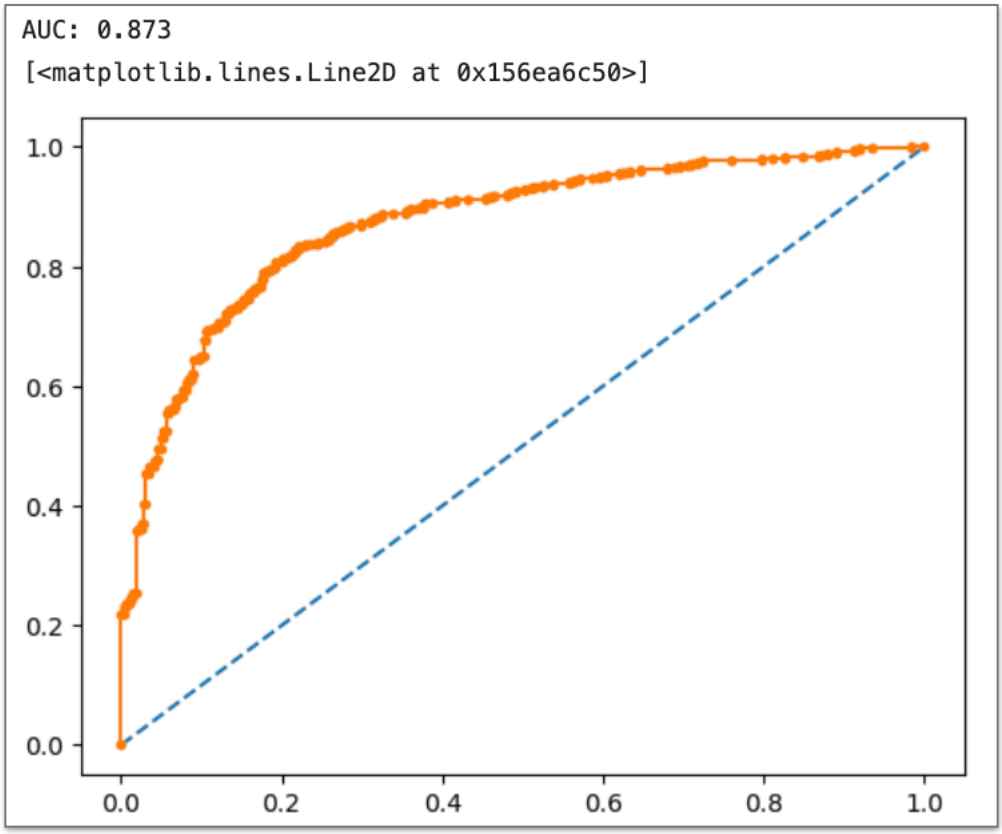


Fig: ROC Curve on Train Data

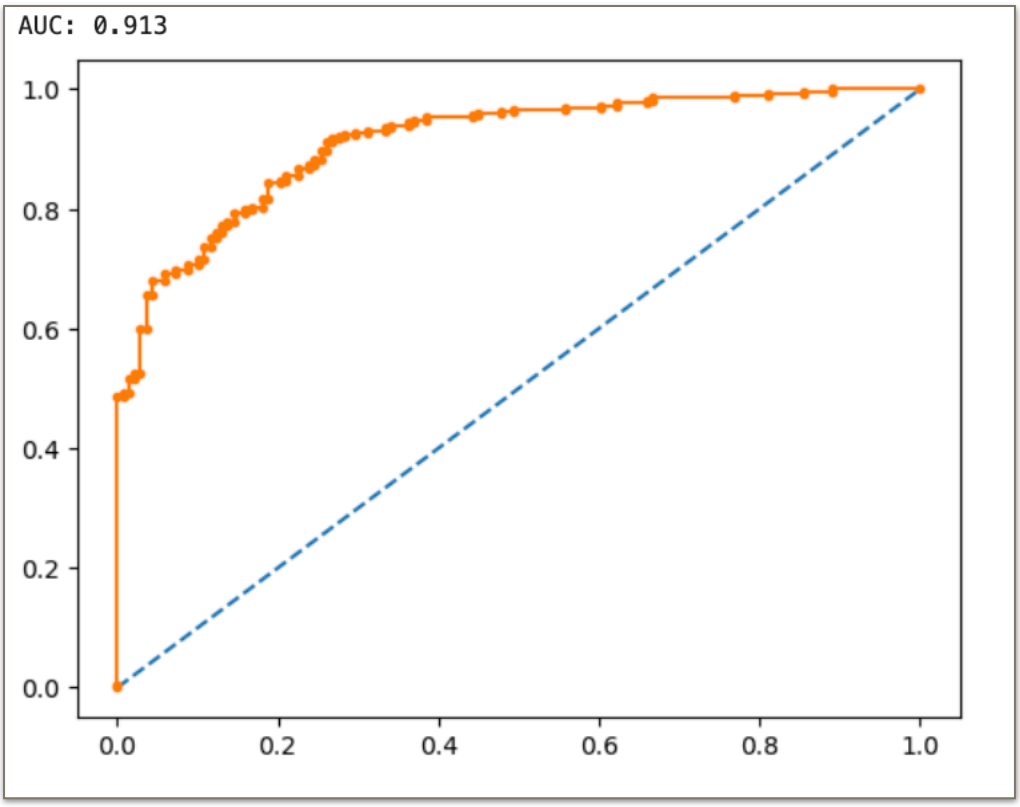
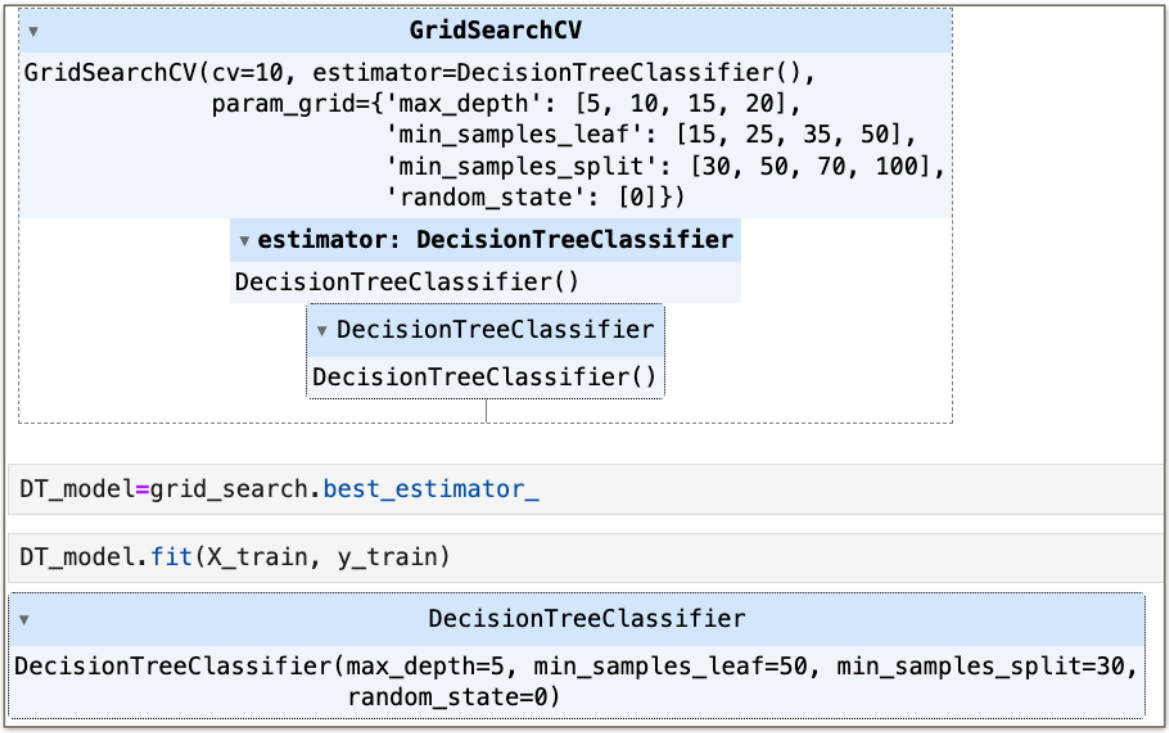


