INFORMATICS INSTITUTE OF TECHNOLOGY

In Collaboration with

ROBERT GORDON UNIVERSITY ABERDEEN

Level: Level 05
Module Code: CM-2604
Module: Machine Learning
Module Leader: Mr. Prasan Yapa

Due Date: 26th of March 2023

Machine Learning Coursework Report

Dineth Hasaranga
IIT ID – 20210537
RGU ID- 2117526

Introduction

This is the report on the coursework for machine learning. The primary goal of this assignment is to conduct a classification to label emails as spam or non-spam based on the word content. Two algorithms KNN and Decision Tree Classification were applied to accomplish this categorization. UCI Machine Learning repository was used to get the dataset

Dataset

The spam-non spam dataset, which has over 4601 rows and 57 characteristics, was used to train the model.

Source of the Dataset	UCI Machine learning repository.
Number of instances	4601
Number of attributes	55-57
Missing values	Yes
Number of classes	02
Relatable Tasks	Classification

Pre – processing techniques

Data Cleaning

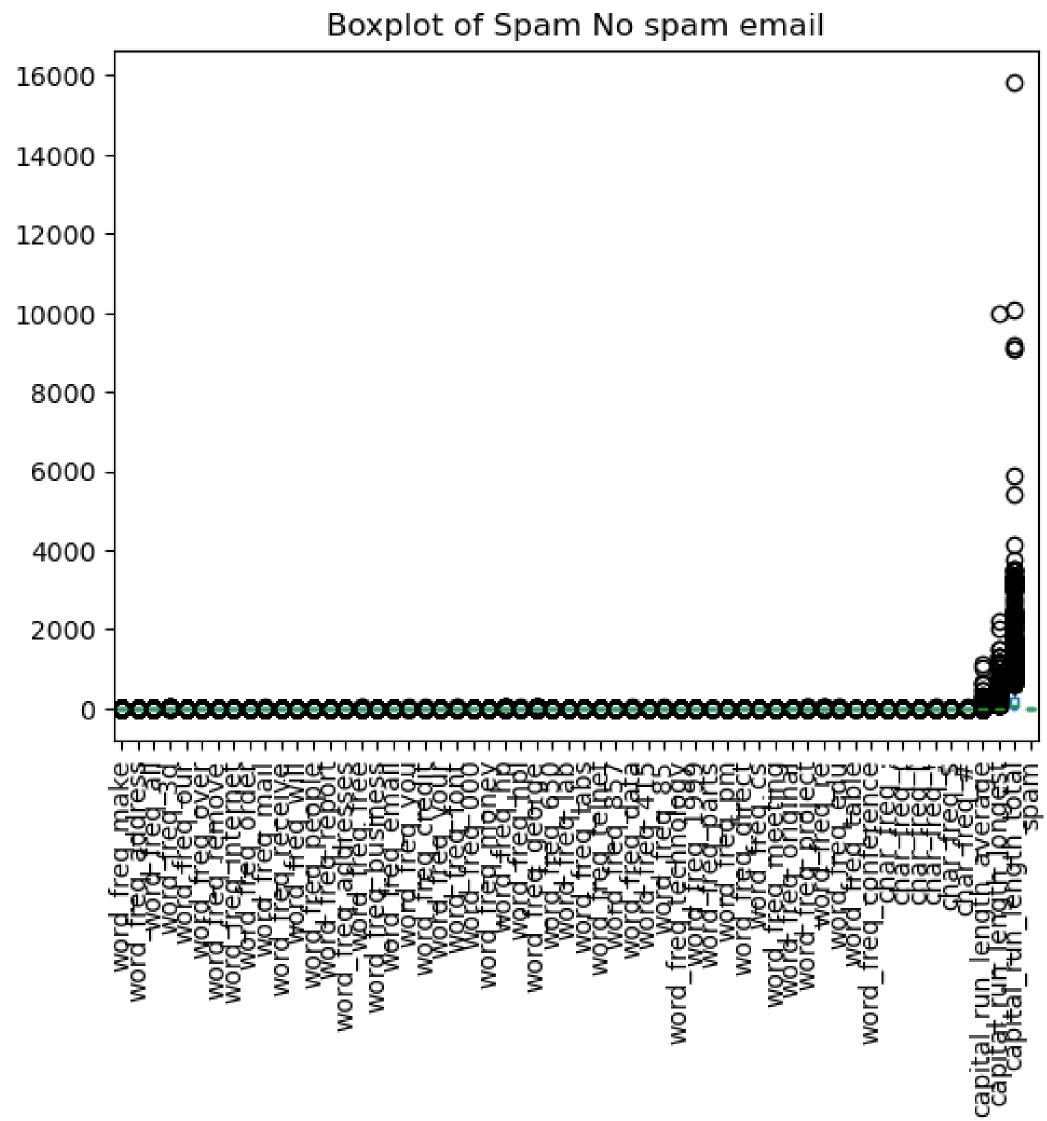
Data cleaning is achieved by removing null duplicates from the Spam base dataset.

No of rows in dataset before removing duplicates	No of rows in dataset after removing duplicates
4601	4210

Data transformation

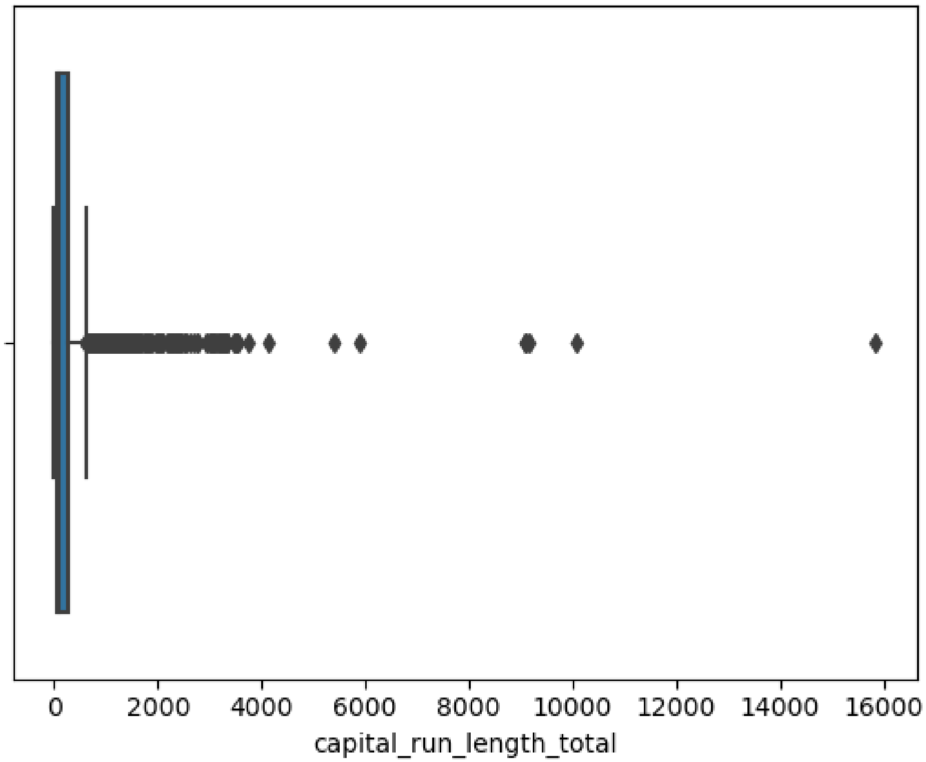
Data transformation is achieved by removing outliers from the dataset and by performing Standard Scaling using *"sklearn.preprocessing import StandardScaler".* The detected outliers were converted to null values and removed from the dataset.

