

## Problem 1:

**You are hired by one of the leading news channel 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.**

In the given dataset, we have 9 variables including the target variable “Vote”, age being the only numerical variable and all other variables being categorical in nature. So we have to consider 1525 entries and build a model that would help in better predictions on the exit polls and uncover the information that which party would win.

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

We now read the excel sheet using the Pandas library and get the following output: -

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

The Unnamed:0 column makes no sense, so we drop it.

	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

We check the number of null entries using `isna().sum()` and get the following output: -

Unnamed: 0	0
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	

Hence, we don't have any missing records in this dataframe.

```
no. of rows: 1525
no. of columns: 9
```

The shape of dataset is given using the shape feature.

Info() is used to get the following output in terms of datatype and missing entries

```
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
```

The describe () is used for the unique values and 5 number summary of the dataset, we have done it twice- for the categorical and the numeric data

	count	unique	top	freq
<b>vote</b>	1525	2	Labour	1063
<b>economic.cond.national</b>	1525	5	3	607
<b>economic.cond.household</b>	1525	5	3	648
<b>Blair</b>	1525	5	4	836
<b>Hague</b>	1525	5	2	624
<b>Europe</b>	1525	11	11	338
<b>political.knowledge</b>	1525	4	2	782
<b>gender</b>	1525	2	female	812

	count	mean	std	min	25%	50%	75%	max
<b>age</b>	1525.0	54.182295	15.711209	24.0	41.0	53.0	67.0	93.0

Now, we can also check for the frequency of different entries in a column using value\_counts().sort\_values(). We are checking them only for the columns which have datatypes as objects. Sort\_values will present the output in an ascending order. Following is the output: -

```

ECONOMIC.COND.NATIONAL : 5
1      37
5      82
2     257
4     542
3     607
Name: economic.cond.national, dtype: int64

ECONOMIC.COND.HOUSEHOLD : 5
1      65
5      92
2     280
4     440
3     648
Name: economic.cond.household, dtype: int64

BLAIR : 5
3      1
1      97
5     153
2     438
4     836
Name: Blair, dtype: int64

HAGUE : 5
3      37
5      73
1     233
4     558
2     624
Name: Hague, dtype: int64

EUROPE : 11
2      79
7      86
10     101
1     109
9     111
8     112
5     124
4     127
3     129
6     209
11     338
Name: Europe, dtype: int64

POLITICAL.KNOWLEDGE : 4
1      38
3     250
0     455
2     782
Name: political.knowledge, dtype: int64

GENDER : 2
male     713
female   812
Name: gender, dtype: int64

```

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

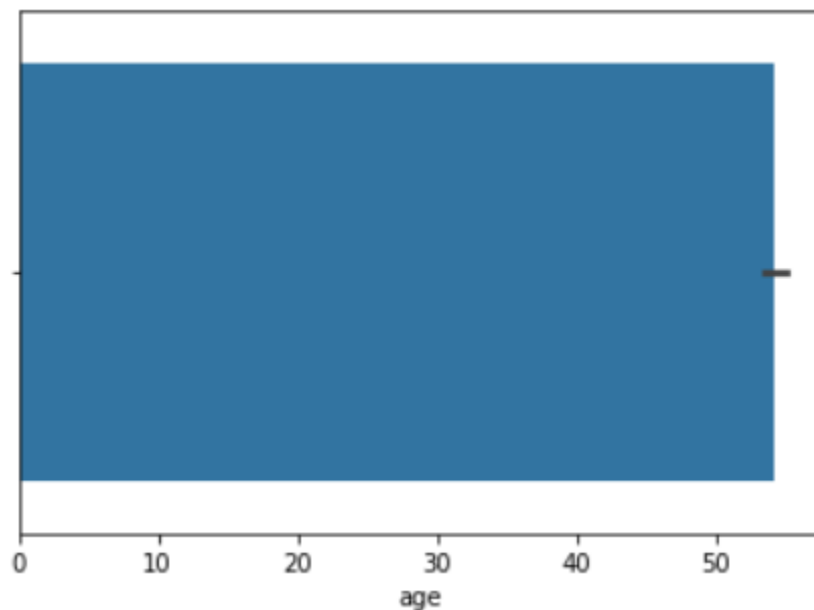
We do different visualizations for our Univariate and Bivariate Analysis and also check for number of duplicates and outliers. Outliers are extreme values in a dataset that should be treated to ensure the consistency of data. Since our data has only one numerical variable, our visual analysis would be limited around it.

To check the duplicates, we use duplicated() and we found that there are 8 duplicates in our data. We'll remove them from the dataset as it won't impact our model.

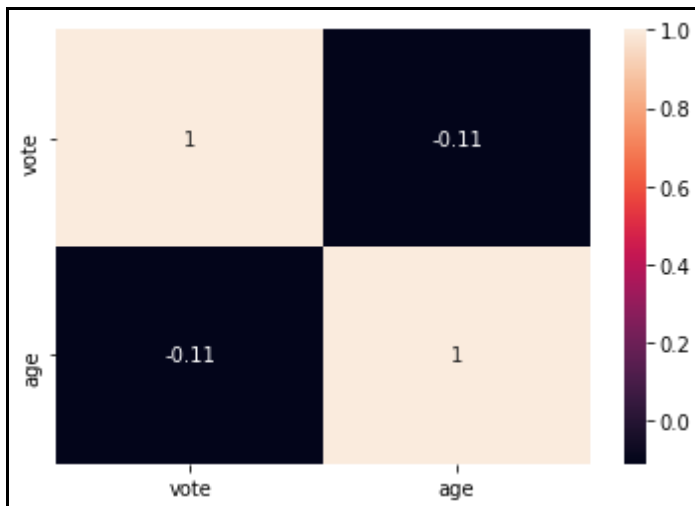
Number of duplicate rows = 8

	vote	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	gender
67	1	35	4	4	5	2	3	2	male
626	1	39	3	4	4	2	5	2	male
870	1	38	2	4	2	2	4	3	male
983	0	74	4	3	2	4	8	2	female
1154	0	53	3	4	2	2	6	0	female
1236	1	36	3	3	2	2	6	2	female
1244	1	29	4	4	4	2	2	2	female
1438	1	40	4	3	4	2	2	2	male

For EDA, we have used various plots like pairplot, heatmap showing correlations, pairplot, stripplot, jointplot, etc for the Univariate and Bivariate Analysis. Following are the snapshots for the same: -