Fig: ROC Curve on Testing Data

Decision Tree

- **Parameter Grid**: The param_grid dictionary defines the grid of hyperparameters to search over during the grid search process. It includes parameters such as min_samples_split, min_samples_leaf, max_depth, and random_state.
- **Grid Search Initialization**: The GridSearchCV object is initialized with the decision tree model (DT_model) and the parameter grid (param_grid). The cv=10 parameter specifies 10-fold cross-validation, splitting the dataset into 10 folds for training and validation.
- **Grid Search Execution**: The grid_search object is created by fitting the grid search to the data. During this process, the algorithm iterates through all possible combinations of hyperparameters defined in the param_grid and evaluates their performance using cross-validation.
- **Cross-Validation**: The cv=10 parameter specifies 10-fold cross-validation, where the dataset is divided into 10 subsets, and the model is trained and evaluated 10 times, each time using a different subset for validation and the remaining data for training.



```
Train =  0.8133836003770029
[[186 136]
 [ 62 677]]
              precision    recall  f1-score   support

           0       0.75      0.58      0.65       322
           1       0.83      0.92      0.87       739

    accuracy                           0.81      1061
   macro avg       0.79      0.75      0.76      1061
weighted avg       0.81      0.81      0.81      1061
```

Based on the provided performance metrics for the decision tree model on the training dataset:

Accuracy on Training Data: The accuracy of the decision tree model on the training data is approximately 81.34%.

Confusion Matrix:

- True Positives (TP): 186
- True Negatives (TN): 677
- False Positives (FP): 136
- False Negatives (FN): 62

The model seems to have relatively higher performance in predicting class 1 (positive class) compared to class 0 (negative class).

Classification Report:

- **Precision:** The precision for class 0 (negative class) is 0.75, and for class 1 (positive class) is 0.83.
- **Recall:** The recall for class 0 is 0.58, and for class 1 is 0.92.
- **F1-score:** The F1-score for class 0 is 0.65, and for class 1 is 0.87.
- **Support:** The number of occurrences of each class in the training data (class 0: 322, class 1: 739).

Observations:

- The model achieves a reasonable accuracy on the training data, indicating its ability to correctly classify instances.
- Precision, recall, and F1-score are relatively balanced between the two classes, suggesting no significant bias towards any class.
- The model's performance on the training data appears consistent across different evaluation metrics, indicating its reliability. However, the lower recall for class 0 suggests that the model may struggle to correctly identify instances of this class.

```
Test =  0.8245614035087719
[[ 79  59]
 [ 21 297]]
              precision    recall  f1-score   support

           0       0.79      0.57      0.66       138
           1       0.83      0.93      0.88       318

    accuracy                           0.82       456
   macro avg       0.81      0.75      0.77       456
weighted avg       0.82      0.82      0.82       456
```

Accuracy on Test Data: The accuracy of the decision tree model on the test data is approximately 82.46%.

Confusion Matrix:

- True Positives (TP): 79
- True Negatives (TN): 297
- False Positives (FP): 59
- False Negatives (FN): 21
- The model seems to have relatively higher performance in predicting class 1 (positive class) compared to class 0 (negative class).

Classification Report:

- **Precision:** The precision for class 0 (negative class) is 0.79, and for class 1 (positive class) is 0.83.
- **Recall:** The recall for class 0 is 0.57, and for class 1 is 0.93.
- **F1-score:** The F1-score for class 0 is 0.66, and for class 1 is 0.88.
- **Support:** The number of occurrences of each class in the test data (class 0: 138, class 1: 318).

