Machine Learning Project Business Report

NAME : SHUBHAM PADOLE

PGD-DSBA Online

Date:23/11/2021

Problem 1:

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

Data Set used : Election_data.xlsx

1.1 Read the dataset. Do the descriptive statistics and do the null value condition check. Write an inference on it. (4 Marks)

- Importing library

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn
import sklearn
from sklearn import tree
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import roc_auc_score,roc_curve
```

- Load the dataset

```
data_df= pd.read_excel("Election_Data.xlsx",sheet_name="Election_Dataset_Two Classes")
```

```
data_df=data_df.drop('Unnamed: 0',axis=1)
```

```
data_df.head()
```

- Head of the dataset and describetion

Since the ‘vote’ variable is the target, we therefore have ‘vote’ as the dependent and rest 8 variables as the independent or predictor variables.
Looking at the first 5 records of the dataset gives the following:

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

```
data_df.describe().T
```

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

- Check the nullvalue and data_info

```
data_df.isnull().sum()
```

```
vote                       0
age                        0
economic.cond.national     0
economic.cond.household    0
Blair                      0
Hague                      0
Europe                     0
political.knowledge        0
gender                     0
dtype: int64
```

```
data_df.info()
```

```
<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
```

Unnamed column has been dropped from the data frame, now we left with only 8 columns in it. All the independent continuous columns has a integer datatype although some of the category columns have object datatype and that can be handled using one hot coding. The dependent column has object datatype

- Dumping the duplicate and data shape

```
dups=data_df.duplicated()
print("Total no of duplicate values = %d" % (dups.sum()))
data_df[dups]
```

```
Total no of duplicate values = 8
```

	vote	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	gender
67	Labour	35	4	4	5	2	3	2	male
626	Labour	39	3	4	4	2	5	2	male
870	Labour	38	2	4	2	2	4	3	male
983	Conservative	74	4	3	2	4	8	2	female
1154	Conservative	53	3	4	2	2	6	0	female
1236	Labour	36	3	3	2	2	6	2	female
1244	Labour	29	4	4	4	2	2	2	female
1438	Labour	40	4	3	4	2	2	2	male

```
data_df.shape
```

```
(1525, 9)
```

Checking the value count

```
data_df.vote.value_counts()
```

```
Labour          1063
Conservative     462
Name: vote, dtype: int64
```

```
for feature in data_df.columns:
    if data_df[feature].dtype=='object':
            print(feature.upper() ," ",data_df[feature].nunique())
            print(data_df[feature].value_counts().sort_values())
```

```
VOTE   2
Conservative     462
Labour          1063
Name: vote, dtype: int64
GENDER   2
male      713
female    812
Name: gender, dtype: int64
```

The male and female voters are briefly divided across “Labour” and “Conservative” parties. People prefer Labour party more over the Conservative parity. Although female voter count over pass the number of male voters.

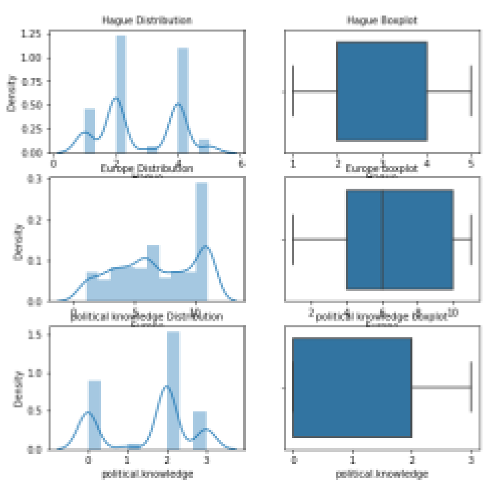
Viewing dtypes

