MACHINE LEARNING 2 - CODED PROJECT

DSBA

Krishnabhamini Sinha

Contents

Data Dictionary for Problem 1: (Election_data:)

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

Problem 1.1 - Define the problem and perform Exploratory Data Analysis

- Problem definition - Check shape, Data types, and statistical summary - Univariate analysis - Multivariate analysis - Use appropriate visualizations to identify the patterns and insights - Key meaningful observations on individual variables and the relationship between variables

Problem Definition:

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.

Objective
The primary objective is to leverage machine learning to build a predictive model capable of forecasting which political party a voter is likely to support. This predictive model, developed based on the provided information, will serve as the foundation for creating an exit poll. The exit poll aims to contribute to the accurate prediction of the overall election outcomes, including determining which party is likely to secure the majority of seats.

Solution:

Shape:

The given dataset has 759 rows and 10 columns.

```
data = pd.read_excel('Election_Data.xlsx')
```

```
data.shape
```

```
(1525, 10)
```

Data Types:

The data type for each column is enlisted as below in the figure given below.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1525 entries, 0 to 1524
Data columns (total 10 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Unnamed: 0              1525 non-null   int64
 1   vote                    1525 non-null   object
 2   age                     1525 non-null   int64
 3   economic.cond.national  1525 non-null   int64
 4   economic.cond.household 1525 non-null   int64
 5   Blair                   1525 non-null   int64
 6   Hague                   1525 non-null   int64
 7   Europe                  1525 non-null   int64
 8   political.knowledge     1525 non-null   int64
 9   gender                  1525 non-null   object
dtypes: int64(8), object(2)
memory usage: 119.3+ KB
```

First five rows:

	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

Last five rows:

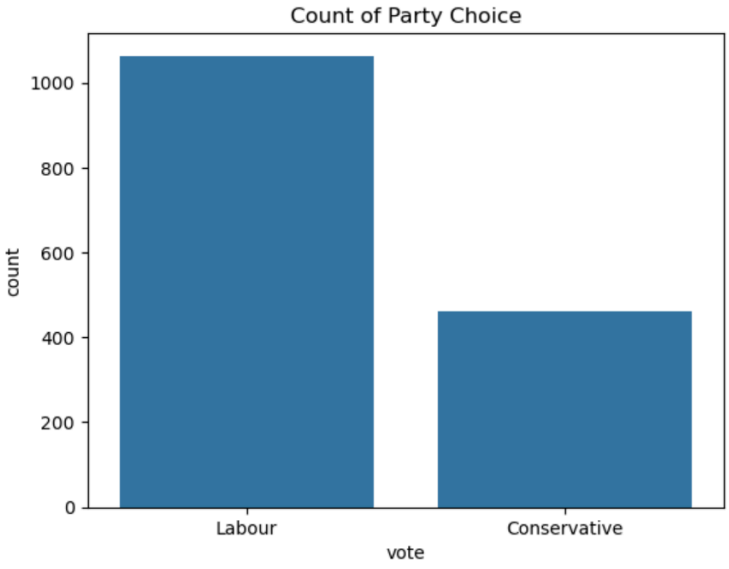
	Unnamed: 0	vote	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	gender
1520	1521	Conservative	67	5	3	2	4	11	3	male
1521	1522	Conservative	73	2	2	4	4	8	2	male
1522	1523	Labour	37	3	3	5	4	2	2	male
1523	1524	Conservative	61	3	3	1	4	11	2	male
1524	1525	Conservative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

Statistical Summary:

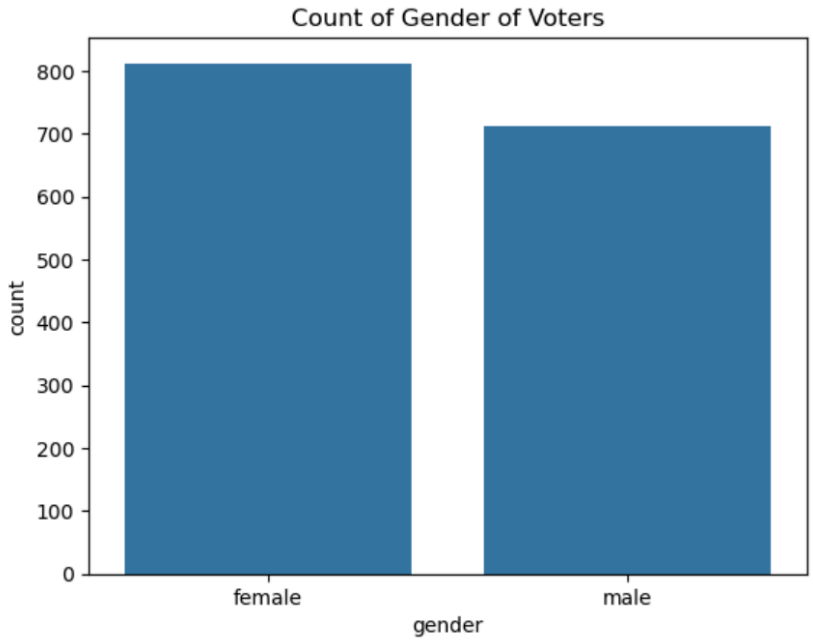