Observations:

- The model achieves a reasonable accuracy on the test data, indicating its ability to correctly classify instances.
- Precision, recall, and F1-score are relatively balanced between the two classes, suggesting no significant bias towards any class.
- The model's performance on the test data appears consistent across different evaluation metrics, indicating its reliability. However, similar to the training data, the lower recall for class 0 suggests that the model may struggle to correctly identify instances of this class.

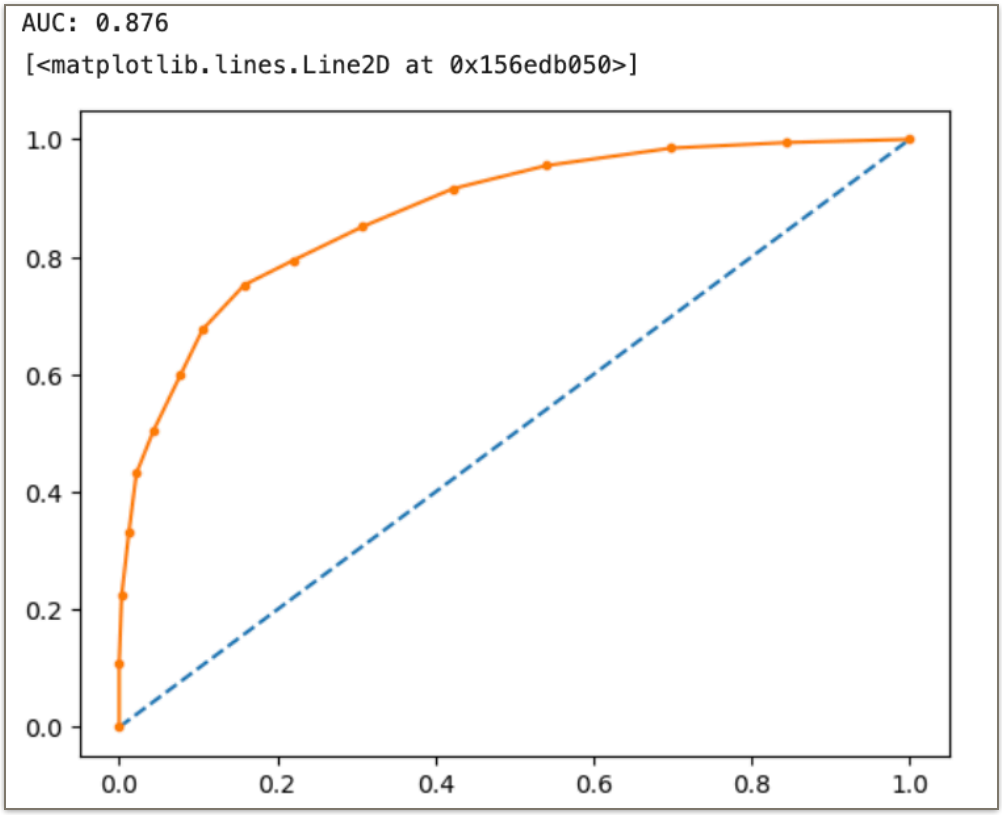


Fig: ROC Curve on Train Data

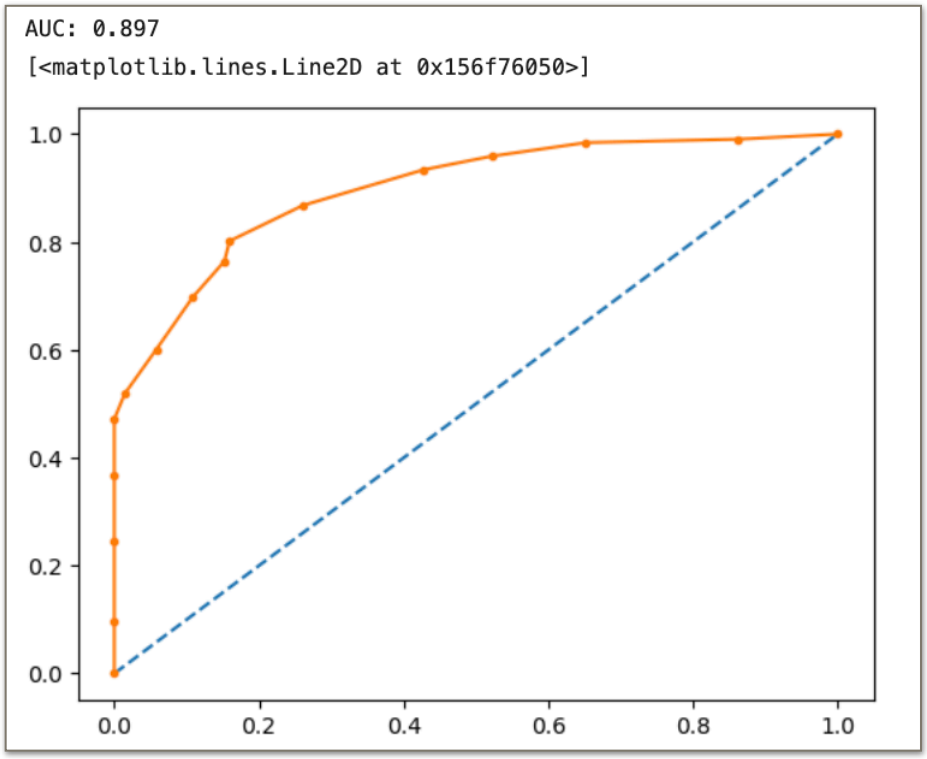


Fig: ROC Curve on Test Data

Gradient Boosting Classifier: The GradientBoostingClassifier from the sklearn.ensemble module is imported and initialized with a random state of 1.
Model Training: The GB classifier is trained on the training data (X_train and y_train) using the fit method.
Feature Importances: The feature importances are computed using the feature_importances_ attribute of the trained GB classifier. This attribute provides the relative importance of each feature in predicting the target variable.
Visualization: The computed feature importances are then visualized using a horizontal bar plot. Features are sorted based on their importances, and their relative importance values are displayed on the plot.