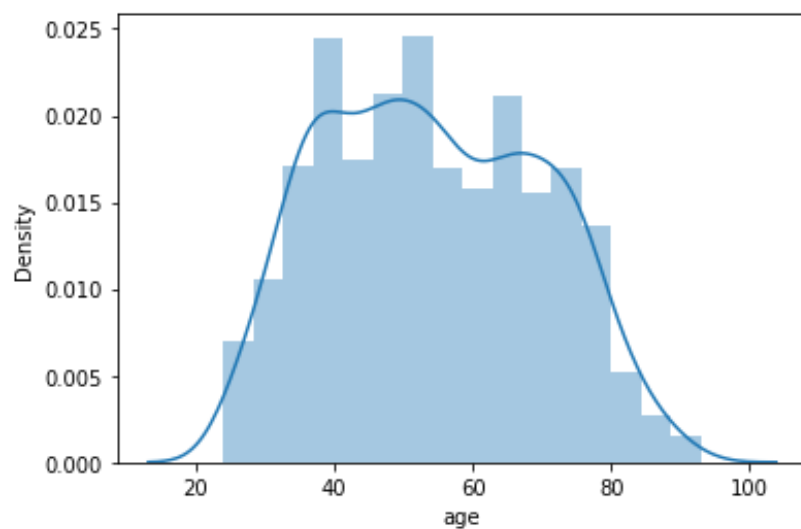
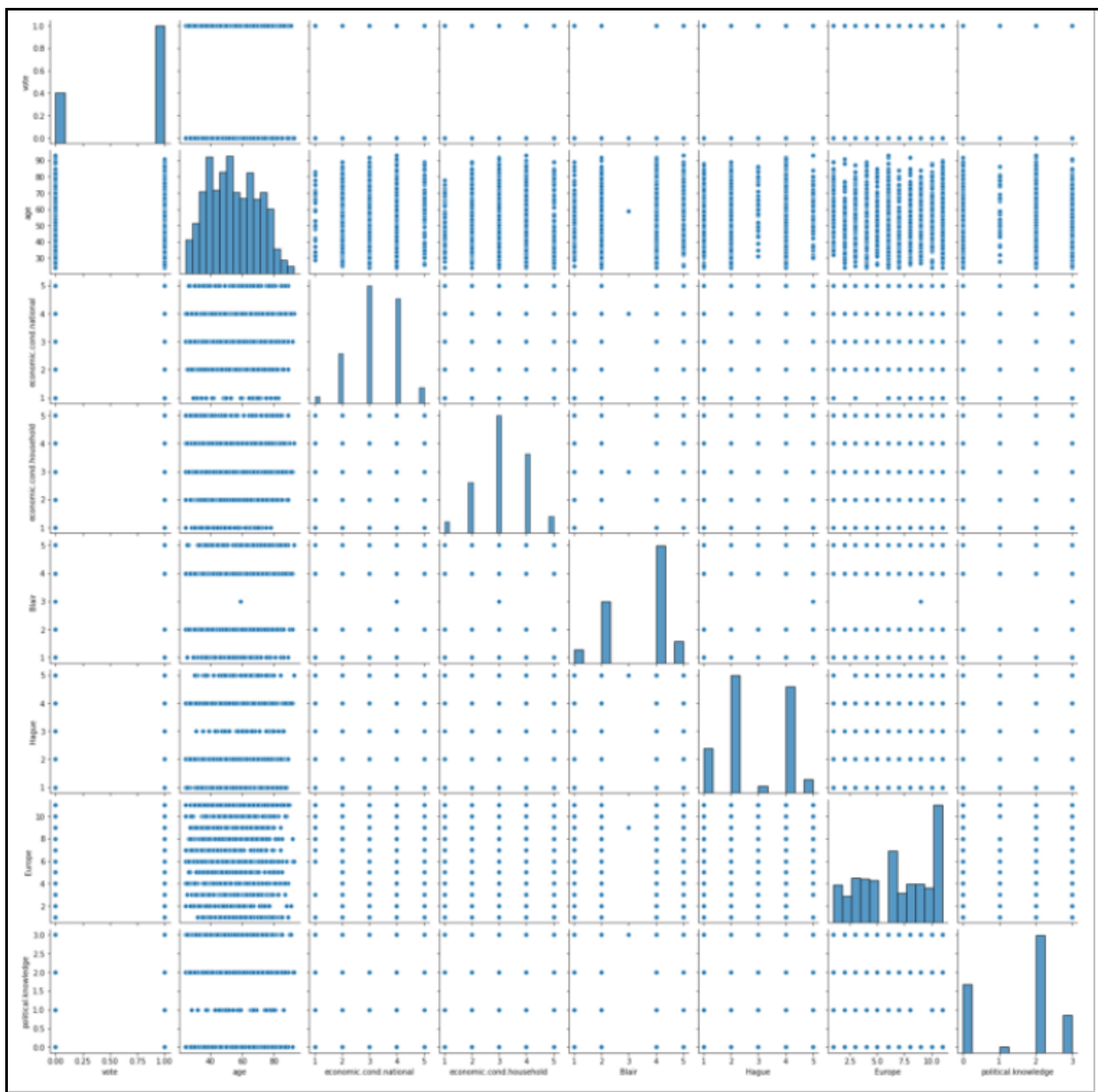


*Barplot for Age*



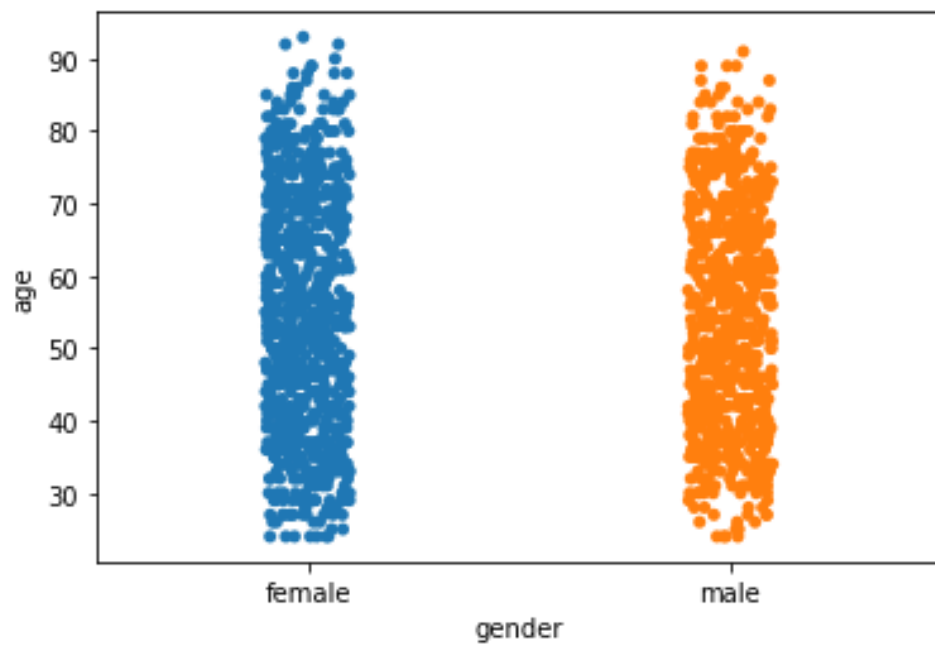
*Heatmap showing correlation*

We can see that there is no high correlation among the numeric variables



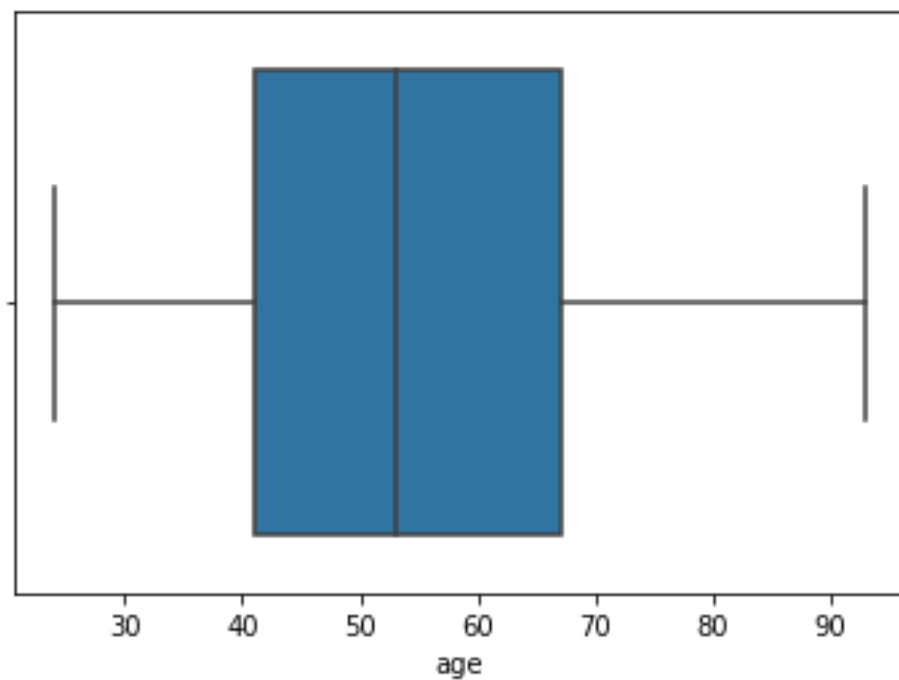
*Distribution Plot for Age*

*The variables isn't normally distributed but majority of population is youth.*



*Strip Plot for Gender vs Age*

*Strip plot shows an even distribution of males and females across ages.*



*BoxPlot for Age*

This explains the Visualization part. The age ranges between 24 and 93 years.

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).

We have also done one hot encoding on our dataset to split the categorical values and removed the first entry in each case to reduce the issue of multi-collinearity. Get\_dummies() is used for the same.

```
1 df=pd.get_dummies(df, columns=cat,drop_first=True)
2 df.head()
```

	vote	age	economic.cond.national_2	economic.cond.national_3	economic.cond.national_4	economic.cond.national_5	economic.cond.household_2
0	1	43	0	1	0	0	0
1	1	38	0	0	1	0	0
2	1	35	0	0	1	0	0
3	1	24	0	0	1	0	1
4	1	41	1	0	0	0	1

Now we do the scaling for variable Age (Min-Max Scaling) to get all the values in a similar scale range.

$$x_{scaled} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

This will make the model values make more sense. Since there is only one variable that is numeric in nature, scaling isn't mandatory, but a good practice. Hence we scale our data in this case. Now we have our values in the range 0-1 which earlier were till 90.

After this we split out data into Train and Test set in a ratio of 70:30. The Train data is used to build our models and the Test data is used to validate the built models. So, we start with splitting our data in X and Y i.e dependant and target variables ("Vote" in our case).