```
data_df.dtypes
```

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

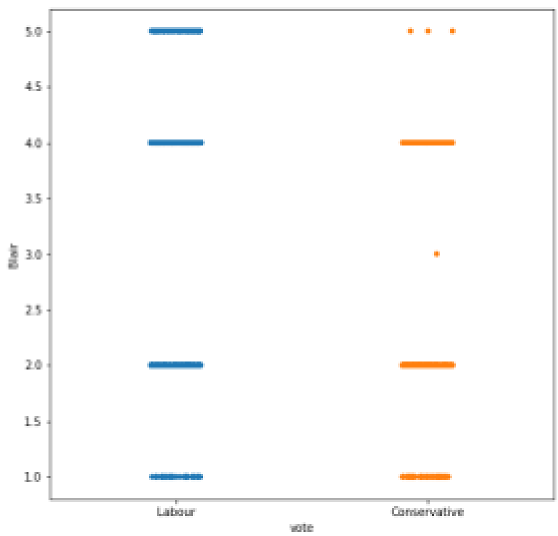
1.2 Perform Univariate and Bivariate Analysis. Do exploratory data analysis. Check for Outliers.

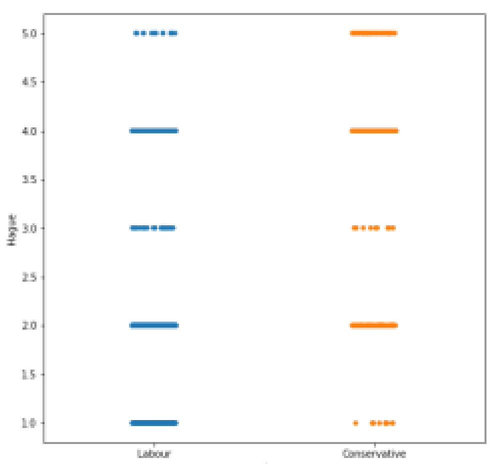
Univariate Analysis

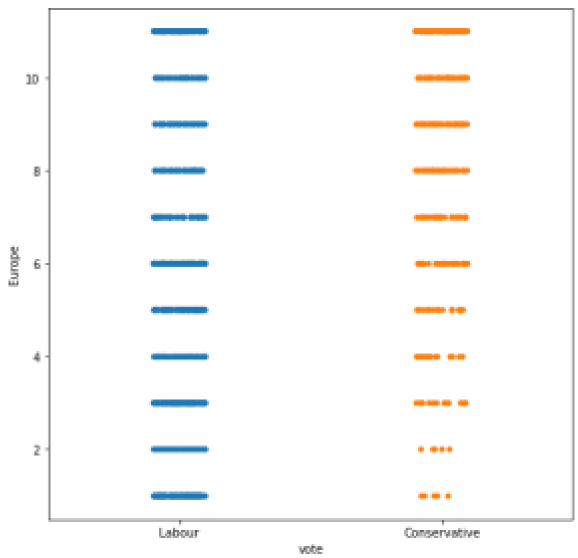


Bivariate and Multivariate Analysis

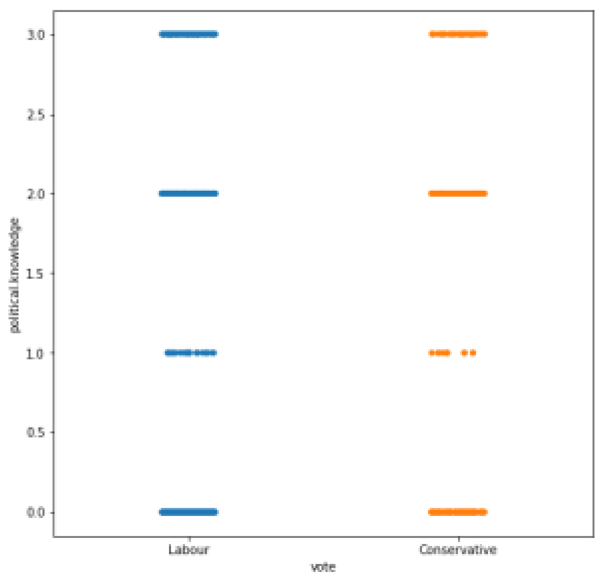
Distribution of Male and Female voters across the age is almost in equal proportion with Females slighter higher in number in comparison to Males, also both female and male voters age lies between 40 to 70 years.





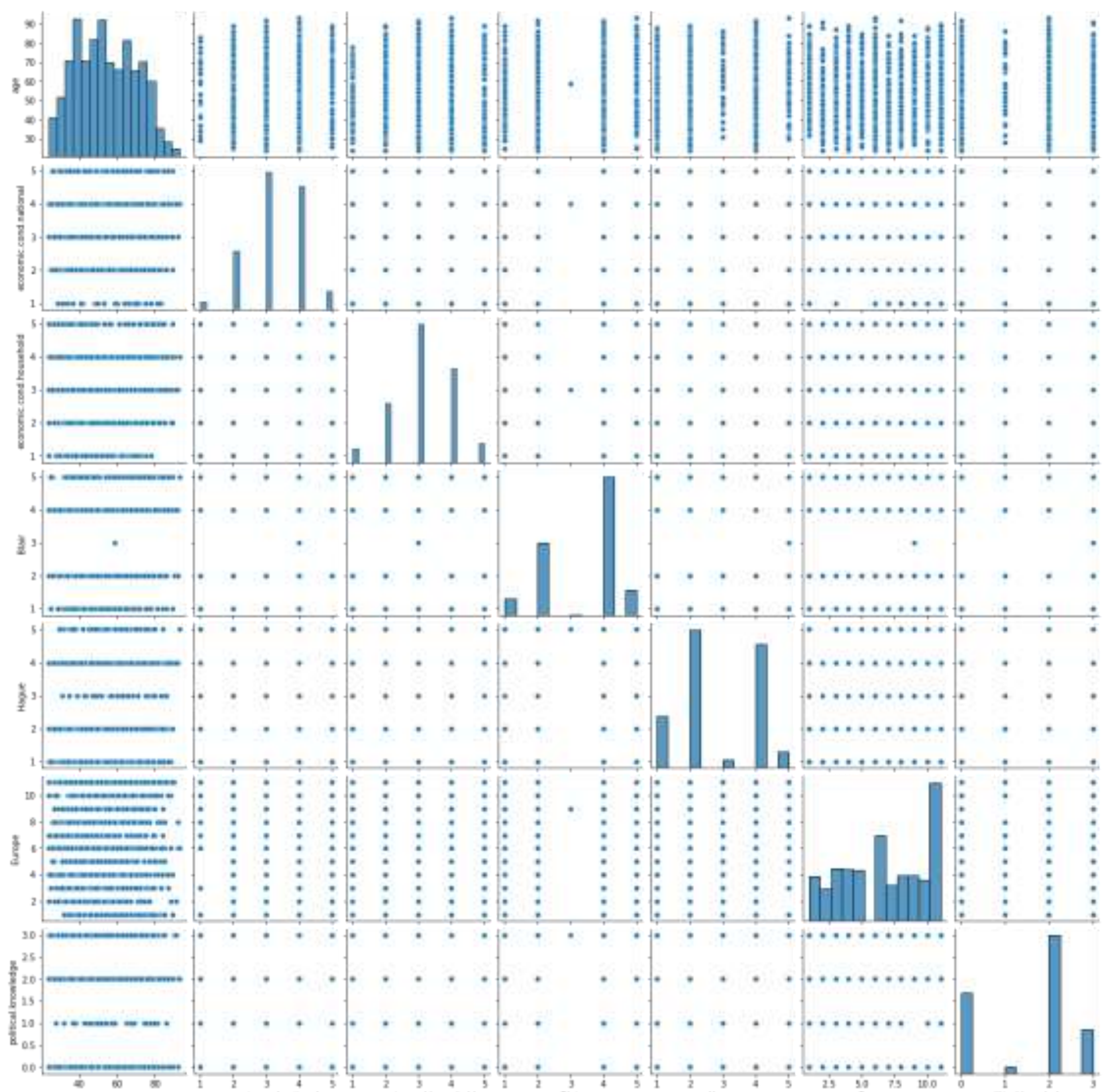


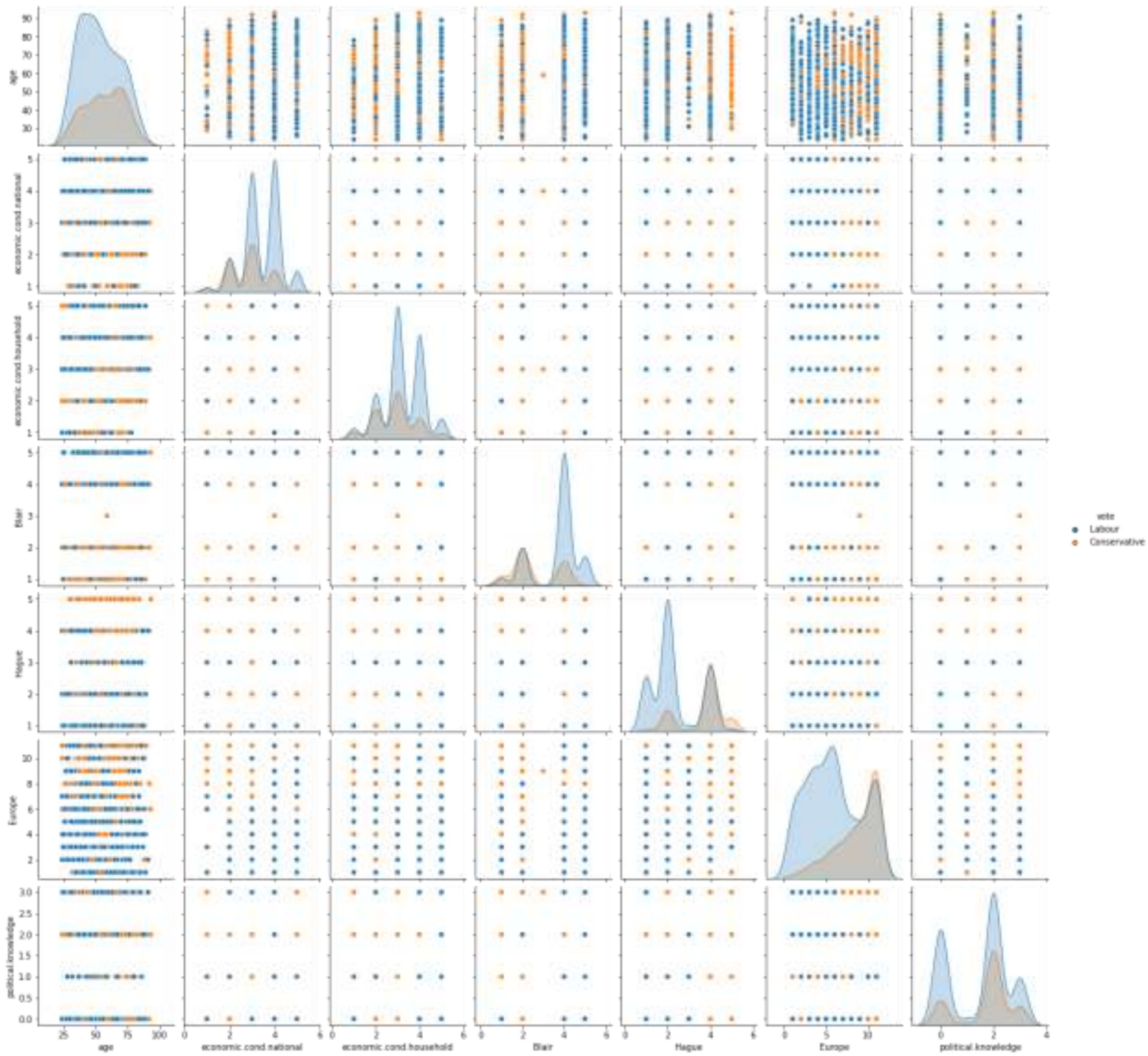
Europe
10
8
6
4
2
Labour
Conservative
vote



political knowledge
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Labour
Conservative
vote

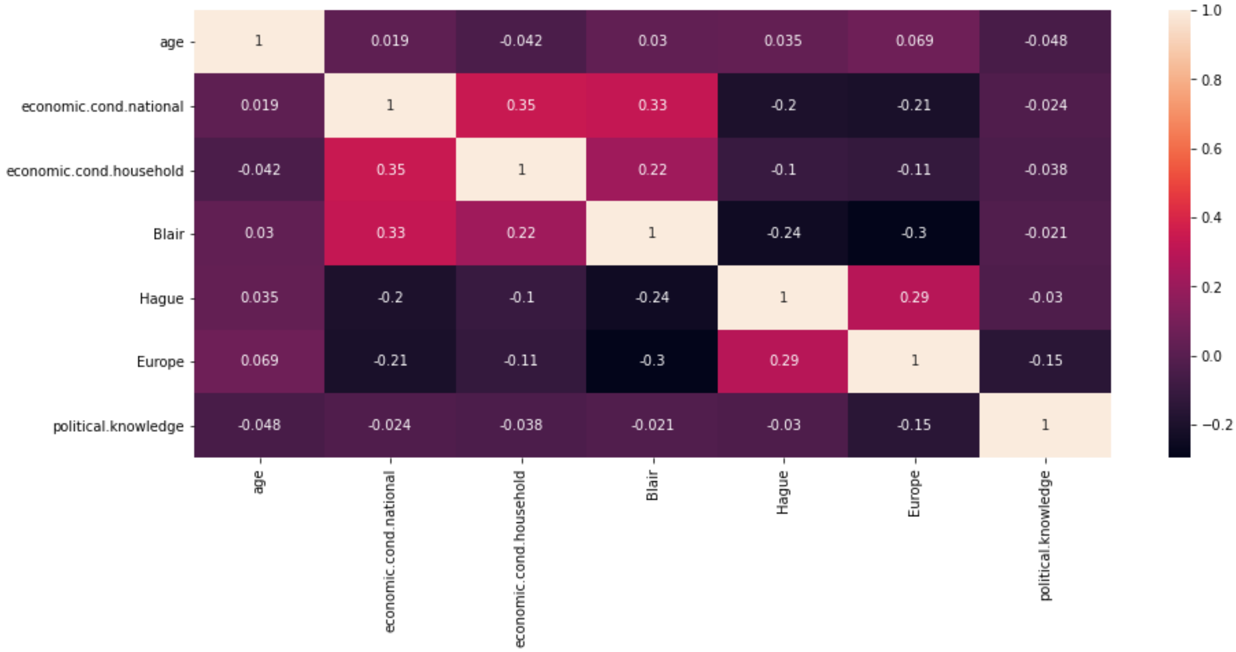
Pair plots


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



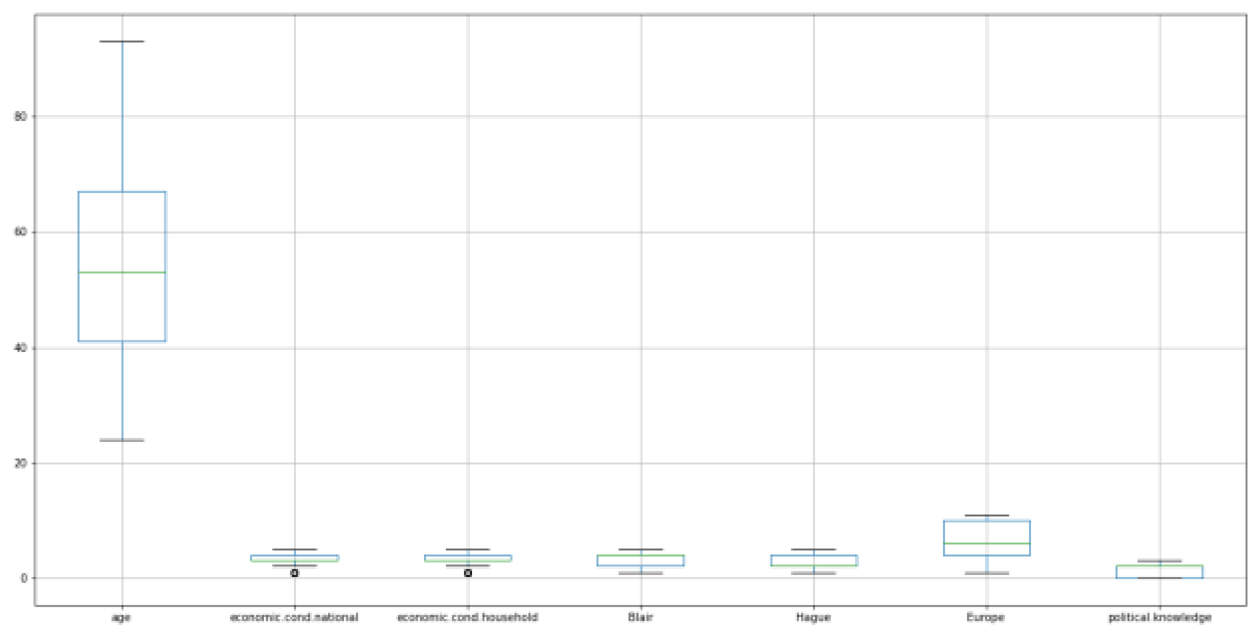
age
economic.cond.national
economic.cond.household
Blair
Hague
Europe
political.knowledge
vote
Labour
Conservative

Heat map

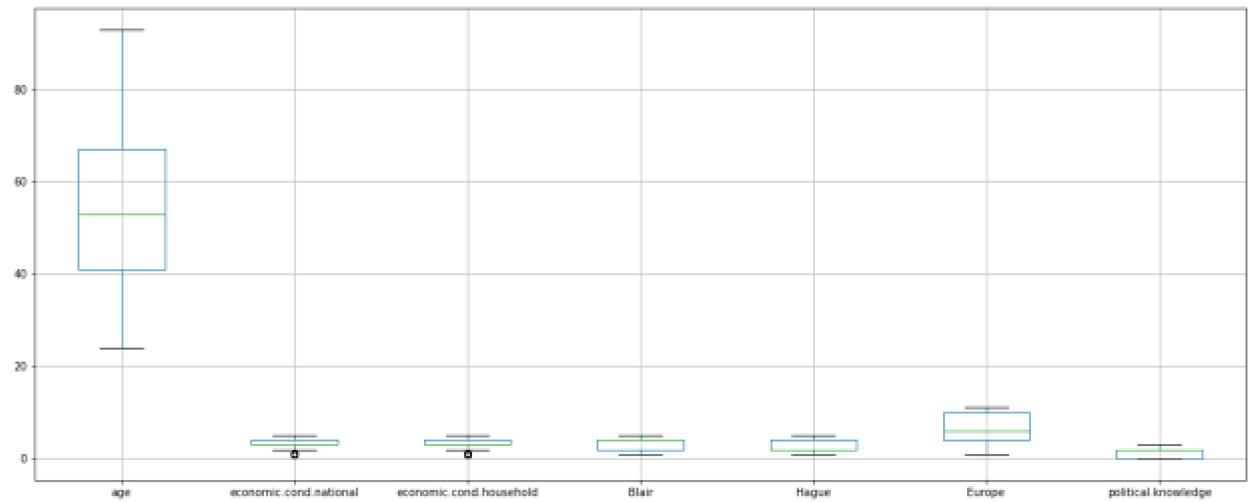


age
economic.cond.national
economic.cond.household
Blair
Hague
Europe
political.knowledge

Box Plot



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



Observation:

- can be easily observed that relatively younger people have voted for "Labour" party in comparison to that of older people who voted for "Conservative" party.
- There is an evenly distributed number of people when it comes to thier knowledge about their party's position on European integration.
- Majority of European people have voted for "Labour" party
- There exists an outlier for economic.cond.household and economic.cond.national.

1.3 Encode the data (having string values) for Modelling. Is Scaling necessary here or not? Data Split: Split the data into train and test (70:30)

