MACHINE LEARNING

PROJECT REPORT

BY,

RAGAVEDHNI K R

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.

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

- The data is read from 'Election_Data.xlsx' dataset, initial set of rows can be viewed as below.

	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

- We can either remove the *'Unnamed: 0'* column or make that column as index column instead of the default index column. Here, we drop that column from our dataframe and can be seen as below.

	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 shape of the data after removing the first column (Unnamed: 0), it has 10 columns and 1525 rows.

```
The shape of the data is  (1525, 9)
```

- The information of the data is seen below.

```
<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
```

- The Descriptive Statistics is shown using the 5 point summary - Count, Mean, Std dev, Min, 25%, 50%, 75%, Max values calculated for all the columns.
- The Mean and Median (50%) are nearly the same for all the columns, as majority of the columns are categorical/object type.
- We can check the description of the data including the object type columns as,

	vote	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	gender
count	1525	1525.000000	1525.000000	1525.000000	1525.000000	1525.000000	1525.000000	1525.000000	1525
unique	2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2
top	Labour	NaN	NaN	NaN	NaN	NaN	NaN	NaN	female
freq	1063	NaN	NaN	NaN	NaN	NaN	NaN	NaN	812
mean	NaN	54.182295	3.245902	3.140328	3.334426	2.746885	6.728525	1.542295	NaN
std	NaN	15.711209	0.880969	0.929951	1.174824	1.230703	3.297538	1.083315	NaN
min	NaN	24.000000	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000	NaN
25%	NaN	41.000000	3.000000	3.000000	2.000000	2.000000	4.000000	0.000000	NaN
50%	NaN	53.000000	3.000000	3.000000	4.000000	2.000000	6.000000	2.000000	NaN
75%	NaN	67.000000	4.000000	4.000000	4.000000	4.000000	10.000000	2.000000	NaN
max	NaN	93.000000	5.000000	5.000000	5.000000	5.000000	11.000000	3.000000	NaN

- The null value check in the dataframe is done and we find no null values are present in the data.

```
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
```

- We can see that 8 rows are duplicated including the target column. Here rows represent individual voter (as per the survey conducted) and we take only unique responses for training and testing the models.

```
Number of duplicate rows :  8
Duplicated rows:
```

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

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

Exploratory Data Analysis:

- There are no null values present in the data.
- The shape of the data frame is checked using df.shape and as seen below.

```
The shape of the data is  (1525, 9)
```

- The data type of the columns are checked using df.dtypes and as seen below.

```
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
```

- For categorical columns, we check the unique values and their counts.

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

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

```
ECONOMIC.COND.NATIONAL :  5
1     37
5     82
2    256
4    538
3    604
Name: economic.cond.national, dtype: int64
```

```
HAGUE :  5
3     37
5     73
1    233
4    557
2    617
Name: Hague, dtype: int64
```

```
ECONOMIC.COND.HOUSEHOLD :  5
1     65
5     92
2    280
4    435
3    645
Name: economic.cond.household, dtype: int64
```

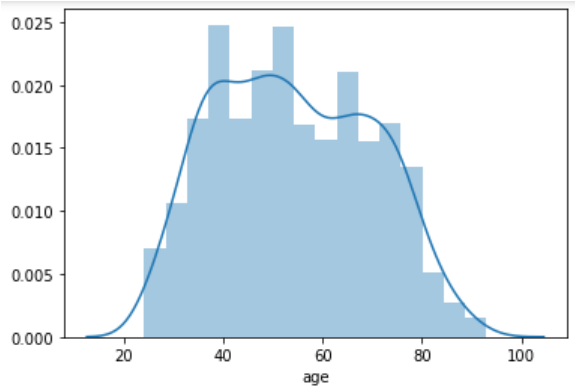
```
BLAIR :  5
3      1
1     97
5    152
2    434
4    833
Name: Blair, dtype: int64
```

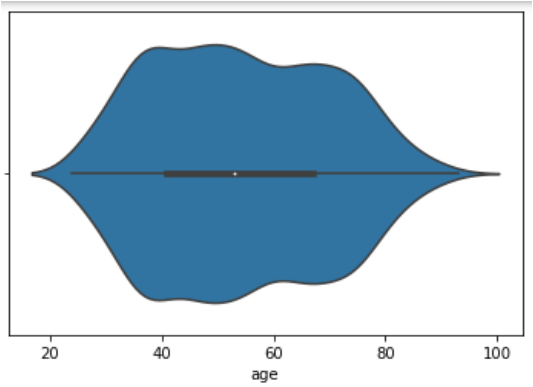
```
POLITICAL.KNOWLEDGE :  4
1     38
3    249
0    454
2    776
Name: political.knowledge, dtype: int64
```

```
EUROPE :  11
2      77
7      86
10    101
1     109
9     111
8     111
5     123
4     126
3     128
6     207
11    338
Name: Europe, dtype: int64
```

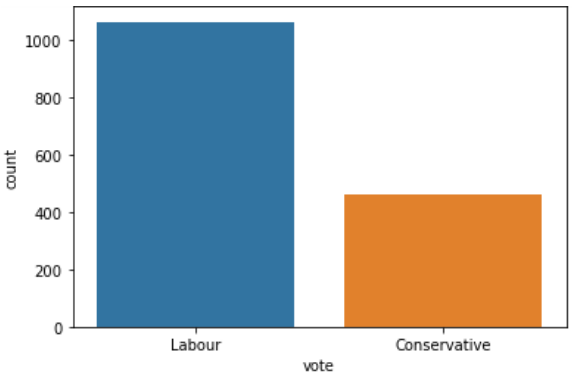
UNIVARIATE ANALYSIS:

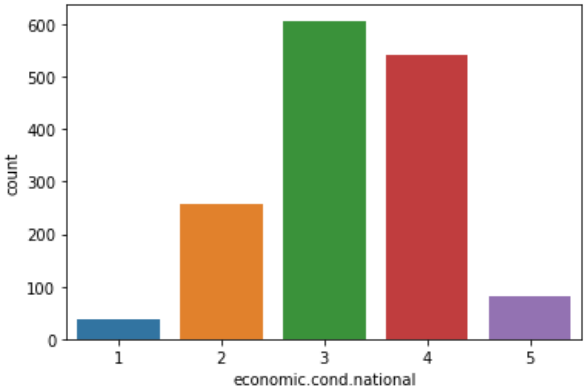
- The plots provide information about the distribution of the observations in the single data variable.
- Here we consider the distplot and violinplot (from sns package) to know the distribution of the continuous variable from the dataset.
- From distplot for the Age column, we can see that data is dense around 35 years to 80 years and very rare voters aged early 20 years or late 80 years.

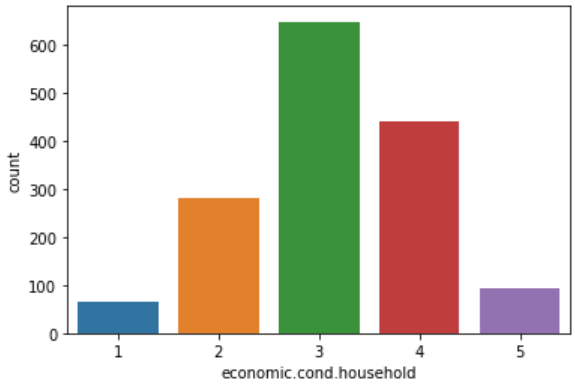


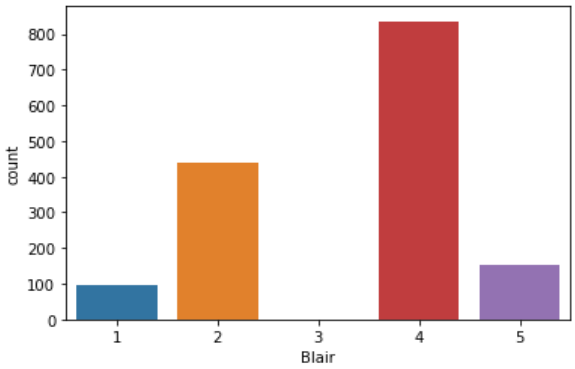


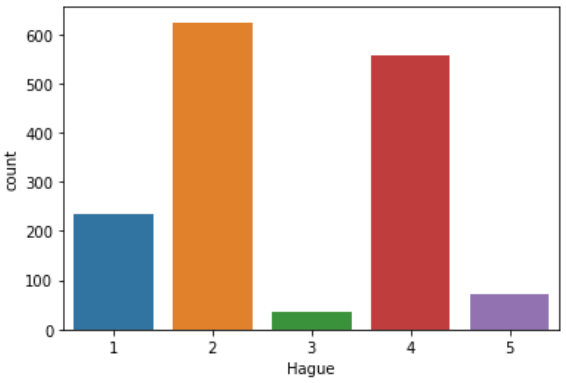
- For categorical variables, we use countplot (from sns package) to get the count of observations from each categorical bins.

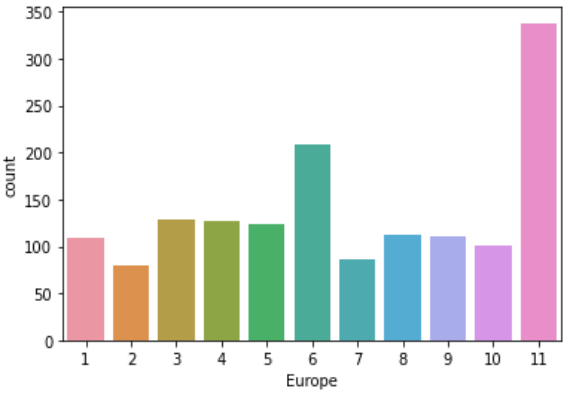


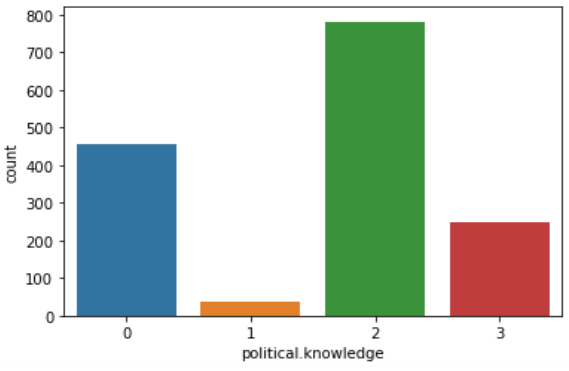


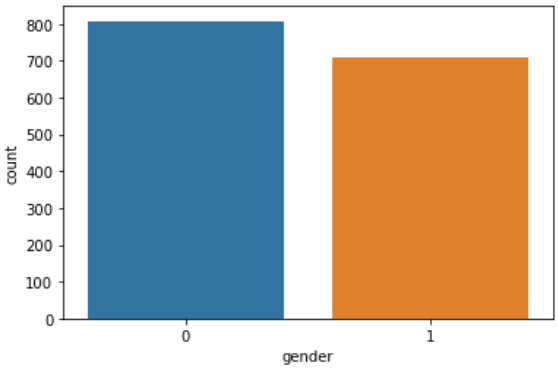








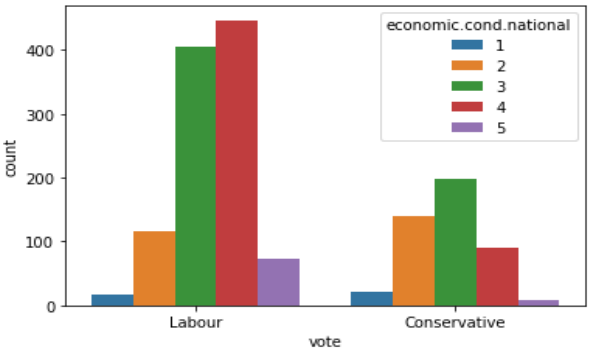




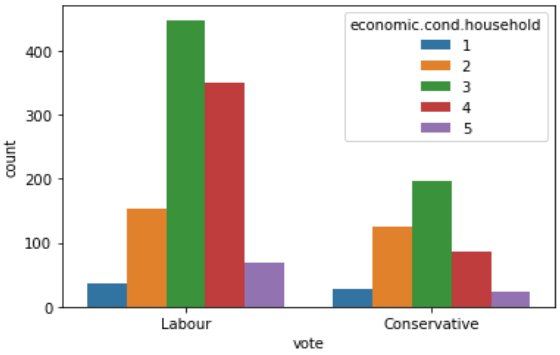
BIVARIATE ANALYSIS:

- We can use both the plots and tabulation for visually understanding the relation between the independent variables and target variable.
- The crosstab from pandas computes a simple cross tabulation of two (or more) factors. By default, it computes a frequency table of the factors.
- To understand through plots, we use count plots along with hue to plot two categorical variables together.