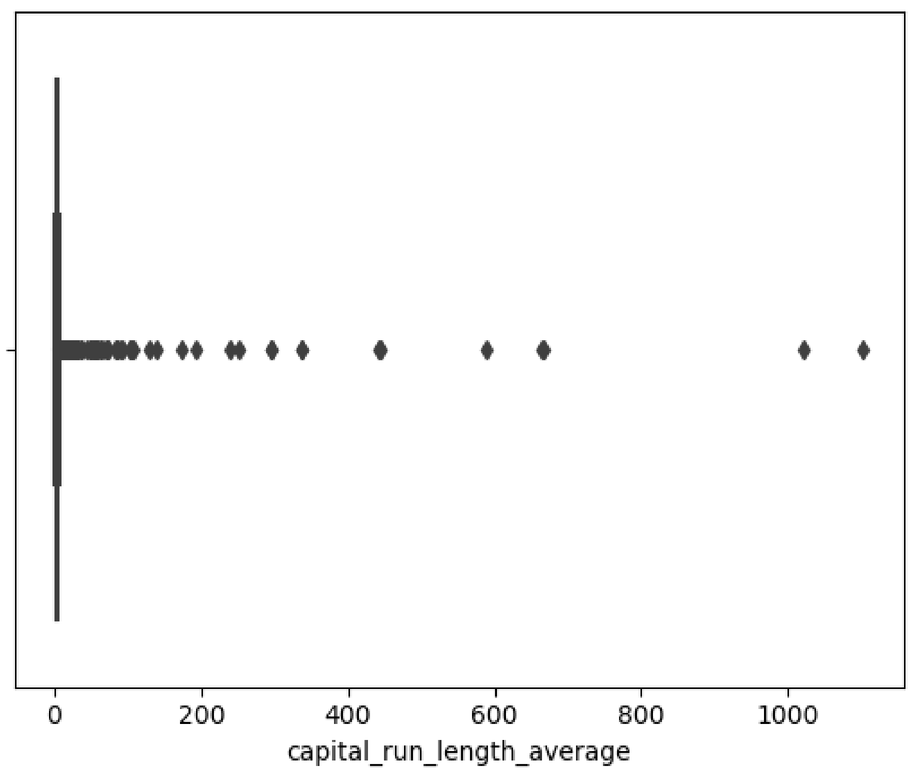
No of rows in dataset before removing outliers	No of rows in dataset after removing outliers
4210	3446

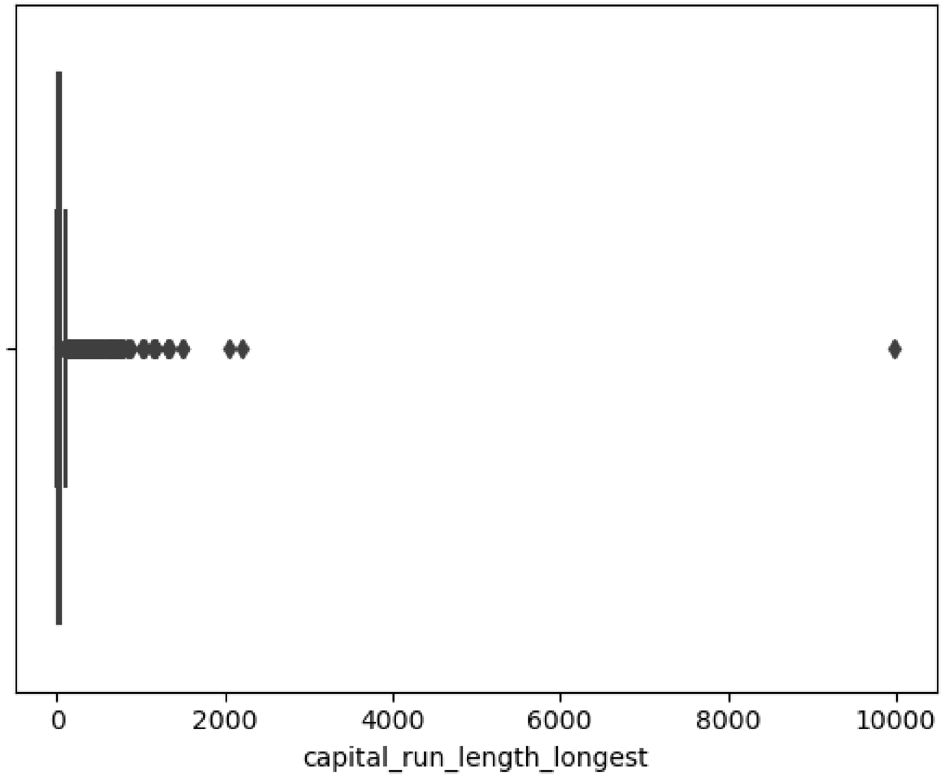


Above box plot shows that outliers are present in the columns; capital_run_length_total, capital_run_length_average and capital_run_length_longest.

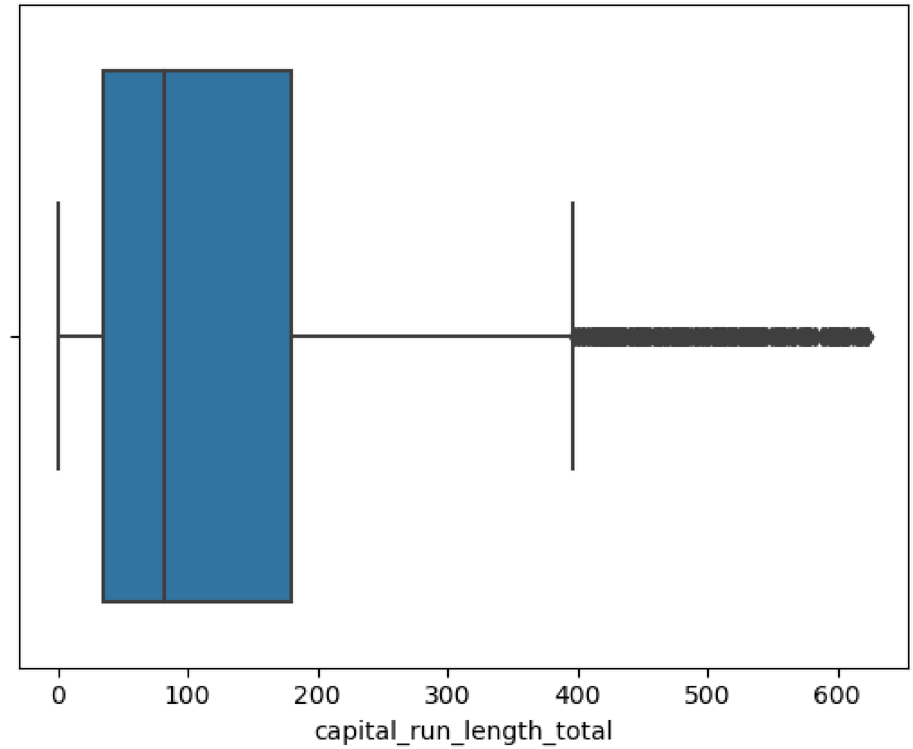
Below are the individual box plots of the above columns before removing the outliers.