After that we split it into train and test sets using :-

```
1 from sklearn.model_selection import train_test_split
2 Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, Y, test_size=0.3 , random_state=1)

1 from sklearn import metrics,model_selection
2 from sklearn.metrics import roc_auc_score,roc_curve,classification_report,confusion_matrix,plot_confusion_matrix
```

We also check the distribution of target variable post splitting the data.

```
TrainSet Target variable proportion:-
1  0.71065
0  0.28935
Name: vote, dtype: float64

TestSet Target variable proportion:-
1  0.664474
0  0.335526
Name: vote, dtype: float64
```

1 : Labour Party

0 : Conservative Party

```
Number of rows and columns of the training set for the independent variables: (1061, 31)
Number of rows and columns of the training set for the dependent variable: (1061,)
Number of rows and columns of the test set for the independent variables: (456, 31)
Number of rows and columns of the test set for the dependent variable: (456,)
```

#### 4. Apply Logistic Regression and LDA (linear discriminant analysis).

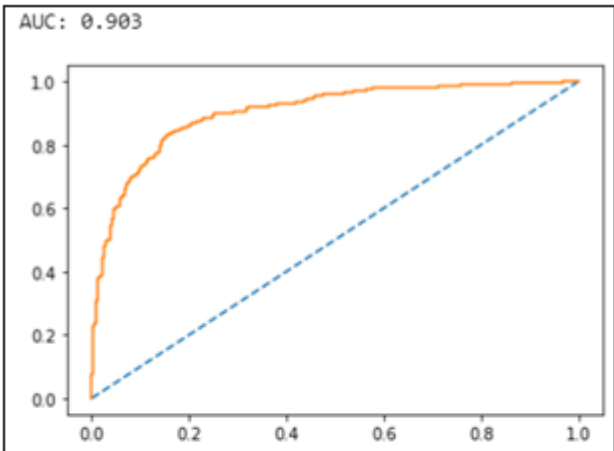
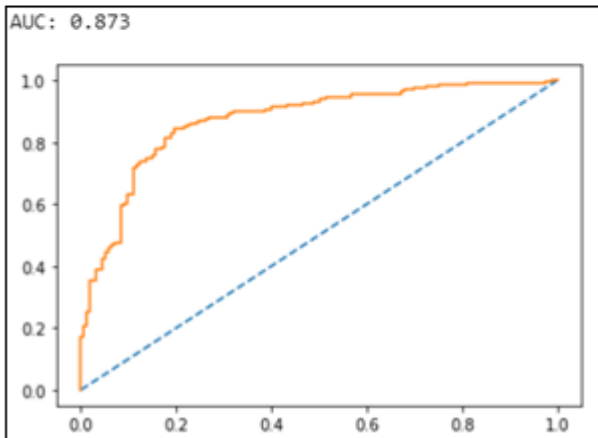
### Logistic Regression

Logistic regression models the probabilities for classification problems with two possible outcomes. It's an extension of the linear regression model for classification problems

We use the following library: -

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
```

On doing Logistic Regression and fitting the model on it, we now check the performance using accuracy, confusion matrix and classification report and get the following result: -

Train And Test Accuracy, Confusion Matrix, Classification Report	AUC and ROC for Train and Test Sets																																																												
<p>Train Set Accuracy:- 0.8473138548539114 Test Set Accuracy:- 0.8245614035087719</p> <p>Confusion Matrix for Train Set:- [[208 99] [ 63 691]]</p> <p>Classification Report for Train Set:-</p> <table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>0</td><td>0.77</td><td>0.68</td><td>0.72</td><td>307</td></tr><tr><td>1</td><td>0.87</td><td>0.92</td><td>0.90</td><td>754</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.85</td><td>1061</td></tr><tr><td>macro avg</td><td>0.82</td><td>0.80</td><td>0.81</td><td>1061</td></tr><tr><td>weighted avg</td><td>0.84</td><td>0.85</td><td>0.84</td><td>1061</td></tr></table> <p>Confusion Matrix for Test Set:- [[104 49] [ 31 272]]</p> <p>Classification Report for Test Set:-</p> <table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>0</td><td>0.77</td><td>0.68</td><td>0.72</td><td>153</td></tr><tr><td>1</td><td>0.85</td><td>0.90</td><td>0.87</td><td>303</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.82</td><td>456</td></tr><tr><td>macro avg</td><td>0.81</td><td>0.79</td><td>0.80</td><td>456</td></tr><tr><td>weighted avg</td><td>0.82</td><td>0.82</td><td>0.82</td><td>456</td></tr></table>		precision	recall	f1-score	support	0	0.77	0.68	0.72	307	1	0.87	0.92	0.90	754	accuracy			0.85	1061	macro avg	0.82	0.80	0.81	1061	weighted avg	0.84	0.85	0.84	1061		precision	recall	f1-score	support	0	0.77	0.68	0.72	153	1	0.85	0.90	0.87	303	accuracy			0.82	456	macro avg	0.81	0.79	0.80	456	weighted avg	0.82	0.82	0.82	456	<p>Train Data</p>  <p>Test Data</p> 
	precision	recall	f1-score	support																																																									
0	0.77	0.68	0.72	307																																																									
1	0.87	0.92	0.90	754																																																									
accuracy			0.85	1061																																																									
macro avg	0.82	0.80	0.81	1061																																																									
weighted avg	0.84	0.85	0.84	1061																																																									
	precision	recall	f1-score	support																																																									
0	0.77	0.68	0.72	153																																																									
1	0.85	0.90	0.87	303																																																									
accuracy			0.82	456																																																									
macro avg	0.81	0.79	0.80	456																																																									
weighted avg	0.82	0.82	0.82	456																																																									

We can see that Logistic Regression performs better on train data.



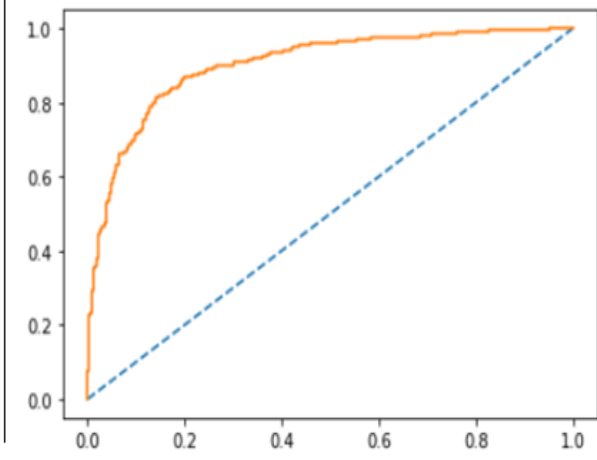
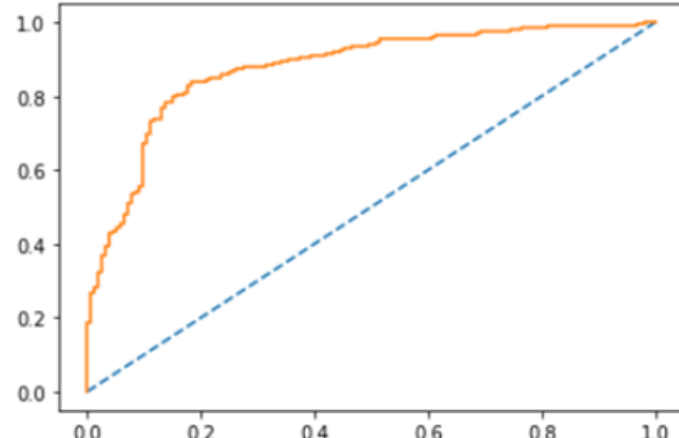
## Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is a dimensionality reduction technique. As the name implies dimensionality reduction techniques reduce the number of dimensions (i.e. variables) in a dataset while retaining as much information as possible.

we use the following library: -

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

On doing LDA and fitting the model on it, we now check the performance using accuracy, confusion matrix and classification report and get the following result: -

Train Set Accuracy, Confusion Matrix, Classification Report and AUC - ROC	Test Set Accuracy, Confusion Matrix, Classification Report and AUC - ROC																																																												
<div>Train Set Accuracy:- 0.8444863336475024</div> <div>Confusion Matrix for Train Set:- [[216 91] [ 74 680]]</div> <div>Classification Report for Train Set:-<table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>0</td><td>0.74</td><td>0.70</td><td>0.72</td><td>307</td></tr><tr><td>1</td><td>0.88</td><td>0.90</td><td>0.89</td><td>754</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.84</td><td>1061</td></tr><tr><td>macro avg</td><td>0.81</td><td>0.80</td><td>0.81</td><td>1061</td></tr><tr><td>weighted avg</td><td>0.84</td><td>0.84</td><td>0.84</td><td>1061</td></tr></table></div> <div>AUC: 0.902</div> <div></div>		precision	recall	f1-score	support	0	0.74	0.70	0.72	307	1	0.88	0.90	0.89	754	accuracy			0.84	1061	macro avg	0.81	0.80	0.81	1061	weighted avg	0.84	0.84	0.84	1061	<div>Test Set Accuracy:- 0.8201754385964912</div> <div>Confusion Matrix for Test Set:- [[107 46] [ 36 267]]</div> <div>Classification Report for Test Set:-<table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>0</td><td>0.75</td><td>0.70</td><td>0.72</td><td>153</td></tr><tr><td>1</td><td>0.85</td><td>0.88</td><td>0.87</td><td>303</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.82</td><td>456</td></tr><tr><td>macro avg</td><td>0.80</td><td>0.79</td><td>0.79</td><td>456</td></tr><tr><td>weighted avg</td><td>0.82</td><td>0.82</td><td>0.82</td><td>456</td></tr></table></div> <div>AUC: 0.877</div> <div></div>		precision	recall	f1-score	support	0	0.75	0.70	0.72	153	1	0.85	0.88	0.87	303	accuracy			0.82	456	macro avg	0.80	0.79	0.79	456	weighted avg	0.82	0.82	0.82	456
	precision	recall	f1-score	support																																																									
0	0.74	0.70	0.72	307																																																									
1	0.88	0.90	0.89	754																																																									
accuracy			0.84	1061																																																									
macro avg	0.81	0.80	0.81	1061																																																									
weighted avg	0.84	0.84	0.84	1061																																																									
	precision	recall	f1-score	support																																																									
0	0.75	0.70	0.72	153																																																									
1	0.85	0.88	0.87	303																																																									
accuracy			0.82	456																																																									
macro avg	0.80	0.79	0.79	456																																																									
weighted avg	0.82	0.82	0.82	456																																																									

LDA does reasonably good on both the sets- train and test. And for selecting the threshold, between 0 and 1, we can see that 0.6 is the better threshold.

0.1

Accuracy Score 0.7785  
F1 Score 0.8636

0.2

Accuracy Score 0.8134  
F1 Score 0.881

0.3

Accuracy Score 0.8351  
F1 Score 0.892

0.4

Accuracy Score 0.8426  
F1 Score 0.8941

0.5

Accuracy Score 0.8445  
F1 Score 0.8918

0.6

Accuracy Score 0.8464  
F1 Score 0.8902

0.7

Accuracy Score 0.8341  
F1 Score 0.8774

0.8

Accuracy Score 0.8011  
F1 Score 0.8465

0.9

Accuracy Score 0.7352  
F1 Score 0.7782

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

KNN Model

The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems.

We use the following library for KNN: -