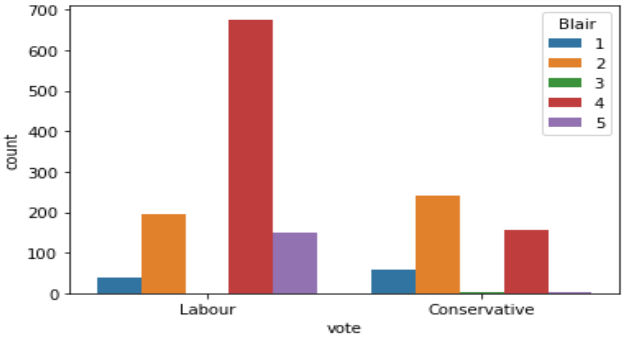
economic.cond.national	Conservative	Labour
1	21	16
2	140	116
3	199	405
4	91	447
5	9	73



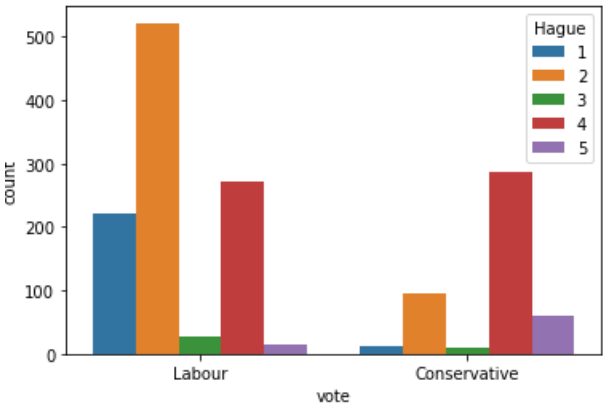
economic.cond.household	Conservative	Labour
1	28	37
2	126	154
3	197	448
4	86	349
5	23	69



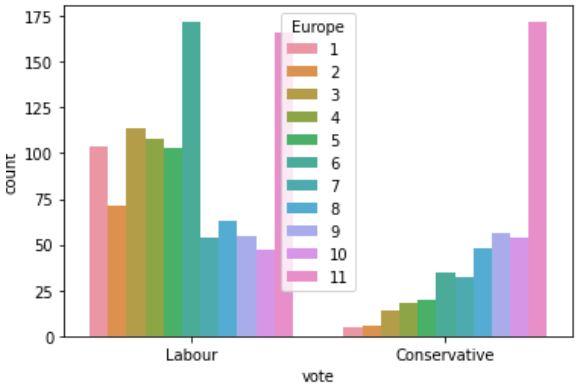
Blair	Conservative	Labour
1	59	38
2	240	194
3	1	0
4	157	676
5	3	149

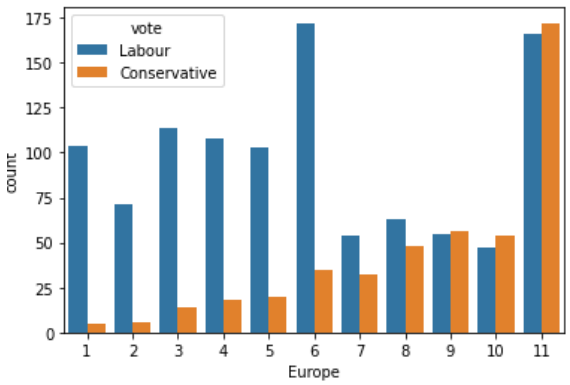


Hague	Conservative	Labour
1	11	222
2	95	522
3	9	28
4	286	271
5	59	14

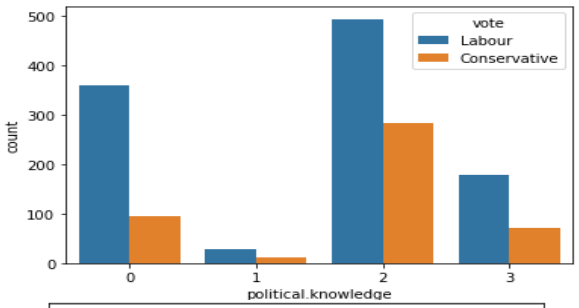


Europe	1	2	3	4	5	6	7	8	9	10	11
vote											
Conservative	5	6	14	18	20	35	32	48	56	54	172
Labour	104	71	114	108	103	172	54	63	55	47	166

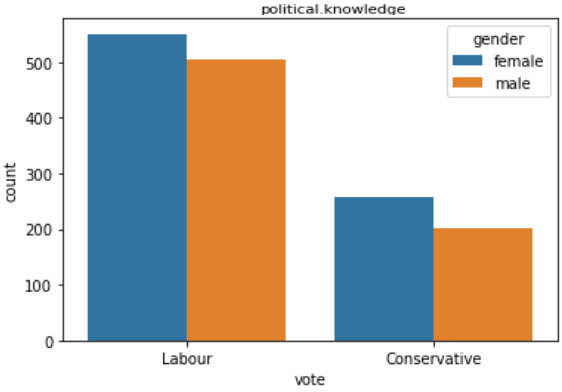




vote	Conservative	Labour
political.knowledge		
0	94	360
1	11	27
2	283	493
3	72	177



vote	Conservative	Labour
gender		
female	257	551
male	203	506



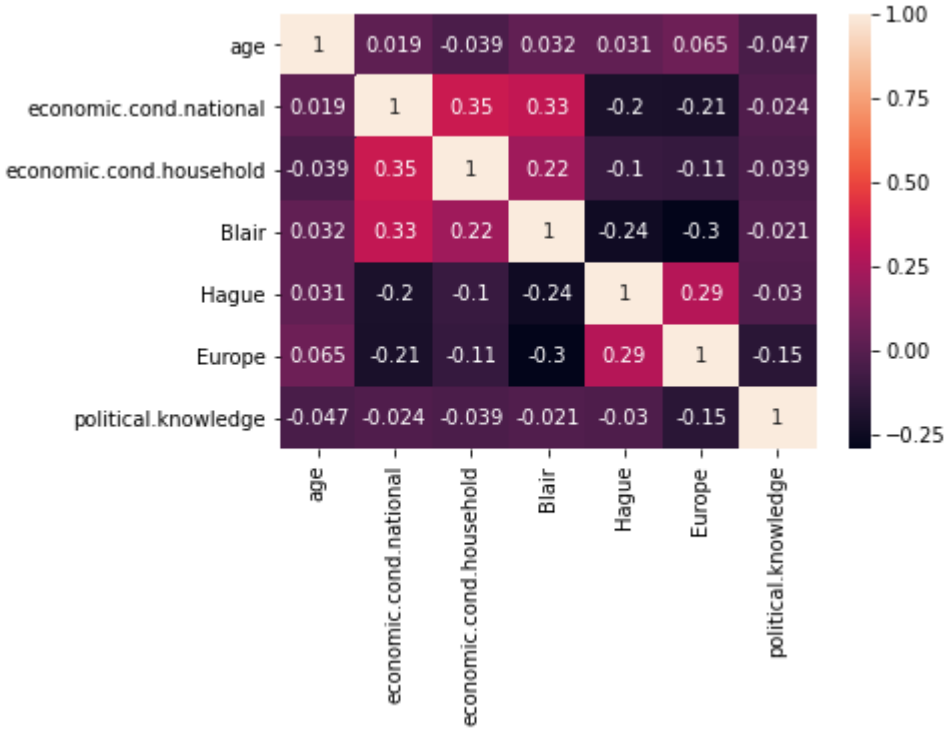
MULTIVARIATE ANALYSIS:

- The correlation across the variables can be found using corr() function in a matrix form.

	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge
age	1.000000	0.018687	-0.038868	0.032084	0.031144	0.064562	-0.046598
economic.cond.national	0.018687	1.000000	0.347687	0.326141	-0.200790	-0.209150	-0.023510
economic.cond.household	-0.038868	0.347687	1.000000	0.215822	-0.100392	-0.112897	-0.038528
Blair	0.032084	0.326141	0.215822	1.000000	-0.243508	-0.295944	-0.021299
Hague	0.031144	-0.200790	-0.100392	-0.243508	1.000000	0.285738	-0.029906
Europe	0.064562	-0.209150	-0.112897	-0.295944	0.285738	1.000000	-0.151197
political.knowledge	-0.046598	-0.023510	-0.038528	-0.021299	-0.029906	-0.151197	1.000000

- To indicate and visualize the clusters within the data using the heatmap (from sns package).
- Generally, all the models requires the independent variables to be uncorrelated or with less multicollinearity.
- Here, all the columns are independent of each other and no correlation is found.

HeatMap:



PairPlot:

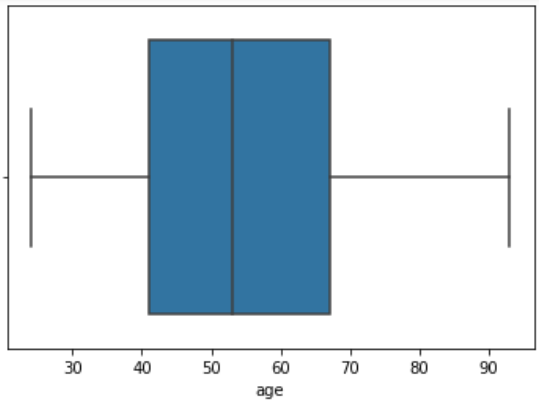
- The pairplot (from sns package) is used for visualizing the pair wise relationship across entire dataframe.
- We plot by taking the target column ‘Vote’ in hue and can observe the distribution of classes in the independent columns.
- As all the columns except Age column is categorical, we don’t find any linear relationship among the columns.

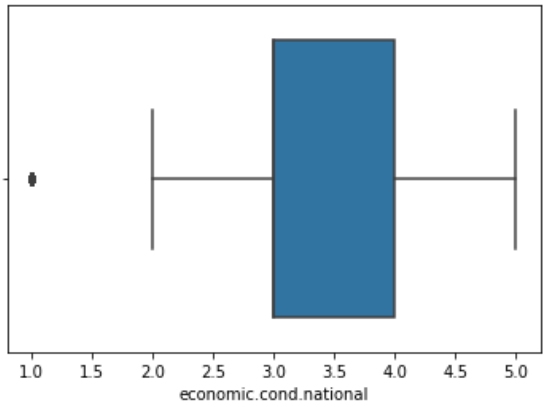
Outliers Check:

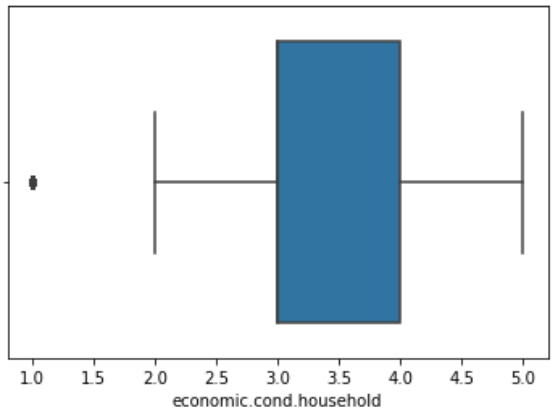
- For checking the outliers in all the columns, we use boxplots (from sns package).

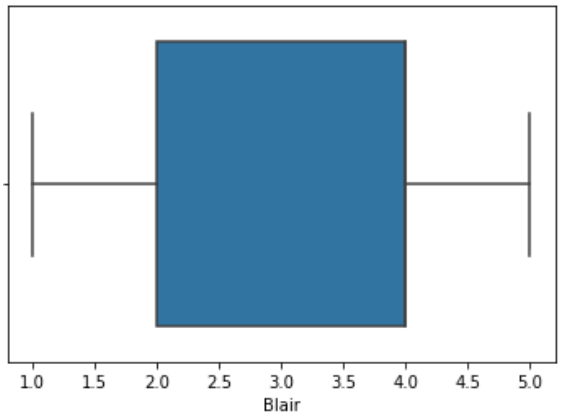
```
Shape of the data without outlier treatment : (1430, 9)
Shape of the original data  : (1517, 9)
```

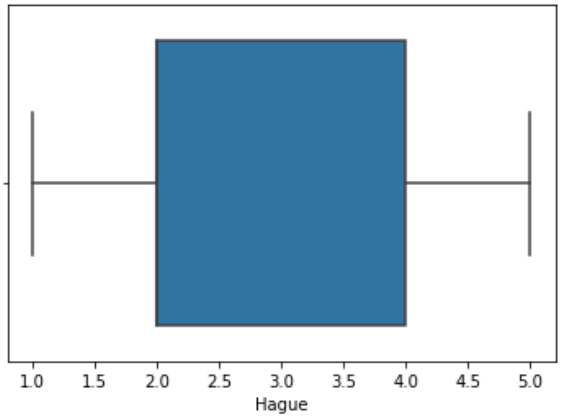
- We usually check the outliers only in the continuous column, here only the continuous column is 'age' and it does not have any outliers present in them.
- We can see that 2 columns economic.cond.national and economic.cond.household are having outliers.
- Both are categorical columns so we don’t handle the outliers in them.
- All other columns are void of outliers.
- We can plot them individually or combined all the columns in single boxplot to understand the relation among the columns.