```
data_df.describe().T
```

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

```
cat1 = ['vote', 'gender']
```

```
df =pd.get_dummies(data_df, columns=cat1,drop_first=True)
```

```
df.head()
```

	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	vote_Labour	gender_male
0	43	3	3	4	1	2	2	1	0
1	36	4	4	4	4	5	2	1	1
2	35	4	4	5	2	3	2	1	1
3	24	4	2	2	1	4	0	1	0
4	41	2	2	1	1	6	2	1	1

```
df= df.rename(columns={'vote_Labour':'IsLabour_or_not' ,'gender_male':'IsMale_or_not'},inplace= False)
```

```
df.sample(10)
```

	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	IsLabour_or_not	IsMale_or_not
296	62	4	2	4	4	6	2	1	1
1036	27	3	5	2	4	6	2	1	0
70	64	4	4	4	2	2	3	1	0
638	41	4	4	4	2	5	2	1	1
1110	49	3	1	4	4	4	0	1	1
307	38	4	4	4	2	4	2	1	1
131	92	3	3	4	4	8	2	0	0
1335	67	4	2	2	2	10	1	1	0
1393	72	2	2	2	5	11	2	0	1
764	25	4	4	4	2	6	2	1	1

We have split the data into train and test

```
X=df.drop('IsLabour_or_not',axis=1)
Y=df['IsLabour_or_not']
```

```
X_train,X_test, Y_train, Y_test=train_test_split(X,Y,train_size=0.70, random_state=1)
```

Observation :

- Vote' and 'Gender' variables are encoded. Also 'Age' variable has been categorized into several bins of age groups. Since Age variable is the only numeric variable and by itself can't be meaningfully used, we plan to categorize the variable into groups, basis the First, Second, Third and Fourth Quartile values.

Is Scaling necessary here or not?

- Scaling doesn't seems to be required here as there are only categorical independent variables in the Dataset.

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

Discriminant Analysis

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

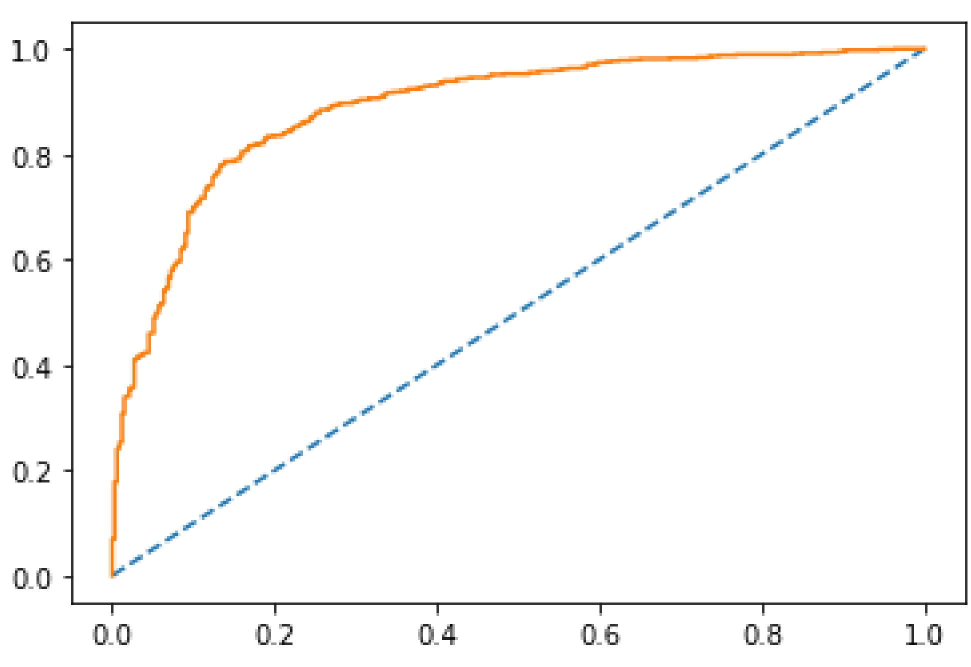
```
LDA_model=LinearDiscriminantAnalysis()
LDA_model.fit(X_train,Y_train)
```

```
LinearDiscriminantAnalysis()
```

```
y_train_predict=LDA_model.predict(X_train)
LDA_model_score=LDA_model.score(X_train,Y_train)
print(LDA_model_score)

print(metrics.confusion_matrix(Y_train,y_train_predict))
print(metrics.classification_report(Y_train,y_train_predict))
```

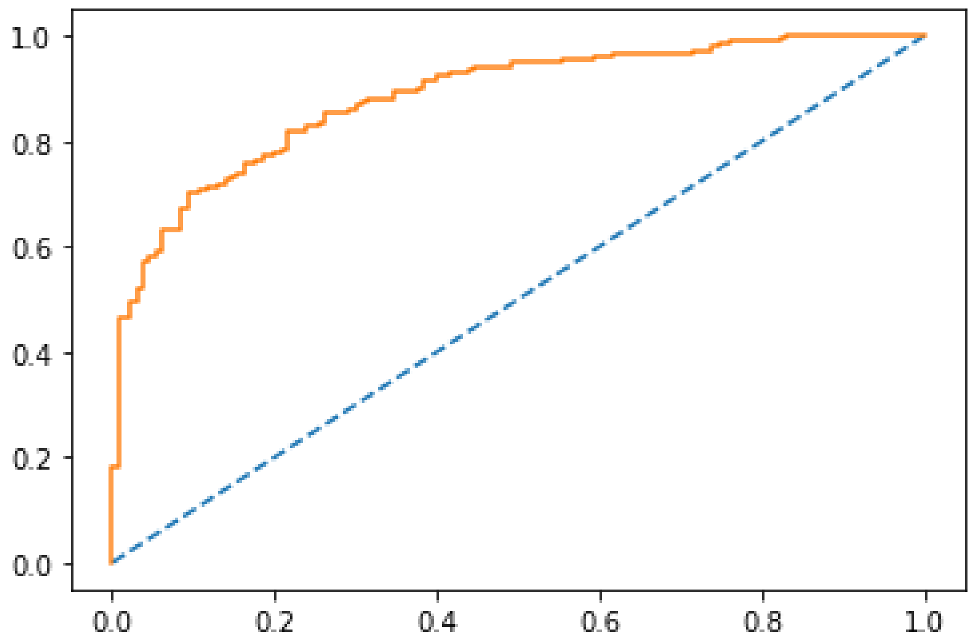
```
0.8369259606373008
[[233  99]
 [ 75 660]]
              precision    recall  f1-score   support

           0       0.76      0.70      0.73       332
           1       0.87      0.90      0.88       735

    accuracy                           0.84      1067
   macro avg       0.81      0.80      0.81      1067
weighted avg       0.83      0.84      0.84      1067
```

AUC ROC curve for LDA Train



```
0.8187772925764192
[[ 86  44]
 [ 39 289]]
              precision    recall  f1-score   support

           0       0.69      0.66      0.67       130
           1       0.87      0.88      0.87       328

    accuracy                           0.82       458
   macro avg       0.78      0.77      0.77       458
weighted avg       0.82      0.82      0.82       458
```

AUC ROC curve for LDA Test



Logistic Regression

```python
from sklearn.linear_model import LogisticRegression
```

Logistic_model=LogisticRegression() Logistic_model.fit(X_train,Y_train)

```python
Logistic_model = LogisticRegression(solver='newton-cg',max_iter=10000,penalty='none',verbose=True,n_jobs=2)
Logistic_model.fit(X_train, Y_train)
```

```
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done   1 out of   1 | elapsed:    7.7s finished
```

```
LogisticRegression(max_iter=10000, n_jobs=2, penalty='none', solver='newton-cg',
                   verbose=True)
```

```python
y_train_predict=Logistic_model.predict(X_train)
Logistic_model_score=Logistic_model.score(X_train,Y_train)
print(Logistic_model_score)

print(metrics.confusion_matrix(Y_train,y_train_predict))
print(metrics.classification_report(Y_train,y_train_predict))
```

```
0.8406747891283973
[[230 102]
 [ 68 667]]
              precision    recall  f1-score   support

           0       0.77      0.69      0.73       332
           1       0.87      0.91      0.89       735