```
from sklearn.neighbors import KNeighborsClassifier
```

To interpret the results, we use different performance parameters or measures like Accuracy, Confusion Matrix, Classification Report, AUC and ROC.

Train Set Accuracy, Confusion Matrix, Classification Report and AUC - ROC

Train Data Accuracy:- 0.8501413760603205

Confusion Matrix:-  
[[212 95]  
[ 64 690]]

Classification Report:-

	precision	recall	f1-score	support
0	0.77	0.69	0.73	307
1	0.88	0.92	0.90	754
accuracy			0.85	1061
macro avg	0.82	0.80	0.81	1061
weighted avg	0.85	0.85	0.85	1061

AUC: 0.924

Test Set Accuracy, Confusion Matrix, Classification Report and AUC - ROC

Test Data Accuracy:- 0.7828947368421053

Confusion Matrix:-  
[[ 91 62]  
[ 37 266]]

Classification Report:-

	precision	recall	f1-score	support
0	0.71	0.59	0.65	153
1	0.81	0.88	0.84	303
accuracy			0.78	456
macro avg	0.76	0.74	0.75	456
weighted avg	0.78	0.78	0.78	456

AUC: 0.823

## Naïve Bayes

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

We use the following library for Naïve Bayes: -

```
from sklearn.naive_bayes import GaussianNB
```

Performance Matrices are given as :-

### Train Set Accuracy, Confusion Matrix, Classification Report and AUC - ROC

```
Train Data Accuracy:- 0.7492931196983977

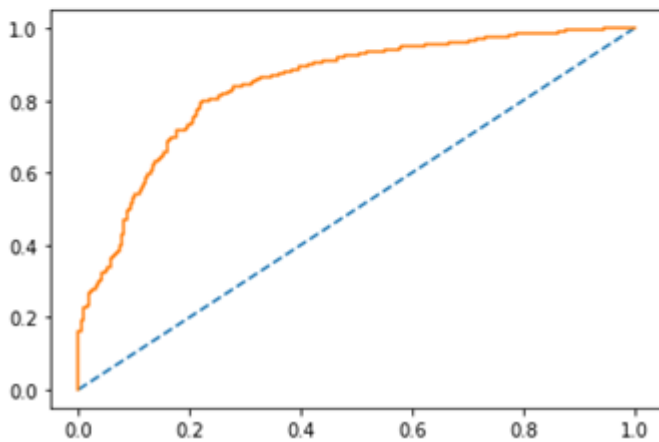
Confusion Matrix:-
[[248  59]
 [207 547]]

Classification Report:-
              precision    recall  f1-score   support

     0       0.55         0.81         0.65         307
     1       0.90         0.73         0.80         754

 accuracy          0.75         1061
 macro avg         0.72         0.77         0.73         1061
 weighted avg      0.80         0.75         0.76         1061
```

AUC: 0.843



### Test Set Accuracy, Confusion Matrix, Classification Report and AUC - ROC

```
Test Data Accuracy:- 0.7346491228070176

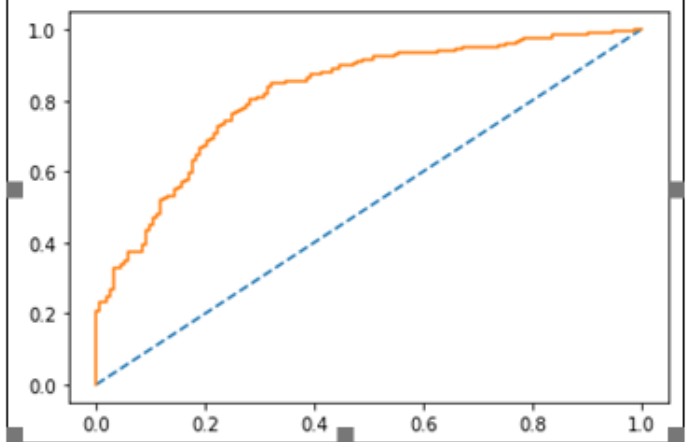
Confusion Matrix:-
[[120  33]
 [ 88 215]]

Classification Report:-
              precision    recall  f1-score   support

     0       0.58         0.78         0.66         153
     1       0.87         0.71         0.78         303

 accuracy          0.73         456
 macro avg         0.72         0.75         0.72         456
 weighted avg      0.77         0.73         0.74         456
```

AUC: 0.819



Here also we can see that the model performs reasonably well on both the sets.

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

Model Tuning is basically enhancing the model using hyper parameters. Newer features are used and feeded to the model and the computation is done on all the features and the best parameters or values are used to build a model and then train on it.

GridSearch and best\_estimators is used for the same.

### Logistic Regression using Model Tuning

The best parameters when model is tuned using Logistic Regression are :-