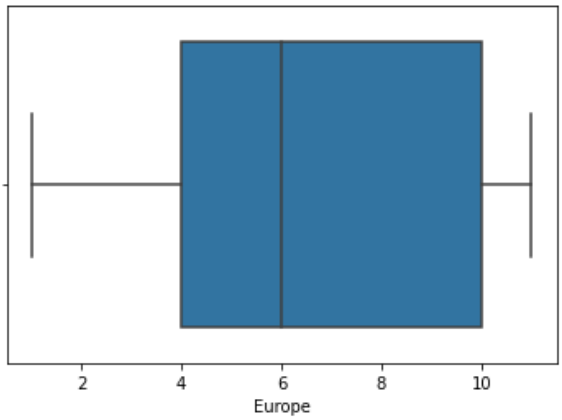


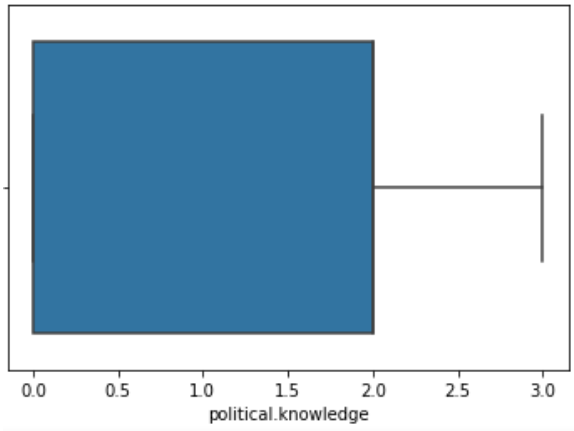


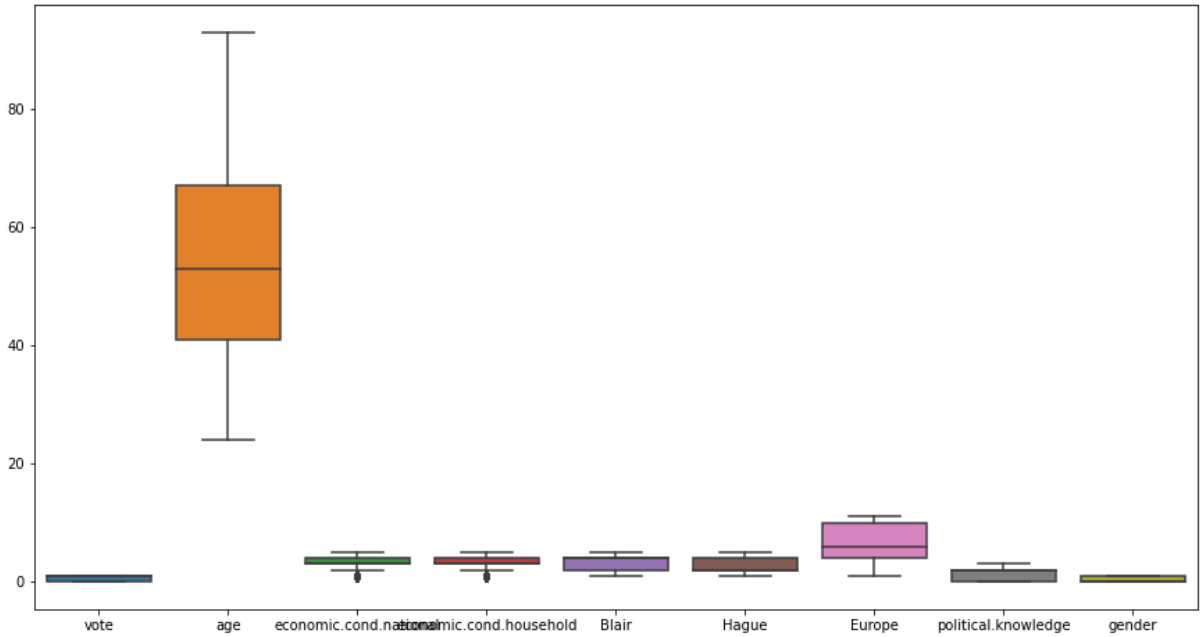












1.3. Encode the data (having string values) for Modelling. Is Scaling necessary here or not? Data Split: Split the data into train and test (70:30)

<u>Encoding the data:</u>

- Checking the data type of the columns and converting the object type columns into

categorical codes using the pd.Categorical() function.

- Here, the columns 'vote' and 'gender' are in object type and they are converted into int8 type as numerical codes.

```
1    1057
0     460
Name: vote, dtype: int64
```

```
0    808
1    709
Name: gender, dtype: int64
```

- After encoding the string values, we get the below dataframe.

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

Scaling:

- As most of the independent variables are categorical (ordinal) in nature, we do not require scaling on the data.
- But we can apply scaling for the Distance based models like KNN model to gives equal weightage to all the features and Gradient descent based models like Logistic Regression to converge quickly.
- Few models like Linear Discriminant Analysis and Naïve Bayes can handle well without scaling, as they are insensitive to scaling of the features.
- We apply zscore (from scipy.stats module) for scaling the numerical column.

Splitting the data:

- Grouping all the columns except the target column into separate variable.
- Checking the classes in the target column, we have Labour class more than twice the Conservative class.

```
1    1057
0     460
Name: vote, dtype: int64
```

- We split the data as X (with all columns except the target column) and Y (target column) used for the model prediction with the ratio 70:30, where 70% of the data for training and 30% of the data for testing.
- Using the train_test_split function from the model_selection module in sklearn package.

1.4. Apply Logistic Regression and LDA (Linear Discriminant Analysis). Interpret the inferences of both models.

Logistic Regression:

- The Logistic Regression model also called Logit is used in the classification problem

for both the Binary and Multi class classifier.

- We apply the Logistic Regression model on the data with default values for the parameters.

```
LogisticRegression(
    penalty='l2',
    *,
    dual=False,
    tol=0.0001,
    C=1.0,
    fit_intercept=True,
    intercept_scaling=1,
    class_weight=None,
    random_state=None,
    solver='lbfgs',
    max_iter=100,
    multi_class='auto',
    verbose=0,
    warm_start=False,
    n_jobs=None,
    l1_ratio=None,
)
```

- The parameters with their default values are:
    - Solver: used in the optimization problem.
    - Penalty: used to specify the norm used in the penalization.
    - Tol: used to specify the tolerance for the stopping criteria.
    - Max_iter: used for specifying maximum number of iterations taken for solvers to converge.
- We fit the train data (X_train and y_train) into the model.
- We predict for the test data and train data to compare the predicted and actual values.

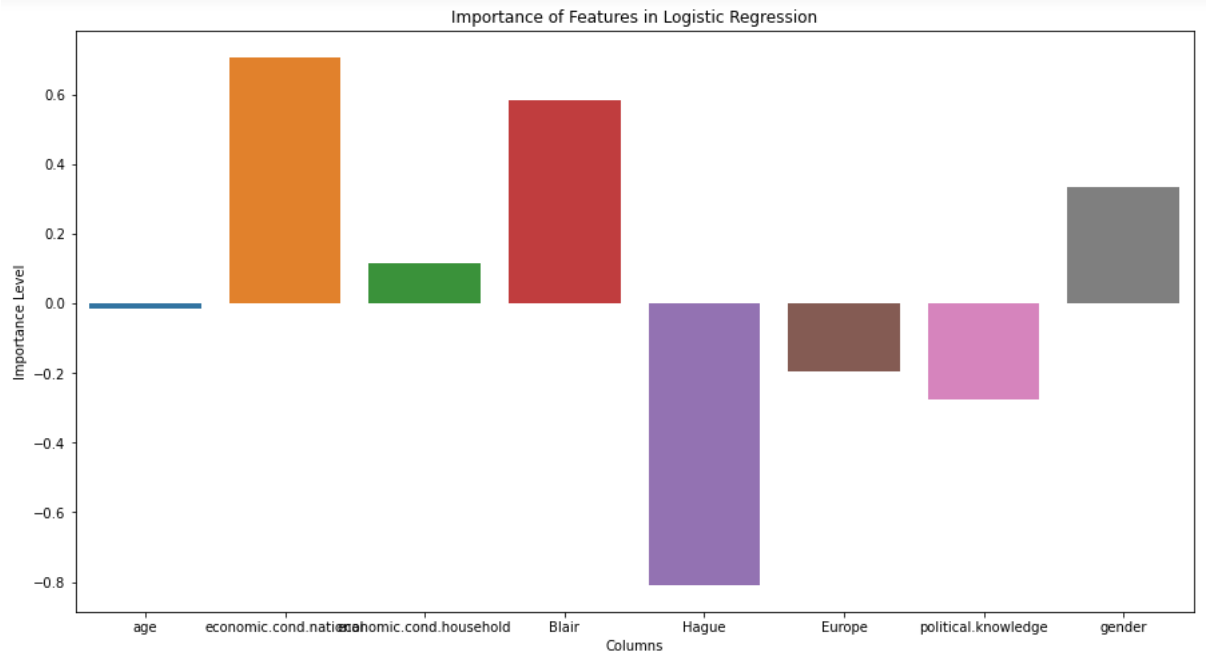
Inference of Logistic Regression:

- We get the coefficients of the variables in the Logistic Regression.

	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	gender	intercept
0	-0.01395	0.707359	0.115391	0.582678	-0.80919	-0.195746	-0.276644	0.334127	1.37743

- Based on the importance of the features in the Logistic Regression.

	Imp
economic.cond.national	0.707359
Blair	0.582678
gender	0.334127
economic.cond.household	0.115391
age	-0.013950
Europe	-0.195746
political.knowledge	-0.276644
Hague	-0.809190



- We can see that the column 'economic.cond.national' is of the highest importance and the column 'Hague' is of least importance in determining the target class for the model.

Linear Discriminant Analysis:

- The Linear Discriminant Analysis (LDA) model is a classifier with a linear decision boundary, generated by fitting class conditional densities to the data and using Bayes' rule.
- We apply the LDA model with the default parameter values.

```
LinearDiscriminantAnalysis(
    *,
    solver='svd',
    shrinkage=None,
    priors=None,
    n_components=None,
    store_covariance=False,
    tol=0.0001,
)
```

- The parameters with their default values are:
    - Solver: svd – Singular Value Decomposition, lsqr – Least Square solution, eigen –Eigen Value decomposition.

- Shrinkage: none – no shrinkage, auto – automatic shrinkage, give float between 0 and 1.
- Priors: class prior probabilities as an array.
- N_components: number of components for dimensionality reduction.
- Tol: absolute threshold.

- We fit the train data for training the LDA model and predict the test data as well as train data to compare with actual values.

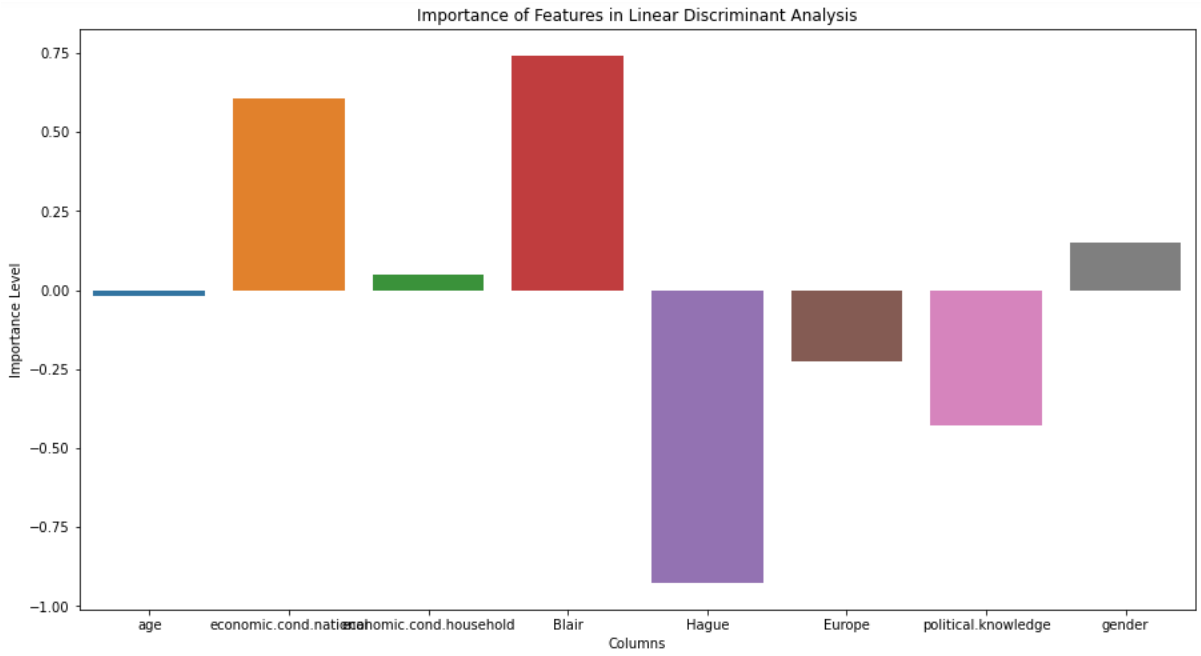
Inferences of LDA Model:

- We get the coefficients of the variables in the LDA.

	age	economic.cond.national	economic.cond.household	Blair	Hague	Europe	political.knowledge	gender	intercept
0	-0.020037	0.60492	0.050069	0.7424	-0.926634	-0.223612	-0.430335	0.14908	2.627616

- Based on the importance of the features in the LDA, we see below:

	Imp
Blair	0.742400
economic.cond.national	0.604920
gender	0.149080
economic.cond.household	0.050069
age	-0.020037
Europe	-0.223612
political.knowledge	-0.430335
Hague	-0.926634



- We can see that the column ‘Blair’ is the highest important column and ‘economic.cond.national’ is of the second highest importance.
- The column ‘Hague’ is of least importance in determining the target class for the

model.