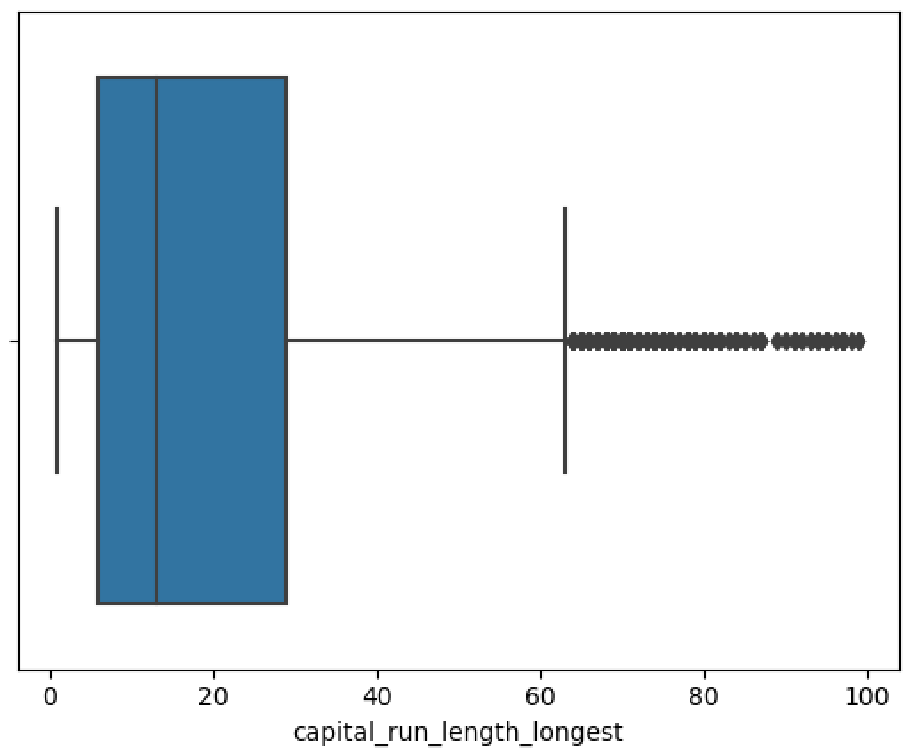


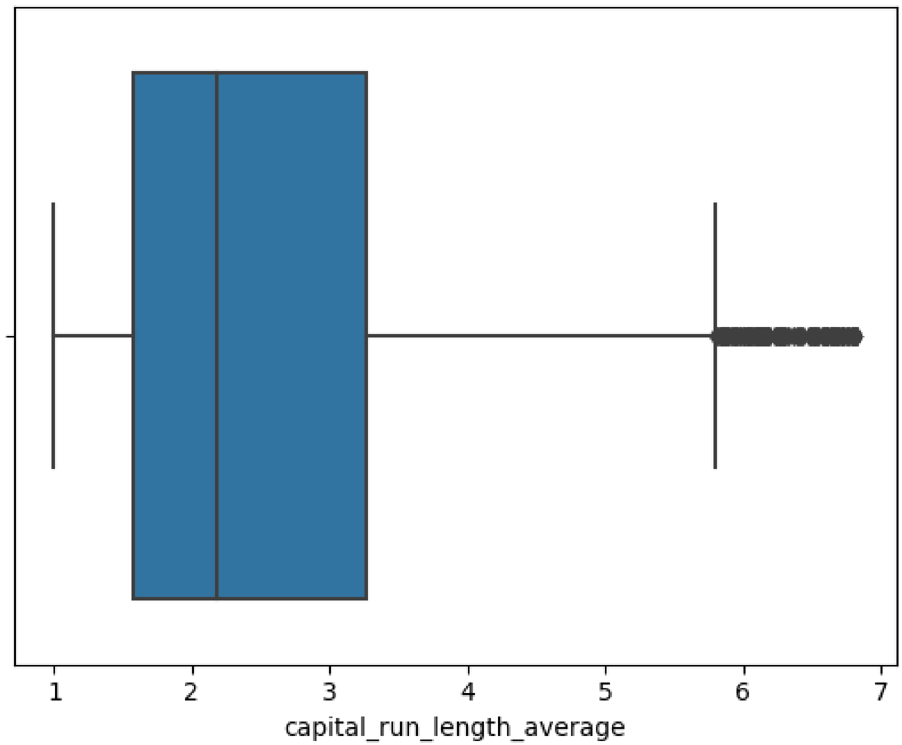



0
2000
4000
6000
8000
10000
capital_run_length_longest

Below are the individual box plots of the above columns after removing the outliers.







Standard Scaling and relevant Visualizations

A KDE (Kernel Density Plot) plot can be used to display the distribution of a feature within the context of the Spam Base dataset.

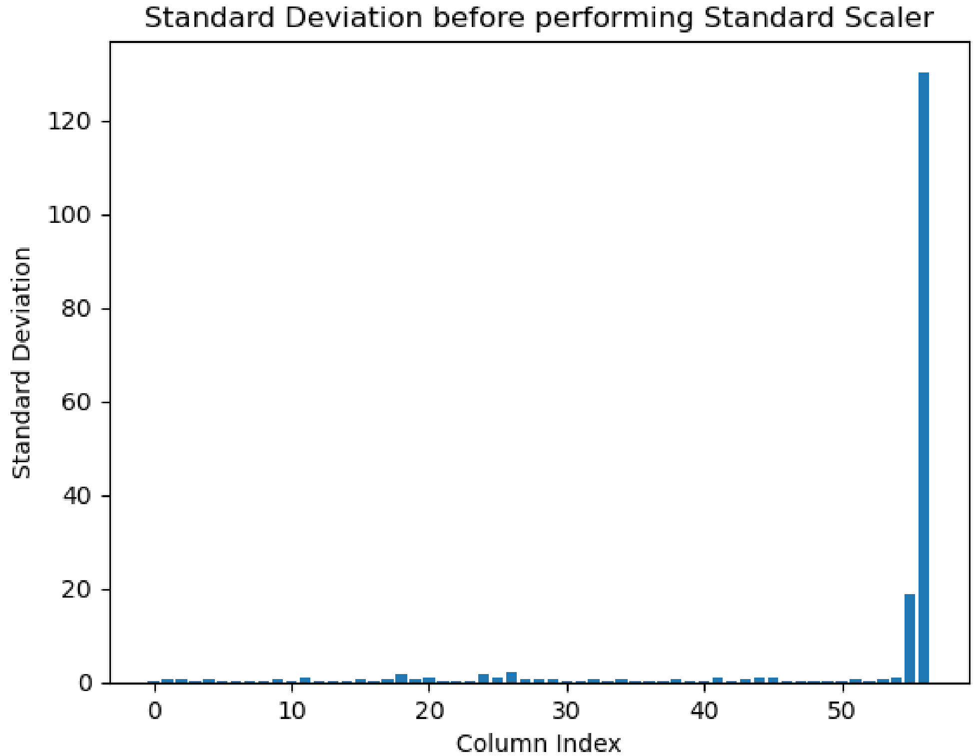
Each feature in the Spam Base dataset has a range of values that can vary dramatically before conventional scaling. This can make it difficult to compare the distribution of different features using a KDE plot.

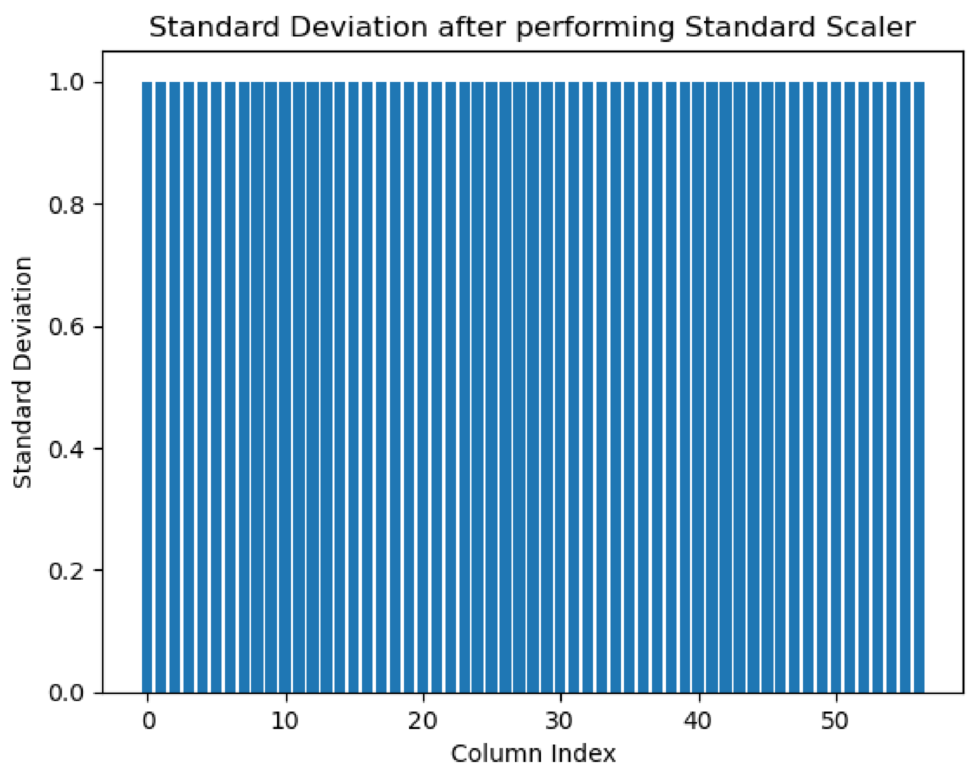
After performing standard scaling, however, the range of values of each feature is normalized to have zero mean and unit variance.