    accuracy                           0.84      1067
   macro avg       0.82      0.80      0.81      1067
weighted avg       0.84      0.84      0.84      1067
```

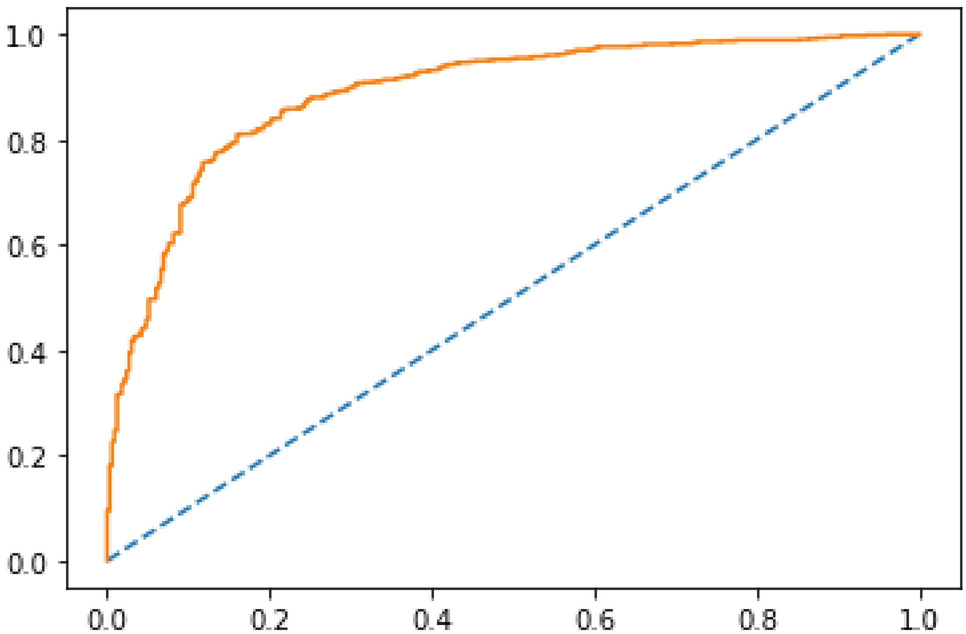
:

	0	1
0	0.616214	0.383786
1	0.186460	0.813540
2	0.187994	0.812006
3	0.163937	0.836063
4	0.052483	0.947517

:
```python
Logistic_model.score(X_train,Y_train)
```

: 0.8406747891283973

AUC ROC curve for Logistic Regression Train



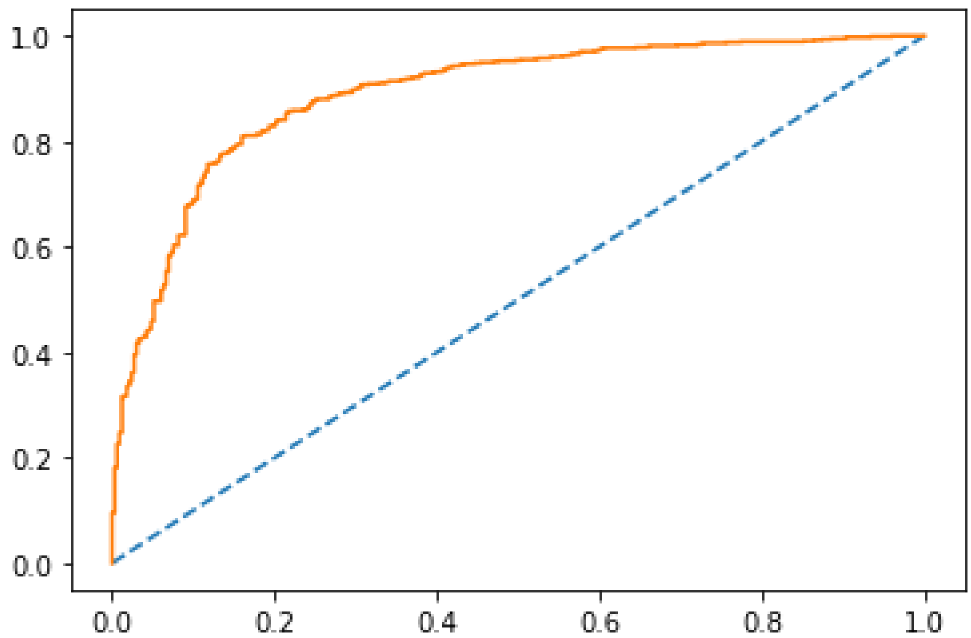

```
y_test_predict=Logistic_model.predict(X_test)
Logistic_model_score=Logistic_model.score(X_test,Y_test)
print(Logistic_model_score)

print(metrics.confusion_matrix(Y_test,y_test_predict))
print(metrics.classification_report(Y_test,y_test_predict))
```

```
0.8231441048034934
[[ 85  45]
 [ 36 292]]
              precision    recall  f1-score   support

           0       0.70      0.65      0.68       130
           1       0.87      0.89      0.88       328

    accuracy                           0.82       458
   macro avg       0.78      0.77      0.78       458
weighted avg       0.82      0.82      0.82       458
```

```
y_test_prob=Logistic_model.predict_proba(X_test)
pd.DataFrame(y_test_prob).head()
```

	0	1
0	0.933648	0.066352
1	0.689194	0.310806
2	0.333480	0.666520
3	0.477407	0.522593
4	0.157152	0.842848

AUC ROC curve for Logistic Regression Test



Observation :

- The model is defined with above parameters and fitted on Train and Test data. The model gives an accuracy score as 82.66% for Train Data and 83.80% on Test Data.

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

KNN

```python
x=df.drop("IsLabour_or_not",axis=1)

y=df["IsLabour_or_not"]
```

```python
x.head()
```

	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	IsMale_or_not
0	43	3	3	4	1	2	2	0
1	36	4	4	4	4	5	2	1
2	35	4	4	5	2	3	2	1
3	24	4	2	2	1	4	0	0
4	41	2	2	1	1	6	2	1

```python
from scipy.stats import zscore
```

```python
x[['age','economic.cond.national','economic.cond.household','Blair','Hague','Europe','political.knowledge','IsMale_or_not']]
```

```python
x.head(10)
```

	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	IsMale_or_not
0	-0.711973	-0.279218	-0.150948	0.566716	-1.419886	-1.434426	0.422643	-0.937059
1	-1.157661	0.856268	0.924730	0.566716	1.018544	-0.524358	0.422643	1.067169
2	-1.221331	0.856268	0.924730	1.418187	-0.607076	-1.131070	0.422643	1.067169
3	-1.921698	0.856268	-1.226625	-1.136225	-1.419886	-0.827714	-1.424148	-0.937059
4	-0.839313	-1.414704	-1.226625	-1.987695	-1.419886	-0.221002	0.422643	1.067169
5	-0.457295	-0.279218	0.924730	0.566716	1.018544	-0.827714	0.422643	1.067169

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y, random_state=1)
```

```
from sklearn.neighbors import KNeighborsClassifier

KNN_model=KNeighborsClassifier()
KNN_model.fit(x_train,y_train)
```

```
KNeighborsClassifier()
```

```
y_train_predict=KNN_model.predict(x_train)
KNN_model_score=KNN_model.score(x_train,y_train)
```

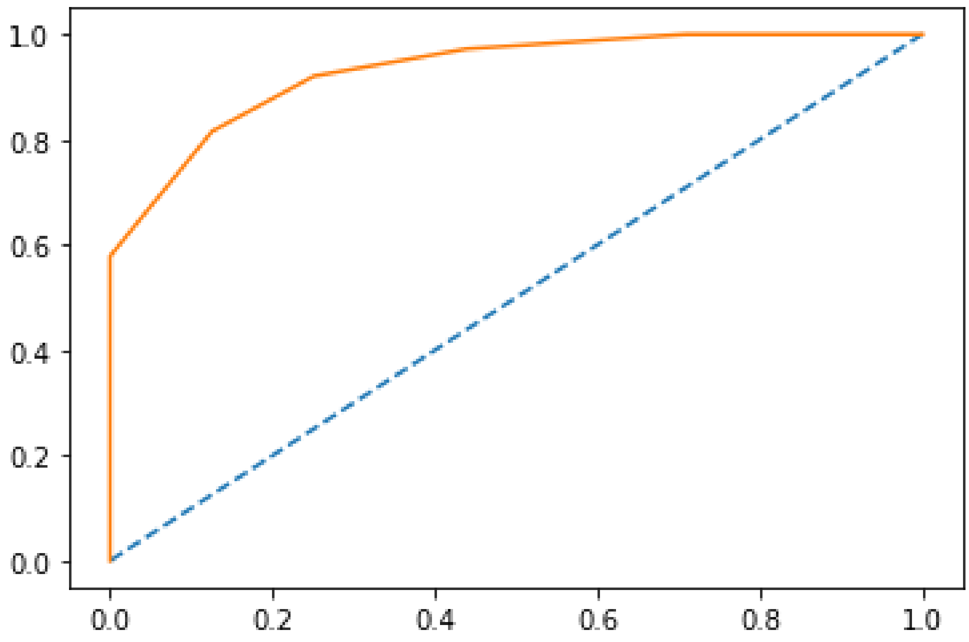
```
print(KNN_model_score)
```

```
0.8678915135608049
```

```
print(metrics.confusion_matrix(y_train,y_train_predict))
print(metrics.classification_report(y_train,y_train_predict))
```

```
[[263  88]
 [ 63 729]]
              precision    recall  f1-score   support

           0       0.81      0.75      0.78       351
           1       0.89      0.92      0.91       792

    accuracy                           0.87      1143
   macro avg       0.85      0.83      0.84      1143
weighted avg       0.87      0.87      0.87      1143
```

AUC ROC Curve KNN Train



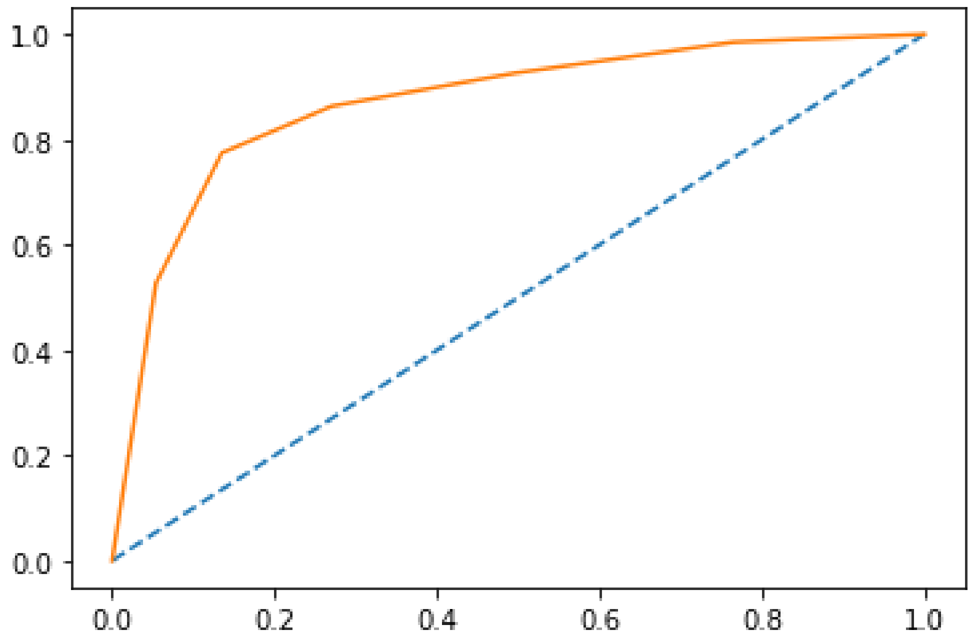

```
y_test_predict=KNN_model.predict(x_test)

KNN_model_score=KNN_model.score(x_test, y_test)

print(KNN_model_score)
```

```
0.824607329842932
```

```
print(metrics.confusion_matrix(y_test,y_test_predict))
print(metrics.classification_report(y_test,y_test_predict))
```

```
[[ 81  30]
 [ 37 234]]
              precision    recall  f1-score   support

           0       0.69      0.73      0.71       111
           1       0.89      0.86      0.87       271

    accuracy                           0.82       382
   macro avg       0.79      0.80      0.79       382
weighted avg       0.83      0.82      0.83       382
```

AUC ROC Curve KNN Test



the auc curve 0.870

```
KNN_model=KNeighborsClassifier(n_neighbors=7)
KNN_model.fit(x_train,y_train)
```

```
KNeighborsClassifier(n_neighbors=7)
```

```
y_train_predict=KNN_model.predict(x_train)
KNN_model_score=KNN_model.score(x_train,y_train)
print(KNN_model_score)
print(metrics.confusion_matrix(y_train,y_train_predict))
print(metrics.classification_report(y_train,y_train_predict))
```

```
0.8530183727034121
[[253  98]
 [ 70 722]]
              precision    recall  f1-score   support

           0       0.78      0.72      0.75       351
           1       0.88      0.91      0.90       792

    accuracy                           0.85      1143
   macro avg       0.83      0.82      0.82      1143
weighted avg       0.85      0.85      0.85      1143
```

```
0.8350785340314136
[[ 84  27]
 [ 36 235]]
              precision    recall  f1-score   support

           0       0.70      0.76      0.73       111
           1       0.90      0.87      0.88       271

    accuracy                           0.84       382
   macro avg       0.80      0.81      0.80       382
weighted avg       0.84      0.84      0.84       382
```

```python
y_train_predict=KNN_model.predict(x_train)
KNN_model_score=KNN_model.score(x_train,y_train)
print(KNN_model_score)
print(metrics.confusion_matrix(y_train,y_train_predict))
print(metrics.classification_report(y_train,y_train_predict))
```

```
0.8678915135608049
[[263  88]
 [ 63 729]]
              precision    recall  f1-score   support

           0       0.81      0.75      0.78       351
           1       0.89      0.92      0.91       792

    accuracy                           0.87      1143
   macro avg       0.85      0.83      0.84      1143
weighted avg       0.87      0.87      0.87      1143
```

```python
y_test_predict=KNN_model.predict(x_test)
KNN_model_score=KNN_model.score(x_test,y_test)
print(KNN_model_score)
print(metrics.confusion_matrix(y_test,y_test_predict))
print(metrics.classification_report(y_test,y_test_predict))
```