1.5. Apply KNN Model and Naïve Bayes Model. Interpret the inferences of each model.

KNN Model:

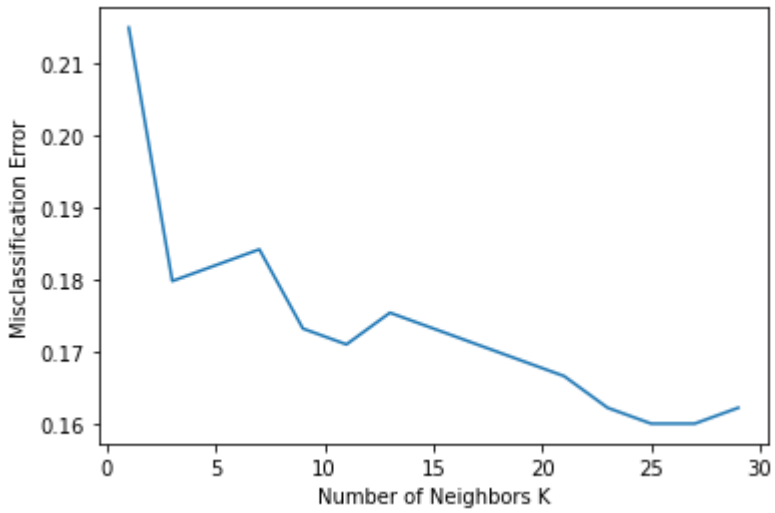
- Before applying the KNN model, we need to scale the numerical columns as it uses distance for calculation.
- Here the column 'Age' is alone numerical, hence we apply zscale (from stats module in scipy library) for that column separately in train and test data.
- KNN model uses voting technique for the classification.
- Applying the KNN model with parameters of their default values.

```
KNeighborsClassifier(
    n_neighbors=5,
    *,
    weights='uniform',
    algorithm='auto',
    leaf_size=30,
    p=2,
    metric='minkowski',
    metric_params=None,
    n_jobs=None,
    **kwargs,
)
```

- We fit the KNN model using the training data and predict for the train and test data to compare with the actual values.
- Taking a list of values and passing them to the KNN model, we calculate the accuracy and misclassification error for the values in the list.
- Below is the accuracy and misclassification error for the values from 1 to 30 (taking only odd numbers).

Accuracy	Misclassification Error
[0.7850877192982456,	[0.2149122807017544,
0.8201754385964912,	0.17982456140350878,
0.8179824561403509,	0.18201754385964908,
0.8157894736842105,	0.1842105263157895,
0.8267543859649122,	0.17324561403508776,
0.8289473684210527,	0.17105263157894735,
0.8245614035087719,	0.17543859649122806,
0.8267543859649122,	0.17324561403508776,
0.8289473684210527,	0.17105263157894735,
0.831140350877193,	0.16885964912280704,
0.8333333333333334,	0.16666666666666663,
0.8377192982456141,	0.16228070175438591,
0.8399122807017544,	0.1600877192982456,
0.8399122807017544,	0.1600877192982456,
0.8377192982456141]	0.16228070175438591]

- Select the best 'k' value which has high accuracy and low misclassification error.
- We can visualize them in the plot and taking K = 25 as the best K value for K-neighbours.



Inferences from KNN model:

- KNN model is lazy learner as it learns from train data when a new query is made
- They are fast to train and slower to query and make predictions.
- Able to recover unstructured partitions of the space and robust in adapting different densities.
- KNN can seamlessly handle a very large number of classes.
- KNN model predicts based on similarity to existing objects.
- KNN is costly inference as to infer the label of an input query, we must find the data points closest to it.

Naïve Bayes Model:

- The Naïve Bayes uses the estimated prior probabilities and conditional probabilities in calculating the posterior probability.
- In Gaussian Naïve Bayes, the continuous independent values are assumed to be in Gaussian distribution (estimate mean and standard deviation) as we cannot calculate conditional probabilities.
- It does not require scaling as it takes the individual independent variables and calculates their probabilities.
- The Gaussian Naïve Bayes model is applied for our data with the parameters of their default value.

```
GaussianNB(*, priors=None, var_smoothing=1e-09)
```

- The parameters with their default values in detail:
    - priors: Prior probabilities of the classes. If specified the priors are not adjusted according to the data.
    - var_smoothing: Portion of the largest variance of all features that is added to variances for calculation stability.
- We fit the model with the training data and predict the test set along with the train set and compare them with the actual results.

Inferences from Naïve Bayes Model:

- As being called 'Naïve' due to the assumption that the features in dataset are mutually independent, but is often violated in real world.
- It is being relatively robust, easy to implement, fast and accurate.
- Absence of rare events in the training set will lead to incorrect probability calculation and classification in the test set.

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

Model Tuning:

- The model tuning refers to the tuning of the hyperparameters that shapes the behavior of the algorithm/model to specific dataset.
- We can use the GridSearchCV from sklearn.model_selection package, to try out different set of values for each parameter and get the best set of parameter.

Logistic Regression Model Tuning:

- GridSearchCV uses cross-validation for the number of times to loop through the predefined hyperparameters and fit our Logistic Regression model on the training set.

```
GridSearchCV(cv=5,
             estimator=LogisticRegression(max_iter=10000, n_jobs=2,
                                          random_state=1),
             n_jobs=-1,
             param_grid={'penalty': ['l2', 'none'],
                         'solver': ['saga', 'lbfgs', 'sag', 'liblinear',
                                    'newton-cg'],
                         'tol': [0.0001, 1e-05, 0.001, 1e-06]},
             scoring='f1')
```

- The hyper parameters are:
    - Solver: used in the optimization problem.
    - Penalty: used to specify the norm used in the penalization.
    - Tol: used to specify the tolerance for the stopping criteria.
    - Max_iter: used for specifying maximum number of iterations taken for solvers to converge.
- We can get the best estimator with the right set of parameters for this data.

```
LogisticRegression(max_iter=10000, n_jobs=2, random_state=1, solver='liblinear')
```

Linear Discriminant Analysis Model Tuning:

- GridSearchCV uses cross-validation for the number of times to loop through the predefined hyperparameters and fit our Linear Discriminant Analysis (LDA) model on the training set.

```
GridSearchCV(cv=3, estimator=LinearDiscriminantAnalysis(),
             param_grid={'solver': ['svd', 'lsqr', 'eigen']})
```

- We apply various solver in the hyper parameters to get the best estimator of LDA model.

```
LinearDiscriminantAnalysis(solver='lsqr')
```

KNN Model Tuning:

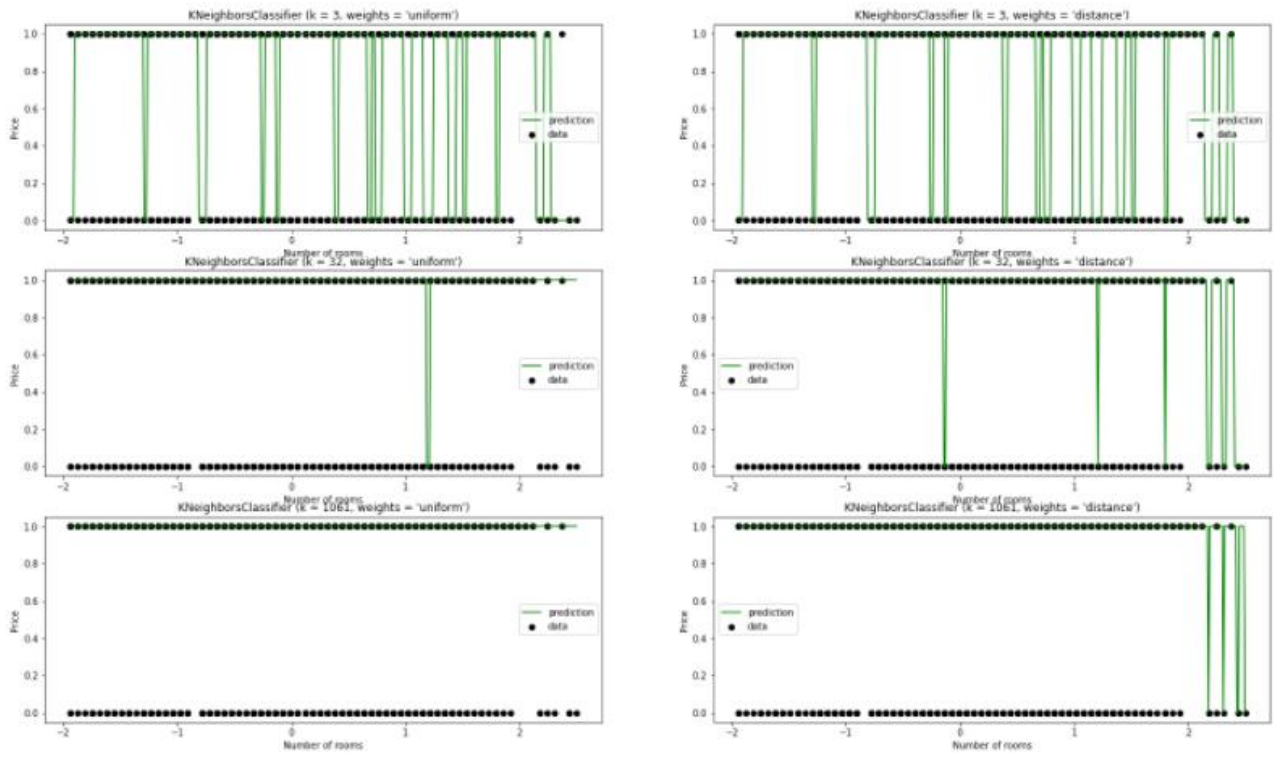
- Here in GridSearchCV we give various hyperparameters as the leaf_size as a list with range of 50 values, n_neighbors as a list with range of 30 values and various p values as Power parameter for the Minkowski metric.

```
GridSearchCV(cv=3, estimator=KNeighborsClassifier(),
             param_grid={'leaf_size': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                                       13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
                                       23, 24, 25, 26, 27, 28, 29, 30, ...],
                         'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                                         13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
                                         23, 24, 25, 26, 27, 28, 29],
                         'p': [1, 2]})
```

- The best estimator is selected for the KNN model.

```
KNeighborsClassifier(leaf_size=1, n_neighbors=20, p=1)
```

- For training the model, we can try for different parameters of weights and n_neighbors with the train set and plot the scatterplot to visualize and select the best estimators for the model.



Naïve Bayes model:

- We can use the partial_fit function while fiiting and training the model, for incremental fit on a batch of samples.
- This is especially useful when the whole dataset is too big to fit in memory at once.

partial_fit(X, y, classes=None, sample_weight=None)

- We apply the Gridsearchcv on the Gaussian Naïve Bayes model, with var_smooting parameter, which handles portion of the largest variance of all features that is added to variances for calculation stability.

```
GridSearchCV(cv=3, estimator=GaussianNB(),
             param_grid={'var_smoothing': array([1.00000000e+00, 8.11130831e-01, 6.57933225e-01, 5.33669923e-01,
       4.32876128e-01, 3.51119173e-01, 2.84803587e-01, 2.31012970e-01,
       1.87381742e-01, 1.51991108e-01, 1.23284674e-01, 1.00000000e-01,
       8.11130831e-02, 6.57933225e-02, 5.33669923e-02, 4.32876128e-02,
       3.51119173e-02, 2.84803587e-02, 2.31...
       1.23284674e-07, 1.00000000e-07, 8.11130831e-08, 6.57933225e-08,
       5.33669923e-08, 4.32876128e-08, 3.51119173e-08, 2.84803587e-08,
       2.31012970e-08, 1.87381742e-08, 1.51991108e-08, 1.23284674e-08,
       1.00000000e-08, 8.11130831e-09, 6.57933225e-09, 5.33669923e-09,
       4.32876128e-09, 3.51119173e-09, 2.84803587e-09, 2.31012970e-09,
       1.87381742e-09, 1.51991108e-09, 1.23284674e-09, 1.00000000e-09])},
             scoring='accuracy', verbose=1)
```

- We get the best estimator for the model.

```
GaussianNB(var_smoothing=0.0006579332246575676)
```

- We can predict the test set as well as for the train set using the best models and their respective best estimators to get better results from the results of the models with default value parameters.

Bagging Model:

- Bagging takes only the sample with replacement of data.
- It can perform well with complex models, as the model prediction gets similar and might return worst final prediction.
- It fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction.
- We execute the bagging model with their default parameter values.

```
BaggingClassifier(
    base_estimator=None,
    n_estimators=10,
    *,
    max_samples=1.0,
    max_features=1.0,
    bootstrap=True,
    bootstrap_features=False,
    oob_score=False,
    warm_start=False,
    n_jobs=None,
    random_state=None,
    verbose=0,
)
```

- The base estimators can be any model to fit on random subsets of the dataset, default estimator is the decision tree.
- The most important parameter for bagged decision trees is the number of trees (*n_estimators*).