```
GridSearchCV(cv=3, estimator=LogisticRegression(max_iter=1000, n_jobs=2),
             param_grid={'penalty': ['l2', 'none'],
                          'solver': ['liblinear', 'lbfgs', 'saga'],
                          'tol': [0.0001, 1e-05]},
             scoring='f1')
```

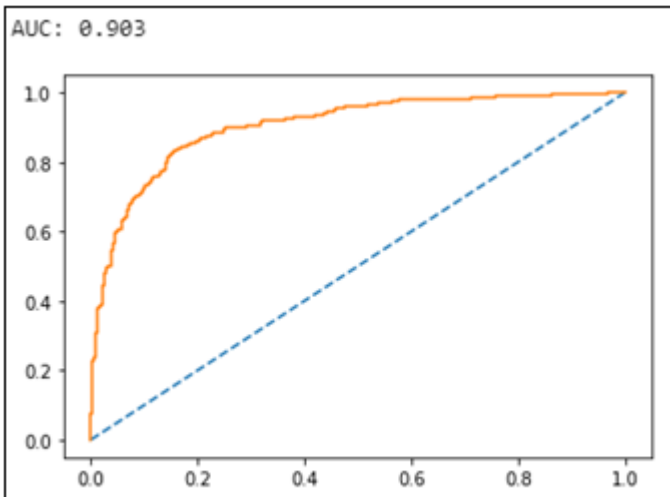
Performance Measures:-

Train Set Accuracy: 0.8473138548539114  
Confusion Matrix:-  
[[208 99]  
 [ 63 691]]

Classification Report:-

	precision	recall	f1-score	support
0	0.77	0.68	0.72	307
1	0.87	0.92	0.90	754
accuracy			0.85	1061
macro avg	0.82	0.80	0.81	1061
weighted avg	0.84	0.85	0.84	1061

AUC: 0.903

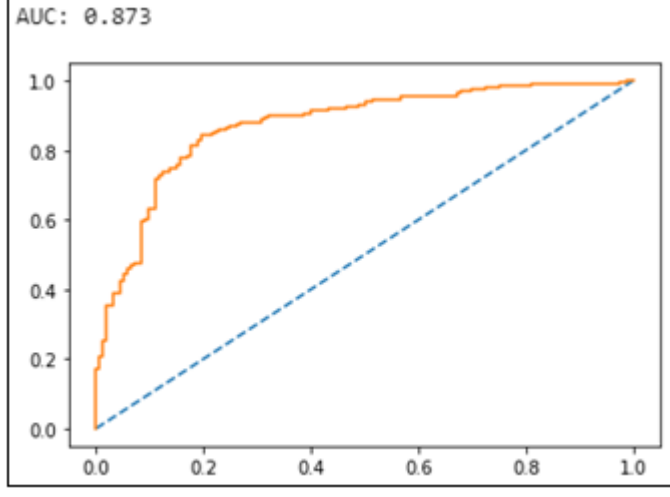


Test Set Accuracy: 0.8245614035087719  
Confusion Matrix:-  
[[104 49]  
 [ 31 272]]

Classification Report:-

	precision	recall	f1-score	support
0	0.77	0.68	0.72	153
1	0.85	0.90	0.87	303
accuracy			0.82	456
macro avg	0.81	0.79	0.80	456
weighted avg	0.82	0.82	0.82	456

AUC: 0.873



When we compare the results with Logistic Regression, we can clearly see how optimised the model has become after Model Tuning.

## Linear Discriminant Analysis using Model Tuning

The best parameters when the model is tuned using LDA are: -

```
Fitting 3 folds for each of 3 candidates, totalling 9 fits

GridSearchCV(cv=3, estimator=LinearDiscriminantAnalysis(), n_jobs=4,
              param_grid={'solver': ['svd'], 'tol': [0.0001, 0.0002, 0.0003]},
              scoring='accuracy', verbose=1)
```

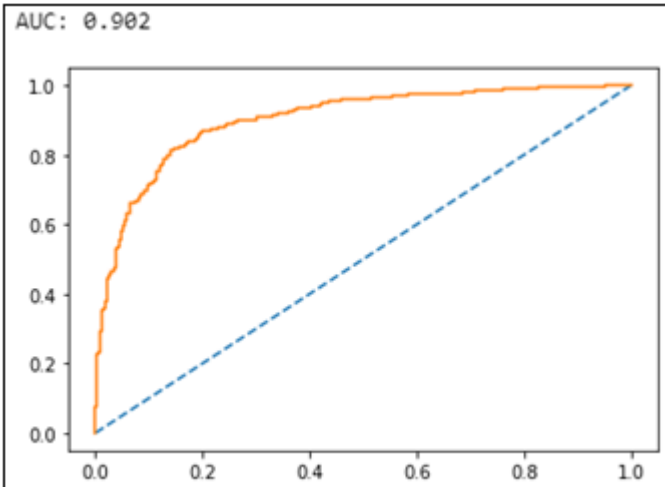
Performance Measures:-

### Train Set Accuracy, Confusion Matrix, Classification Report and AUC – ROC

Train Set Accuracy: 0.8444863336475024  
Confusion Matrix:-  
[[216 91]  
 [ 74 680]]

Classification Report:-

	precision	recall	f1-score	support
0	0.74	0.70	0.72	307
1	0.88	0.90	0.89	754
accuracy			0.84	1061
macro avg	0.81	0.80	0.81	1061
weighted avg	0.84	0.84	0.84	1061

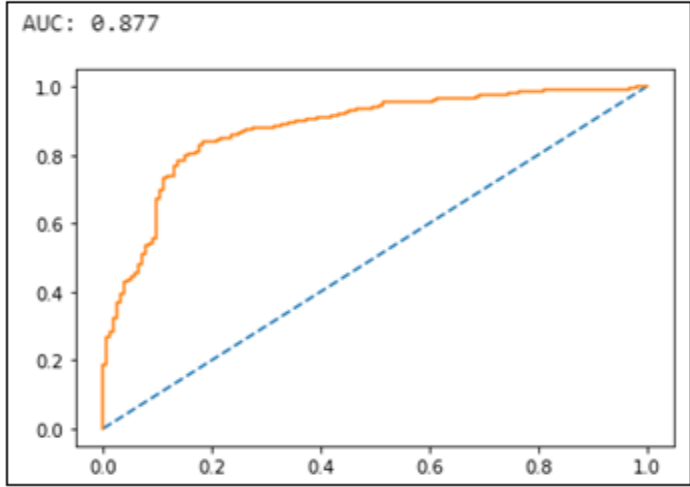


### Test Set Accuracy, Confusion Matrix, Classification Report and AUC - ROC

Test Set Accuracy: 0.8201754385964912  
Confusion Matrix:-  
[[107 46]  
 [ 36 267]]

Classification Report:-

	precision	recall	f1-score	support
0	0.75	0.70	0.72	153
1	0.85	0.88	0.87	303
accuracy			0.82	456
macro avg	0.80	0.79	0.79	456
weighted avg	0.82	0.82	0.82	456



## KNN with Model Tuning

The best parameters when model is tuned using KNN :-

```
GridSearchCV(estimator=KNeighborsClassifier(n_jobs=-1), n_jobs=1,  
            param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],  
                        'leaf_size': [5, 10, 15], 'n_jobs': [-1],  
                        'n_neighbors': [3, 5, 7, 9],  
                        'weights': ['uniform', 'distance']})
```

Performance Measures :-

### Train Set Accuracy, Confusion Matrix, Classification Report and AUC – ROC

Train Set Accuracy: 0.8388312912346843

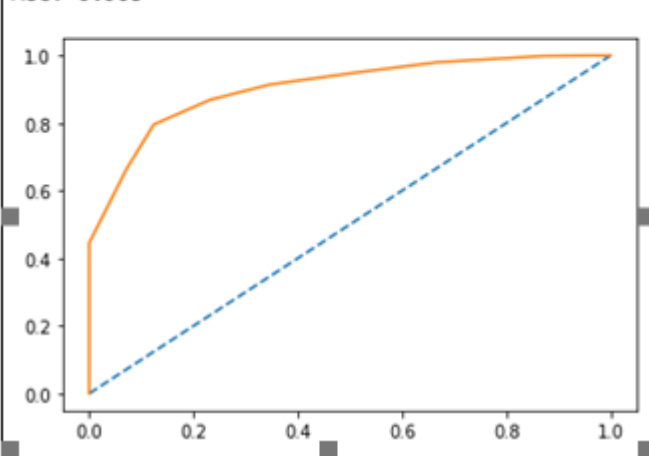
Confusion Matrix:-

```
[[201 106]  
 [ 65 689]]
```

Classification Report:-

	precision	recall	f1-score	support
0	0.76	0.65	0.70	307
1	0.87	0.91	0.89	754
accuracy			0.84	1061
macro avg	0.81	0.78	0.80	1061
weighted avg	0.83	0.84	0.84	1061

AUC: 0.905



### Test Set Accuracy, Confusion Matrix, Classification Report and AUC – ROC

Test Set Accuracy: 0.7675438596491229

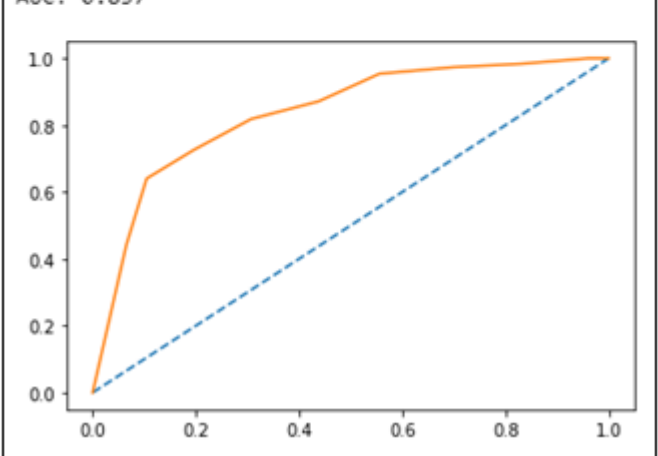
Confusion Matrix:-

```
[[ 86  67]  
 [ 39 264]]
```

Classification Report:-

	precision	recall	f1-score	support
0	0.69	0.56	0.62	153
1	0.80	0.87	0.83	303
accuracy			0.77	456
macro avg	0.74	0.72	0.73	456
weighted avg	0.76	0.77	0.76	456

AUC: 0.837



## Naïve Bayes using Model Tuning

The best parameters when model is tuned using Naïve Bayes: -