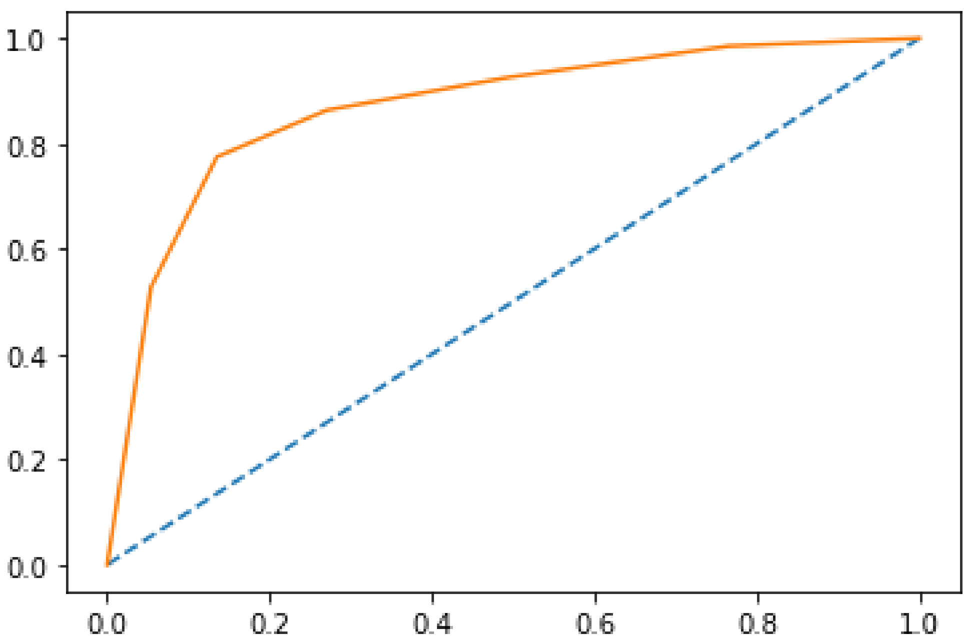
```
0.824607329842932
[[ 81  30]
 [ 37 234]]
              precision    recall  f1-score   support

           0       0.69      0.73      0.71       111
           1       0.89      0.86      0.87       271

    accuracy                           0.82       382
   macro avg       0.79      0.80      0.79       382
weighted avg       0.83      0.82      0.83       382
```

```python
ac_score=[]

for k in range(1,20,2):
    knn= KNeighborsClassifier(n_nei
    knn.fit(x_train, y_train)
    scores=knn.score(x_test,y_test]
    ac_score.append(scores)

MCE=[1-x for x in ac_score]
MCE
```

```
[0.23298429319371727,
 0.19633507853403143,
 0.17539267015706805,
 0.16492146596858637,
 0.17801047120418845,
 0.17277486910994766,
 0.17539267015706805,
 0.18586387434554974,
 0.17801047120418845,
 0.17277486910994766]
```

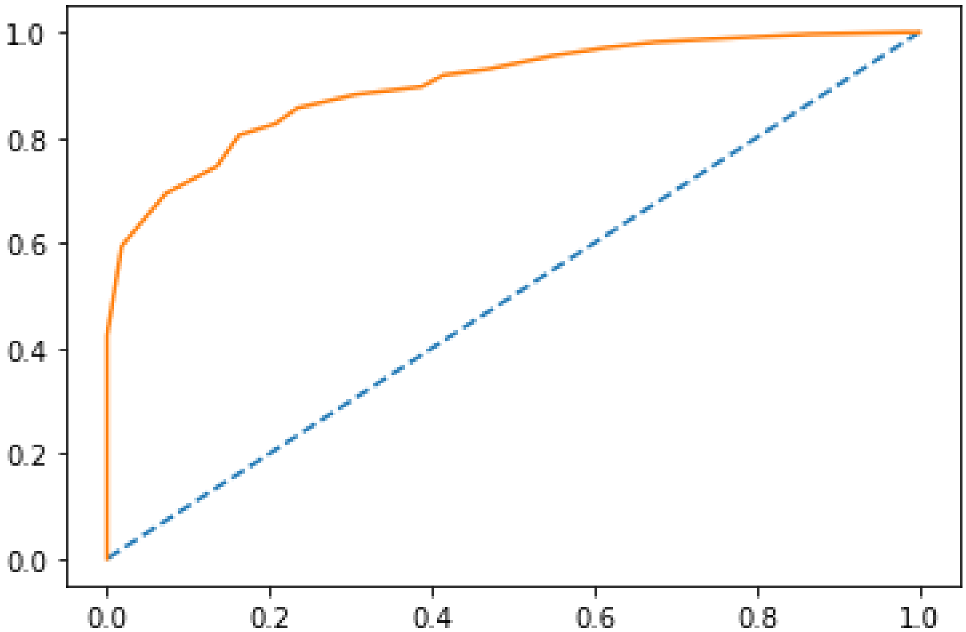
AUC ROC curve after n classifier for train data set

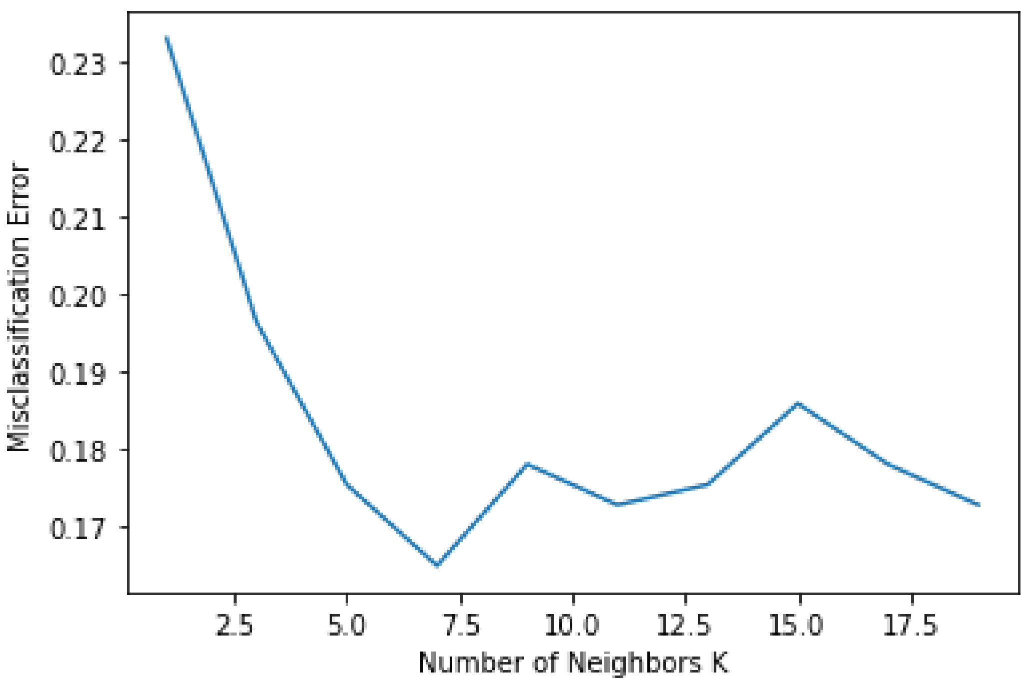
```
the auc curve 0.904
```



AUC ROC curve after n classifier for test data set

```
the auc curve 0.900
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Misclassification Error
0.23
0.22
0.21
0.20
0.19
0.18
0.17
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Number of Neighbors K

Naive Bayes

```python
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
```

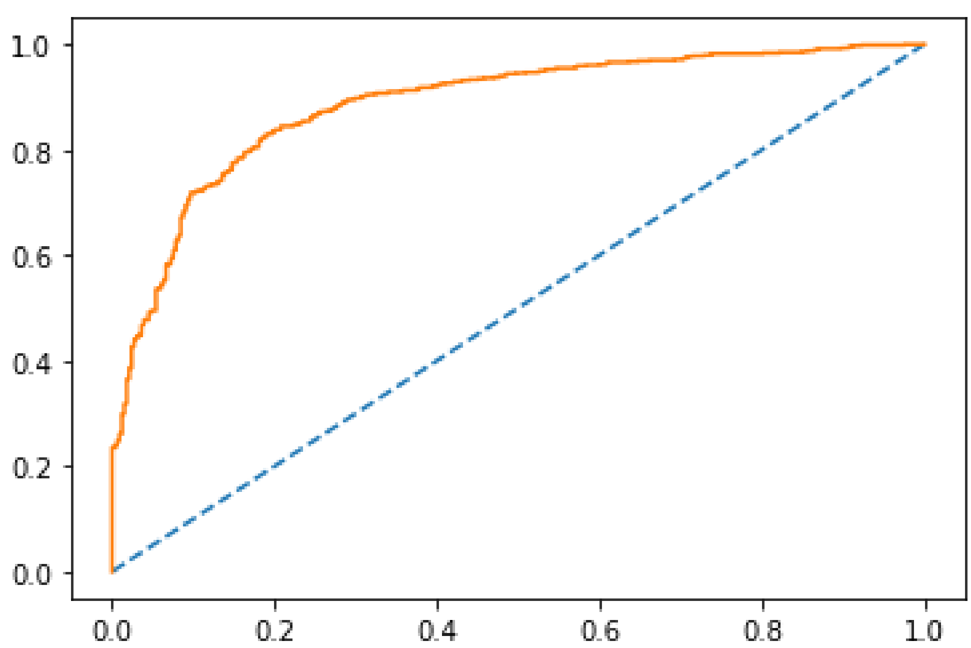
```python
NB_model=GaussianNB()
NB_model.fit(X_train, Y_train)
```