Observations:

- The plot helps in identifying the most important features for the classification task.
- Features with higher relative importance contribute more to the model's decision-making process.
- This visualization aids in feature selection, model interpretation, and understanding the underlying relationships between features and the target variable.

Analysis

Hague:

- Has the highest relative importance among all features.
- Indicates that the “Hague” feature significantly influences the outcome.

Blair:

- Has moderate importance.
- Contributes to the model’s predictions.

Europe:

- Also has moderate importance.
- Likely related to European context or location.

Political knowledge:

- Plays a role in predictions.
- Indicates that understanding of politics matters.

Age:

- Has some importance.
- Age-related factors impact the outcome.

Economic condition (national):

- Moderately important.
- National economic factors affect predictions.

Economic condition (household):

- Similar to national economic condition.
- Household-level economic factors matter.

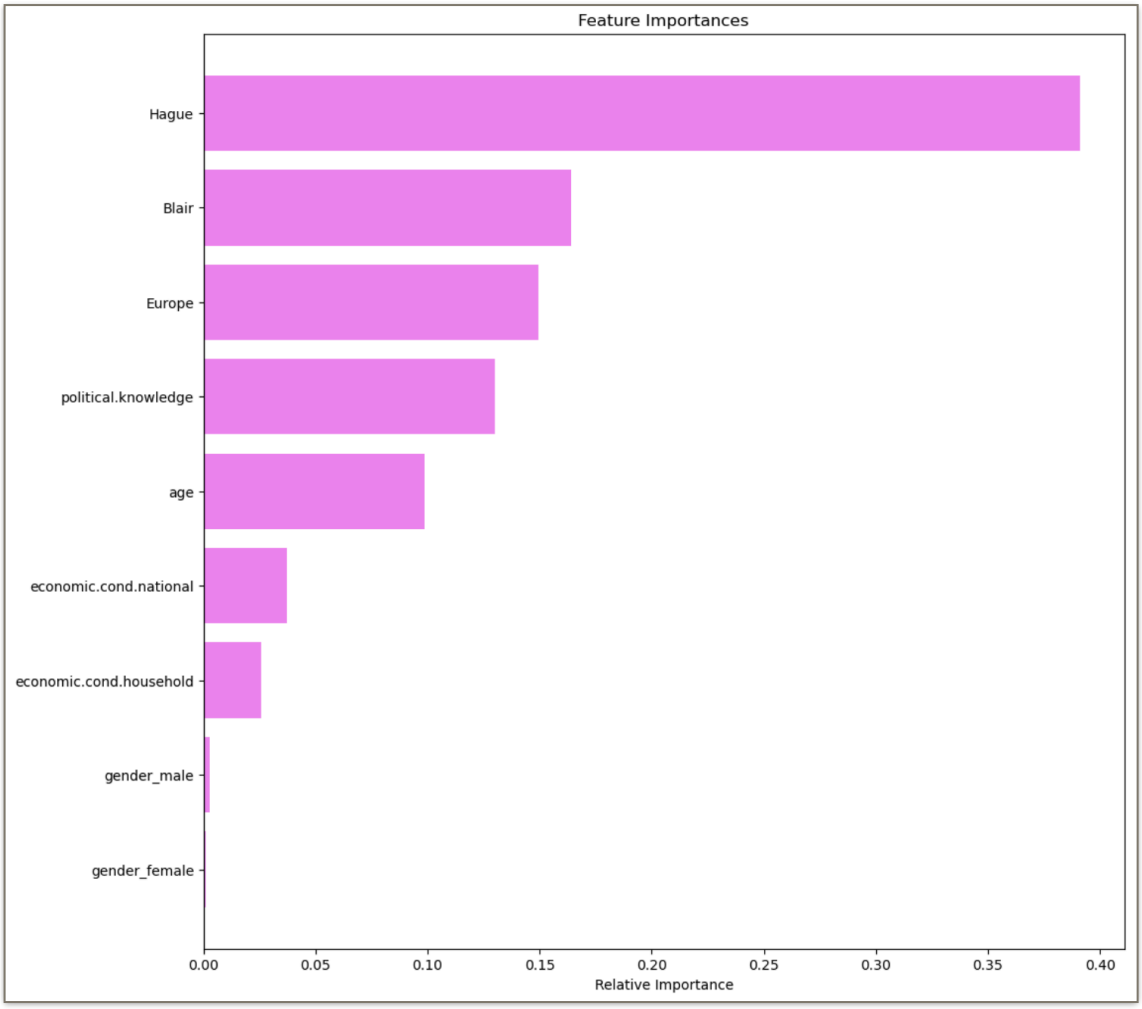


Fig: Feature Importance vs Relative Importance

Gender (male):

- Has lower importance.
- Gender may not strongly influence predictions.

Gender (female):

- Similar to male gender.
- Gender alone isn't a dominant predictor.

1.4 Model Performance evaluation

Comparing the AUC, ROC curve on the train data of all the tuned models:

In all the models, tuned ones are better than the regular models. So, we compare only the tuned models and describe which model is the best/optimised.