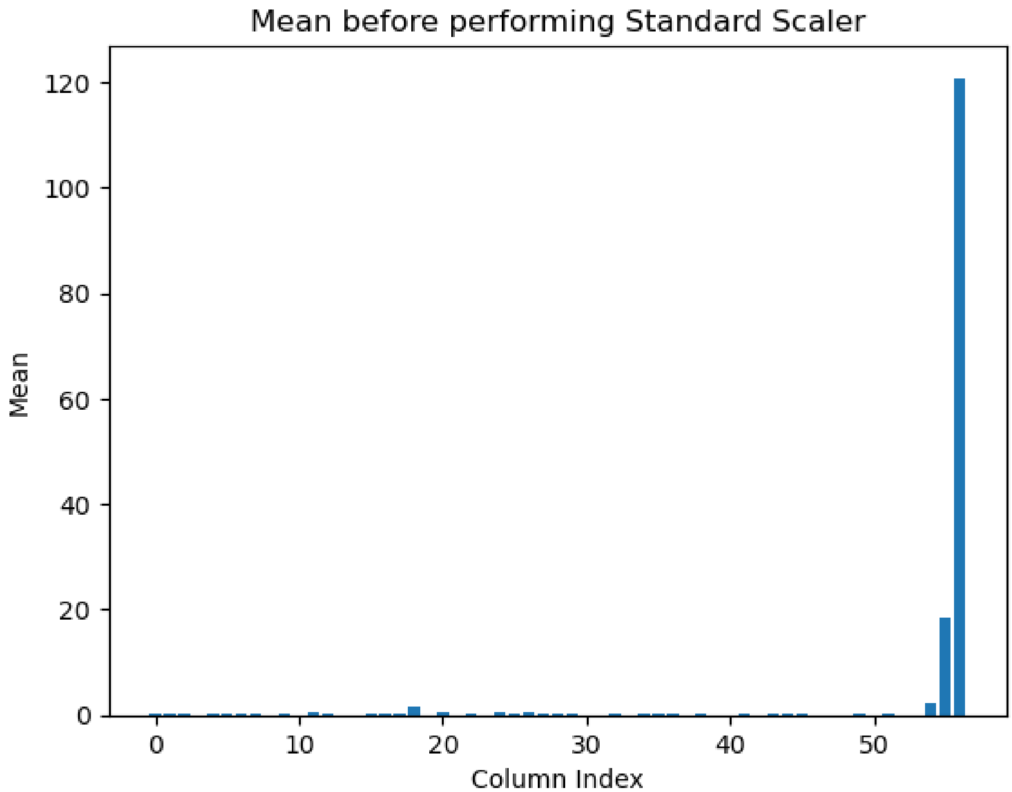
As a result, the scales of the features are now comparable, which makes it simpler to compare the distributions of the features using a KDE plot.

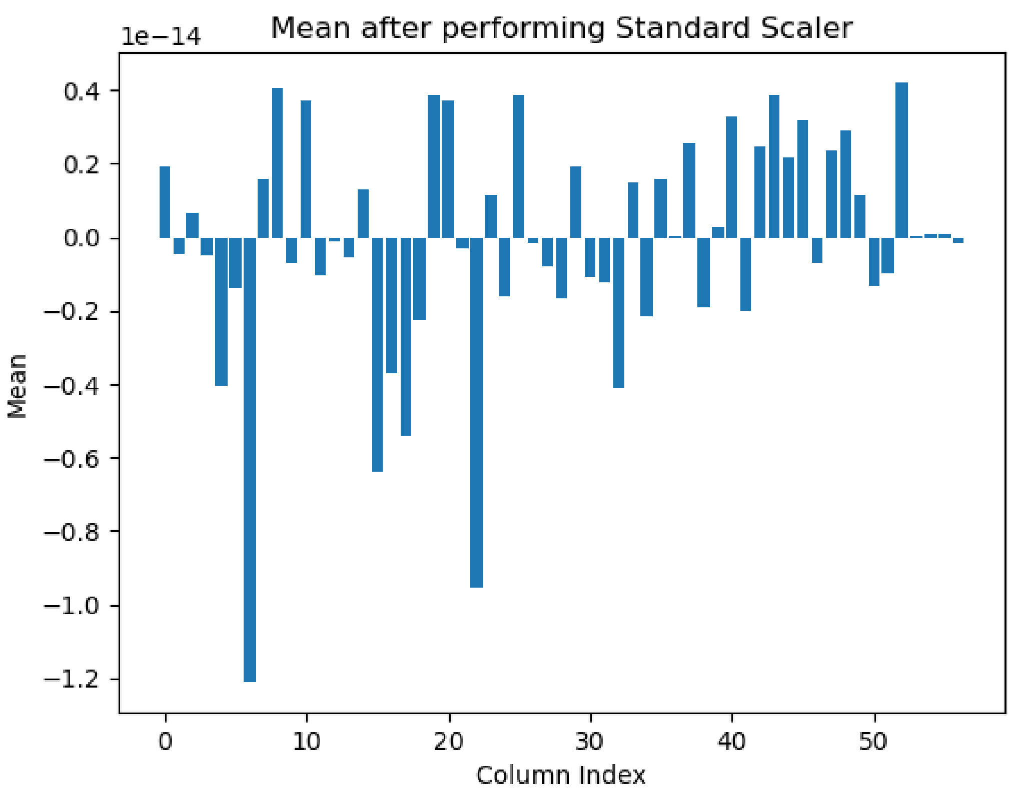
The x-axis represents the values of the feature of the Spam Base dataset, and the y-axis represents the estimated density of those values.

Changes achieved after performing Standard Scaler.