	count	unique	top	freq	mean	std	min	25%	50%	75%	max
Unnamed: 0	1525.0	NaN	NaN	NaN	763.0	440.373894	1.0	382.0	763.0	1144.0	1525.0
vote	1525	2	Labour	1063	NaN	NaN	NaN	NaN	NaN	NaN	NaN
age	1525.0	NaN	NaN	NaN	54.182295	15.711209	24.0	41.0	53.0	67.0	93.0
economic.cond.national	1525.0	NaN	NaN	NaN	3.245902	0.880969	1.0	3.0	3.0	4.0	5.0
economic.cond.household	1525.0	NaN	NaN	NaN	3.140328	0.929951	1.0	3.0	3.0	4.0	5.0
Blair	1525.0	NaN	NaN	NaN	3.334426	1.174824	1.0	2.0	4.0	4.0	5.0
Hague	1525.0	NaN	NaN	NaN	2.746885	1.230703	1.0	2.0	2.0	4.0	5.0
Europe	1525.0	NaN	NaN	NaN	6.728525	3.297538	1.0	4.0	6.0	10.0	11.0
political.knowledge	1525.0	NaN	NaN	NaN	1.542295	1.083315	0.0	0.0	2.0	2.0	3.0
gender	1525	2	female	812	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Univariate Analysis:

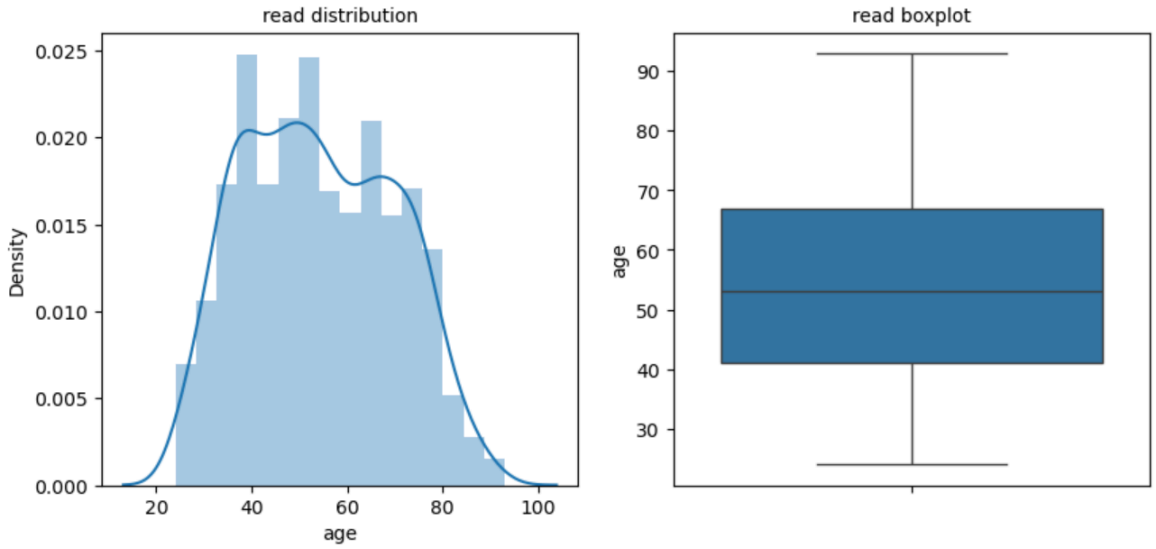
Observation on Party Choice:



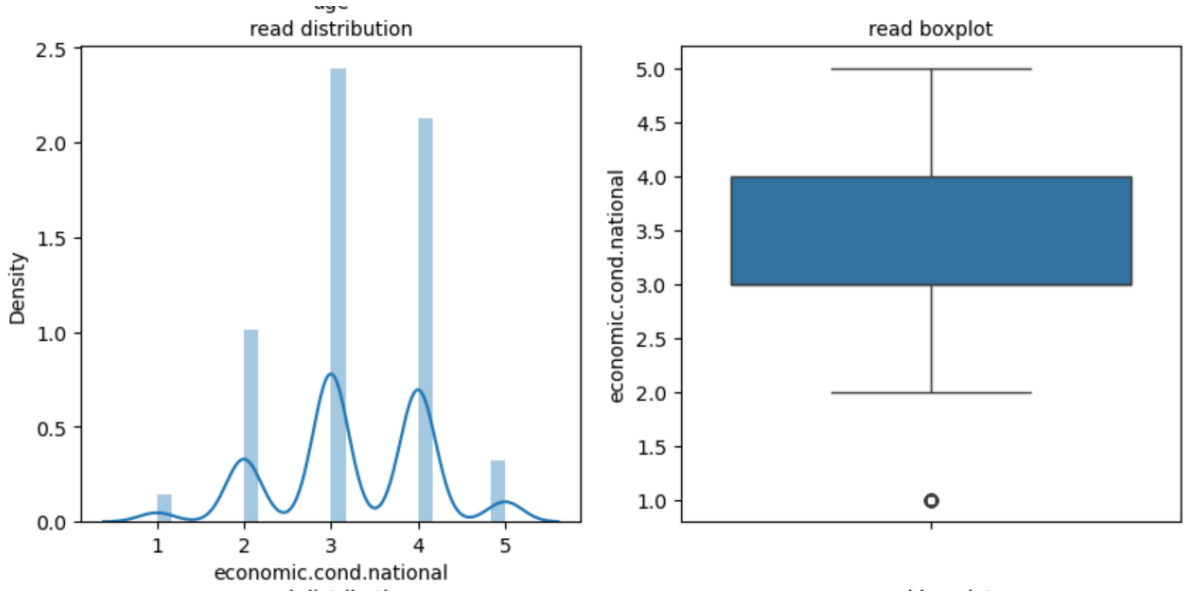
Observation on gender of voters:



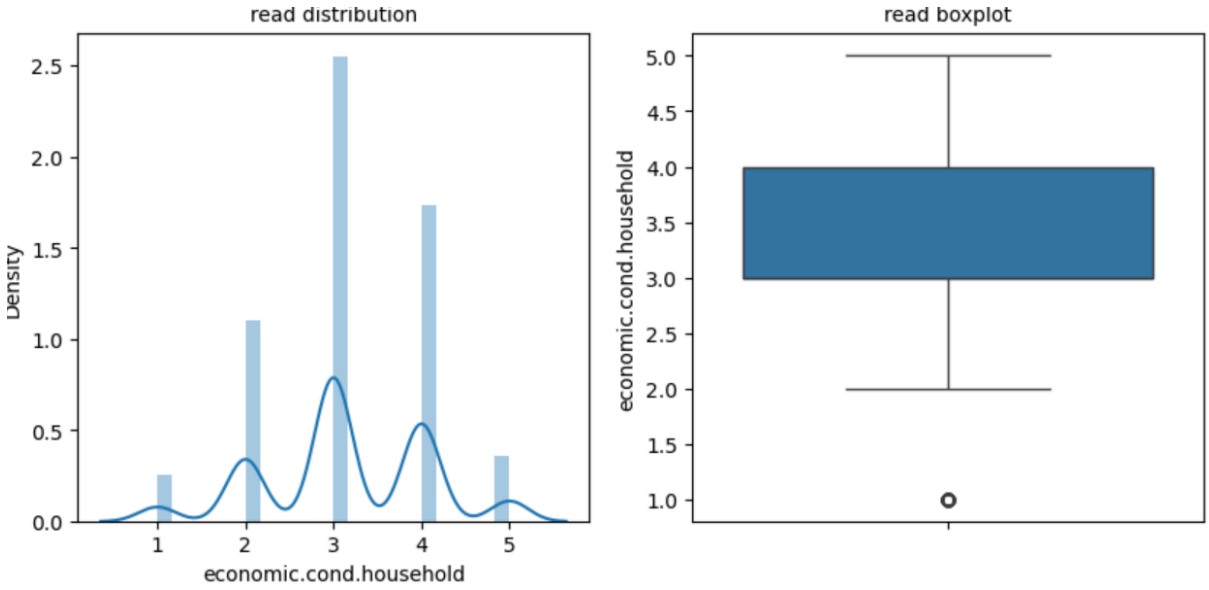
Observation on age:



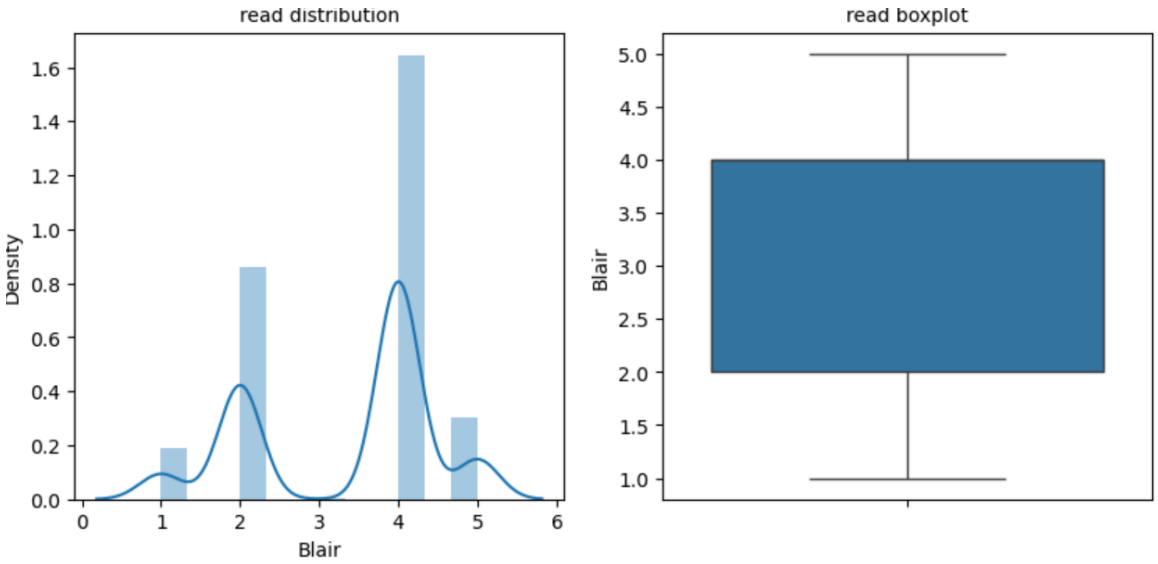
Observations on number of current national economic condition:



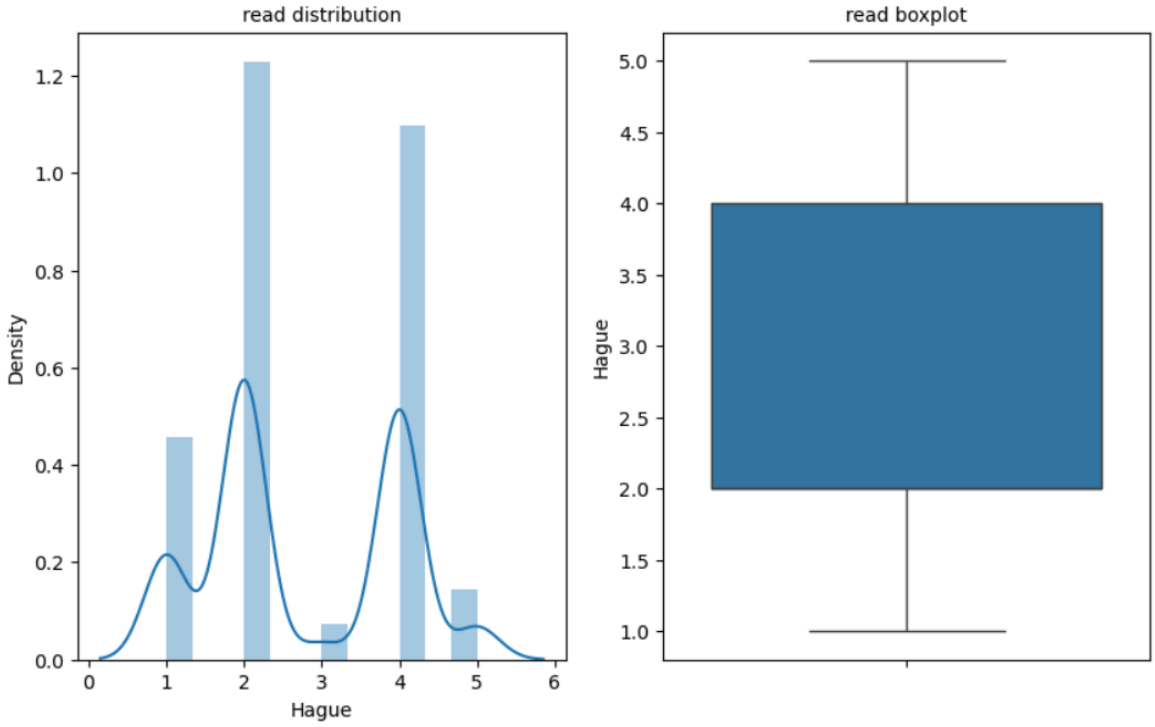
Observations on number of current household economic condition:



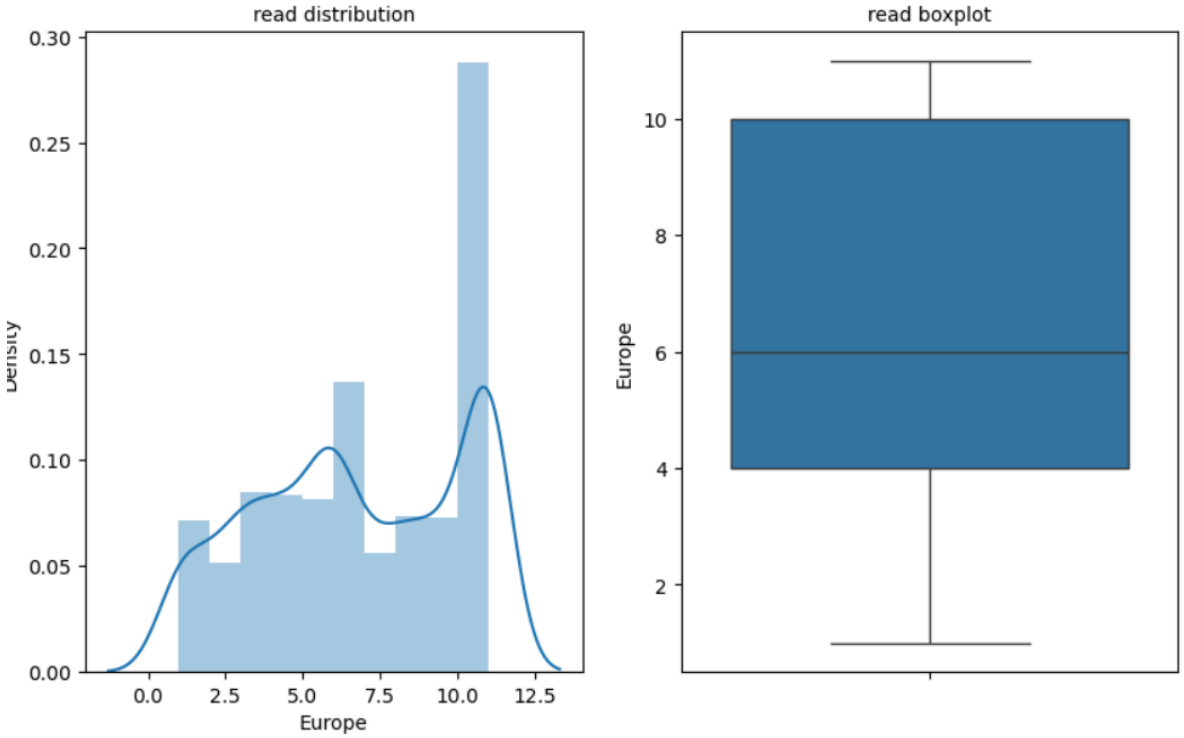
Observations on Blair:



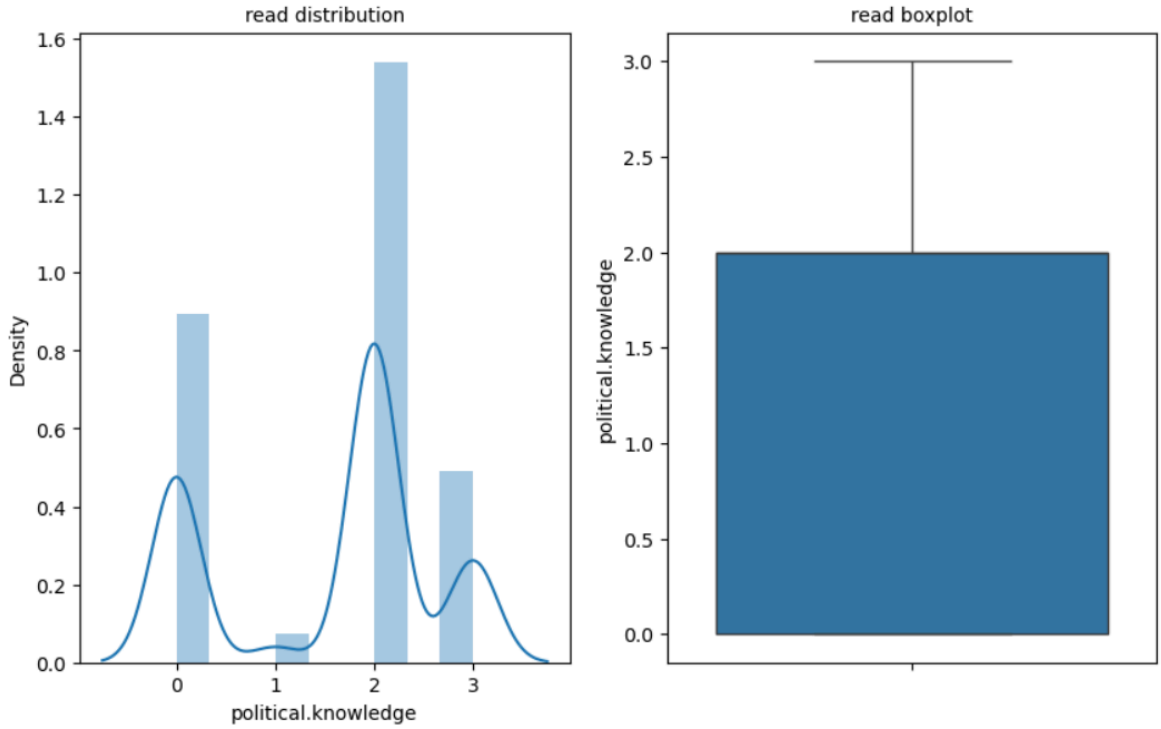
Observations on Hague:



Observations on Europe:

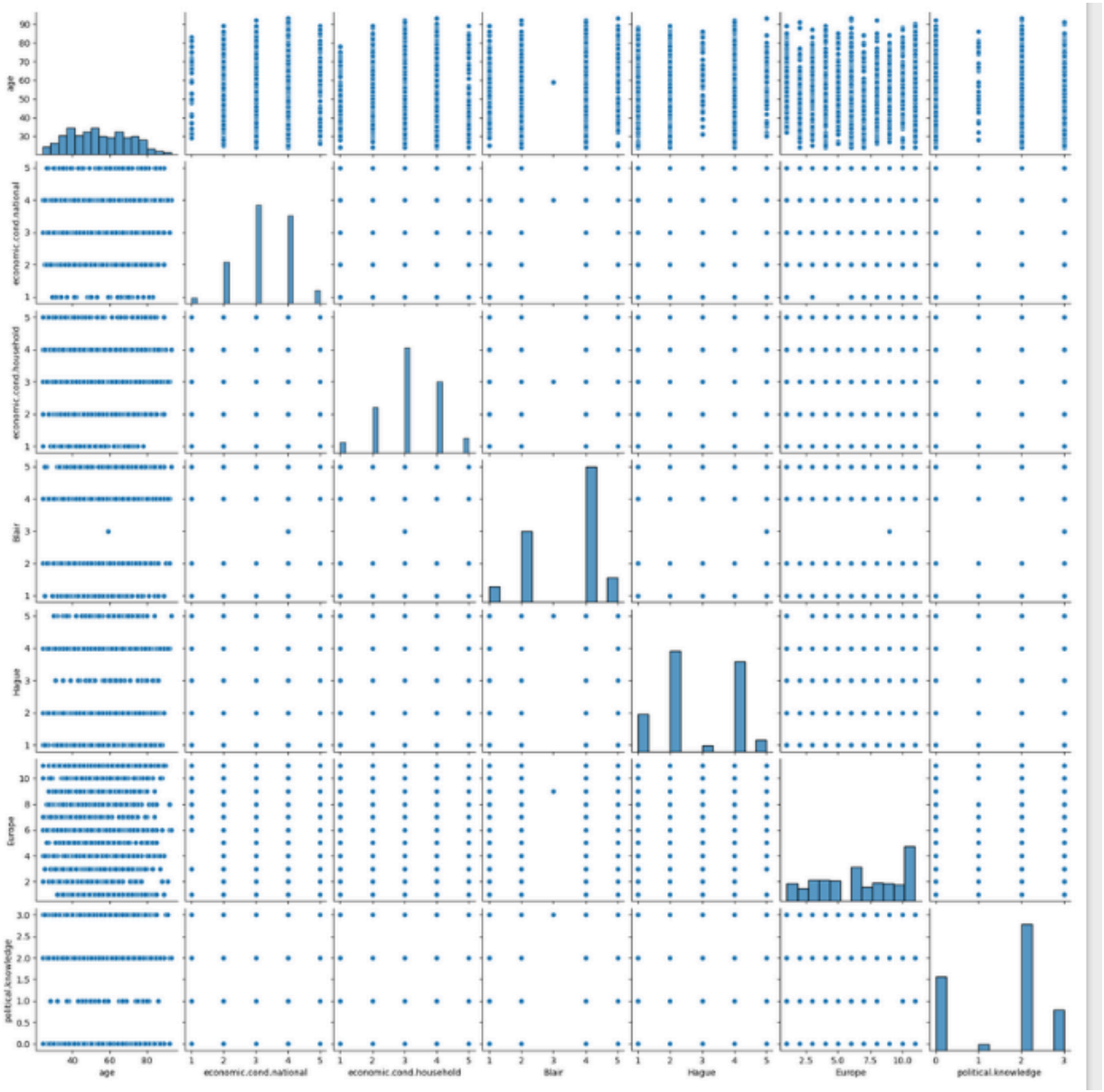


Observations on political knowledge:

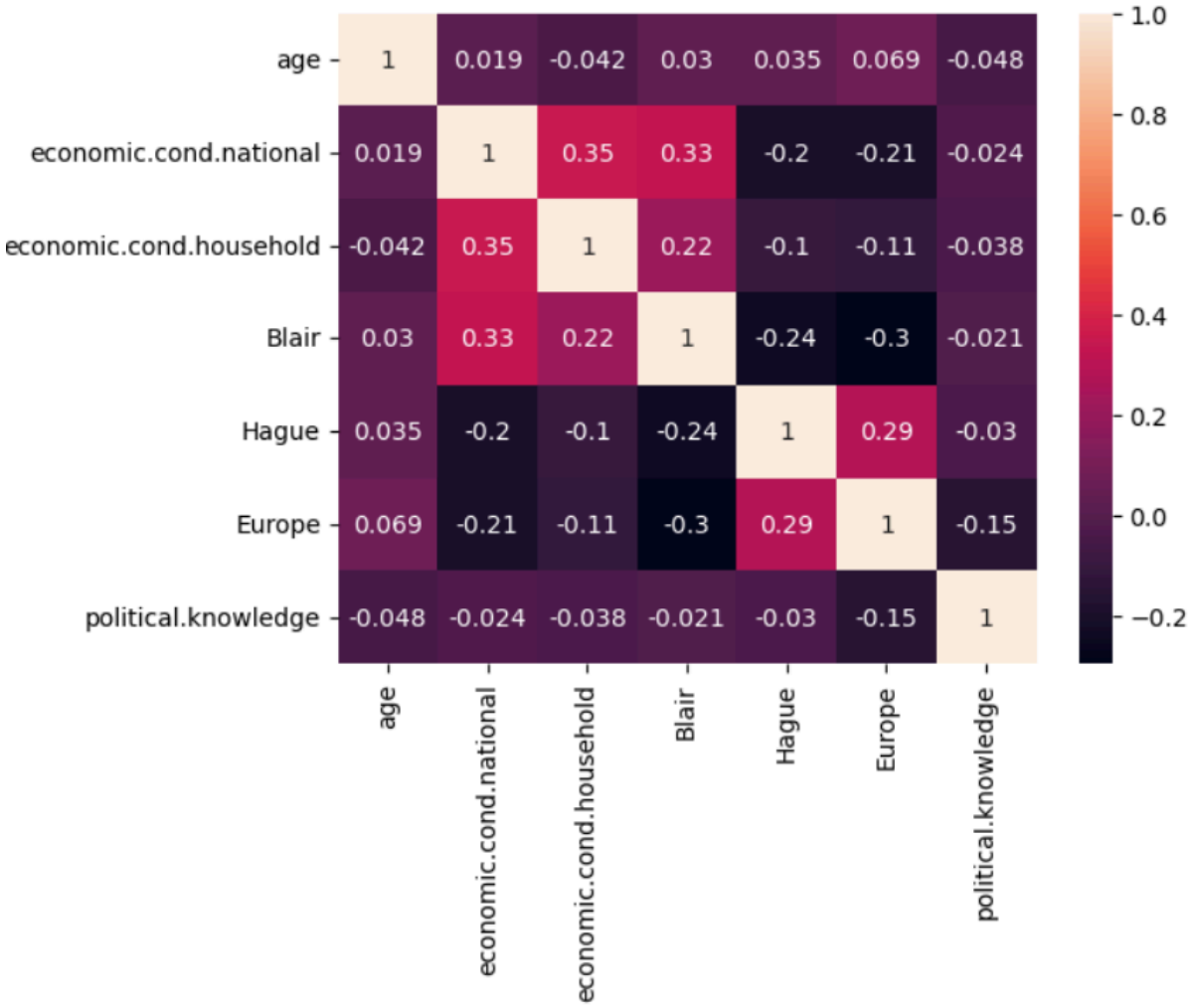


Multivariate analysis

Pairplot of numerical variables:



Heatmap of numerical variables:



Observations:

- Voters from the Labour party are more.
- Number of female voters are higher in number.
- Most of the voters are between the ages of 40 to 70.
- People under current national economic condition And current household economic condition number 3 at the highest.
- Knowledge of parties Position on European Integration is highest for 2.

Insights:

- We can expect votes mostly from the labour party and people who are under National economic condition and current household economic condition number 3.
- There is high scope for hi Eurosceptic sentiment.

Problem 1.2 - Data Preprocessing

Prepare the data for modeling: - Outlier Detection(treat, if needed)) - Encode the data - Data split - Scale the data (and state your reasons for scaling the features)

Solution:

Outlier treatment: There is no need for outlier treatment because the variables - economic.cond.national and economic.cond.household have max values as 5 and these are just labels.

Encode the data