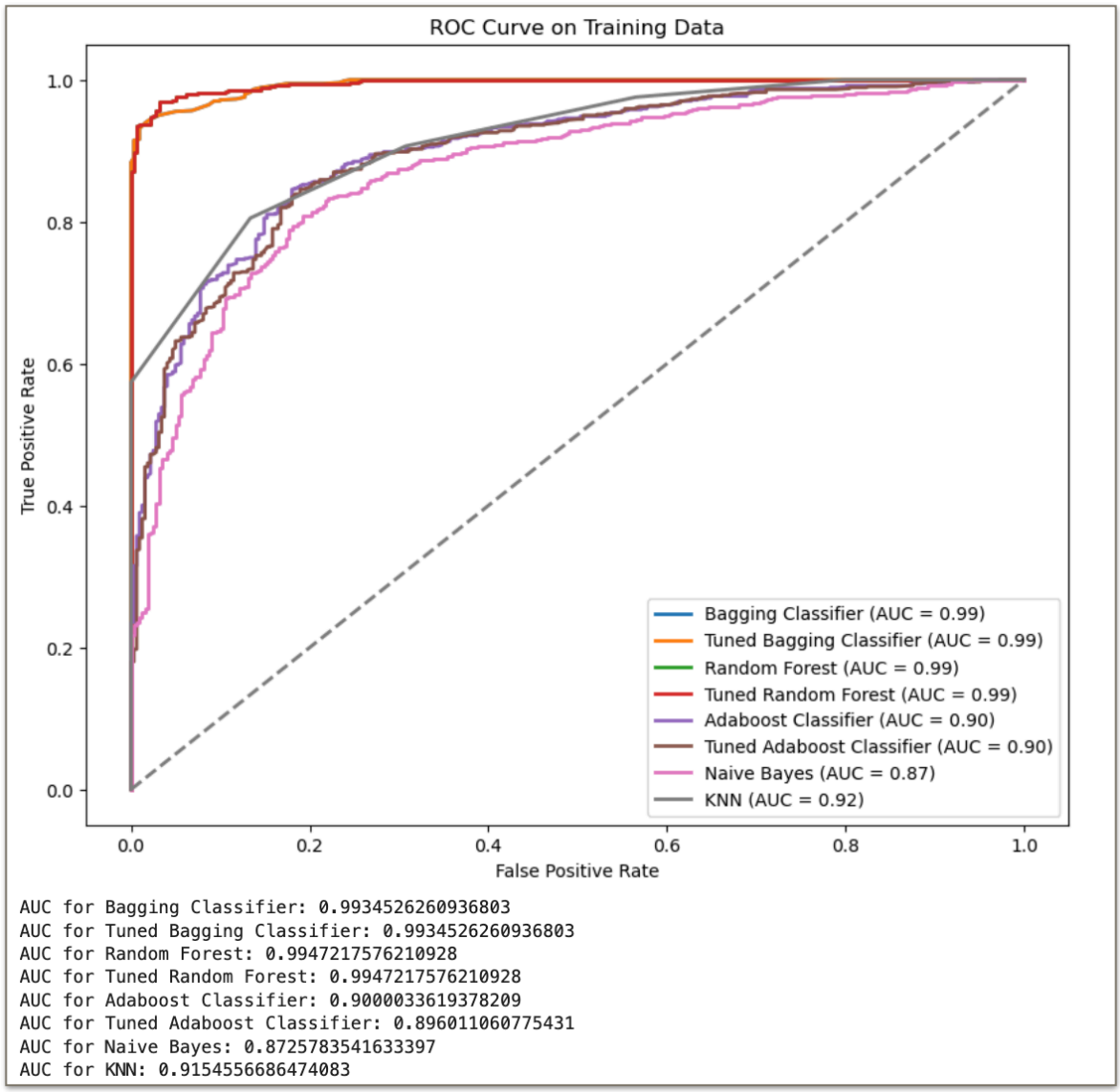


Fig: ROC Curve on Training Data

AUC for Bagging Classifier: 99.35%
AUC for Tuned Bagging Classifier: 99.35%
AUC for Random Forest: 99.47%
AUC for Tuned Random Forest: 99.47%
AUC for Ada-boost Classifier: 90.00%

AUC for Tuned Ada-boost Classifier: 89.60
AUC for Naive Bayes: 87.26%
AUC for KNN: 91.55%

Comparing the AUC, ROC curve on the test data of all the tuned models:

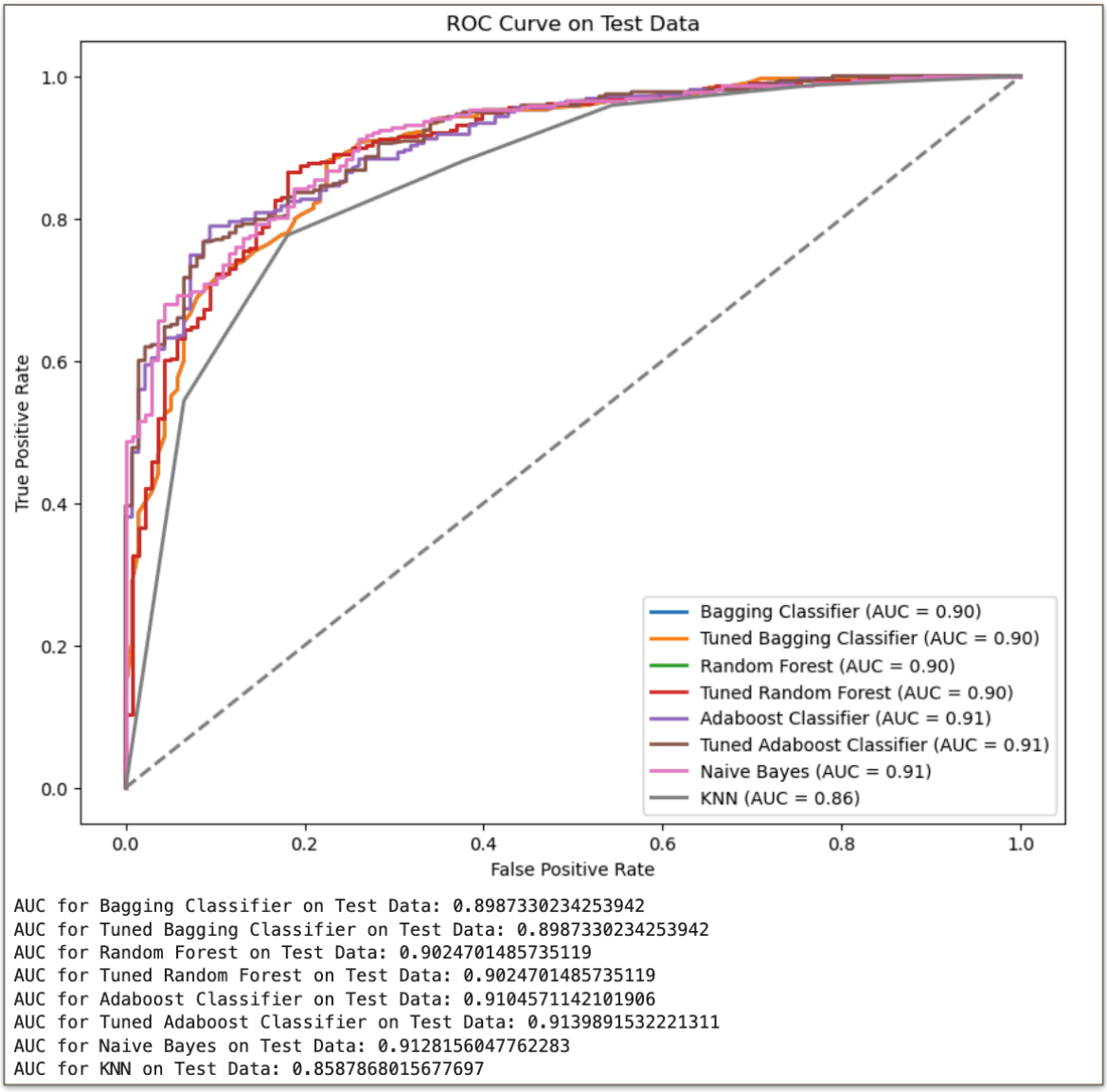


Fig: ROC Curve on Test Data

AUC for Bagging Classifier on Test Data: 89.87%
AUC for Tuned Bagging Classifier on Test Data: 89.87%
AUC for Random Forest on Test Data: 90.25%
AUC for Tuned Random Forest on Test Data: 90.25%
AUC for Ada-boost Classifier on Test Data: 91.05%
AUC for Tuned Ada-boost Classifier on Test Data: 91.39%
AUC for Naive Bayes on Test Data: 91.28%
AUC for KNN on Test Data: 85.88%

Observations:
The AUC scores provide insights into the discriminative power of each classifier.

On Train Data:

- Bagging Classifier and Tuned Bagging Classifier exhibit very high AUC scores, indicating strong performance in distinguishing between classes.
- Random Forest and Tuned Random Forest also demonstrate excellent performance, with slightly higher AUC scores compared to Bagging Classifier.
- Adaboost Classifier and Tuned Adaboost Classifier have relatively lower AUC scores compared to the ensemble methods (Bagging and Random Forest), but still perform reasonably well.
- Naive Bayes and KNN show moderate AUC scores, suggesting decent performance but with some room for improvement.

On Test Data:

- The AUC scores on the test data generally follow a similar trend to the train data, indicating consistent performance across train and test sets for most models.
- Adaboost Classifier and Tuned Adaboost Classifier exhibit the highest AUC scores on the test data, indicating robust generalization performance.
- Naive Bayes also performs well on the test data, showing competitive AUC scores compared to the ensemble methods.
- KNN shows slightly lower AUC scores on the test data compared to the train data, suggesting some overfitting or limitations in generalization capability.

Overall, the ensemble methods (Bagging, Random Forest, Adaboost) along with Naive Bayes demonstrate strong performance on both train and test data, while KNN shows comparatively lower performance, especially on the test data.

1.6 Final Model Selection

Comparison of Models:

- Bagging Classifier: AUC - 99.35% (Train), 89.87% (Test)
- Tuned Bagging Classifier: AUC - 99.35% (Train), 89.87% (Test)
- Random Forest: AUC - 99.47% (Train), 90.25% (Test)
- Tuned Random Forest: AUC - 99.47% (Train), 90.25% (Test)
- AdaBoost Classifier: AUC - 90.00% (Train), 91.05% (Test)
- Tuned AdaBoost Classifier: AUC - 89.60% (Train), 91.39% (Test)
- Naive Bayes: AUC - 87.26% (Train), 91.28% (Test)
- KNN: AUC - 91.55% (Train), 85.88% (Test)

Selection of Final Model:

- Based on the comparison, both Random Forest and Tuned Random Forest models exhibit the highest AUC scores on both train and test data, indicating superior performance and generalisation capability.
- Random Forest and Tuned Random Forest models consistently outperform other models on both train and test data, suggesting robustness and reliability.

Most Important Features in the Final Model:

- Random Forest and Tuned Random Forest models can provide insights into the importance of features used in classification.
- By examining feature importance scores provided by these models, we can identify the key factors contributing to the classification outcome.
- Features with higher importance scores are indicative of their significant influence on the classification decision.
- Drawing inferences from these important features can offer valuable insights into the underlying patterns and characteristics of the dataset.