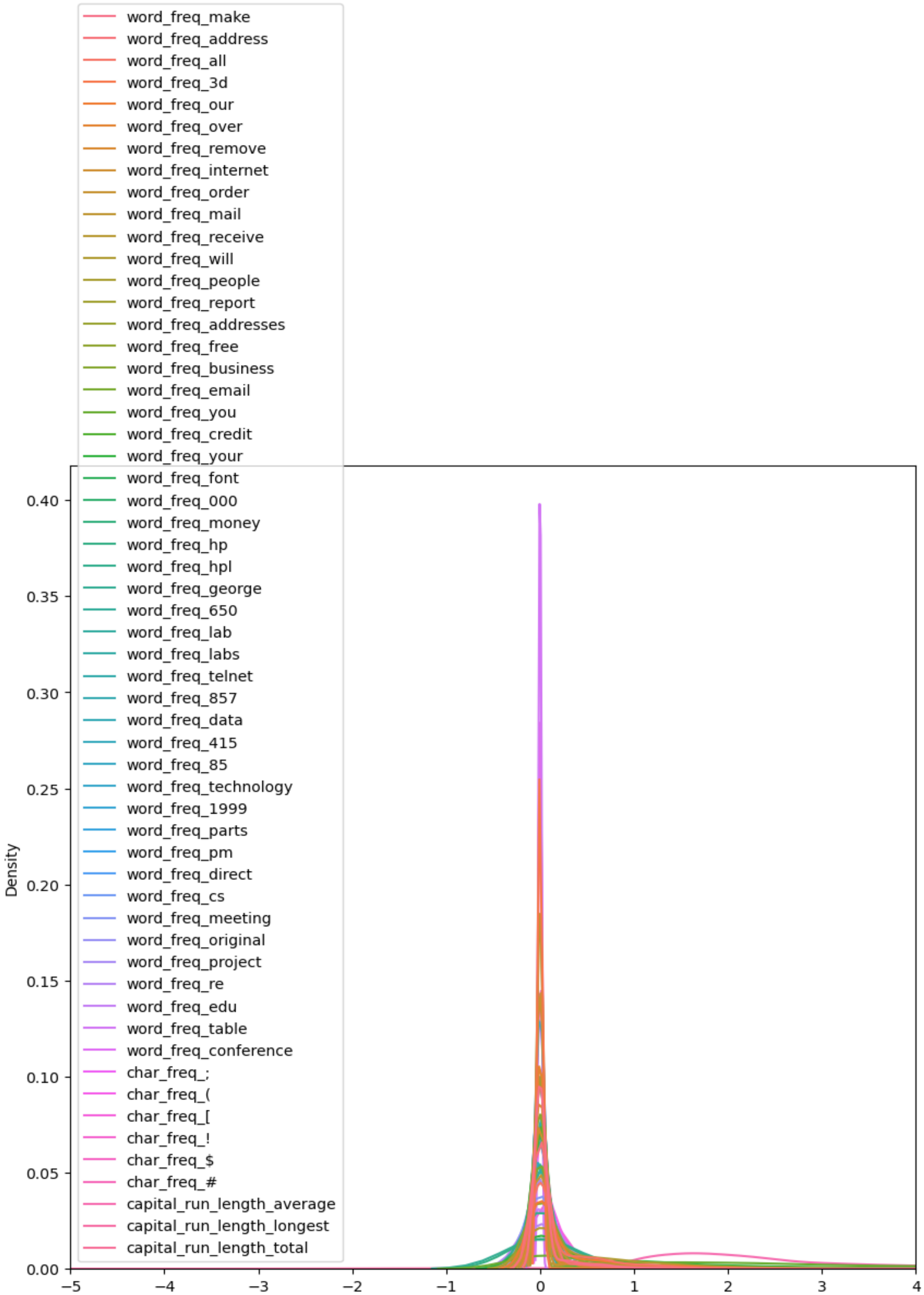




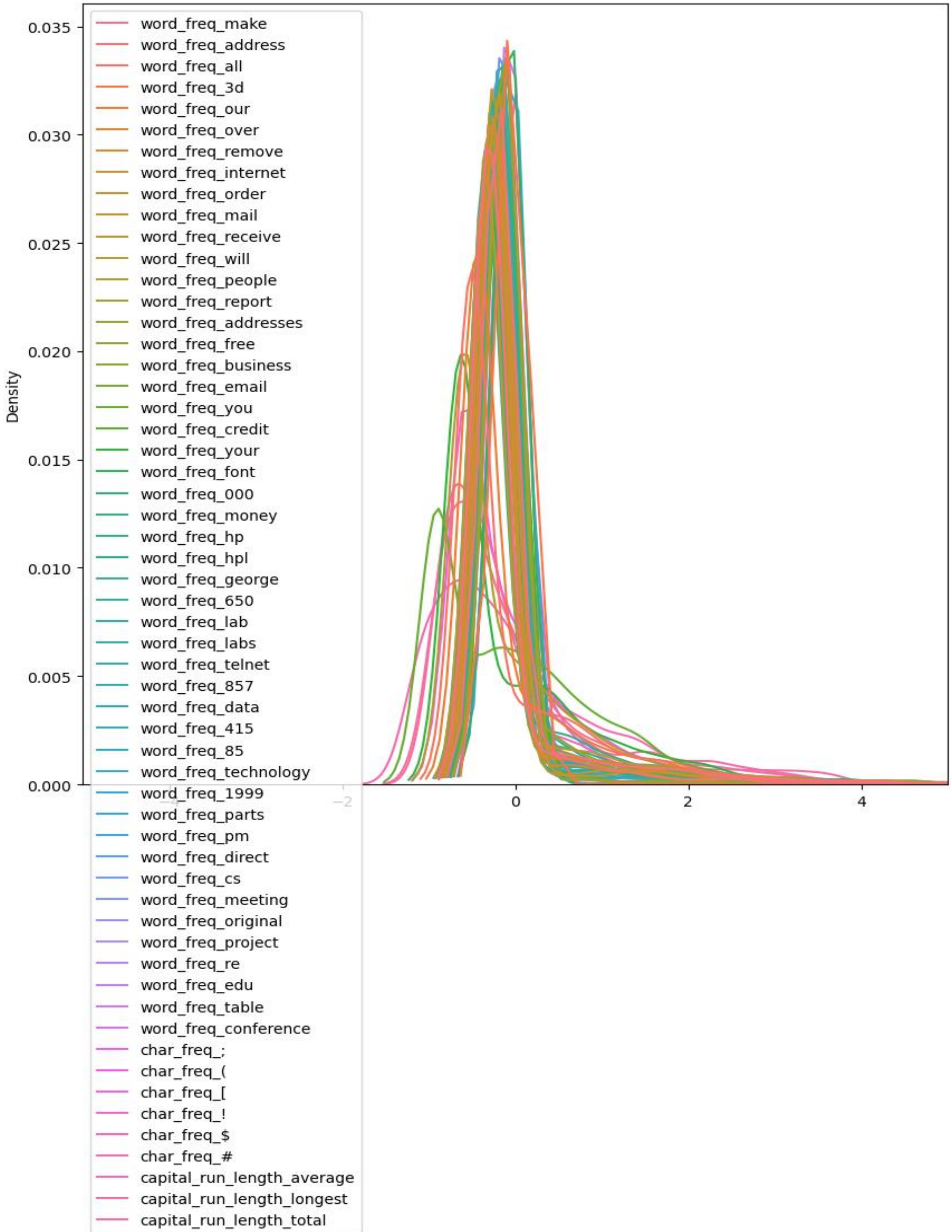

Mean before performing Standard Scaler
Mean
Column Index
0
20
40
60
80
100
120
0
10
20
30
40
50


Mean after performing Standard Scaler
1e−14
Mean
Column Index
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−1.2
0
10
20
30
40
50

Kernel Density Estimate plot before Standard Scaling



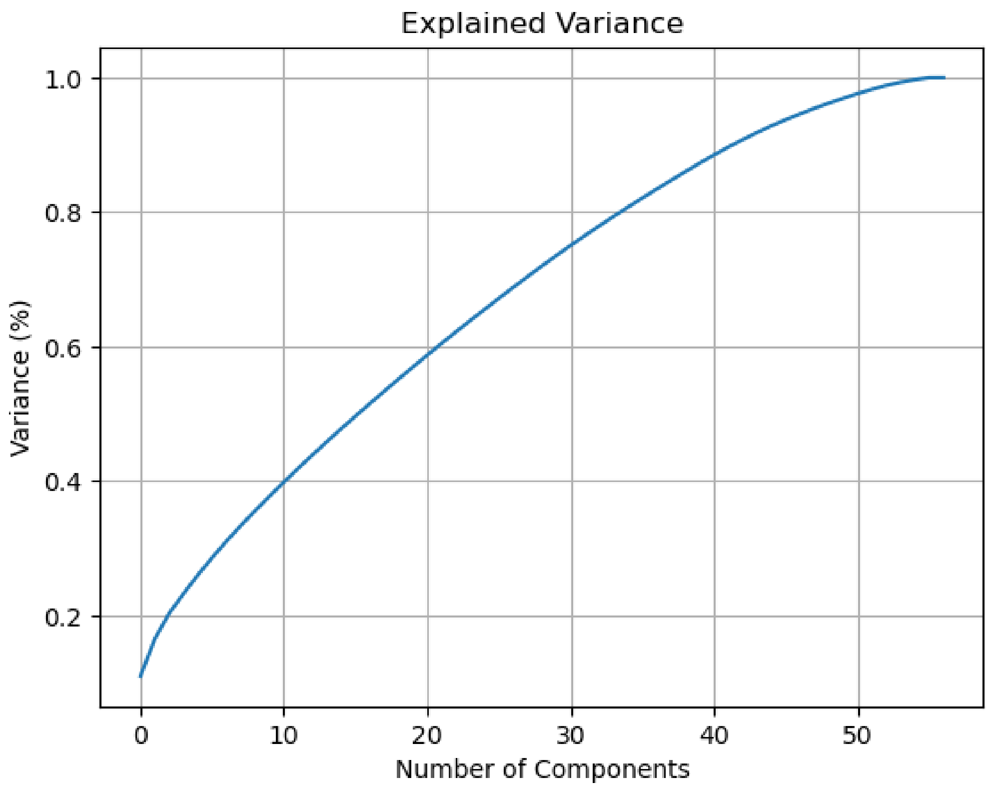
Kernel Density Estimate plot after Standard Scaling



Dimensionality reduction techniques

Dimensionality reduction is a feature selection approach that allows us to use fewer features than the original dataset while maintaining a high level of information in the final model.