- We have tried the Bagging model with other best estimator models like Random Forest model, KNN model, Gaussian Naïve bayes model.
- We can compare the predictions and get the best Bagging model.
- Default Model:

```
BaggingClassifier(random_state=1)
```

- KNN Model:

```
BaggingClassifier(base_estimator=KNeighborsClassifier(leaf_size=1,
                                                      n_neighbors=20, p=1),
                  random_state=1)
```

- Gaussian Naïve Bayes Model:

```
BaggingClassifier(base_estimator=GaussianNB(var_smoothing=0.0006579332246575676),
                  random_state=1)
```

- Random Forest Model:

```
BaggingClassifier(base_estimator=RandomForestClassifier(max_depth=8,
                                                        max_features=5,
                                                        min_samples_leaf=18,
                                                        min_samples_split=45,
                                                        n_estimators=101),
                  random_state=1)
```

- We can give various numbers of trees in n_estimators in the GridsearchCV.

```
GridSearchCV(cv=3, estimator=BaggingClassifier(random_state=1),
             param_grid={'n_estimators': [101, 301]})
```

- The best estimator of Bagging model from GridsearchCV as below.

```
BaggingClassifier(n_estimators=301, random_state=1)
```

Boosting Model:

- The Boosting model executes in sequential order, i.e., it executes for one model, changes the data (updates the weight of misclassified values or takes the residual values) and is predicted by another model.
- The 2 types of Boosting models are:
  - Ada Boosting (Adaptive Boosting) – the successive learner focuses on the ill fitted data of previous learner.
  - Gradient Boosting – It fits on the modified version of original data, residuals from previous learner.
- Boosting can perform well on the simple models, as it takes high computational time for fitting on the data.

AdaBoosting:

- The Adaptive boosting classifier is built on the default values of the parameters.

```
AdaBoostClassifier(
    base_estimator=None,
    *,
    n_estimators=50,
    learning_rate=1.0,
    algorithm='SAMME.R',
    random_state=None,
)
```

- We can tune the n_estimators parameters in the GridsearchCV for getting the best AdaBoosting model.

```
GridSearchCV(cv=3, estimator=AdaBoostClassifier(),
             param_grid={'n_estimators': [101, 301]})
```

- The best estimator for the model is given as,

```
AdaBoostClassifier(n_estimators=101)
```

- We can fit the model with the train data for training the model.
- Using the fitted model, we can predict for the test data as well as for the train data.

Gradient Boosting:

- The Gradient Boosting is loaded with the default values for the parameters.

```
GradientBoostingClassifier(
    *,
    loss='deviance',
    learning_rate=0.1,
    n_estimators=100,
    subsample=1.0,
    criterion='friedman_mse',
    min_samples_split=2,
    min_samples_leaf=1,
    min_weight_fraction_leaf=0.0,
    max_depth=3,
    min_impurity_decrease=0.0,
    min_impurity_split=None,
    init=None,
    random_state=None,
    max_features=None,
    verbose=0,
    max_leaf_nodes=None,
    warm_start=False,
    presort='deprecated',
    validation_fraction=0.1,
    n_iter_no_change=None,
    tol=0.0001,
    ccp_alpha=0.0,
)
```

- We can tune the n_estimators, min_samples_split, max_depth and min_samples_leaf hyperparameters in the GridsearchCV for getting the best Gradient Boosting model.

```
GridSearchCV(cv=3, estimator=GradientBoostingClassifier(),
             param_grid={'max_depth': [3, 4], 'min_samples_leaf': [1, 3, 5],
                         'min_samples_split': [2, 3, 4],
                         'n_estimators': [101, 301]})
```

- The best estimators are given by,

```
{'max_depth': 3,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 101}
```

- Using the best parameters we can fit the model from the train data and predict the test data and train data.

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.

- We can view the Accuracy scores of each models.

Logistic Regression Accuracy:

```
Accuracy of the Logistic Regression for Train data: 0.8360037700282752
Accuracy of the Logistic Regression for Test data: 0.8245614035087719
```

```
Accuracy of the Logistic Regression for Train data in GridsearchCV: 0.84
Accuracy of the Logistic Regression for Test data in GridsearchCV: 0.83
```

Linear Discriminant Analysis Accuracy:

```
Accuracy of Train in LDA model: 0.8341187558906692
Accuracy of Test in LDA model: 0.8333333333333334
Accuracy of Train in LDA model: 0.83
Accuracy of Test in LDA model: 0.83
Accuracy of Train in LDA model in GridsearchCV: 0.83
Accuracy of Test in LDA model in GridsearchCV: 0.83
```

KNN Model Accuracy:

```
Accuracy of Train data in KNN model: 0.8614514608859567
Accuracy of Test data in KNN model: 0.8179824561403509
Accuracy of the Train data in KNN Model using GridsearcCV:  0.84
Accuracy of the Test data in KNN Model using GridsearcCV:  0.83
```

Naïve Bayes Model Accuracy:

```
Accuracy of the Train data in Gradient Naive Bayes: 0.8350612629594723
Accuracy of the Test data in Gradient Naive Bayes: 0.8223684210526315
Accuracy of the Train data in Gradient Naive Bayes using GridsearchCV: 0.83
Accuracy of the Train data in Gradient Naive Bayes using GridsearchCV: 0.85
```

Random Forest Accuracy:

```
Accuracy of the Train data in Random Forest model : 1.0
Accuracy of the Train data in Random Forest model : 0.8289473684210527
Accuracy of the Train data in Random Forest using GridsearchCV: 0.85
Accuracy of the Train data in Random Forest using GridsearchCV: 0.83
```

Bagging Accuracy:

```
Accuracy of the Train data in Bagging: 0.9868049010367578
Accuracy of the Test data in Bagging: 0.7982456140350878
Accuracy of the Train data in Bagging using Random Forest: 0.8397737983034873
Accuracy of the Test data in Bagging using Random Forest: 0.8135964912280702
Accuracy of the Train data in Bagging using GridsearchCV: 1.0
Accuracy of the Test data in Bagging using GridsearchCV: 0.82
```

Ada Boosting Accuracy:

```
Accuracy of the Train data in AdaBoosting: 0.8407163053722903
Accuracy of the Test data in AdaBoosting: 0.8289473684210527
Accuracy of the Train data in AdaBoosting using GridsearchCV: 0.84
Accuracy of the Test data in AdaBoosting using GridsearchCV: 0.84
```

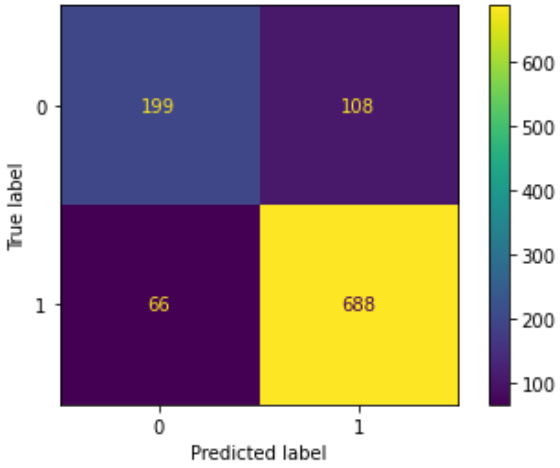
Gradient Boosting Accuracy:

```
Accuracy of the Train data in Gradient Boosting: 0.8868991517436381
Accuracy of the Test data in Gradient Boosting: 0.8377192982456141
```

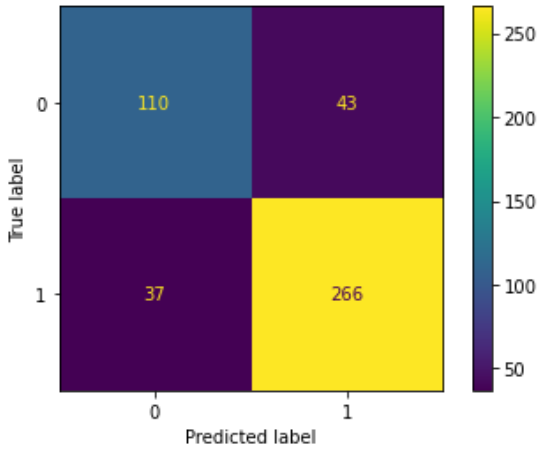
```
Accuracy of the Train data in Gradient Boosting using GridsearchCV: 0.89
Accuracy of the Train data in Gradient Boosting using GridsearchCV: 0.84
```

- The confusion matrix for all the models.

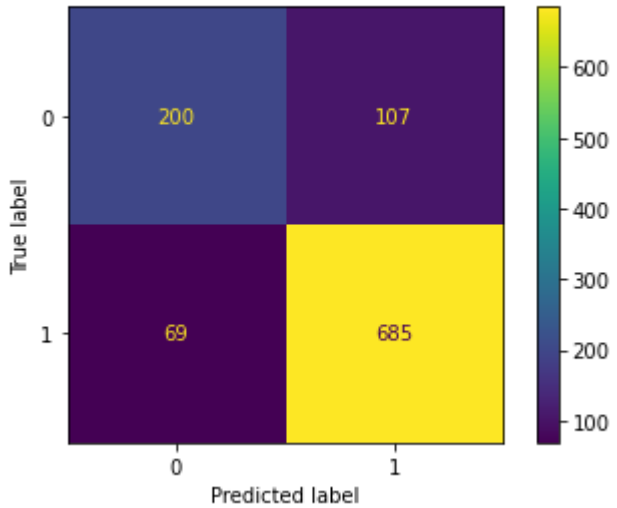
```
Confusion Matrix of the Logistic Regression for Train data:
```



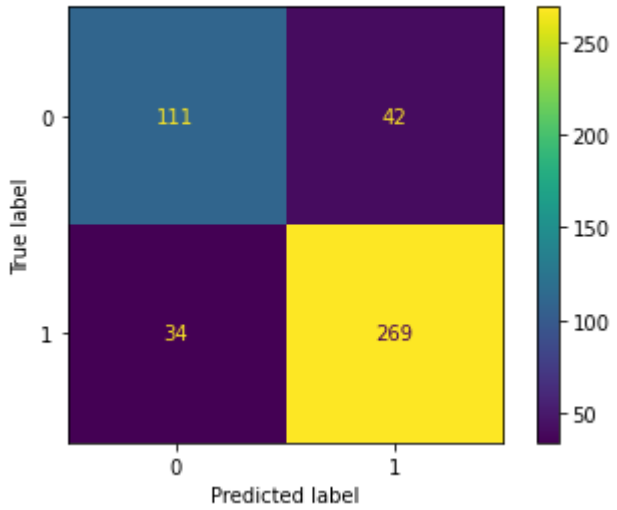

```
Confusion Matrix of the Logistic Regression for Test data:
```




```
Confusion Matrix of the LDA for Train data:
```



Confusion Matrix of the LDA for Test data:



Confusion Matrix, Classification Report for KNN Model:

```
Confusion Matrix of the KNN for Train data:
[[223  84]
 [ 63 691]]
Classification Report of the KNN for Train data:
              precision    recall  f1-score   support

           0       0.78      0.73      0.75       307
           1       0.89      0.92      0.90       754

    accuracy                           0.86      1061
   macro avg       0.84      0.82      0.83      1061
weighted avg       0.86      0.86      0.86      1061


Accuracy of Test data in KNN model: 0.8179824561403509
Confusion Matrix of the KNN for Test data:
[[100  53]
 [ 30 273]]
Classification Report of the KNN for Test data:
              precision    recall  f1-score   support

           0       0.77      0.65      0.71       153
           1       0.84      0.90      0.87       303

    accuracy                           0.82       456
   macro avg       0.80      0.78      0.79       456
weighted avg       0.81      0.82      0.81       456
```

Confusion Matrix, Classification Report for Naïve Bayes Model:

```
Accuracy of the Train data in Gradient Naive Bayes: 0.822808671065033
Confusion matrix for Train data in Gradient Naive Bayes:
[[173 134]
 [ 54 700]]
Classification Report for Train data in Gradient Naive Bayes:
              precision    recall  f1-score   support

           0       0.76      0.56      0.65       307
           1       0.84      0.93      0.88       754

    accuracy                           0.82      1061
   macro avg       0.80      0.75      0.76      1061
weighted avg       0.82      0.82      0.81      1061
```

```
Accuracy of the Test data in Gradient Naive Bayes: 0.8114035087719298
Confusion matrix for Test data in Gradient Naive Bayes:
[[ 99  54]
 [ 32 271]]
Classification Report for Test data in Gradient Naive Bayes:
              precision    recall  f1-score   support

           0       0.76      0.65      0.70       153
           1       0.83      0.89      0.86       303

    accuracy                           0.81       456
   macro avg       0.79      0.77      0.78       456
weighted avg       0.81      0.81      0.81       456
```