```
GaussianNB()
```

```python
Y_train_predict=NB_model.predict(X_train)
model_score=NB_model.score(X_train, Y_train)
print(model_score)
print(metrics.confusion_matrix(Y_train,Y_train_predict))

print(metrics.classification_report(Y_train,Y_train_predi
```

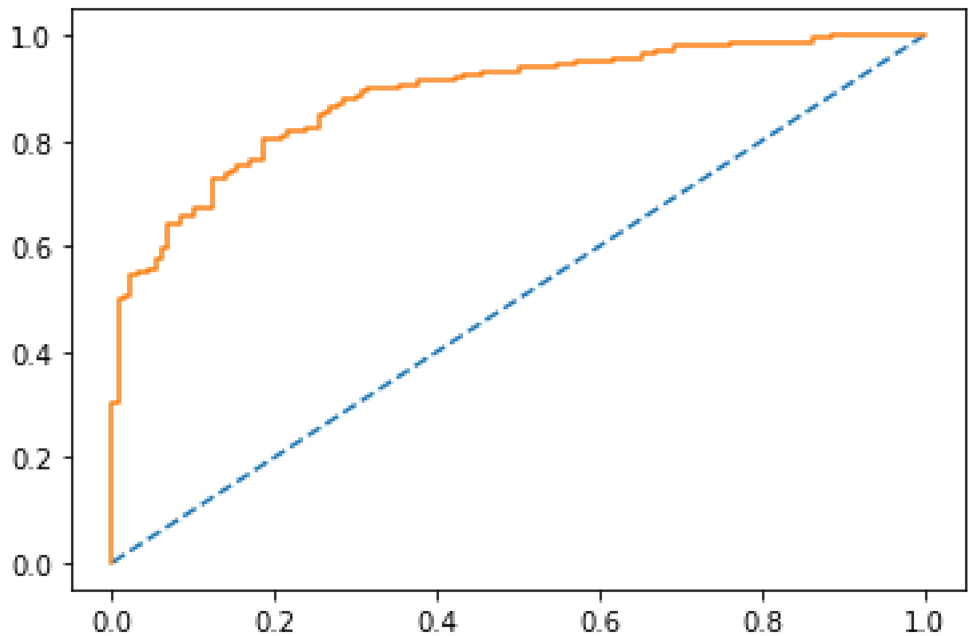
```
0.8331771321462043
[[240  92]
 [ 86 649]]
              precision    recall  f1-score   support

           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
```

the auc 0.886



```
0.8253275109170306
[[ 94  36]
 [ 44 284]]
              precision    recall  f1-score   support

           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
```

```
the auc curve 0.885
```



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

Bagging Train

```python
from sklearn.model_selection import train_test_split
X_train,X_test, Y_train,Y_test= train_test_split(X,Y,test_size=0.30,
```

```python
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
```

```python
cart=DecisionTreeClassifier()
Bagging_model=BaggingClassifier(base_estimator=cart,n_estimators=100

Bagging_model.fit(X_train,Y_train)
```

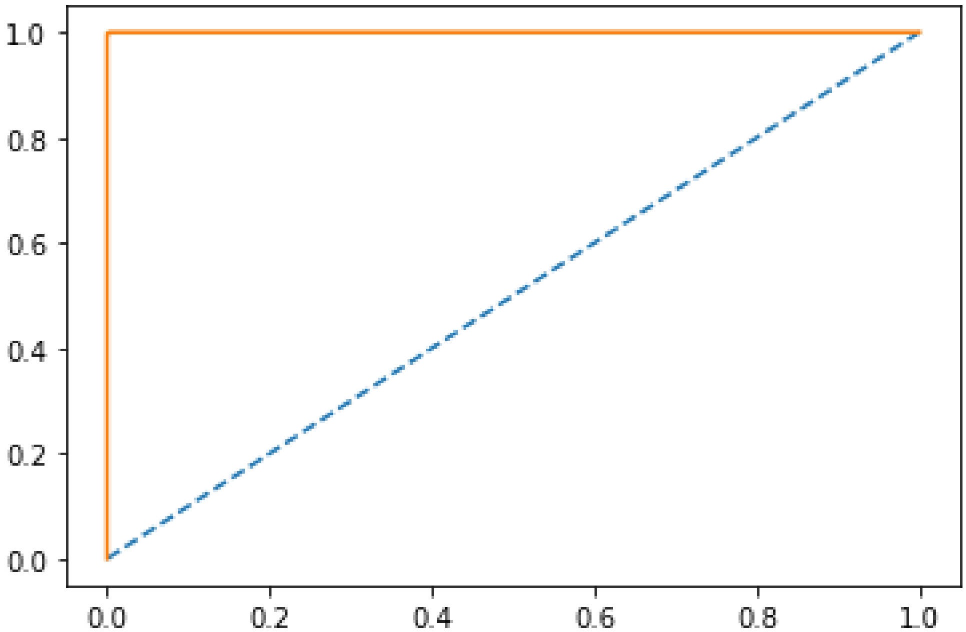
```
BaggingClassifier(base_estimator=DecisionTreeClassifier(), n_estimators=1
                  random_state=1)
```

```python
y_train_predict=Bagging_model.predict(X_train)
Bagging_model_score=Bagging_model.score(X_train,Y_train)
print(Bagging_model_score)

print(metrics.confusion_matrix(Y_train,y_train_predict))
print(metrics.classification_report(Y_train,y_train_predict))
```

```
0.9990627928772259
[[331   1]
 [  0 735]]
              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
```

AUC _ROC Curve Bagging Train

```
AUC: 1.000
```

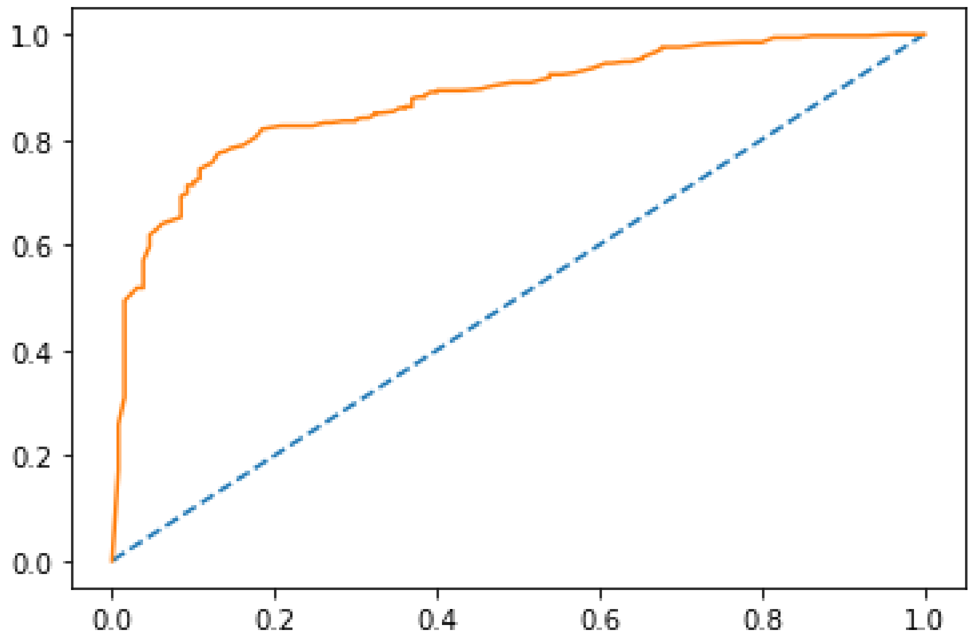


Bagging Test

```
0.7969432314410481
[[ 83  47]
 [ 46 282]]
              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
```

AUC _ROC Curve Bagging Test

AUC: 0.877



1.7 Performance Metrics: Check the performance of Predictions on Train and Test sets using Accuracy, Confusion Matrix, Plot ROC curve and get ROC_AUC score for each model. Final Model: Compare the models and write inference which model is best/optimized.

```
from sklearn.ensemble import AdaBoostClassifier
```

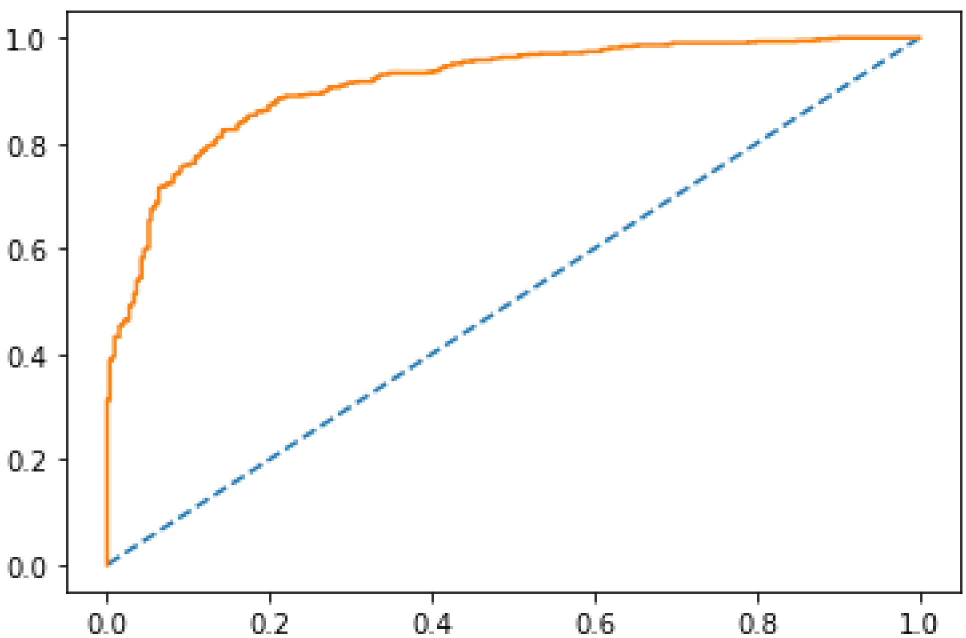
```
ADB_model=AdaBoostClassifier(n_estimators=100,random_st
ADB_model.fit(X_train,Y_train)
```

```
AdaBoostClassifier(n_estimators=100, random_state=1)
```

```
y_train_predict=ADB_model.predict(X_train)
ADB_model_score=ADB_model.score(X_train,Y_train)
print(ADB_model_score)

print(metrics.confusion_matrix(Y_train,y_train_predict)
print(metrics.classification_report(Y_train,y_train_pre
```

```
0.8472352389878163
[[238  94]
 [ 69 666]]
              precision    recall  f1-score   support

           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
```

AUC: 0.913



Gradient Boosting

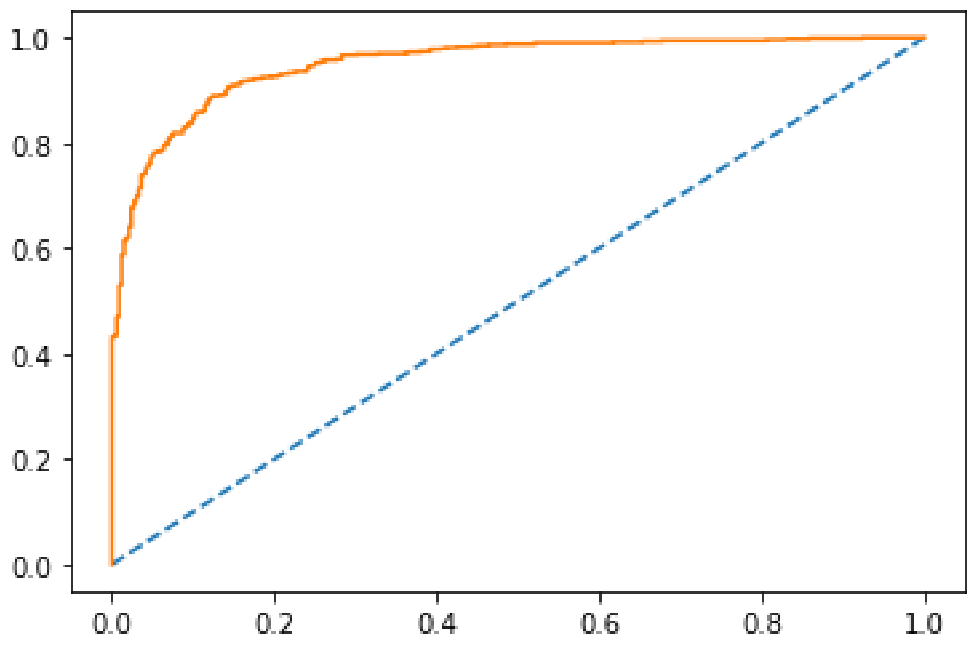
```
1 from sklearn.ensemble import GradientBoostingClassifier
2 gbc_model=GradientBoostingClassifier(random_state=1)
3 gbc_model.fit(X_train, Y_train)
```