Principal component analysis is used to pick features using dimensionality reduction methods. As implied by the name, it extracts the primary components from the data.



From the diagram above, 44 principal components explain almost 90% of the variance in data.

So, instead of giving all the columns as inputs, 44 principal components of the data are entered to the machine learning algorithm.

Splitting the dataset into the Training set and Test set

The training set is given 80% of the dataset and testing set is given 20% of the dataset.

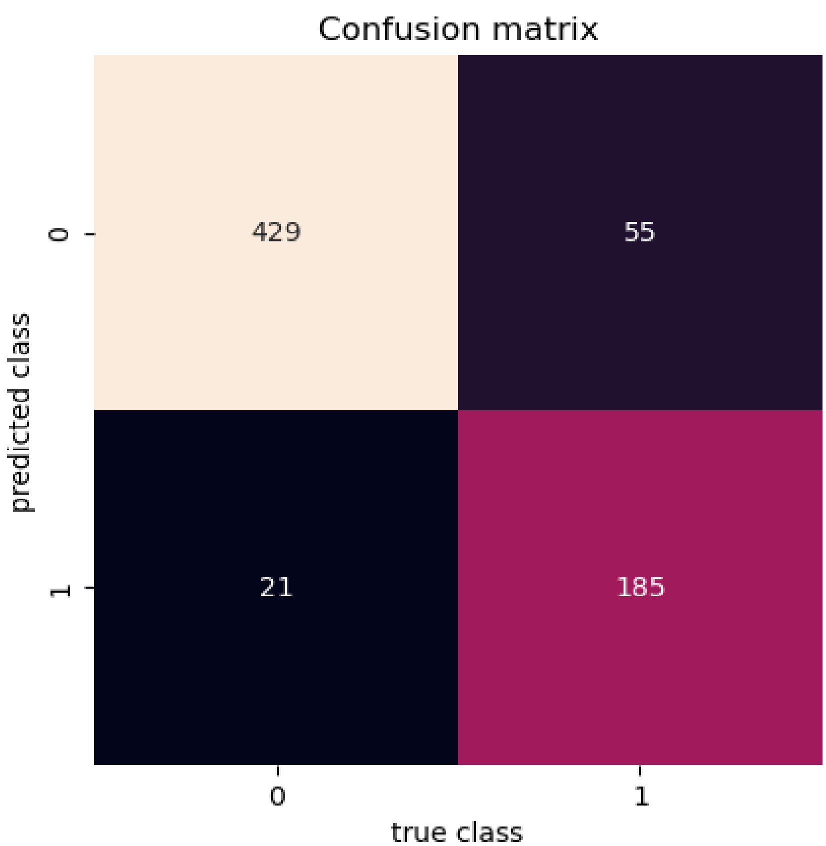
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
```

K Nearest Neighbors (KNN) Classification

Classification Report of KNN

```
Classification Report :

              precision    recall  f1-score   support

           0       0.89      0.95      0.92       450

           1       0.90      0.77      0.83       240

    accuracy                           0.89       690
   macro avg       0.89      0.86      0.87       690
weighted avg       0.89      0.89      0.89       690
```

Confusion matrix of KNN



True Positive: 429 mails are predicted as Not Spam and it is correct.

False Positive: 55 mails are predicted as Spam, but it is Not Spam.

True Negative: 21 mails are predicted as Spam, and it is correct.

False Negative: 185 mails are predicted as Spam, but it is Not spam.

Accuracy of testing dataset

Accuracy score of email prediction using KNN : 88.98550724637681

Accuracy of training dataset

Accuracy score of email prediction using KNN : 92.8156748911466

Decision Tree Classification

Accuracy of testing dataset of Decision Tree Classification

Accuracy score of email prediction using Decision Trees = 0.8695652173913043

Summary of the test dataset

```
Classification Report :

              precision    recall  f1-score   support

           0       0.88      0.92      0.90       450
           1       0.84      0.78      0.81       240

    accuracy                           0.87       690
   macro avg       0.86      0.85      0.85       690
weighted avg       0.87      0.87      0.87       690
```

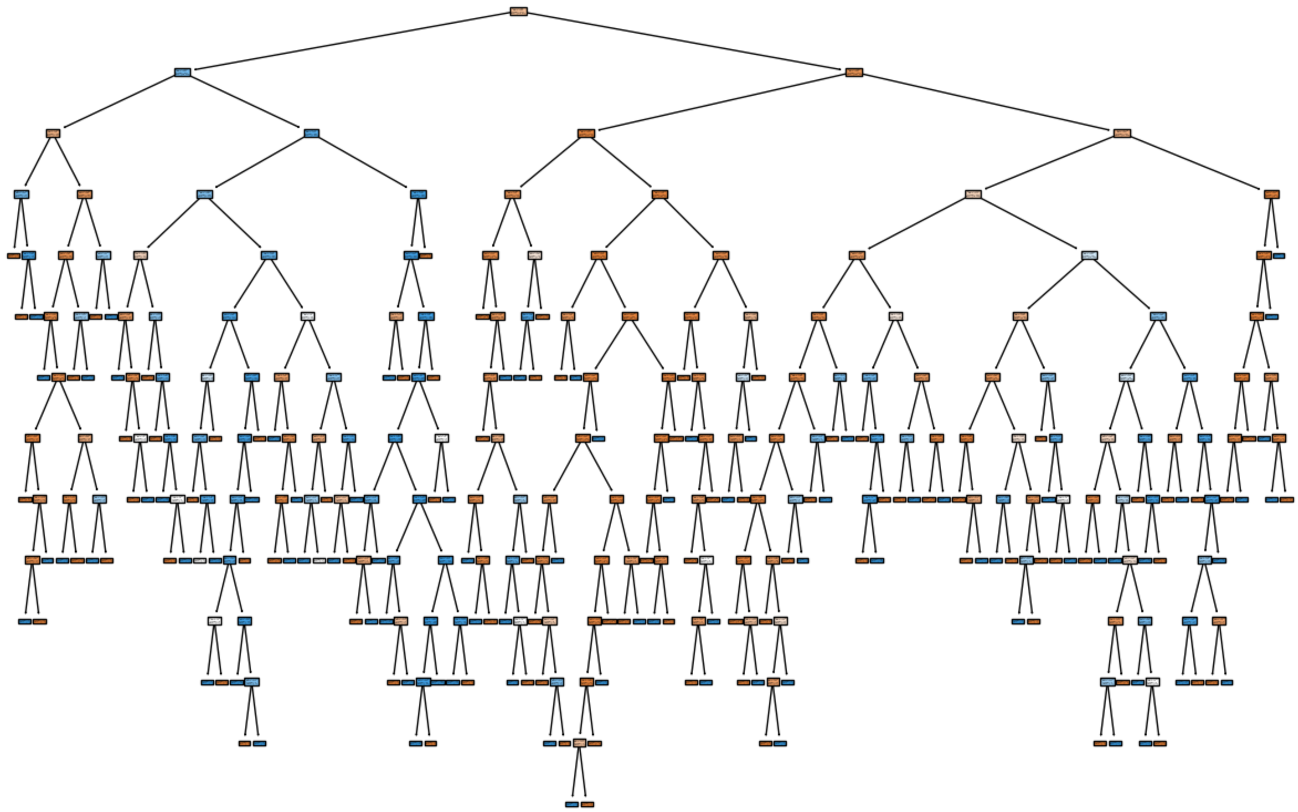
Accuracy of training dataset of Decision Tree Classification

Accuracy score of email prediction using Decision Trees = 0.9992743105950653

Summary of the training dataset