```
{'var_smoothing': 0.8111308307896871}  
  
GaussianNB(var_smoothing=0.8111308307896871)  
GridSearchCV(estimator=GaussianNB(), n_jobs=1,
```

Performance Measures: -

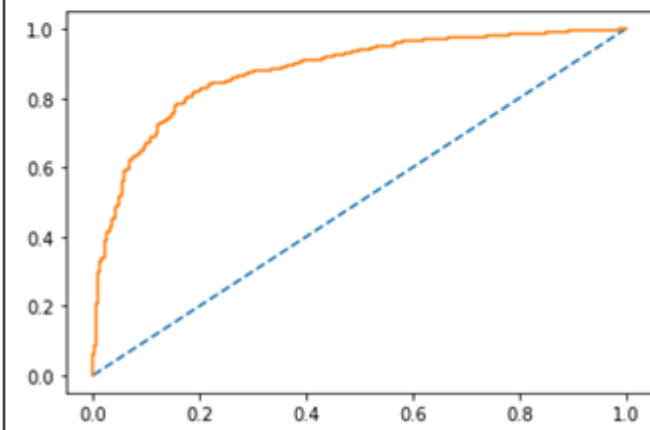
### Train Set Accuracy, Confusion Matrix, Classification Report and AUC – ROC

Train Set Accuracy: 0.819038642789821  
Confusion Matrix:-  
[[200 107]  
 [ 85 669]]

Classification Report:-

	precision	recall	f1-score	support
0	0.70	0.65	0.68	307
1	0.86	0.89	0.87	754
accuracy			0.82	1061
macro avg	0.78	0.77	0.78	1061
weighted avg	0.82	0.82	0.82	1061

AUC: 0.879



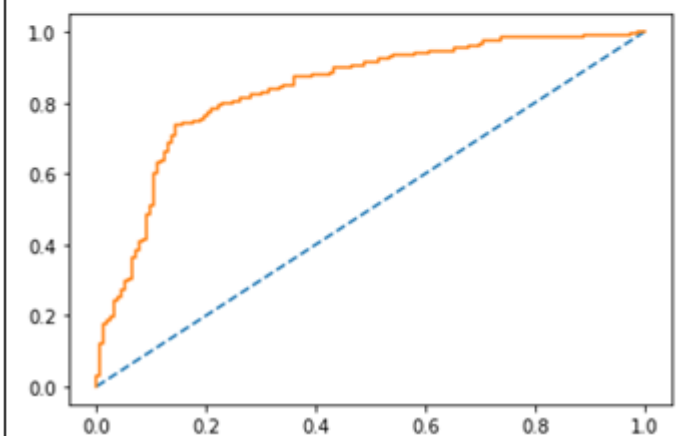
### Test Set Accuracy, Confusion Matrix, Classification Report and AUC – ROC

Test Set Accuracy: 0.7850877192982456  
Confusion Matrix:-  
[[101 52]  
 [ 46 257]]

Classification Report:-

	precision	recall	f1-score	support
0	0.69	0.66	0.67	153
1	0.83	0.85	0.84	303
accuracy			0.79	456
macro avg	0.76	0.75	0.76	456
weighted avg	0.78	0.79	0.78	456

AUC: 0.838





## Bagging and Boosting

**Bagging** is a homogeneous weak learners' model that learns from each other independently in parallel and combines them for determining the model average.

### Example of Bagging:

The Random Forest model uses Bagging, where decision tree models with higher variance are present. It makes random feature selection to grow trees. Several random trees make a Random Forest.

**Boosting** is also a homogeneous weak learners' model but works differently from Bagging. In this model, learners learn sequentially and adaptively to improve model predictions of a learning algorithm.

### Example of Boosting:

The AdaBoost uses Boosting techniques, where a 50% less error is required to maintain the model. Here, Boosting can keep or discard a single learner. Otherwise, the iteration is repeated until achieving a better learner.

### Difference between Bagging and Boosting

- Bagging follows parallel approach and overcomes the problem of overfitting as it only considers a portion from dataset with replacement. It is a simple model. Whereas Boosting is a simple model which is slower as it uses a sequential approach.
- Bagging decreases variance, not bias, and solves over-fitting issues in a model. Boosting decreases bias, not variance.

## Bagging using Random Forest Classifier

```
BaggingClassifier(base_estimator=RandomForestClassifier(), n_estimators=100,  
                  random_state=1)
```

The performance of the model after Bagging using Random Forest Classifier are :-

### Train Set Accuracy, Confusion Matrix, Classification Report and AUC – ROC

Train Data Accuracy:- 0.9679547596606974

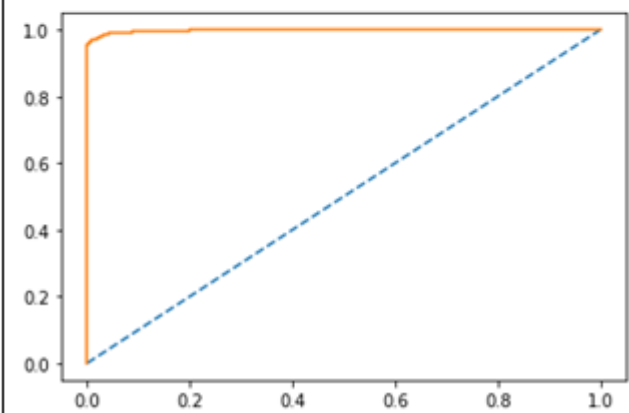
Confusion Matrix:-

```
[[277 30]  
 [ 4 750]]
```

Classification Report:-

	precision	recall	f1-score	support
0	0.99	0.90	0.94	307
1	0.96	0.99	0.98	754
accuracy			0.97	1061
macro avg	0.97	0.95	0.96	1061
weighted avg	0.97	0.97	0.97	1061

AUC: 0.998



### Test Set Accuracy, Confusion Matrix, Classification Report and AUC – ROC

Test Data Accuracy:- 0.8179824561403509

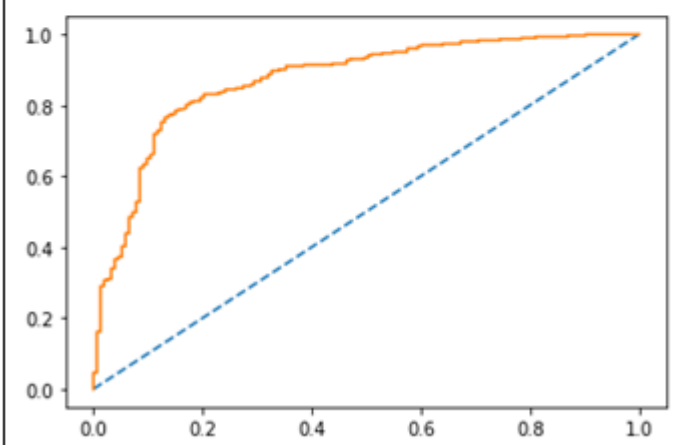
Confusion Matrix:-

```
[[ 97 56]  
 [ 27 276]]
```

Classification Report:-

	precision	recall	f1-score	support
0	0.78	0.63	0.70	153
1	0.83	0.91	0.87	303
accuracy			0.82	456
macro avg	0.81	0.77	0.78	456
weighted avg	0.81	0.82	0.81	456

AUC: 0.873



## Adaptive Boosting

```
1 from sklearn.ensemble import AdaBoostClassifier
2
3 ADaBo = AdaBoostClassifier(n_estimators=100, random_state=1)
4 ADaBo.fit(Xtrain, Ytrain)
```

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

The performance measures are: -

### Train Set Accuracy, Confusion Matrix, Classification Report and AUC – ROC

Train Data Accuracy:- 0.8473138548539114

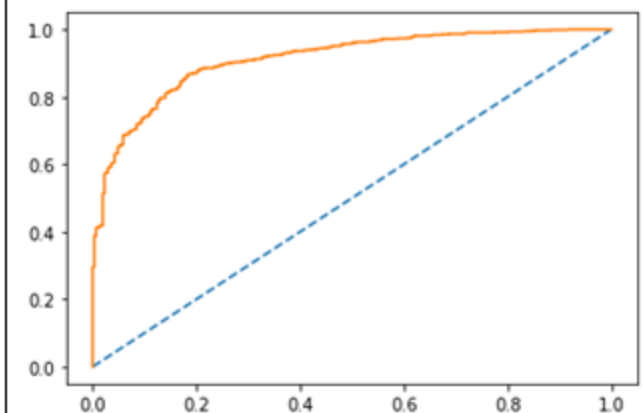
Confusion Matrix:-