1.7 Actionable Insights & Recommendations

- The Labour Party garnered over twice the number of votes compared to the Conservative Party.
- The majority of respondents rated the national economic condition with scores of 3 and 4, yielding an average score of 3.245221. Similarly, for household economic condition, most people gave scores of 3 and 4, resulting in an average score of 3.137772.
- Blair received more votes than Hague, and Blair's scores were notably higher than Hague's.
- Blair's average score stands at 3.335531, whereas Hague's is 2.749506, indicating Blair's stronger performance.
- Approximately 30% of the population exhibited no political knowledge, rating 0 on a scale of 0 to 3.
- Despite giving a low score of 1 to a particular party, individuals still opted to vote for the same party, possibly due to a lack of political awareness.
- Eurosceptic individuals tended to vote for the Conservative Party, while those with lower Eurosceptic sentiment favoured the Labour Party.
- Out of 454 individuals who scored 0 for political knowledge, 360 voted for the Labour Party, while 94 voted for the Conservative Party.
- All models demonstrated satisfactory performance on both training and test datasets, with tuned models outperforming regular models.
- Overfitting was not observed in any model, except for the Random Forest and Bagging regular models.

Business recommendations

- Optimal model construction heavily relies on tuning hyper-parameters, but this process demands substantial computational resources due to the vast combinations involved. However, exploring various parameter sets could yield superior results.
- Expanding the dataset can bolster model training, thereby enhancing predictive accuracy.
- Implementing a sequential prediction function across all models can offer deeper insights into outcomes and their probabilities, fostering better comprehension.

Problem 2

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

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

Code Snippet to extract the three speeches:

"

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

"

2.1 Problem 2 - Define the problem and Perform Exploratory Data Analysis

```
Speech of President Roosevelt:
Number of Characters: 7571
Number of Words: 1526
Number of Sentences: 68

Speech of President Kennedy:
Number of Characters: 7618
Number of Words: 1543
Number of Sentences: 52

Speech of President Nixon:
Number of Characters: 9991
Number of Words: 2006
Number of Sentences: 68
```

Number of Characters

```
The No. of characters in President Franklin D. Roosevelt's speech in 1941 is 7571
The No. of characters in President John F. Kennedy's speech in 1961 is 7618
The No. of characters in President Richard Nixon's speech in 1973 is 9991
```

- The speech delivered by President Franklin D. Roosevelt contains **7571** characters, including spaces.
- President John F. Kennedy's speech comprises **7618** characters, including spaces.
- President Richard Nixon's speech totals **9991** characters, including spaces.

Number of Words

- President Franklin D. Roosevelt's speech contains 1526 words.
- President John F. Kennedy's speech includes 1543 words.
- President Richard Nixon's speech consists of 2006 words.

Number of Sentence

- President Franklin D. Roosevelt's speech contains 68 sentences.
- President John F. Kennedy's speech consists of 52 sentences.
- President Richard Nixon's speech comprises 68 sentences.

Average word count

	Speech	avg_word
0	On each national day of inauguration since 178...	4.783825
1	Vice President Johnson, Mr. Speaker, Mr. Chief...	4.626100
2	Mr. Vice President, Mr. Speaker, Mr. Chief Jus...	4.713397

Stop words count

	Speech	stopwords
0	On each national day of inauguration since 178...	632
1	Vice President Johnson, Mr. Speaker, Mr. Chief...	618
2	Mr. Vice President, Mr. Speaker, Mr. Chief Jus...	899

- President Franklin D. Roosevelt's speech contains 632 stopwords.
- President John F. Kennedy's speech consists of 618 stopwords.
- President Richard Nixon's speech comprises 899 stopwords.

Hashtag count

	Speech	hastags
0	On each national day of inauguration since 178...	0
1	Vice President Johnson, Mr. Speaker, Mr. Chief...	0
2	Mr. Vice President, Mr. Speaker, Mr. Chief Jus...	0

- There is no hastag in all the three speeches.

Count of Numbers

	Speech	numerics
0	On each national day of inauguration since 178...	14
1	Vice President Johnson, Mr. Speaker, Mr. Chief...	7
2	Mr. Vice President, Mr. Speaker, Mr. Chief Jus...	10

- President Franklin D. Roosevelt's speech contains 14 numeric values.
- President John F. Kennedy's speech consists of 7 numeric values.
- President Richard Nixon's speech comprises 10 numeric values.

2.2 Text cleaning

Remove all the stop-words from all three speeches.

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

1. **Stopwords Removal:**
- The nltk.corpus library provides a package called “stopwords” for removing common stopwords.
- Stopwords include words like “and,” “a,” “is,” “to,” etc., which typically don’t carry significant meaning in sentiment analysis or machine learning algorithms.

- These universally accepted stopwords can be added or removed based on specific requirements.
- Specify the language (e.g., English) before defining functions, as there are language-specific packages available.

2. **Stemming:**
- Stemming is a process that reduces words to their base or root form.
- It aids in understanding words with similar meanings.
- For example, “eating,” “eats,” and “eaten” are all reduced to “eat” during stemming.

	Speech	char_count	word_count
Franklin D. Roosevelt	national day inauguration since 1789 people re...	4562	618
John F. Kennedy	vice president johnson speaker chief justice p...	4665	663
Richard Nixon	vice president speaker chief justice senator c...	5771	781

After the removal of stop-words,
- President Franklin D. Roosevelt's speech have 618 words.
- President John F. Kennedy's speech have 663 words.
- President Richard Nixon's speech have 781 words.