```
from sklearn.preprocessing import MinMaxScaler
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
data['vote']= label_encoder.fit_transform(data['vote'])
data['gender'] = label_encoder.fit_transform(data['gender'])
```

```
data.head(1)
```

	vote	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	gender
0	1	43	3	3	4	1	2	2	0

Column each and gender have been level label encoded.

Data split

Data has been split into training and test sets in a 70:30 ratio.

```
from sklearn.model_selection import train_test_split
```

```
X = data.drop('vote', axis=1)
y = data['vote']
```

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=1)
```

Scaling of data:

```
scaler = MinMaxScaler()
scaled = scaler.fit_transform(data)
```

Scaling the data is important for maintaining uniformity in the data machine learning process. it ensures that the data follows a normal distribution which makes creating models like KNN easier.

Problem 1.3 - Model Building

- Metrics of Choice (Justify the evaluation metrics) - Model Building (KNN, Naive bayes, Bagging, Boosting) - Metrics of Choice (Justify the evaluation metrics) - Model Building (KNN, Naive bayes, Bagging, Boosting)

Problem 1.4 - Model Performance evaluation

- Metrics of Choice (Justify the evaluation metrics) - Model Building (KNN, Naive bayes, Bagging, Boosting) - Metrics of Choice (Justify the evaluation metrics) - Model Building (KNN, Naive bayes, Bagging, Boosting)

Solution:

Comparison of performance by different models

Naive Bayes Model

Accuracy - 0.8253275109170306

	precision	recall	f1-score	support for test set.
0	0.68	0.72	0.70	130
1	0.89	0.87	0.88	328
accuracy			0.83	458
macro avg	0.78	0.79	0.79	458

weighted avg	0.83	0.83	0.83	458

precision recall f1-score support for training set.

0	0.74	0.72	0.73	332
1	0.88	0.88	0.88	735
accuracy			0.83	1067
macro avg	0.81	0.80	0.80	1067
weighted avg	0.83	0.83	0.83	1067

Accuracy - 0.8331771321462043

KNN model

precision recall f1-score support for training set

0	0.80	0.74	0.77	332
1	0.89	0.92	0.90	735
accuracy			0.86	1067
macro avg	0.84	0.83	0.83	1067
weighted avg	0.86	0.86	0.86	1067

Accuracy - 0.8612933458294283

precision recall f1-score support for test set

0	0.68	0.72	0.70	130
1	0.89	0.87	0.88	328
accuracy			0.83	458
macro avg	0.78	0.79	0.79	458
weighted avg	0.83	0.83	0.83	458

Accuracy - 0.7860262008733624

Ada Boost

precision recall f1-score support for training set

0	0.78	0.72	0.74	332
1	0.88	0.91	0.89	735
accuracy			0.85	1067
macro avg	0.83	0.81	0.82	1067
weighted avg	0.84	0.85	0.85	1067

Accuracy - 0.8472352389878163

precision	recall	f1-score	support for test set
0	0.68	0.69	0.68	130
1	0.88	0.87	0.87	328
accuracy			0.82	458
macro avg	0.78	0.78	0.78	458
weighted avg	0.82	0.82	0.82	458

accuracy - 0.8187772925764192

Gradient boosting

precision	recall	f1-score	support for training set
0	0.84	0.79	0.81	332
1	0.91	0.93	0.92	735
accuracy			0.89	1067
macro avg	0.87	0.86	0.87	1067
weighted avg	0.89	0.89	0.89	1067

Accuracy - 0.8865979381443299

precision	recall	f1-score	support
0	0.69	0.74	0.71	130
1	0.89	0.87	0.88	328
accuracy			0.83	458

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

Accuracy - 0.8318777292576419

Bagging

precision	recall	f1-score	support
0	1.00	1.00	1.00	332
1	1.00	1.00	1.00	735
accuracy			1.00	1067
macro avg	1.00	1.00	1.00	1067
weighted avg	1.00	1.00	1.00	1067

Accuracy - 0.9990627928772259

precision	recall	f1-score	support
0	0.64	0.64	0.64	130
1	0.86	0.86	0.86	328
accuracy			0.80	458
macro avg	0.75	0.75	0.75	458
weighted avg	0.80	0.80	0.80	458

Accuracy - 0.7969432314410481

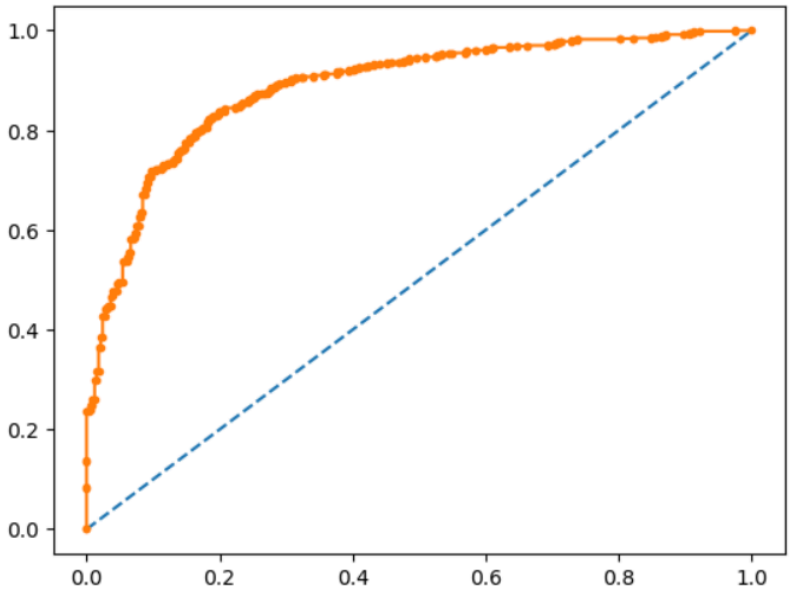
- After looking at the scores and the overfit and underfit percentages, we finalize Naive Bayes as the best model as it has the least difference between f1 scores of both classes (0 and 1) for training and test sets.
- Also, since the dataset is balanced, we can determine the appropriate model from the accuracy scores. Naive Bayes model has the least difference along with good fitting scores between both classes in training and test sets respectively.