```
[[211  96]
 [ 66 688]]
```

Classification Report:-

	precision	recall	f1-score	support
0	0.76	0.69	0.72	307
1	0.88	0.91	0.89	754
accuracy			0.85	1061
macro avg	0.82	0.80	0.81	1061
weighted avg	0.84	0.85	0.84	1061

AUC: 0.912



### Test Set Accuracy, Confusion Matrix, Classification Report and AUC – ROC

Test Data Accuracy:- 0.8135964912280702

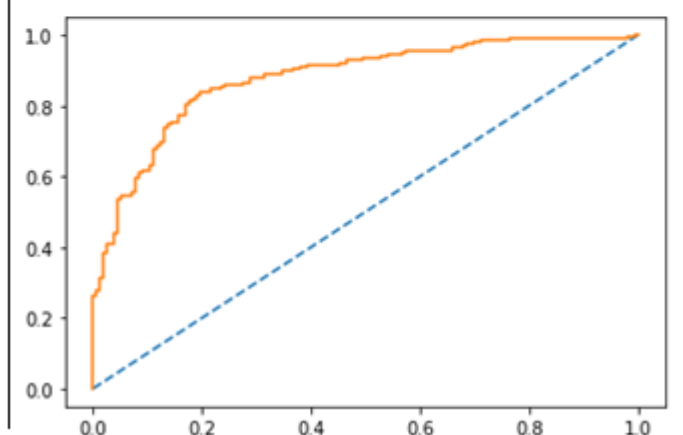
Confusion Matrix:-

```
[[100  53]
 [ 32 271]]
```

Classification Report:-

	precision	recall	f1-score	support
0	0.76	0.65	0.70	153
1	0.84	0.89	0.86	303
accuracy			0.81	456
macro avg	0.80	0.77	0.78	456
weighted avg	0.81	0.81	0.81	456

AUC: 0.878



## Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier
GBo = GradientBoostingClassifier(random_state=1)
GBo = GBo.fit(Xtrain, Ytrain)
```

The performance measures are: -

### Train Set Accuracy, Confusion Matrix, Classification Report and AUC – ROC

Train Data Accuracy 0.884071630537229

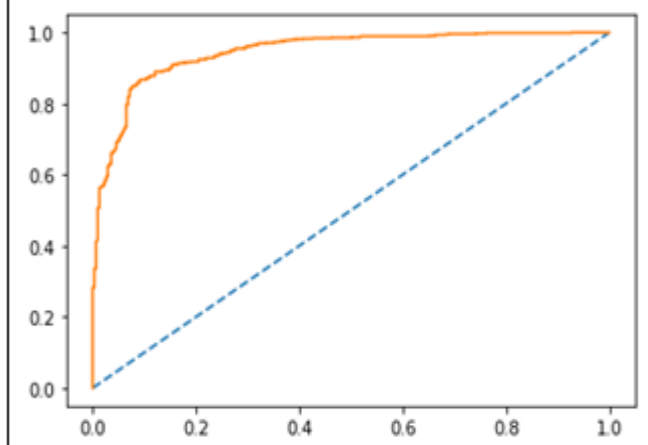
Confusion Matrix:-

```
[[227  80]
 [ 43 711]]
```

Classification Report:-

	precision	recall	f1-score	support
0	0.84	0.74	0.79	307
1	0.90	0.94	0.92	754
accuracy			0.88	1061
macro avg	0.87	0.84	0.85	1061
weighted avg	0.88	0.88	0.88	1061

AUC: 0.945



### Test Set Accuracy, Confusion Matrix, Classification Report and AUC – ROC

Test Data Accuracy 0.8223684210526315

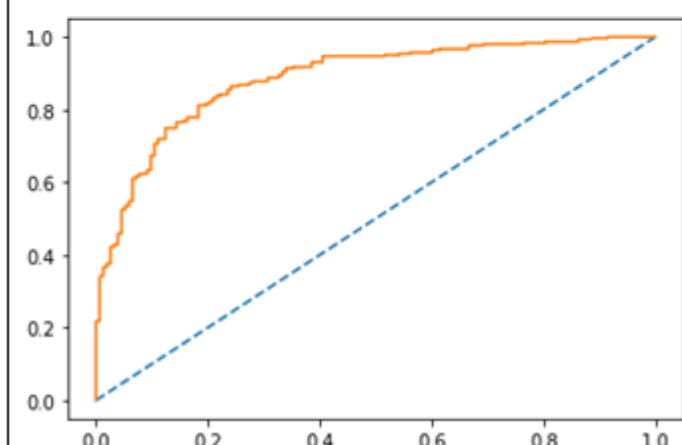
Confusion Matrix:-

```
[[101  52]
 [ 29 274]]
```

Classification Report:-

	precision	recall	f1-score	support
0	0.78	0.66	0.71	153
1	0.84	0.90	0.87	303
accuracy			0.82	456
macro avg	0.81	0.78	0.79	456
weighted avg	0.82	0.82	0.82	456

AUC: 0.886



Other ways of Optimization can be using SMOTE if the models are overfit or imbalanced

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.*

To compare the performance we have summarised in one table :-

	Accuracy	AUC	Recall for 0	Recall for 1	Precision for 0	Precision for 1	F1 Score for 0	F1 Score for 1
LR Train	0.85	0.90	0.68	0.92	0.77	0.87	0.72	0.90
LR Test	0.82	0.87	0.68	0.90	0.77	0.85	0.72	0.87
LR BestParameters Train	0.85	0.90	0.68	0.92	0.77	0.87	0.72	0.90
LR BestParameters Test	0.82	0.87	0.68	0.90	0.77	0.85	0.72	0.87
LDA Train	0.84	0.90	0.70	0.90	0.74	0.88	0.72	0.89
LDA Test	0.82	0.88	0.70	0.88	0.75	0.85	0.72	0.87
LDA BestParameters Train	0.84	0.90	0.70	0.90	0.74	0.88	0.72	0.89
LDAM BestParameters Test	0.82	0.88	0.70	0.88	0.75	0.85	0.72	0.87
AdaBoost Train	0.85	0.91	0.69	0.91	0.76	0.88	0.72	0.89
AdaBoost Test	0.81	0.88	0.65	0.89	0.76	0.84	0.70	0.86
GradientBoost Train	0.88	0.95	0.74	0.94	0.84	0.90	0.79	0.92
GradientBoost Test	0.82	0.89	0.66	0.90	0.78	0.84	0.71	0.87
Decision Tree Train	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Decision Tree Test	0.77	0.73	0.60	0.85	0.68	0.81	0.64	0.83
Random Forest Train	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Random Forest Test	0.80	0.87	0.61	0.90	0.76	0.82	0.67	0.86
Random Forest BestParameters Train	0.90	0.96	0.72	0.97	0.90	0.89	0.80	0.93
Random Forest BestParameters Test	0.81	0.87	0.58	0.92	0.79	0.81	0.67	0.86
Bagging with Decision Tree Train	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Bagging with Decision Tree Test	0.80	0.87	0.67	0.87	0.73	0.84	0.70	0.86
Bagging with Random Forest Train	0.97	1.00	0.90	0.99	0.99	0.96	0.94	0.98
Bagging with Random Forest Test	0.82	0.87	0.63	0.91	0.78	0.83	0.70	0.87
KNN Train	0.85	0.92	0.69	0.92	0.77	0.88	0.73	0.90
KNN Test	0.78	0.82	0.59	0.88	0.71	0.81	0.65	0.84
KNNM BestParameters Train	0.84	0.90	0.65	0.91	0.76	0.87	0.70	0.89
KNNM BestParameters Test	0.77	0.84	0.56	0.87	0.69	0.80	0.62	0.83
KNN with SMOTE Train	0.88	0.97	0.96	0.80	0.83	0.95	0.89	0.87
KNN with SMOTE Test	0.74	0.79	0.76	0.74	0.59	0.86	0.66	0.79
Naive Bayes Train	0.75	0.84	0.81	0.73	0.55	0.90	0.65	0.80
Naive Bayes Test	0.73	0.82	0.78	0.71	0.58	0.87	0.66	0.78
Naive Bayes BestParameters Train	0.82	0.88	0.65	0.89	0.70	0.86	0.68	0.87
Naive Bayes BestParameters Test	0.79	0.84	0.66	0.85	0.69	0.83	0.67	0.84
Naive Bayes with SMOTE Train	0.82	0.90	0.92	0.71	0.76	0.90	0.83	0.80
Naive Bayes with SMOTE Test	0.75	0.84	0.84	0.70	0.59	0.89	0.69	0.79

**Accuracy** : Accuracy is the number of correct predictions over the output size

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{True Positive} + \text{True Negative} + \text{False Positive} + \text{False Negative}}$$

**Precision** : It only measures the rate of false positives

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

**Recall** : Recall is the opposite of precision, it measures false negatives against true positives.

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

**F1 Score**: F1-score as the harmonic mean of precision and recall

$$\text{F1 - Score} = 2 * \frac{\text{Recall} * \text{Precision}}{\text{Recall} + \text{Precision}}$$

In terms of Accuracy, almost all the model have been doing well since they predict it better than 75%, but the Model that is build on Tuning the Hyperparameters of Random Forest does extremely well.