```
Classification Report :

precision    recall  f1-score   support

           0       1.00      1.00      1.00      1879
           1       1.00      1.00      1.00       877

    accuracy                           1.00      2756
   macro avg       1.00      1.00      1.00      2756
weighted avg       1.00      1.00      1.00      2756
```

Visualizing final decision tree



Confusion Matrix before pruning the decision tree.

The Training Dataset accuracy is very high when compared to the Testing Dataset accuracy. This depicts that the model is overfitted. To avoid overfitting the Decision tree should be Pruned.

A decision tree can be pruned to minimize its size by removing branches that don't significantly improve the tree's ability to classify data. Pruning results, a decision tree that is simpler and performs better when generalizing to new data.

In the context of a Decision tree, the (TPR) true positive rate measures the proportion of positive instances correctly identified as positive.

The equation is below: TPR = TP / (TP + FN)

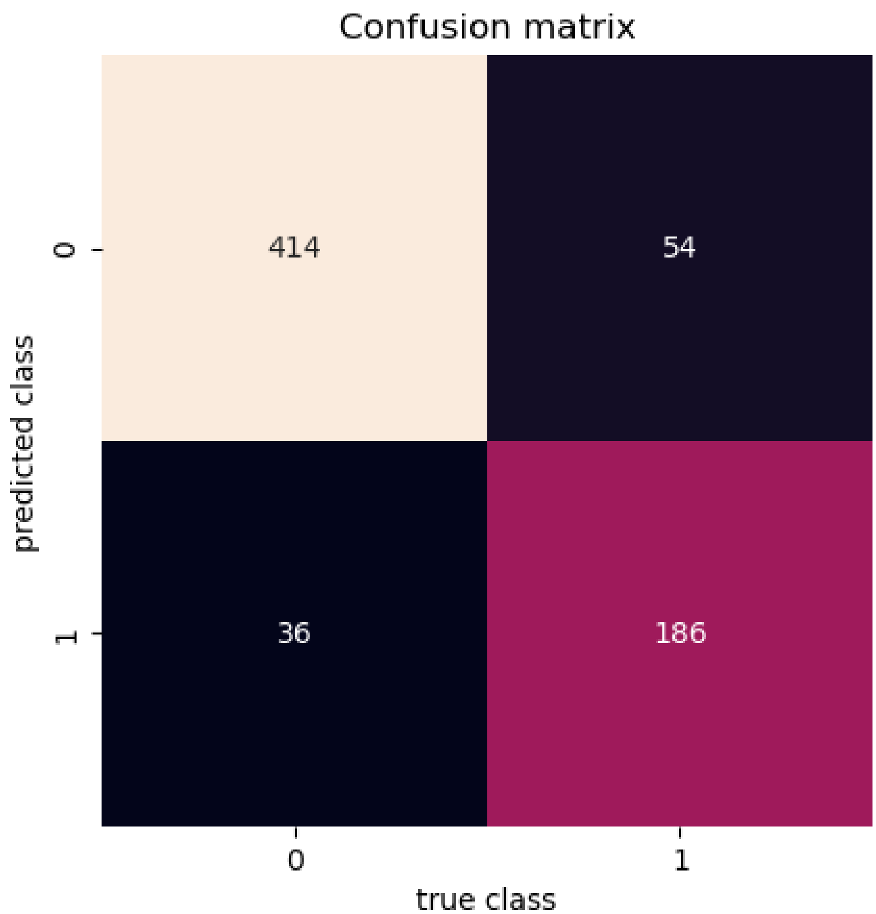
FN (False negative) is the number of positive instances that are incorrectly classified as positive.

In the context of a Decision tree, the (FPR) false positive rate measures the proportion of negative instances incorrectly identified as positive.

The equation is below: FPR = FP / (FP + TN)

TN (True negative) is the number of negative instances that are correctly classified as negative.

A good decision tree classifier model must have a high TPR while maintaining a lower FPR.



True Positive: 414 mails are predicted as Not Spam and it is correct.

False Positive: 54 mails are predicted as Spam, but it is Not Spam.

True Negative: 186 mails are predicted as Spam, and it is correct.

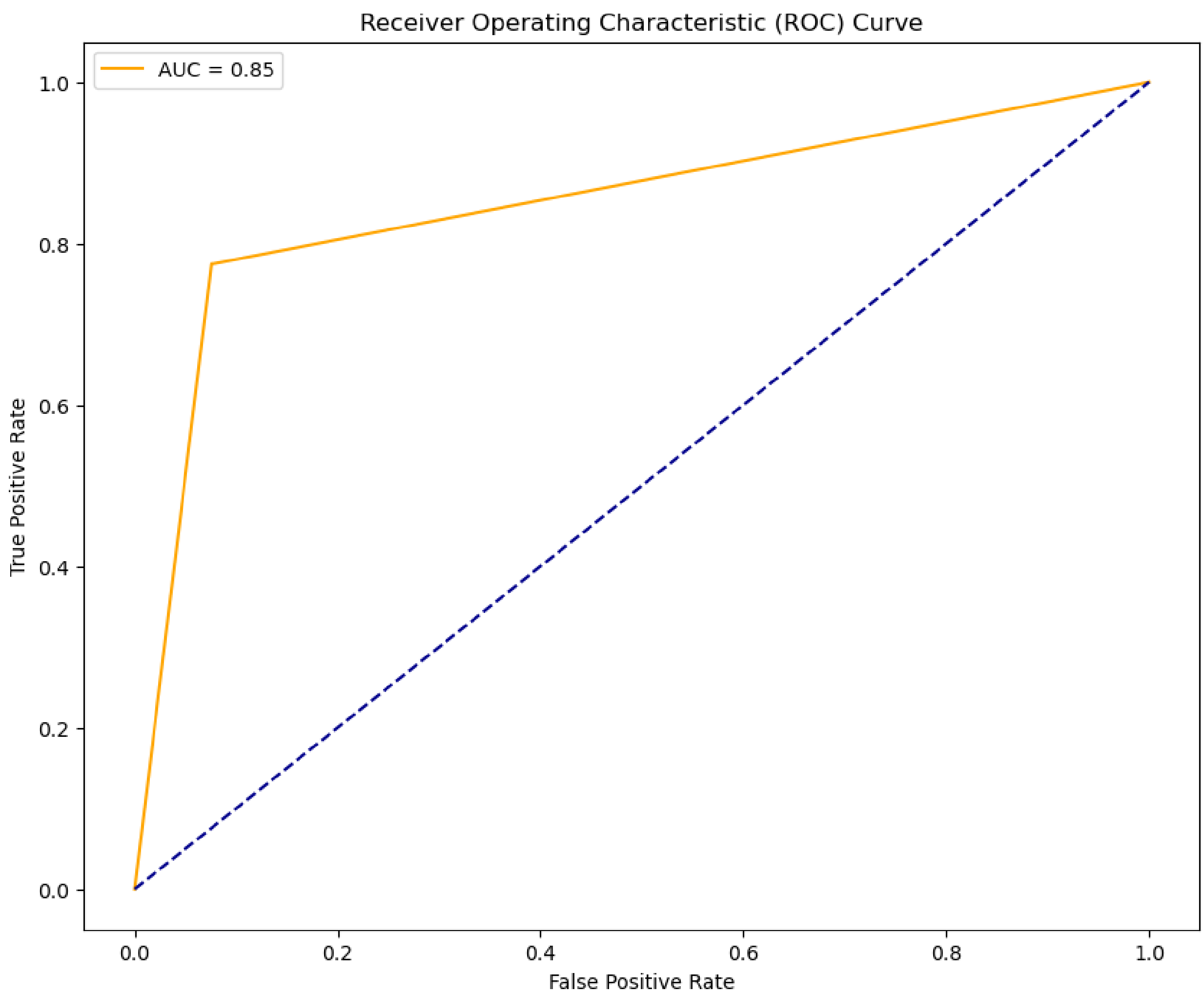
False Negative: 36 mails are predicted as Spam, but it is Not spam.

ROC Curve