Confusion Matrix, Classification Report for Bagging using Random Forest Model:

```
Accuracy of the Train data in Bagging using Random Forest: 0.8397737983034873
Confusion matrix for Train data in Bagging using Random Forest:
[[188 119]
 [ 51 703]]
Classification Report for Train data in Bagging using Random Forest:
              precision    recall  f1-score   support

           0       0.79      0.61      0.69       307
           1       0.86      0.93      0.89       754

    accuracy                           0.84      1061
   macro avg       0.82      0.77      0.79      1061
weighted avg       0.84      0.84      0.83      1061
```

```
Accuracy of the Test data in Bagging using Random Forest: 0.8135964912280702
Confusion matrix for Test data in Bagging using Random Forest:
[[ 93  60]
 [ 25 278]]
Classification Report for Test data in Bagging using Random Forest:
              precision    recall  f1-score   support

           0       0.79      0.61      0.69       153
           1       0.82      0.92      0.87       303

    accuracy                           0.81       456
   macro avg       0.81      0.76      0.78       456
weighted avg       0.81      0.81      0.81       456
```

Confusion Matrix, Classification Report for AdaBoosting Model:

```
Accuracy of the Train data in AdaBoosting: 0.8407163053722903
Confusion matrix for Train data in AdaBoosting:
[[237  98]
 [ 71 655]]
Classification Report for Train data in AdaBoosting:
              precision    recall  f1-score   support

           0       0.77      0.71      0.74       335
           1       0.87      0.90      0.89       726

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

```
Accuracy of the Test data in AdaBoosting: 0.8289473684210527
Confusion matrix for Test data in AdaBoosting:
[[ 81  44]
 [ 34 297]]
Classification Report for Test data in AdaBoosting:
              precision    recall  f1-score   support

           0       0.70      0.65      0.68       125
           1       0.87      0.90      0.88       331

    accuracy                           0.83       456
   macro avg       0.79      0.77      0.78       456
weighted avg       0.83      0.83      0.83       456
```

Confusion Matrix, Classification Report for Gradient Boosting Model:

```
Accuracy of the Train data in Gradient Boosting: 0.8868991517436381
Confusion matrix for Train data in Gradient Boosting:
[[261  74]
 [ 46 680]]
Classification Report for Train data in Gradient Boosting:
              precision    recall  f1-score   support

           0       0.85      0.78      0.81       335
           1       0.90      0.94      0.92       726

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

```
Accuracy of the Test data in Gradient Boosting: 0.8377192982456141
Confusion matrix for Test data in Gradient Boosting:
[[ 83  42]
 [ 32 299]]
Classification Report for Test data in Gradient Boosting:
              precision    recall  f1-score   support

           0       0.72      0.66      0.69       125
           1       0.88      0.90      0.89       331

    accuracy                           0.84       456
   macro avg       0.80      0.78      0.79       456
weighted avg       0.83      0.84      0.84       456
```

Table of Accuracy, ROC_AUC Score, Recall, Precision, F1 score for Logit and LDA models:

	Logit Train	Logit Test	LDA Train	LDA Test
Accuracy	0.84	0.83	0.83	0.83
AUC Score	0.89	0.88	0.89	0.89
Recall	0.92	0.88	0.91	0.89
Precision	0.86	0.86	0.86	0.86
F1 Score	0.89	0.87	0.89	0.88

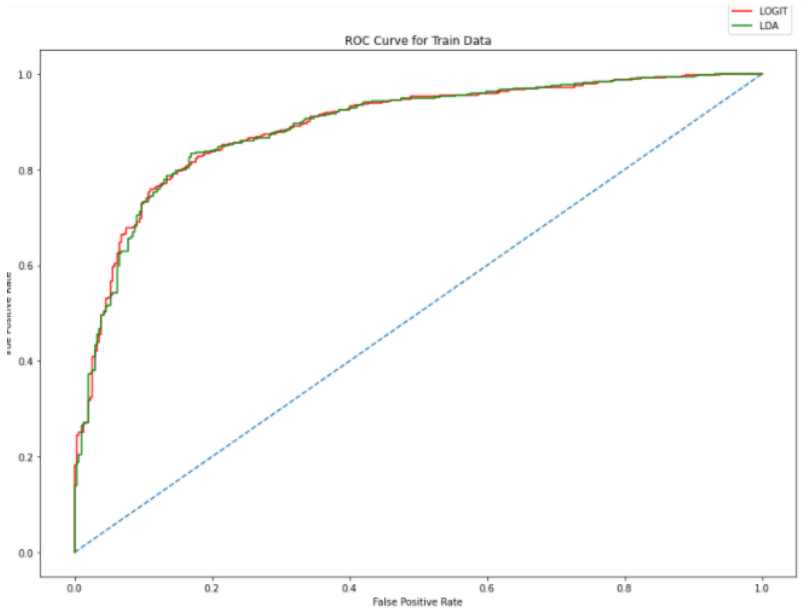
Table of Accuracy, ROC_AUC Score, Recall, Precision, F1 score for KNN and NB models:

	KNN Train	KNN Test	Gaussian NB Train	Gaussian NB Test
Accuracy	0.84	0.83	0.83	0.85
AUC Score	0.90	0.90	0.88	0.90
Recall	0.90	0.89	0.90	0.87
Precision	0.88	0.86	0.88	0.87
F1 Score	0.89	0.88	0.89	0.87

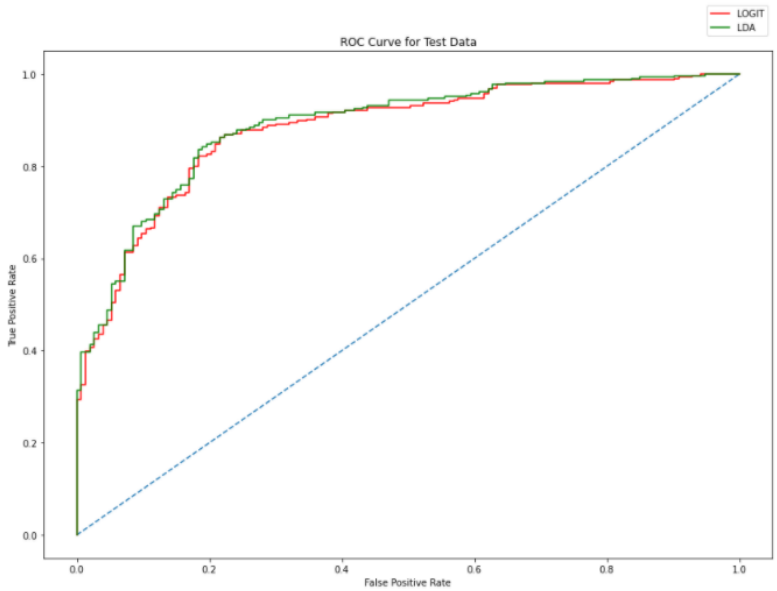
Table of Accuracy, ROC_AUC Score, Recall, Precision, F1 score for RF, Bagging, AdaBoosting and Gradient Boosting models:

	RF Train	RF Test	Bagging Train	Bagging Test	AdaBoost Train	AdaBoost Test	GradientBoost Train	GradientBoost Test
Accuracy	0.85	0.83	0.84	0.83	0.84	0.84	0.89	0.84
AUC Score	0.91	0.89	0.90	0.88	0.91	0.88	0.95	0.88
Recall	0.92	0.91	0.91	0.92	0.91	0.90	0.94	0.91
Precision	0.88	0.84	0.86	0.86	0.87	0.88	0.90	0.88
F1 Score	0.90	0.87	0.89	0.89	0.89	0.89	0.92	0.89

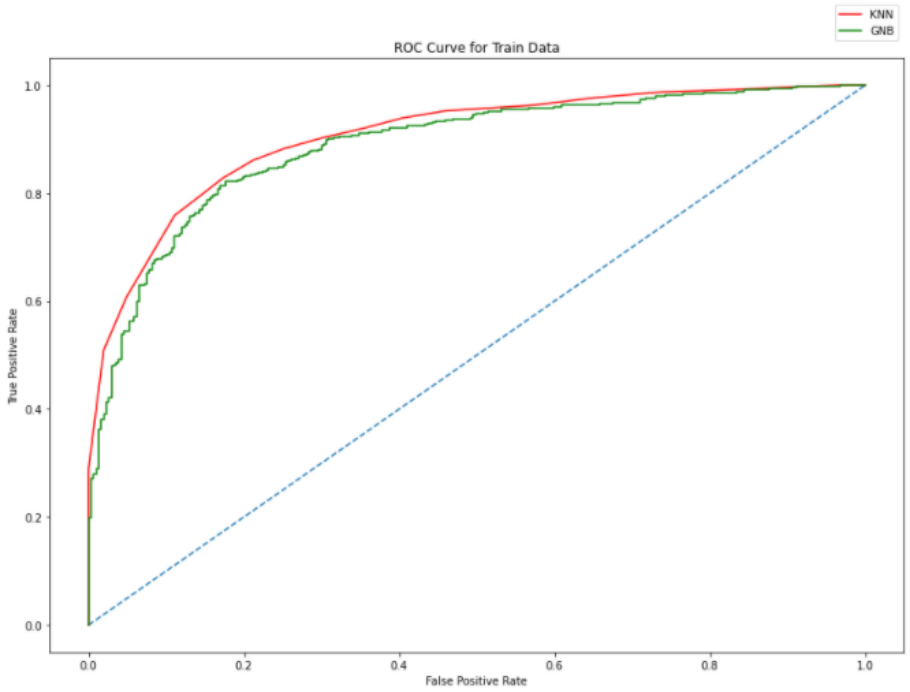
ROC Curve for Train data for Logit and LDA models:



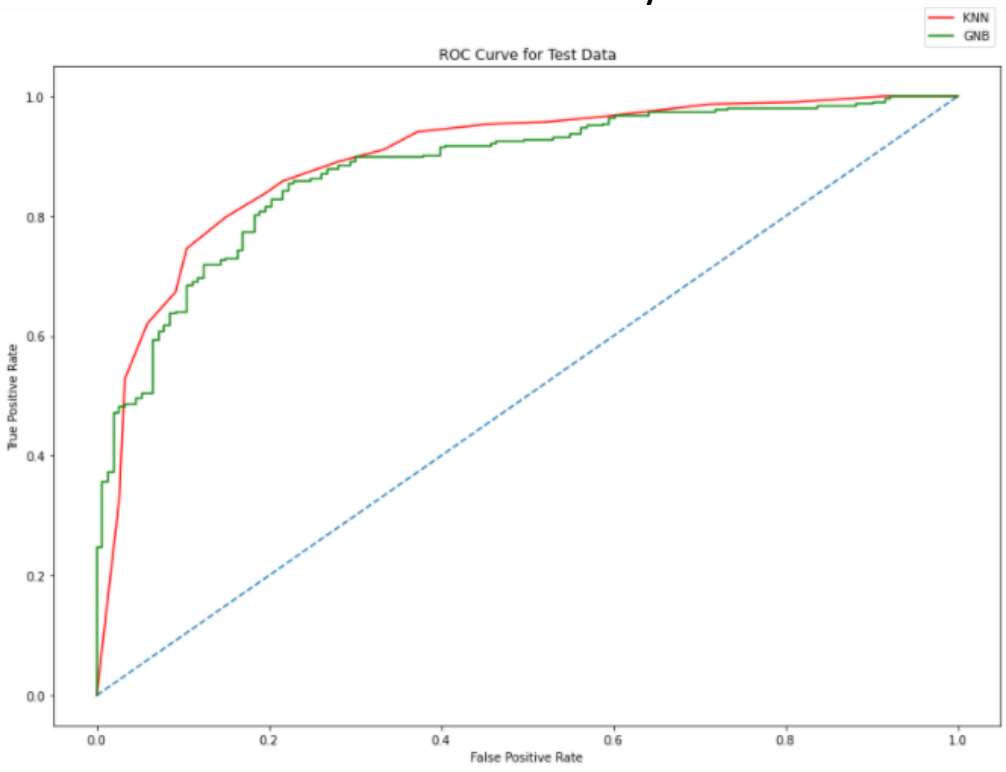
ROC Curve for Test data for Logit and LDA models:



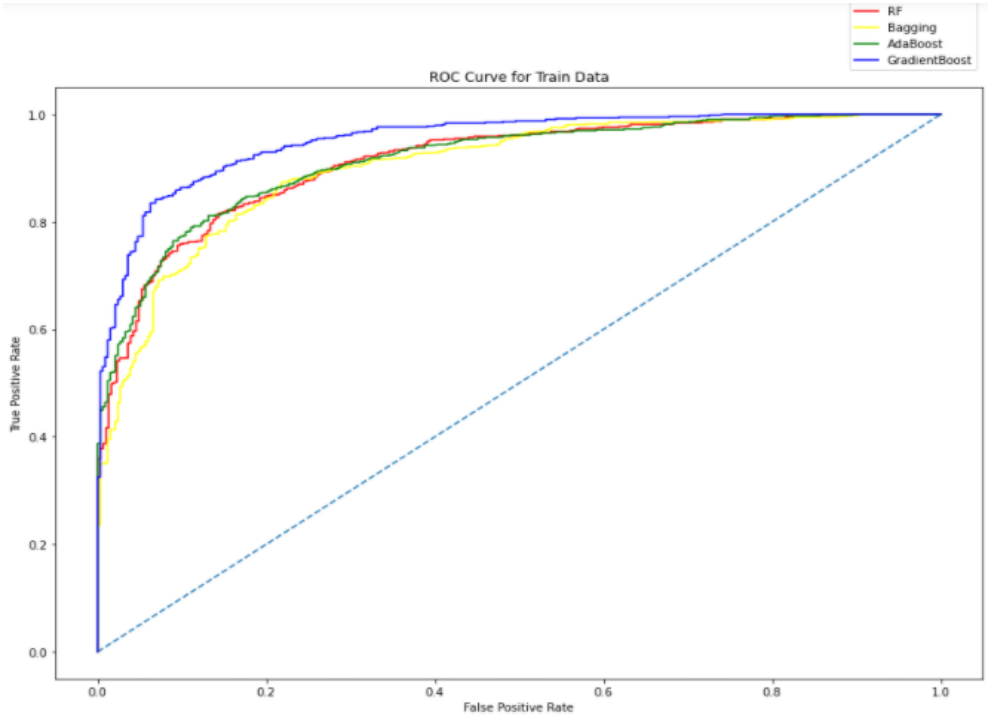
ROC Curve for Train data for KNN and Gaussian Naïve Bayes models:



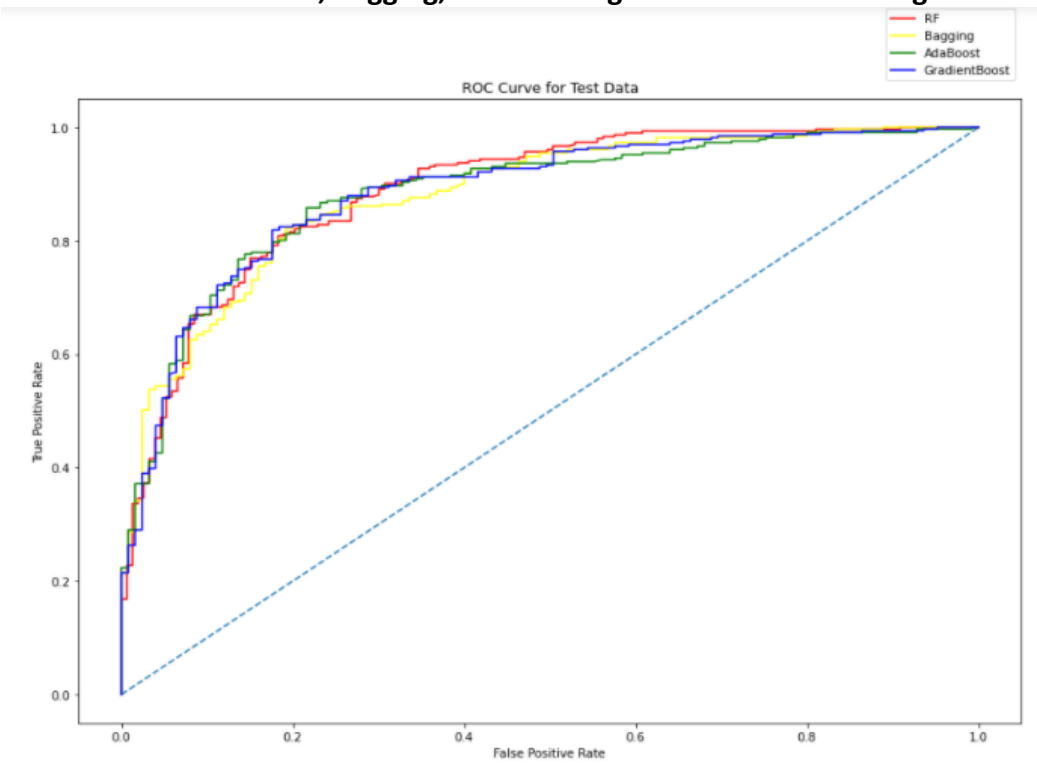
ROC Curve for Test data for KNN and Gaussian Naïve Bayes models:



ROC Curve for Train data for RF, Bagging, AdaBoosting and Gradient Boosting models:



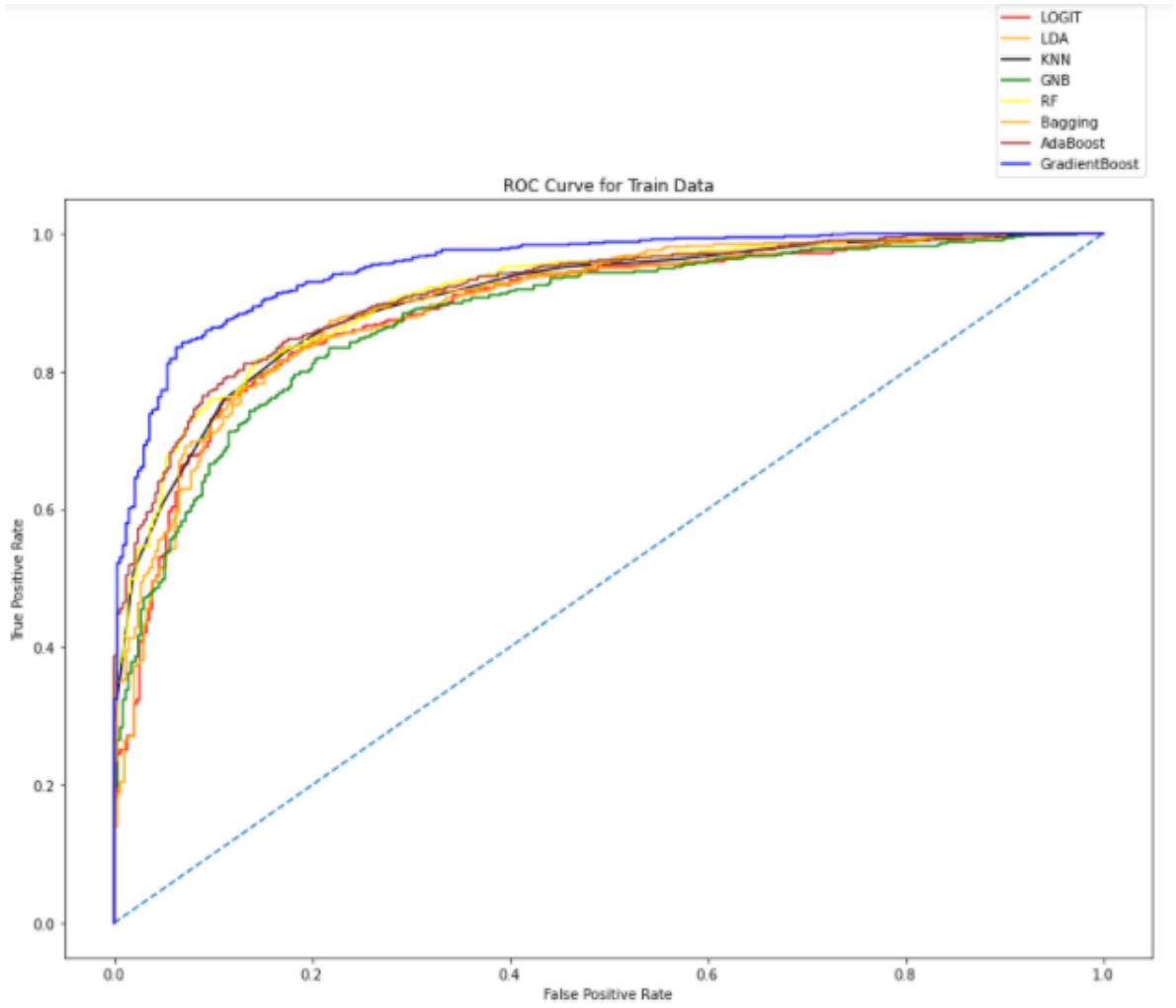
ROC Curve for Test data for RF, Bagging, AdaBoosting and Gradient Boosting models:



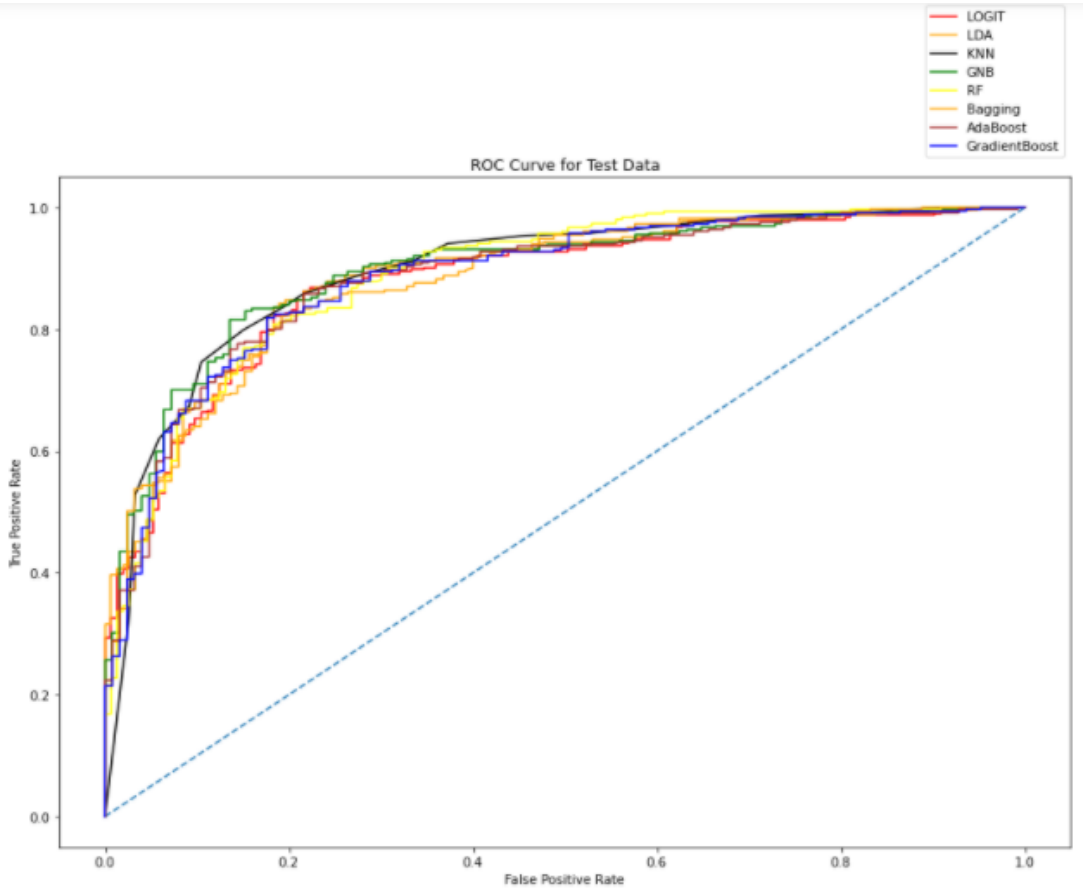
Comparing all the models – Accuracy, AUC score, Recall, Precision, F1 Score:

	Logit Train	Logit Test	LDA Train	LDA Test	KNN Train	KNN Test	Gaussian NB Train	Gaussian NB Test	RF Train	RF Test	Bagging Train	Bagging Test	AdaBoost Train	AdaBoost Test	GradientBoost Train	GradientBoost Test
Accuracy	0.84	0.83	0.83	0.83	0.84	0.83	0.83	0.85	0.85	0.83	1.00	0.82	0.84	0.84	0.89	0.84
AUC Score	0.89	0.88	0.89	0.89	0.90	0.90	0.88	0.90	0.91	0.89	0.90	0.88	0.91	0.88	0.95	0.88
Recall	0.92	0.88	0.91	0.89	0.90	0.89	0.89	0.91	0.92	0.91	1.00	0.87	0.91	0.90	0.94	0.91
Precision	0.86	0.86	0.86	0.86	0.88	0.86	0.86	0.89	0.88	0.84	1.00	0.85	0.87	0.88	0.90	0.88
F1 Score	0.89	0.87	0.89	0.88	0.89	0.88	0.88	0.90	0.90	0.87	0.89	1.00	0.86	0.89	0.92	0.89

Comparing all the models – ROC Curve for the Train Data:



Comparing all the models – ROC Curve for the Test Data:



- The Gradient Boosting model gives the highest and best prediction results compared to all other models.
- It gives 89% accuracy for train data and 84% accuracy for test data.
- The overall prediction is better for all the metrics in the Gradient Boosting model.

	GradientBoost Train	GradientBoost Test
Accuracy	0.89	0.84
AUC Score	0.95	0.88
Recall	0.94	0.91
Precision	0.90	0.88
F1 Score	0.92	0.89

- The Gradient Naïve Bayes model has good prediction rate of test data than with the train data.

	Gradient NB Train	Gradient NB Test
Accuracy	0.83	0.85
AUC Score	0.88	0.90
Recall	0.89	0.91
Precision	0.86	0.89
F1 Score	0.88	0.90

- Still the evalution for train and test data is better predicted by the Gradient Boosting model.

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

Business Insights and Recommendations:

- We check the classes in the target variable.