When it comes to Precision and Recall, all the Models that are tuned tend to predict and perform better and similar on Train and Test Sets.

KNN, Naïve Bayes, Logistic Regression, LDA, AdaBoost and models with SMOTE perform similarly on Train and Test Sets whereas Decision Tree, Random Forest aren't as consistent as them

All the model have good F1 score but LDA and Logistic Regression in particular are better on both sets.

8. Based on these predictions, what are the insights?

Based on the analysis, we found that the data predictions for 1 i.e Labour Party is more accurate compared to Conservative Party. This is because of the imbalance in data. And to reduce the problems caused by imbalance of data, we use Bagging and Boosting.

<b>female</b>	<b>812</b>
Conservative	259
Labour	553
<b>male</b>	<b>713</b>
Conservative	203
Labour	510

Most of the males and females support the Labour party.

<b>Conservative</b>	<b>795</b>	<b>33.80%</b>
0	0	0.00%
1	11	1.38%
2	568	71.45%
3	216	27.17%
<b>Labour</b>	<b>1557</b>	<b>66.20%</b>
0	0	0.00%
1	27	1.73%
2	996	63.97%
3	534	34.30%

The proportion of voters with political knowledge on a scale from 0-3 seem to be in commensurate.

Row Labels	Sum of economic.cond.national2	Sum of economic.cond.household
<b>Conservative</b>	<b>26.55%</b>	<b>27.92%</b>
1	1.60%	3.44%
2	21.31%	28.05%
3	45.66%	44.35%
4	28.01%	22.21%
5	3.42%	1.94%
<b>Labour</b>	<b>73.45%</b>	<b>72.08%</b>
1	0.44%	0.90%
2	6.44%	9.21%
3	33.58%	36.76%
4	49.50%	45.19%
5	10.04%	7.94%
<b>Grand Total</b>	<b>100.00%</b>	<b>100.00%</b>

The class with lower and middle class economical conditions seem to prefer Conservative Party, where as the upper middle and upper classes tend to incline more towards the Labour Party.

So, we would recommend the same that we have stated above with numbers in order to make a tight grip or hold over the voter bases. Both the parties have different strengths and target audiences and should make them their potential bases to ensure wins.

## Problem 2:

In this 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

```
import re
import nltk
import string

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

Text Analysis is used in python to analyse the texts, interpret the syntax and semantics , find patterns and make the most out of it. Different libraries like nltk, string and re are used for the same.

In the given task we use 3 text files or speeches given by the former Presidents of United States and analyze them.

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

To find the number of sentences, we use :-

**len(inaugural.sents())**

To find the number of words, we use :-

**len(inaugural.words())**

To find the number of characters, we use :-

**len(inaugural.raw())**

1941-Roosevelt.txt		
Total sentences:- 68	Total words:- 1536	Total characters :- 7571
Total sentences:- 52	Total words:- 1546	Total characters :- 7618
1973-Nixon.txt		
Total sentences:- 69	Total words:- 2028	Total characters :- 9991



2. *Remove all the stopwords from all the three speeches.*

Stopwords are the most occurring words in the language or the other words that are used for the grammatical purposes but don't have meaning to it and they don't affect the meaning or context that changes the dynamics of what is being conveyed.

We use the following code for removing codewords : -

```
nltk.download('stopwords')
stopwords = nltk.corpus.stopwords.words('english') + list(string.punctuation)
from nltk.tokenize import word_tokenize
rv_tokens = word_tokenize(rv)
ken_tokens = word_tokenize(ken)
nix_tokens = word_tokenize(nix)

rv_clean = [word for word in (x.lower() for x in rv_tokens) if word not in stopwords]
ken_clean = [word for word in (x.lower() for x in ken_tokens) if word not in stopwords]
nix_clean = [word for word in (x.lower() for x in nix_tokens) if word not in stopwords]
print(rv_clean)
print('\n\n')
print(ken_clean)
print('\n\n')
print(nix_clean)
```

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 cleaned the text, but it is in a format where all the words are separated, we need to join them first to make them a complete text and then do stemming. Also some of the unwanted characters like – have to be removed.

```
rv_para = ' '.join([str(elem) for elem in rv_clean])
ken_para = ' '.join([str(elem) for elem in ken_clean])
nix_para = ' '.join([str(elem) for elem in nix_clean])
rv_ultra_clean = rv_para.replace('[^\w\s]', '').replace('--', '')
ken_ultra_clean = ken_para.replace('[^\w\s]', '').replace('--', '')
nix_ultra_clean = nix_para.replace('[^\w\s]', '').replace('--', '')
```

```
from nltk.stem import PorterStemmer
st = PorterStemmer()

rv_stem= " ".join([st.stem(word) for word in rv_ultra_clean.split()])
ken_stem= " ".join([st.stem(word) for word in ken_ultra_clean.split()])
nix_stem= " ".join([st.stem(word) for word in nix_ultra_clean.split()])

print(rv_stem)
print('\n\n')
print(ken_stem)
print('\n\n')
print(nix_stem)
```

After completing this, we can now count the frequency of most occurring words in all the speeches using FreqDist and Counter.

Following is the code:-

```
from nltk.probability import FreqDist
from collections import Counter

rv_freq = nltk.FreqDist(rv_stem)
ken_freq = nltk.FreqDist(ken_stem)
nix_freq = nltk.FreqDist(nix_stem)

Count_rv = Counter(rv_stem.split())
Count_ken = Counter(ken_stem.split())
Count_nix = Counter(nix_stem.split())

rv_MOW = Count_rv.most_common(3)
ken_MOW = Count_ken.most_common(3)
nix_MOW = Count_nix.most_common(3)
```

After this we get the output as :-

```
[('nation', 17), ('know', 10), ('peopl', 9)]
[('let', 16), ('us', 12), ('power', 9)]
[('us', 26), ('let', 22), ('america', 21)]
```

In Roosevelt's speech, NATION, KNOW and PEOPLE are most frequent words occurring 17,10 and 9 times respectively.

In Kennedy's speech LET,US and POWER are most frequent words occurring 16, 12 and 9 times respectively.

In Nixon's speech US, LET and AMERICA are most frequent words occurring 26, 22 and 21 times respectively.

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

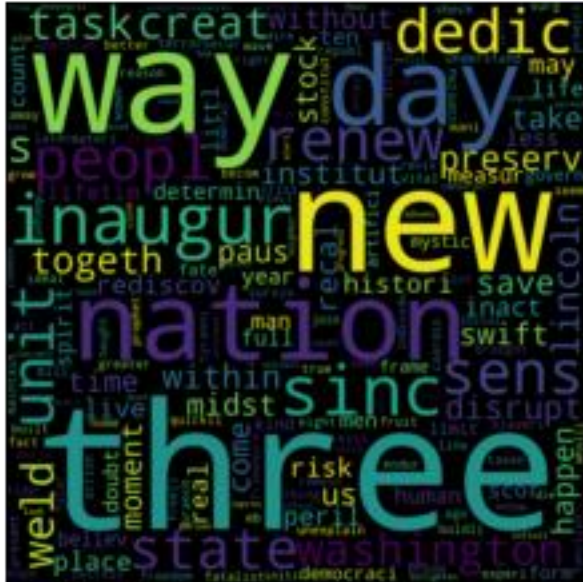
```
from wordcloud import WordCloud

rv_string = ' '.join([str(elem) for elem in Count_rv])
ken_string = ' '.join([str(elem) for elem in Count_ken])
nix_string = ' '.join([str(elem) for elem in Count_nix])
rv_fin = rv_string.replace('[^\w\s]', '')
ken_fin = ken_string.replace('[^\w\s]', '')
nix_fin = nix_string.replace('[^\w\s]', '')
```

Following is the code used for making a visualization of words in a presentable format.

## ROOSEVELT WORD CLOUD-

```
wordcloud = WordCloud(width = 1000, height = 1000, min_font_size = 10).generate(rv_fin)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.figure(figsize=(10,10))
plt.show()
```



## KENNEDY WORD CLOUD.

```
wordcloud = WordCloud(width = 1000, height = 1000, min_font_size = 10).generate(ken_fin)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.figure(figsize=(10,10))
plt.show()
```



## NIXON'S WORD CLOUD-

```
wordcloud = WordCloud(width = 1000, height = 1000, min_font_size = 10).generate(nix_fin)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.figure(figsize=(10,10))
plt.show()
```