Given below are the ROC and AUC scores and plots:

-Naive Bayes Model:

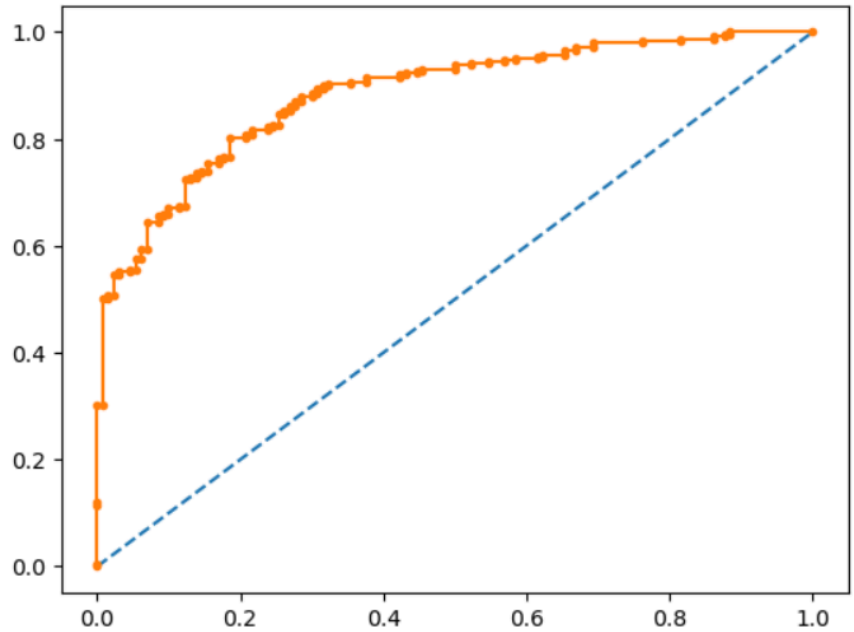
For training set-

```
AUC: 0.886
[<matplotlib.lines.Line2D at 0x1bd11a17860>]
```



for test set-

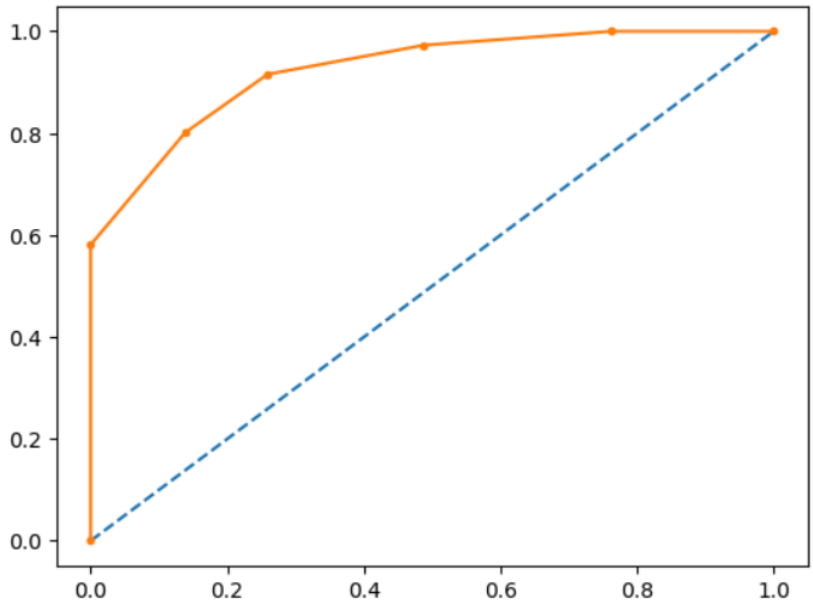
```
AUC: 0.885
```



KNN Model:

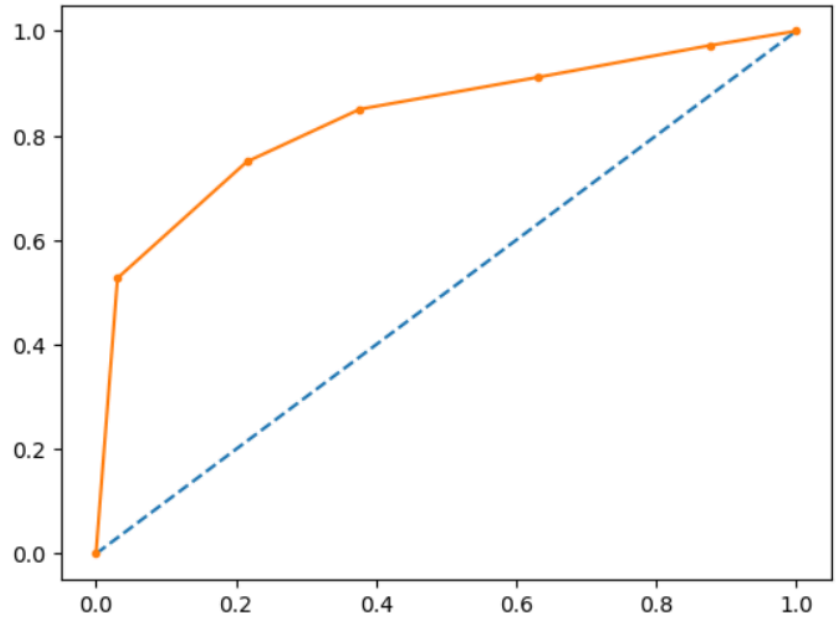
For draining set -

```
AUC: 0.924
[<matplotlib.lines.Line2D at 0x1bd14f663c0>]
```



for test set-

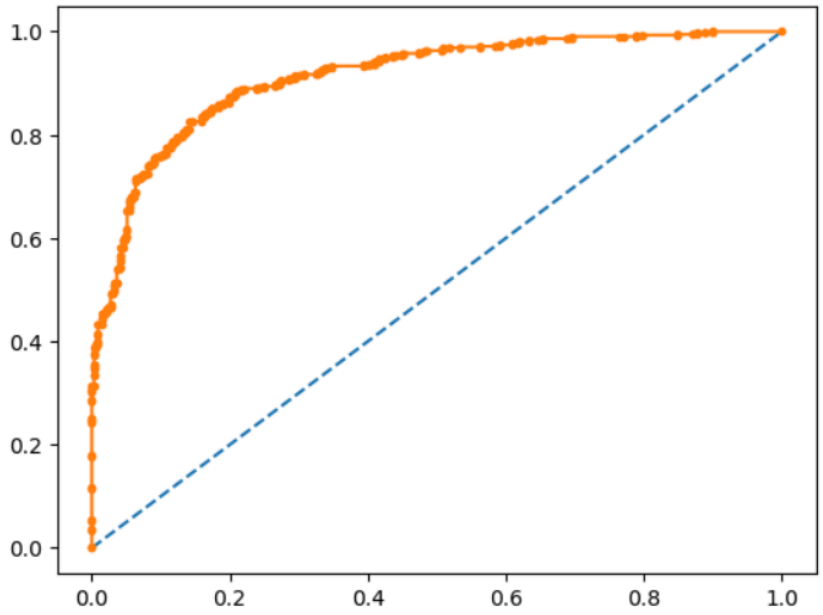
```
AUC: 0.832
```



AdaBoost Model:

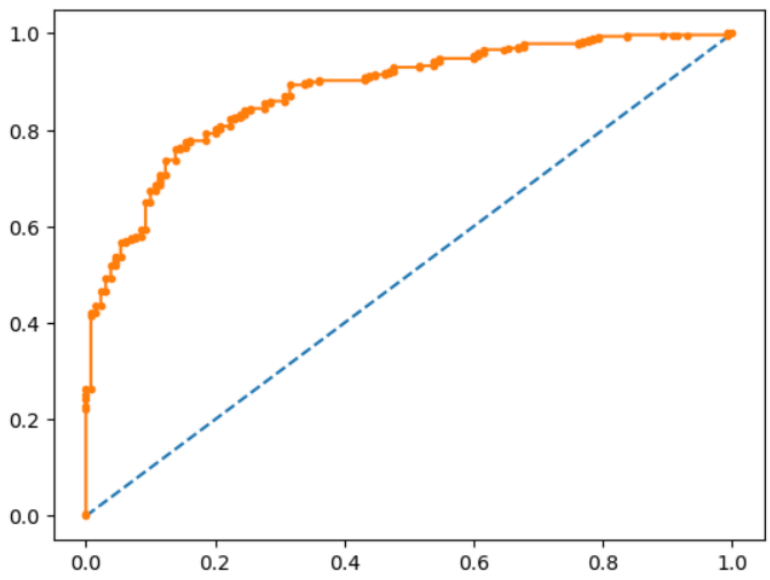
For training set -

```
AUC: 0.913
[<matplotlib.lines.Line2D at 0x1bd15077e30>]
```