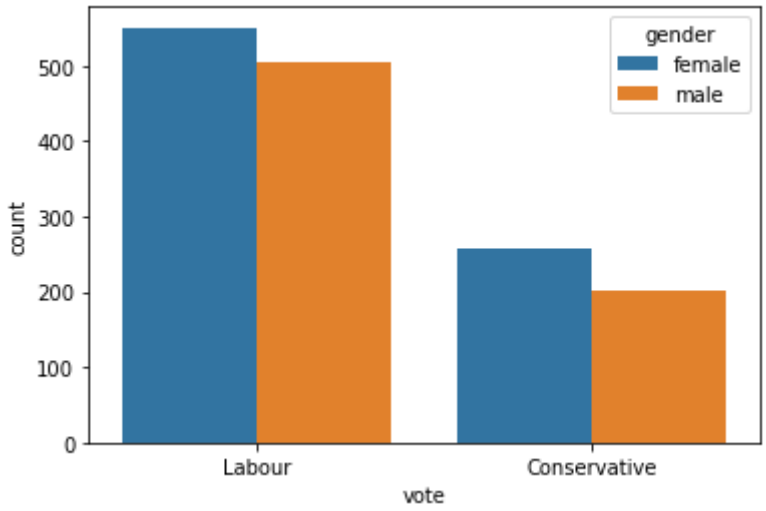
```
VOTE :  2
Conservative     460
Labour          1057
Name: vote, dtype: int64
```

- From this we can say that the Labour party has won the election (by survey) with nearly 70% of the total votes.
- We can see that there is class imbalance in the dataset, as per survey many people have voted for Labour party more than twice the Conservative party.
- Despite the class imbalance, we get better model prediction for both the train set and test set.
- The Gradient Boosting model gives the highest and best prediction results compared to all other models.
- Considering that the Blair and Hague is a categorical variable with 1 being the worst and 5 being the best.
- The ***Blair*** is voted for about 64% votes with 4th grade rating for the Labour party, this tells that the Labour party was more favoured leader in the election.
- The ***Hague*** is voted for about 52% negative voting for the Labour party, and he was very minimal chances for getting elected by the people.

vote	Conservative	Labour
Blair		
1	59	38
2	240	194
3	1	0
4	157	676
5	3	149

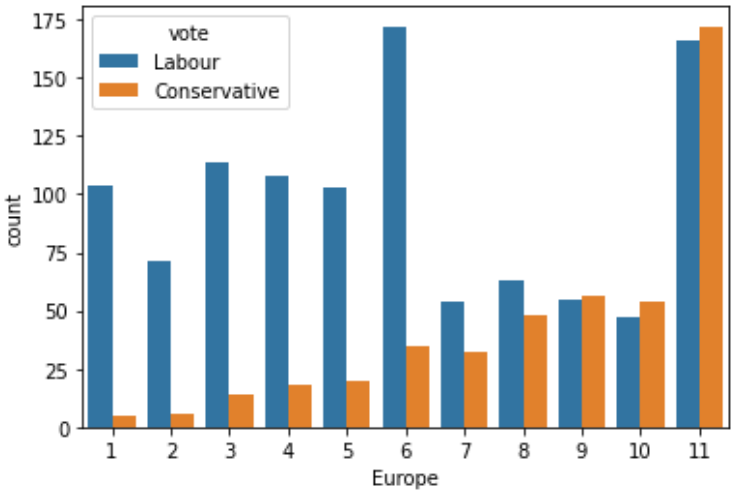
vote	Conservative	Labour
Hague		
1	11	222
2	95	522
3	9	28
4	286	271
5	59	14

- In the gender wise calculation of voting, nearly 68% of the female vote to the Labour party and 71% of the male vote to the Labour party.



- The people feel that the Labour party helps in the European integration of about
- The Conservative party is more skeptical in the integration than the Labour party.

vote	Conservative	Labour
Europe		
1	5	104
2	6	71
3	14	114
4	18	108
5	20	103
6	35	172
7	32	54
8	48	63
9	56	55
10	54	47
11	172	166



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.

- We can find the number of Characters by applying the .raw() function for that document and use len() function to get the count.
- We can find the number of Words by applying the .words() function for that document and use len() function to get the count.
- To find the number of Words, another method is to apply word_tokenize() from nltk library and use .len() function to get the count.
- We can find the number of Sentences by applying the .sents() function for that document and use len() function to get the count.
- To find the number of Sentences, another method is to apply sent_tokenize() from nltk library and use .len() function to get the count.

In President Franklin D. Roosevelt in 1941:

```
Total Number of Characters in "1941-Roosevelt.txt" speech: 7571
Total Number of Words in "1941-Roosevelt.txt" speech: 1536
Total Number of Sentences in "1941-Roosevelt.txt" speech: 68
```

In President John F. Kennedy in 1961:

```
Total Number of Characters in "1961-Kennedy.txt" speech: 7618
Total Number of Words in "1961-Kennedy.txt" speech: 1546
Total Number of Sentences in "1961-Kennedy.txt" speech: 52
```

In President Richard Nixon in 1973:

```
Total Number of Characters in "1973-Nixon.txt" speech: 9991
Total Number of Words in "1973-Nixon.txt" speech: 2028
Total Number of Sentences in "1973-Nixon.txt" speech: 69
```

2.2. Remove all the stopwords from all the three speeches.

Cleaning the Text:

- As a part of cleaning the texts in the speech, we lower the case of all the texts for easy handling of the text mining.
- We can get the list of punctuations from the string.punctuation module and additional punctuation '- -' is added to the list before removing the punctuations.
- We remove the stopwords from the corpus module in nltk library.

nltk.corpus.stopwords.words('english')

- We can see that the words 'let' and 'us' is occurring in the document list even after removing the stopwords, hence we add them in the list of stopwords to be removed if required.

Roosevelt in 1941:

```
['national',
 'day',
 'inauguration',
 'since',
 '1789',
 'people',
 'renewed',
 'sense',
 'dedication',
 'united',
 'states',
 'washington',
 'day',
 'task',
 'people',
 'create',
 'weld',
 'together',
 'nation',
```

Kennedy in 1961:

```
['vice',
 'president',
 'johnson',
 'mr',
 'speaker',
 'mr',
 'chief',
 'justice',
 'president',
 'eisenhower',
 'vice',
 'president',
 'nixon',
 'president',
 'truman',
 'reverend',
 'clergy',
 'fellow',
 'citizens',
```

Richard Nixon in 1973:

```
['mr',
 'vice',
 'president',
 'mr',
 'speaker',
 'mr',
 'chief',
 'justice',
 'senator',
 'cook',
 'mrs',
 'eisenhower',
 'fellow',
 'citizens',
 'great',
 'good',
 'country',
 'share',
 'together',
```

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).

- Removing the punctuations and stopwords in each document.
- We find the most frequent words in each document.

In President Franklin D. Roosevelt in 1941:

```
The frequent occuring words in "1941-Roosevelt.txt" speech:

FreqDist({'nation': 12, 'know': 10, 'spirit': 9, 'life': 9, 'democracy': 9, 'us': 8, 'people': 7, 'america': 7, 'years': 6, 'fr
eedom': 6, ...})
```

In President John F. Kennedy in 1961:

```
The frequent occuring words in "1961-Kennedy.txt" speech:

FreqDist({'let': 16, 'us': 12, 'world': 8, 'sides': 8, 'new': 7, 'pledge': 7, 'citizens': 5, 'power': 5, 'shall': 5, 'free': 5,
...})
```

In President Richard Nixon in 1973:

```
The frequent occuring words in "1973-Nixon.txt" speech:

FreqDist({'us': 26, 'let': 22, 'america': 21, 'peace': 19, 'world': 18, 'new': 15, 'nation': 11, 'responsibility': 11, 'governm
ent': 10, 'great': 9, ...})
```

- The top three most occurring words in each documents is taken as below.

```
The top 3 frequent occuring words in "1941-Roosevelt.txt" speech:
['nation', 'know', 'spirit']
 The top 3 frequent occuring words in "1961-Kennedy.txt" speech:
 ['let', 'us', 'world']
The top 3 frequent occuring words in "1973-Nixon.txt" speech:
['us', 'let', 'america']
```

- We can remove the words ‘let’ and ‘us’ from the document as they are stopwords.

In President Franklin D. Roosevelt in 1941:

```
The frequent occuring words in "1941-Roosevelt.txt" speech:
FreqDist({'nation': 12, 'know': 10, 'spirit': 9, 'life': 9, 'democracy': 9, 'people': 7, 'america': 7, 'years': 6, 'freedom': 6, 'human': 5, ...})
```

In President John F. Kennedy in 1961:

```
The frequent occuring words in "1961-Kennedy.txt" speech:
FreqDist({'world': 8, 'sides': 8, 'new': 7, 'pledge': 7, 'citizens': 5, 'power': 5, 'shall': 5, 'free': 5, 'nations': 5, 'ask': 5, ...})
```

In President Richard Nixon in 1973:

```
The frequent occuring words in "1973-Nixon.txt" speech:
FreqDist({'america': 21, 'peace': 19, 'world': 18, 'new': 15, 'nation': 11, 'responsibility': 11, 'government': 10, 'great': 9, 'home': 9, 'abroad': 8, ...})
```

- The top three most occurring words in each documents after removing the ‘let’ and ‘us’ as stopwords.

```
The top 3 frequent occuring words in "1941-Roosevelt.txt" speech:
['nation', 'know', 'spirit']

The top 3 frequent occuring words in "1961-Kennedy.txt" speech:
['world', 'sides', 'new']

The top 3 frequent occuring words in "1973-Nixon.txt" speech:
['america', 'peace', 'world']
```

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

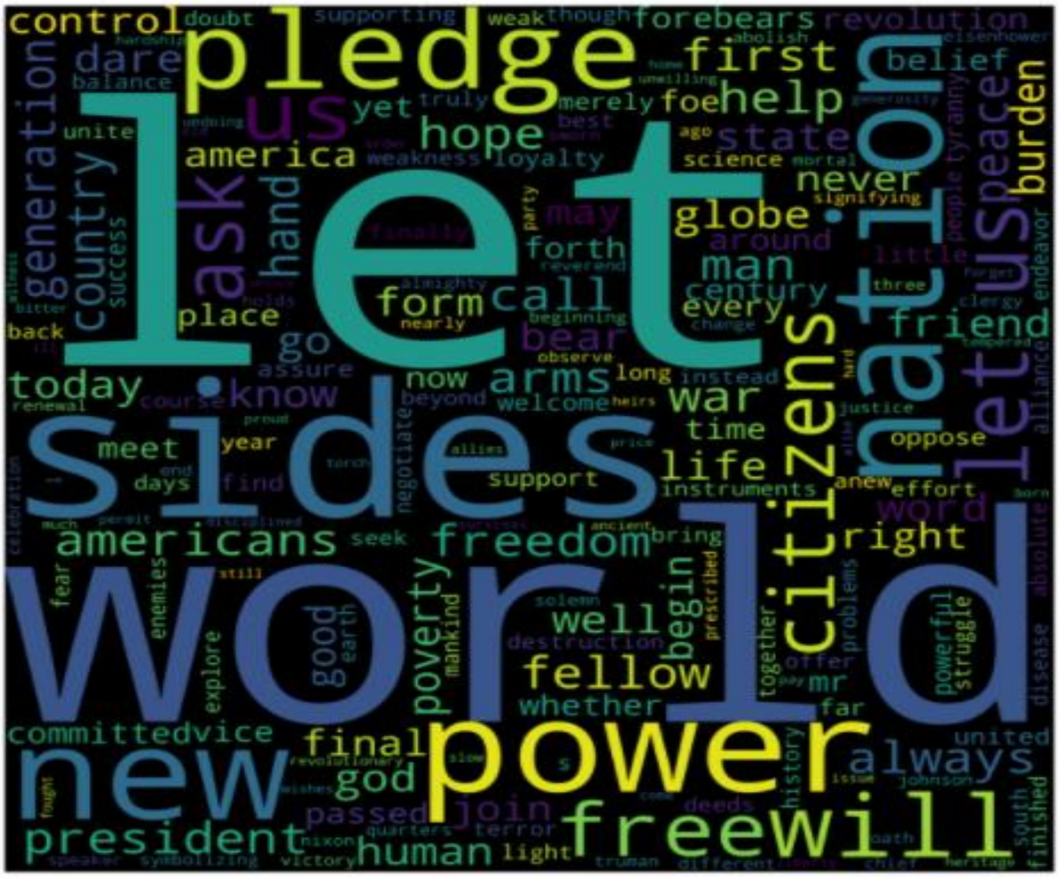
- We plot the word cloud using the WordCloud module in the wordcloud library.
- We use the WordCloud function as below,

wordcloud = WordCloud(width = 3000, height = 3000,
background_color ='black',
min_font_size = 10, random_state=100).generate(n_wc_a)

- Passing the corpus in the generate () function.

Word Cloud for President Franklin D. Roosevelt-1941 speech (after cleaning)!!

Word Cloud for President John F. Kennedy-1961 speech (after cleaning)!!



Word Cloud for President Richard Nixon-1973 speech (after cleaning)!!