```
GradientBoostingClassifier(random_state=1)
```

```
1 y_train_predict = gbc_model.predict(X_train)
2 gbc_model_score = gbc_model.score(X_train, Y_train)
3 print(gbc_model_score)
4 print(metrics.confusion_matrix(Y_train, y_train_predict))
5 print(metrics.classification_report(Y_train, y_train_predi
```

```
0.8865979381443299
[[240  92]
 [ 86 649]]
              precision    recall  f1-score   support

           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
```

AUC _ROC Curve Boosting Train



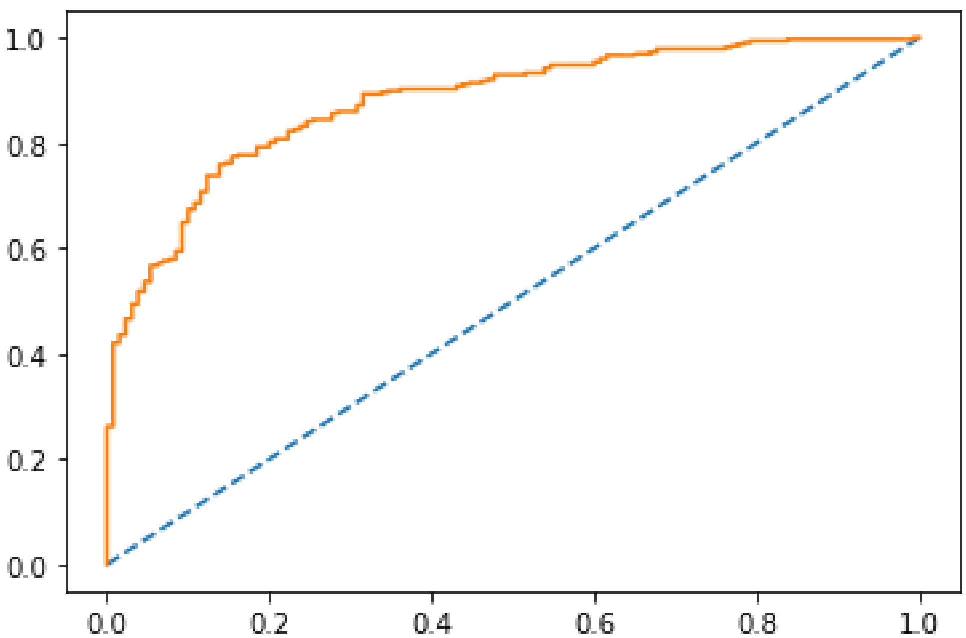
ADA Boosting Test

```
y_test_predict = ADB_model.predict(X_test)
ADB_model_score = ADB_model.score(X_test, Y_test)
print(ADB_model_score)
print(metrics.confusion_matrix(Y_test, Y_test_predict))
print(metrics.classification_report(Y_test, Y_test_predict
```

```
0.8187772925764192
[[ 94  36]
 [ 44 284]]
              precision    recall  f1-score   support

           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
```

AUC _ROC Curve Boosting Test

```
AUC: 0.879
```



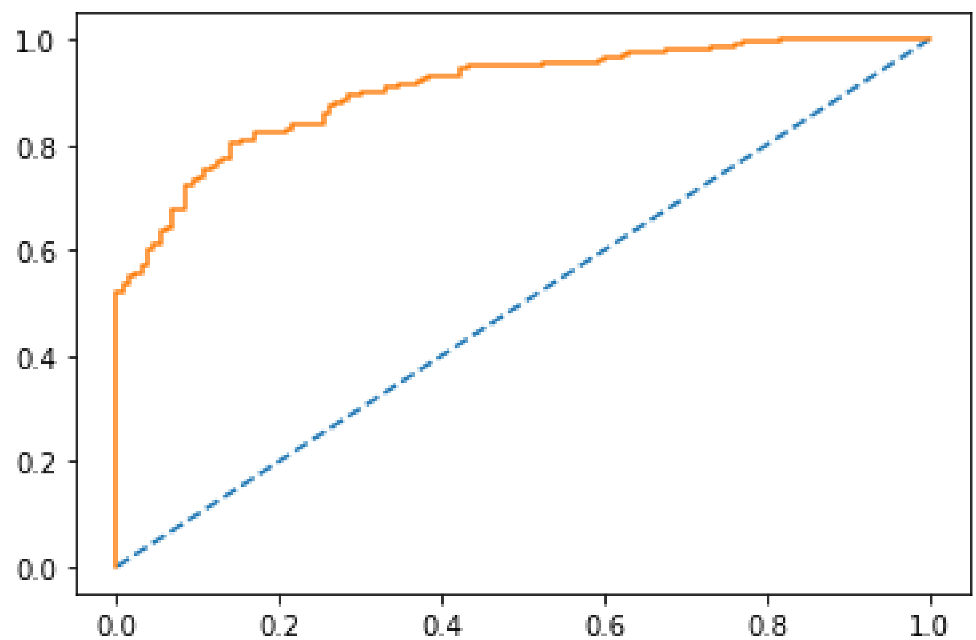
Gradient Boosting Test

```python
y_test_predict = gbc_model.predict(X_test)
gbc_model_score = gbc_model.score(X_test, Y_test)
print(gbc_model_score)
print(metrics.confusion_matrix(Y_test, Y_test_predict))
print(metrics.classification_report(Y_test, Y_test_predi
```

```
0.8318777292576419
[[ 94  36]
 [ 44 284]]
              precision    recall  f1-score   support

           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
```

Gradient Boosting AUC_ROC Curve Test

```
AUC: 0.904
```



Problem 2:

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

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

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

```
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')
```

```
[nltk_data] Downloading package inaugural to
[nltk_data]     C:\Users\GHOST\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\inaugural.zip.
```

```
'Mr. Vice President, Mr. Speaker, Mr. Chief Justice, Senator Cook, Mrs. Eiser
good country we share together:\n\nWhen we met here four years ago, America w
f seemingly endless war abroad and of destructive conflict at home.\n\nAs we
```

```python
import nltk
```

```python
nltk.download('inaugural')
```

```
[nltk_data] Downloading package inaugural to
[nltk_data]     C:\Users\GHOST\AppData\Roaming\nltk_data...
[nltk_data]   Package inaugural is already up-to-date!
```

```
True
```

```python
from nltk.corpus import inaugural
```

```python
inaugural.fileids()
```

```
['1789-Washington.txt',
 '1793-Washington.txt',
 '1797-Adams.txt',
 '1801-Jefferson.txt',
 '1805-Jefferson.txt',
 '1809-Madison.txt',
 '1813-Madison.txt',
 '1817-Monroe.txt',
 '1821-Monroe.txt',
 '1825-Adams.txt',
 '1829-Jackson.txt',
 '1833-Jackson.txt',
 '1837-VanBuren.txt',
 '1841-Harrison.txt',
 '1845-Polk.txt',
 '1849-Taylor.txt',
 '1853-Pierce.txt',
 '1857-Buchanan.txt',
 '1861-Lincoln.txt',
 '1865-Lincoln.txt',
 '1869-Grant.txt',
 '1873-Grant.txt',
 '1877-Hayes.txt',
```

We have imported inaugural module of nltk.corpus library to get speeches of the Presidents of the United States of America.
After that we have used inaugural modules raw method to extract the speech of President Franklin D. Roosevelt, President John F. Kennedy and President Richard Nixon as below

We have used python's len function on each row speech to identify Number of characters in speech

```
Roosevelt = inaugural.raw('1941-Roosevelt.txt')
Kennedy= inaugural.raw('1961-Kennedy.txt')
Nixon =inaugural.raw('1973-Nixon.txt')
```

After importing the text file, we would first count the total number of characters in each file separately.

```
number_of_characters = len(Roosevelt)
print('Number of Character in Roosevelt file:',number_of_characters)
number_of_characters = len(Kennedy)
print('number of characters in Kennedy file:', number_of_characters)
number_of_characters =len(Nixon)
print('Number of characters in Nixon file :',number_of_characters)
```

```
Number of Character in Roosevelt file: 7571
number of characters in Kennedy file: 7618
Number of characters in Nixon file : 9991
```

Number of words in each text file: Below we are counting the total number of words from each file separately. Here we are using the split() to split up the words based on space between each word and we are counting the total number of words by using the len() function.

number of words in Kennedy

```
x = inaugural.raw('1961-Kennedy.txt')
words = x.split()
print('Number of words in Kennedy file:', len(words))
```

```
Number of words in Kennedy file: 1390
```

number of words in Nixon

```
x=inaugural.raw('1973-Nixon.txt')
words = x.split()
print('Number of words in Nixon file: ', len(words))
```

```
Number of words in Nixon file:  1819
```

number of words in Roosevelt

```
x=inaugural.raw('1941-Roosevelt.txt')
words = x.split()
print('Number of words in Roosevelt file: ', len(words))
```

```
Number of words in Roosevelt file:  1360
```

Number of Sentences.

Below we are counting the total number of sentence in each text file, by using lambda function. We are using pd.Dataframe to move the data as dictionary and then with lambda function we are checking each sentece which ends with "." Using endswith() function and the below code and output is as below

```
#number of sentence  in Nixon

y = pd.DataFrame({'Text':inaugural.raw('1973-Nixon.txt')}, index = [0])
y['sentences']= y['Text'].apply(lambda x:len([x for x in x.split()if x.endswith('.')]))
y
```

	Text	sentences
0	Mr. Vice President, Mr. Speaker, Mr. Chief Jus...	68

```python
y = pd.DataFrame({'Text':inaugural.raw('1961-Kennedy.txt')}, index = [0])
y['sentences']= y['Text'].apply(lambda x:len([x for x in x.split()if x.endswith('.')]))
y
```

	Text	sentences
0	Vice President Johnson, Mr. Speaker, Mr. Chief...	52

```python
y = pd.DataFrame({'Text':inaugural.raw('1941-Roosevelt.txt')}, index = [0])
y['sentences']= y['Text'].apply(lambda x:len([x for x in x.split()if x.endswith('.')]))
y
```

	Text	sentences
0	On each national day of inauguration since 178...	67

2.2 Remove all the stopwords from all three speeches.

We would use the library from nltk.corpus import stopwords from nltk.tokenize import word_tokenize. We need these to remove all the English predefined words from each text file separately and with the help of tokenize we would separate each word and remove all the words from the text file. 3 |

```python
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
nltk.download('stopwords')
nltk.download('punkt')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\GHOST\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\GHOST\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping tokenizers\punkt.zip.
```

```
True
```

```python
stop_words = set(stopwords.words('english'))
word_tokens = word_tokenize(Roosevelt)
filtered_sentence = [w for w in word_tokens if not w in stop_words]
filtered_sentence =[]
for w in word_tokens:
    if w not in stop_words:
        filtered_sentence.append(w)
print(word_tokens)
print(filtered_sentence)

```

```
['On', 'each', 'national', 'day', 'of', 'inauguration', 'since', '1789', ',', 'the', 'people', 'have', 'renewed', 'their',
'sense', 'of', 'dedication', 'to', 'the', 'United', 'States', '.', 'In', 'Washington', "'s", 'day', 'the', 'task', 'of', 'th
e', 'people', 'was', 'to', 'create', 'and', 'weld', 'together', 'a', 'nation', '.', 'In', 'Lincoln', "'s", 'day', 'the', 'ta
sk', 'of', 'the', 'people', 'was', 'to', 'preserve', 'that', 'Nation', 'from', 'disruption', 'from', 'within', '.', 'In', 't
his', 'day', 'the', 'task', 'of', 'the', 'people', 'is', 'to', 'save', 'that', 'Nation', 'and', 'its', 'institutions', 'fro
m', 'disruption', 'from', 'without', '.', 'To', 'us', 'there', 'has', 'come', 'a', 'time', ',', 'in', 'the', 'midst', 'of',
```

Remove all the stopwords from the Kennedy speeches.