For test set-

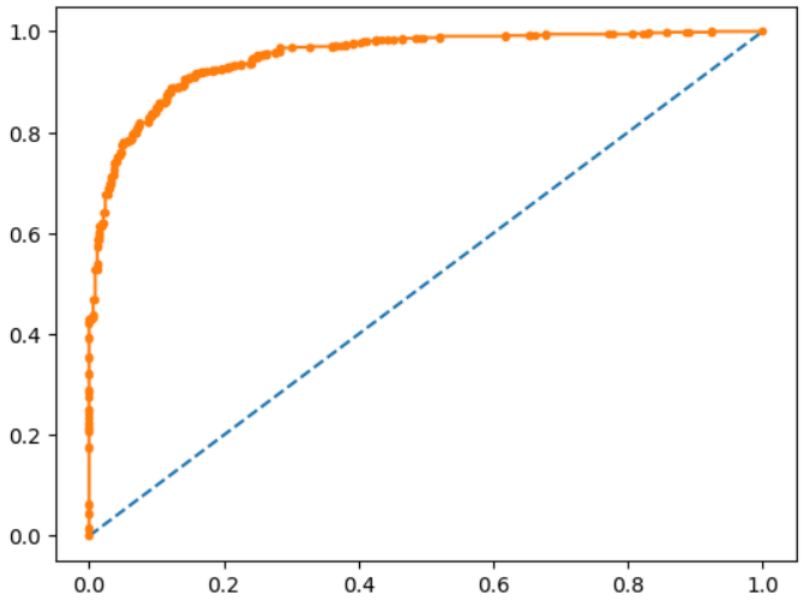
```
AUC: 0.879
```



Gradient Boosting-

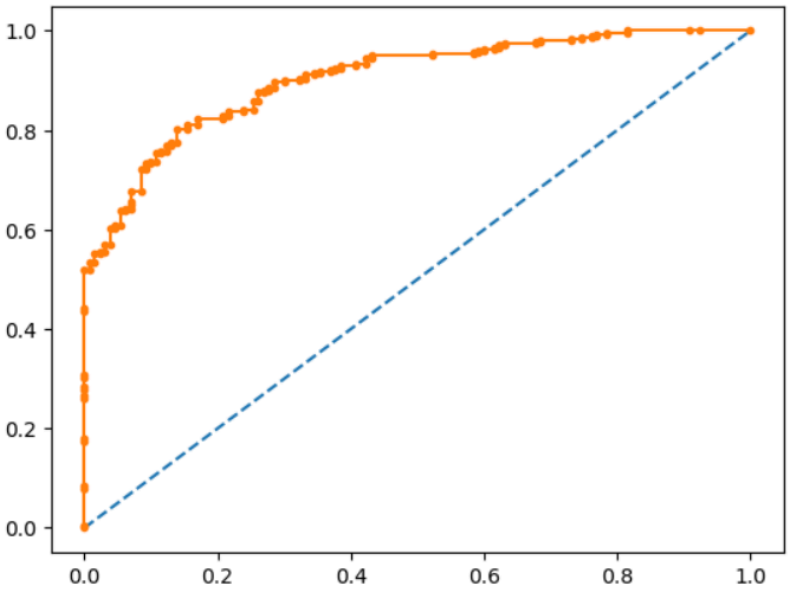
For training set-

```
AUC: 0.950
[<matplotlib.lines.Line2D at 0x1bd150a5f40>]
```



For test set-

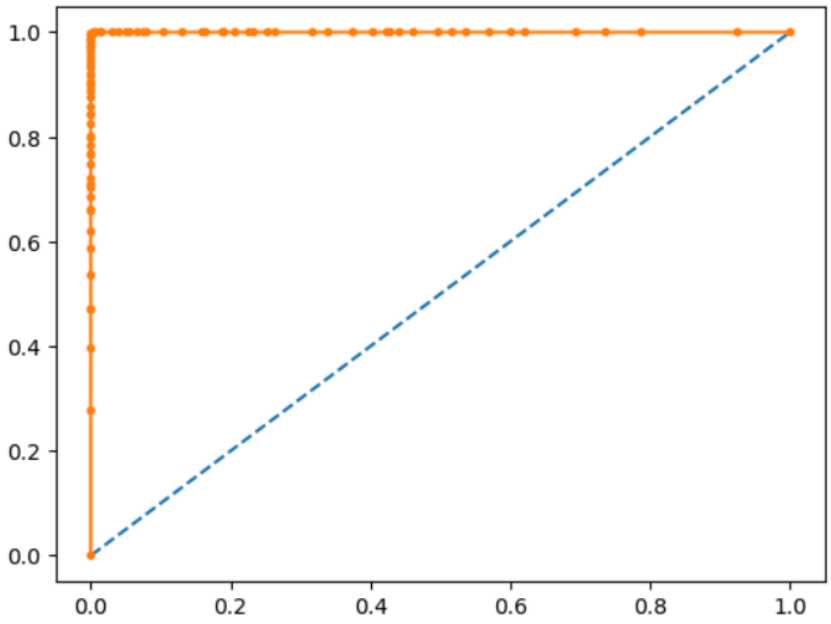
```
AUC: 0.904
```



Bagging

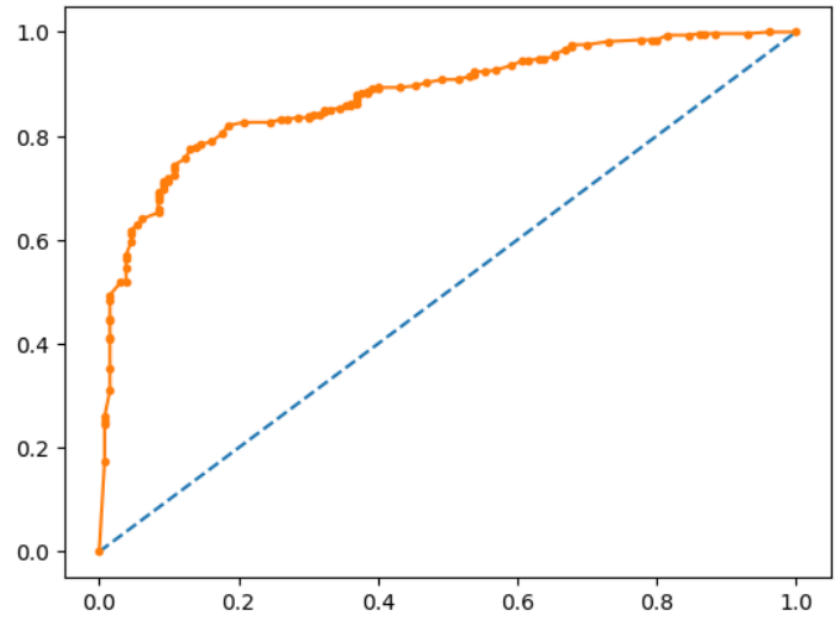
For training set-

```
AUC: 1.000
[<matplotlib.lines.Line2D at 0x1bd1549aa80>]
```



For test set-

```
AUC: 0.877
```



We looked at the ROC-AUC scores and respective plots in the above section. Let us look at the confusion matrices:

For Naive Bayes-

Training set-
[[240 92]
 [86 649]]

Test set-
[[94 36]
 [44 284]]

For KNN-

Training set-
[[246 86]
 [62 673]]

Test set-
[[94 36]
 [44 284]]

For Gradient Boosting-

Training set-
[[262 70]
 [51 684]]

Test set-
[[96 34]
 [43 285]]

For AdaBoost-

Training set-

[[238 94]
 [69 666]]

Test set-
[[90 40]
 [43 285]]

Bagging

Training set-
[[331   1]
 [0 735]]

Test set-
[[83 47]
 [46 282]]

As we can see in Naive Bayes most of the classes have been correctly predicted according to their numbers in the dataset.

Problem 1.5 - Model Performance improvement

- Improve the model performance of bagging and boosting models by tuning the model - Comment on the model performance improvement on training and test data

Solution:

We use SMOTE to tune the performance of bagging and boosting models. Let us look at the performances and scores of both the models after tuning the data set.