Most common words in all three speeches

```
Top 3 most common words used in all three speeches:
us: 46
nation: 40
let: 39
```

The top 3 word
all three speeches

us - 46
nation - 40
let - 39

The top 10 most common words used in all three speeches

```
Speech of President Roosevelt:
Top 10 most common words:
nation: 17
know: 10
peopl: 9
spirit: 9
life: 9
democraci: 9
us: 8
america: 8
live: 7
year: 7

Speech of President Kennedy:
Top 10 most common words:
let: 16
us: 12
power: 9
world: 8
nation: 8
side: 8
new: 7
pledg: 7
ask: 6
citizen: 5

Speech of President Nixon:
Top 10 most common words:
us: 26
let: 22
america: 21
peac: 19
world: 18
respons: 17
new: 15
nation: 15
govern: 10
great: 9
```

Top 3 most common word in President Franklin D. Roosevelt's speech are:

- nation - 17
- know - 10
- peopl - 9

Top 3 most common word in President John F. Kennedy's speech are:

- let - 16
- us - 12
- power - 9

Top 3 most common word in President Richard Nixon's speech are:

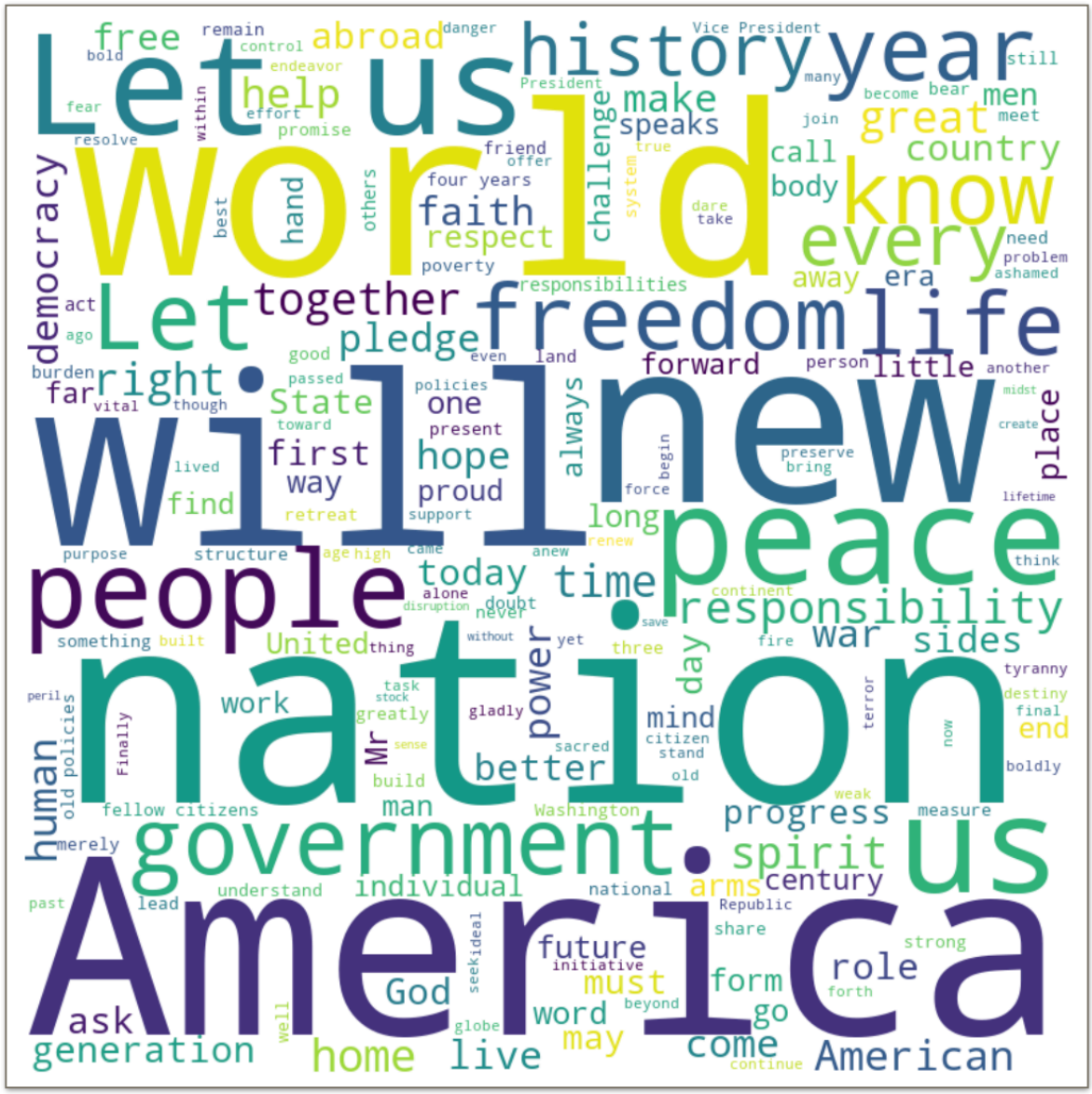
- us - 26
- let - 22
- america - 21

2.3 Plot Word cloud of all three speeches

A word cloud is a visual representation of text data, where the size of each word indicates its frequency or importance within the text. In a word cloud, words are typically arranged randomly or in a way that emphasises the most prominent words by making them larger and more centrally located.

Word clouds are commonly used to quickly visualise the most important words or themes in a body of text, making it easier to identify patterns, trends, and key concepts. They are often employed in text analysis, sentiment analysis, content summarisation, and topic modelling tasks.

WordCloud for all three speeches



The most highlighted words are “nation”, “America”, “world”, “will”, “let” and “us”.

WordCloud of President Franklin D. Roosevelt's speech

The most highlighted words are “nation”, “know”, “peopl”, “spirit”, ”life” and “democraci”.

WordCloud of President John F. Kennedy's speech

The most highlighted words are “world”, “power”, “let”, “nation”, ”free” and “side”.

WordCloud of President Richard Nixon's speech

The most highlighted words are “world”, “america”, “let”, “us”, ”nation” and “peac”.

Insights

- The primary objective was to analyze all three speeches comprehensively, focusing on their strength and sentiment.
- Through our examination, we identified common words that were prevalent across all three speeches.
- These shared words likely encapsulate themes that resonated with many individuals and played a role in securing the presidency.
- Notably, words such as "us", "nation", and "let" emerged prominently in all three speeches.
- The prevalence of these words suggests their significance and potential impact on the audience.

———END———