```
stop_words = set(stopwords.words('english'))
word_tokens = word_tokenize(Kennedy)
filtered_sentence = [w for w in word_tokens if not w in stop_words]
filtered_sentence =[]
for w in word_tokens:
    if w not in stop_words:
        filtered_sentence.append(w)
print(word_tokens)
print(filtered_sentence)
```

```
['Vice', 'President', 'Johnson', ',', 'Mr.', 'Speaker', ',', 'Mr.', 'Chief', 'Justice', ',', 'President', 'Eisenhower', ',',
'Vice', 'President', 'Nixon', ',', 'President', 'Truman', ',', 'reverend', 'clergy', ',', 'fellow', 'citizens', ',', 'we',
'observe', 'today', 'not', 'a', 'victory', 'of', 'party', ',', 'but', 'a', 'celebration', 'of', 'freedom', '--', 'symbolizin
g', 'an', 'end', ',', 'as', 'well', 'as', 'a', 'beginning', '--', 'signifying', 'renewal', ',', 'as', 'well', 'as', 'chang
e', '.', 'For', 'I', 'have', 'sworn', 'I', 'before', 'you', 'and', 'Almighty', 'God', 'the', 'same', 'solemn', 'oath', 'ou
r', 'forebears', 'l', 'prescribed', 'nearly', 'a', 'century', 'and', 'three', 'quarters', 'ago', '.', 'The', 'world', 'is',
'very', 'different', 'now', '.', 'For', 'man', 'holds', 'in', 'his', 'mortal', 'hands', 'the', 'power', 'to', 'abolish', 'al
l', 'forms', 'of', 'human', 'poverty', 'and', 'all', 'forms', 'of', 'human', 'life', '.', 'And', 'yet', 'the', 'same', 'revo
lutionary', 'beliefs', 'for', 'which', 'our', 'forebears', 'fought', 'are', 'still', 'at', 'issue', 'around', 'the', 'glob
e', '--', 'the', 'belief', 'that', 'the', 'rights', 'of', 'man', 'come', 'not', 'from', 'the', 'generosity', 'of', 'the', 's
tate', ',', 'but', 'from', 'the', 'hand', 'of', 'God', '.', 'We', 'dare', 'not', 'forget', 'today', 'that', 'we', 'are', 'th
e', 'heirs', 'of', 'that', 'first', 'revolution', '.', 'Let', 'the', 'word', 'go', 'forth', 'from', 'this', 'time', 'and',
'place', ',', 'to', 'friend', 'and', 'foe', 'alike', ',', 'that', 'the', 'torch', 'has', 'been', 'passed', 'to', 'a', 'new',
'generation', 'of', 'Americans', '--', 'born', 'in', 'this', 'century', ',', 'tempered', 'by', 'war', ',', 'disciplined', 'b
y', 'a', 'hard', 'and', 'bitter', 'peace', ',', 'proud', 'of', 'our', 'ancient', 'heritage', '--', 'and', 'unwilling', 'to',
'witness', 'or', 'permit', 'the', 'slow', 'undoing', 'of', 'those', 'human', 'rights', 'to', 'which', 'this', 'Nation', 'ha
s', 'always', 'been', 'committed', ',', 'and', 'to', 'which', 'we', 'are', 'committed', 'today', 'at', 'home', 'and', 'aroun
d', 'the', 'world', '.', 'Let', 'every', 'nation', 'know', ',', 'whether', 'it', 'wishes', 'us', 'well', 'or', 'ill', ',',
'that', 'we', 'shall', 'pay', 'any', 'price', ',', 'bear', 'any', 'burden', ',', 'meet', 'any', 'hardship', ',', 'support',
```

Remove all the stopwords from the Nixon speeches

```
stop_words = set(stopwords.words('english'))
word_tokens = word_tokenize(Nixon)
filtered_sentence = [w for w in word_tokens if not w in stop_words]
filtered_sentence =[]
for w in word_tokens:
    if w not in stop_words:
        filtered_sentence.append(w)
print(word_tokens)
print(filtered_sentence)
```

```
['Mr.', 'Vice', 'President', ',', 'Mr.', 'Speaker', ',', 'Mr.', 'Chief', 'Justice', ',', 'Senator', 'Cook', ',', 'Mrs.', 'Ei
senhower', ',', 'and', 'my', 'fellow', 'citizens', 'of', 'this', 'great', 'and', 'good', 'country', 'we', 'share', 'togethe
r', ':', 'When', 'we', 'met', 'here', 'four', 'years', 'ago', ',', 'America', 'was', 'bleak', 'in', 'spirit', ',', 'depresse
d', 'by', 'the', 'prospect', 'of', 'seemingly', 'endless', 'war', 'abroad', 'and', 'of', 'destructive', 'conflict', 'at', 'h
ome', '.', 'As', 'we', 'meet', 'here', 'today', ',', 'we', 'stand', 'on', 'the', 'threshold', 'of', 'a', 'new', 'era', 'of',
'peace', 'in', 'the', 'world', '.', 'The', 'central', 'question', 'before', 'us', 'is', ':', 'How', 'shall', 'we', 'use', 't
hat', 'peace', '?', 'Let', 'us', 'resolve', 'that', 'this', 'era', 'we', 'are', 'about', 'to', 'enter', 'will', 'not', 'be',
'what', 'other', 'postwar', 'periods', 'have', 'so', 'often', 'been', ':', 'a', 'time', 'of', 'retreat', 'and', 'isolation',
'that', 'leads', 'to', 'stagnation', 'at', 'home', 'and', 'invites', 'new', 'danger', 'abroad', '.', 'Let', 'us', 'resolve',
'that', 'this', 'will', 'be', 'what', 'it', 'can', 'become', ':', 'a', 'time', 'of', 'great', 'responsibilities', 'greatly',
'borne', ',', 'in', 'which', 'we', 'renew', 'the', 'spirit', 'and', 'the', 'promise', 'of', 'America', 'as', 'we', 'enter',
'our', 'third', 'century', 'as', 'a', 'nation', '.', 'This', 'past', 'year', 'saw', 'far-reaching', 'results', 'from', 'ou
r', 'new', 'policies', 'for', 'peace', '.', 'By', 'continuing', 'to', 'revitalize', 'our', 'traditional', 'friendships',
',', 'and', 'by', 'our', 'missions', 'to', 'Peking', 'and', 'to', 'Moscow', ',', 'we', 'were', 'able', 'to', 'establish', 't
he', 'base', 'for', 'a', 'new', 'and', 'more', 'durable', 'pattern', 'of', 'relationships', 'among', 'the', 'nations', 'of',
'the', 'world', '.', 'Because', 'of', 'America', "'s", 'bold', 'initiatives', ',', '1972', 'will', 'be', 'long', 'remembere
d', 'as', 'the', 'year', 'of', 'the', 'greatest', 'progress', 'since', 'the', 'end', 'of', 'World', 'War', 'II', 'toward',
'a', 'lasting', 'peace', 'in', 'the', 'world', '.', 'The', 'peace', 'we', 'seek', 'in', 'the', 'world', 'is', 'not', 'the',
'flimsy', 'peace', 'which', 'is', 'merely', 'an', 'interlude', 'between', 'wars', ',', 'but', 'a', 'peace', 'which', 'can',
```

2.3 Which word occurs the most number of times in his inaugural address for each president? Mention the top three words. (after removing the stopwords)

We have already removed the stopwatch in previous code using stopwords. Now we are loop to look for any word and count the total number of occurrences. And we see from Roosevelt file we have the below words which are highly used in during the speech by president. Top 3 Words : Nation , Know, Spirit.

For roosevelt

```
[('Nation', 12),
 ('Know', 10),
 ('Spirit', 9),
 ('Life', 9),
 ('Democracy', 9),
 ('Us', 8),
 ('People', 7),
 ('America', 7),
 ('Years', 6),
 ('Freedom', 6),
 ('Human', 5),
 ('New', 5),
 ('Body', 5),
 ('Mind', 5),
 ('Speaks', 5),
 ('Day', 4),
 ('States', 4),
 ('Government', 4),
 ('Must', 4),
```

For Kennedy

In [162]:

```python
from nltk.corpus import stopwords
from nltk.tokenize import RegexpTokenizer
from collections import Counter

tokenizer =RegexpTokenizer(r'\w+')
Kennedy_no_punc = tokenizer.tokenize(Roosevelt)
set(w.title() for w in roosevelt_no_punc if w.lower() not in stopwords.words())
word_count_dict = Counter(w.title() for w in Kennedy_no_punc if w.lower() not in stopwords.words())
word_count_dict.most_common()
```

Out[162]:

```
[('Nation', 12),
 ('Know', 10),
 ('Spirit', 9),
 ('Life', 9),
 ('Democracy', 9),
 ('Us', 8),
 ('People', 7),
 ('America', 7),
 ('Years', 6),
 ('Freedom', 6),
 ('Human', 5),
 ('New', 5),
 ('Body', 5),
 ('Mind', 5),
 ('Speaks', 5),
 ('Day', 4),
 ('States', 4),
 ('Government', 4),
 ('Must', 4),
```

For Nixon

```python
from nltk.corpus import stopwords
from nltk.tokenize import RegexpTokenizer
from collections import Counter

tokenizer =RegexpTokenizer(r'\w+')
Nixon_no_punc = tokenizer.tokenize(Roosevelt)
set(w.title() for w in roosevelt_no_punc if w.lower() not in stopwords.words())
word_count_dict = Counter(w.title() for w in Nixon_no_punc if w.lower() not in stopwords.words())
word_count_dict.most_common()
```

```
[('Nation', 12),
 ('Know', 10),
 ('Spirit', 9),
 ('Life', 9),
 ('Democracy', 9),
 ('Us', 8),
 ('People', 7),
 ('America', 7),
 ('Years', 6),
 ('Freedom', 6),
 ('Human', 5),
 ('New', 5),
 ('Body', 5),
 ('Mind', 5),
 ('Speaks', 5),
```