For Gradient Boosting with SMOTE:

We can see that the model is overfit for class 0 by 20%.

```
y_train_predict = gbcl_SM.predict(x_train_res)
model_score = gbcl_SM.score(x_train_res, y_train_res)
print(model_score)
print(metrics.confusion_matrix(y_train_res, y_train_predict))
print(metrics.classification_report(y_train_res, y_train_predict)
```

```
0.9027210884353741
[[674  61]
 [ 82 653]]
              precision    recall  f1-score   support

           0       0.89      0.92      0.90       735
           1       0.91      0.89      0.90       735

    accuracy                           0.90      1470
   macro avg       0.90      0.90      0.90      1470
weighted avg       0.90      0.90      0.90      1470
```

```
y_test_predict = gbcl_SM.predict(x_test)
model_score = gbcl_SM.score(x_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.8078602620087336
[[103  27]
 [ 61 267]]
              precision    recall  f1-score   support

           0       0.63      0.79      0.70       130
           1       0.91      0.81      0.86       328

    accuracy                           0.81       458
   macro avg       0.77      0.80      0.78       458
weighted avg       0.83      0.81      0.81       458
```

For AdaBoost with SMOTE:

We can see that the model is overfit for class 0 by 17%

```
y_train_predict = ADB_SM_model.predict(x_train_res)
model_score = ADB_SM_model.score(x_train_res, y_train_res)
print(model_score)
print(metrics.confusion_matrix(y_train_res, y_train_predict))
print(metrics.classification_report(y_train_res, y_train_predict))
```

```
0.8551020408163266
[[632 103]
 [110 625]]
              precision    recall  f1-score   support

           0       0.85      0.86      0.86       735
           1       0.86      0.85      0.85       735

    accuracy                           0.86      1470
   macro avg       0.86      0.86      0.86      1470
weighted avg       0.86      0.86      0.86      1470
```

```
y_test_predict = ADB_SM_model.predict(x_test)
model_score = ADB_SM_model.score(x_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.8013100436681223
[[100  30]
 [ 61 267]]
              precision    recall  f1-score   support

           0       0.62      0.77      0.69       130
           1       0.90      0.81      0.85       328

    accuracy                           0.80       458
   macro avg       0.76      0.79      0.77       458
weighted avg       0.82      0.80      0.81       458
```

For Bagging with SMOTE:

We can see that the bagging model is highly overfit as differences between f1 scores of both classes for training and test sets are more than 10%. Hence, this model is not recommended at all.

```
y_train_predict = Bagging_SM_model.predict(x_train_res)
model_score =Bagging_SM_model.score(x_train_res, y_train_res)
print(model_score)
print(metrics.confusion_matrix(y_train_res, y_train_predict))
print(metrics.classification_report(y_train_res, y_train_predict))
```

```
0.998639455782313
[[733   2]
 [  0 735]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       735
           1       1.00      1.00      1.00       735

    accuracy                           1.00      1470
   macro avg       1.00      1.00      1.00      1470
weighted avg       1.00      1.00      1.00      1470
```

```
y_test_predict = Bagging_SM_model.predict(x_test)
model_score = Bagging_SM_model.score(x_test, y_test)
print(model_score)
print(metrics.confusion_matrix(y_test, y_test_predict))
print(metrics.classification_report(y_test, y_test_predict))
```

```
0.8056768558951966
[[ 96  34]
 [ 55 273]]
              precision    recall  f1-score   support

           0       0.64      0.74      0.68       130
           1       0.89      0.83      0.86       328

    accuracy                           0.81       458
   macro avg       0.76      0.79      0.77       458
weighted avg       0.82      0.81      0.81       458
```

Although after tuning we can see that the AdaBoost method has the least overfit issues among the three, we would prefer to use the Naive Bayes model.

Problem 1.6 - Final Model Selection

We have already discussed that the final model is the Naive Bayes model. Let us add some reasons why:

Fast Training and Prediction: It is computationally efficient and requires less training time compared to more complex models, which is particularly beneficial for large datasets or real-time applications.

Effective with Small Data: Naive Bayes can perform surprisingly well even with a small amount of training data, as it doesn't require a large dataset to estimate parameters.

Resilience to Noise: The model is generally less affected by irrelevant features, thanks to its independence assumption (features are treated as independent of each other given the class label).

Overfitting Prevention: Due to its simplicity and the strong independence assumptions it makes, Naive Bayes is less likely to overfit compared to more complex models, particularly when dealing with a large number of features.

Handling Missing Data: In some cases, Naive Bayes can handle missing data effectively by simply ignoring the missing values during the probability calculation, which can be an advantage in practical scenarios where data might be incomplete - which could be very beneficial in our context of predicting election choices since sometimes all surveys do not have complete details.

Problem 1.7 - Actionable Insights & Recommendations

- Compare all four models - Conclude with the key takeaways for the business

Solution:

- Class 0 (Conservative Party):

Precision: 0.68 (Test Set) - Indicates that 68% of the predictions made by the model for Conservative Party were correct. Recall: 0.72 (Test Set) - Indicates that the model correctly identified 72% of the actual instances of Conservative Party. F1-Score: 0.70 (Test Set) - Balances the precision and recall, showing a moderate performance for predicting Conservative Party

- Class 1 (Labour Party):

Precision: 0.89 (Test Set) - Indicates that 89% of the predictions made for the Labour Party were correct. Recall: 0.87 (Test Set) - Indicates that the model correctly identified 87% of the actual instances of the Labour Party. F1-Score: 0.88 (Test Set) - Shows strong performance in predicting the Labour Party which has more instances in the dataset.

- The Labour Party is more likely to be predicted correctly by the model, possibly because it has more instances in the dataset, which leads to better training for this class.

- Conservative Party predictions are less accurate, likely due to fewer instances in the data. This could mean the model has less information to learn from or that Conservative Party's voting patterns are more difficult to predict.

- Model Reliability: The relatively high accuracy and balanced performance across precision and recall suggest the model can be reliably used to predict which party is more likely to get votes. However, the imbalance in class distribution should be considered when interpreting the results.

Problem 2.1 - Define the problem and Perform Exploratory Data Analysis

-Problem Definition - Find the number of Characters, words & sentences in all three speeches.

Solution:

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

President Franklin D. Roosevelt in 1941
President John F. Kennedy in 1961
President Richard Nixon in 1973
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')
"

We download the speeches and remove stopwords, perform stemming and find frequency distribution for the first problem.

```
speeches = {
    "Roosevelt_1941": inaugural.raw('1941-Roosevelt.txt'),
    "Kennedy_1961": inaugural.raw('1961-Kennedy.txt'),
    "Nixon_1973": inaugural.raw('1973-Nixon.txt')
}
```

After analysis of each speech the results are:

```
Analysis for Roosevelt_1941:
Number of characters: 7571
Number of words: 1526
Number of sentences: 68


Analysis for Kennedy_1961:
Number of characters: 7618
Number of words: 1543
Number of sentences: 52


Analysis for Nixon_1973:
Number of characters: 9991
Number of words: 2006
Number of sentences: 68
```

Problem 2.2 - Text cleaning

- Stopword removal - Stemming - find the 3 most common words used in all three speeches.

Solution:

We have already done the stopword removal and stemming.

```
# Remove stopwords and perform stemming
filtered_words = [ps.stem(word.lower()) for word in speeches if word.isalnum() and word.lower() not in stop_words]
```

```
Top 3 most common words in Roosevelt_1941's speech:
[('nation', 17), ('know', 10), ('peopl', 9)]


Top 3 most common words in Kennedy_1961's speech:
[('let', 16), ('us', 12), ('power', 9)]


Top 3 most common words in Nixon_1973's speech:
[('us', 26), ('let', 22), ('america', 21)]
```

Problem 2.3 - Plot Word cloud of all three speeches

- Show the most common words used in all three speeches in the form of word clouds

Solution:

Word Cloud for Kennedy_1961

Word Cloud for Nixon_1973