An illustration of the performance of a binary classifier, such as a decision tree classifier, when the discrimination threshold is changed can be shown using a ROC (Receiver Operating Characteristic) curve.

A decision tree classifier's ROC curve will be in the top left corner of the plot if it performs well at classifying data.

The decision tree classifier's overall performance is assessed by the area under the ROC curve, with an AUC of 1.0 denoting excellent classification performance and an AUC of 0.5 denoting random guessing.



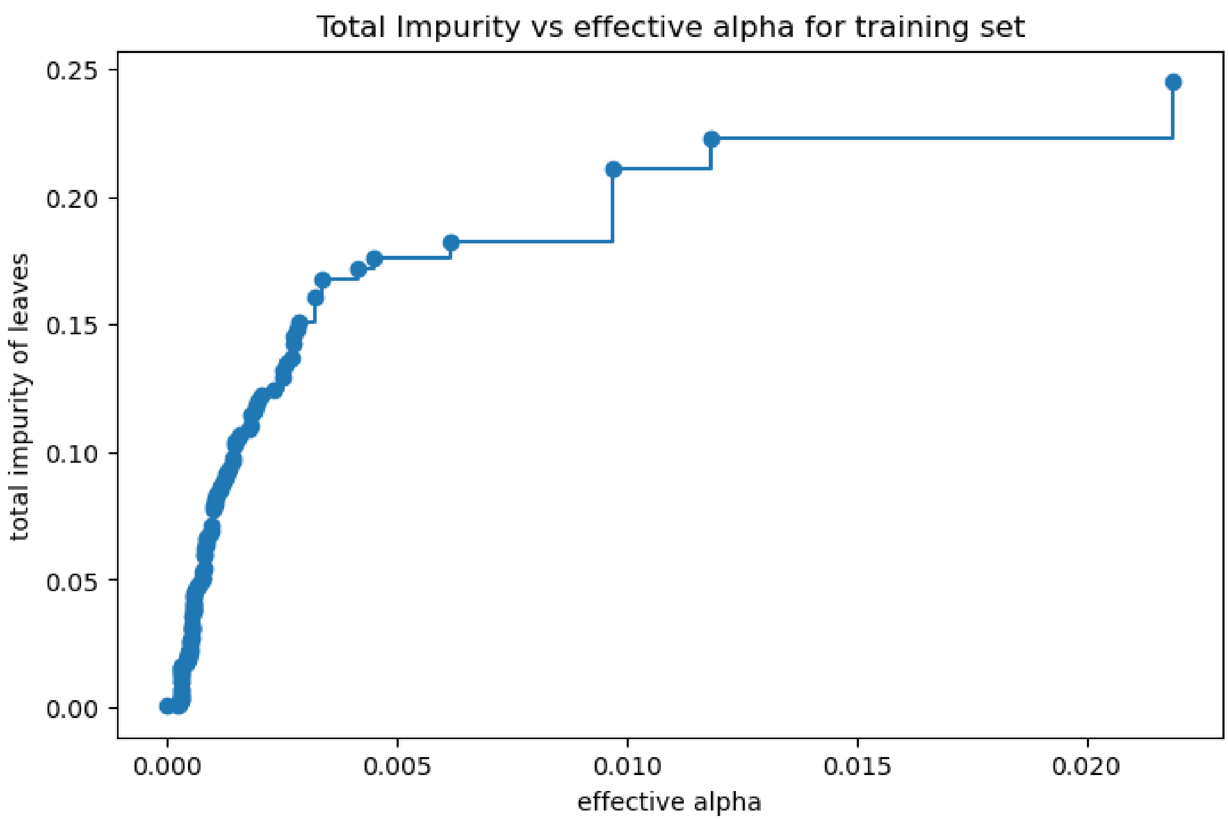
The AUC score for the above graph is 0.8475

Pruning of Decision Tree

Since the training dataset was overfitted the tree needs to be pruned. The pruning process is below.

A graph is plotted between 'Total impurity of leaves' as Y axis and 'Effective alpha' as the X axis. Using the graph, we can find the optimal 'ccp alpha' value required for pruning.

The regularization parameter ccp alpha balances the accuracy and model complexity in decision trees. It helps the model perform better when it comes to generalization and prevents overfitting.



Accuracy of testing dataset after pruning

Accuracy score of email prediction after pruning the decision tree : 0.8782608695652174

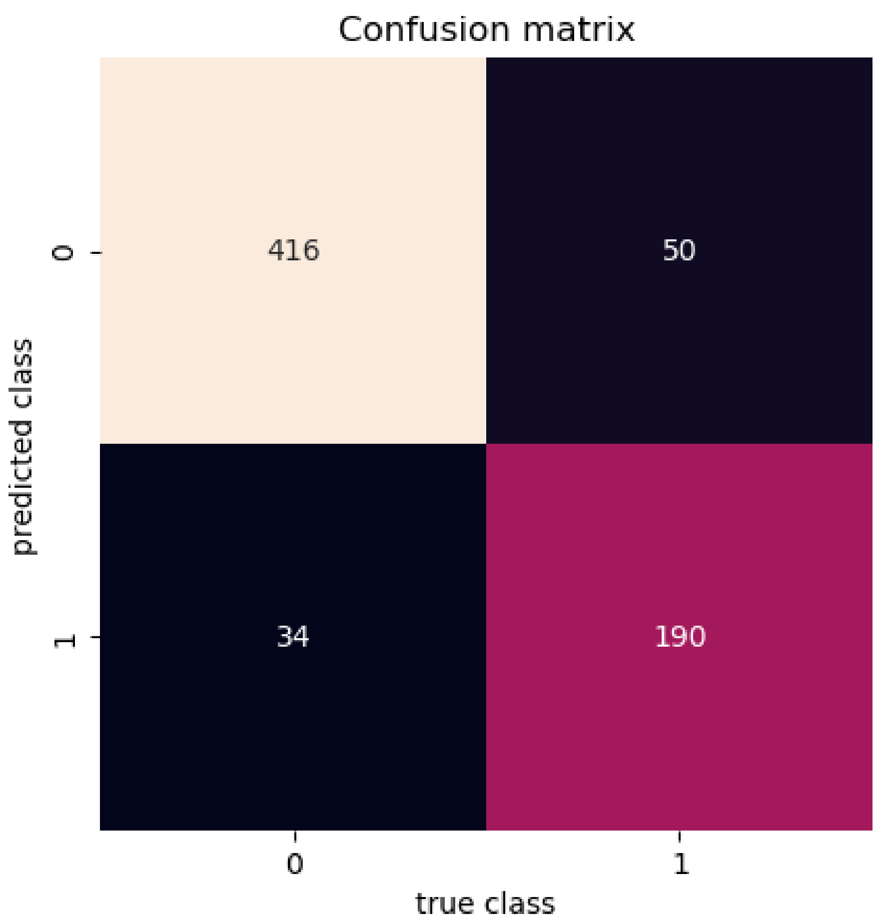
Accuracy of training dataset after pruning

Accuracy score of email prediction after pruning the decision tree : 0.918722786647315

The above values, confirms that overfitting of the training dataset is treated properly.

Classification Report and Confusion Matrix of Pruned Decision Tree.

```
Classification Report :
              precision    recall  f1-score   support

           0       0.89      0.92      0.91       450
           1       0.85      0.80      0.82       240

    accuracy                           0.88       690
   macro avg       0.87      0.86      0.87       690
weighted avg       0.88      0.88      0.88       690
```



True Positive: 416 mails are predicted as Not Spam and it is correct.

False Positive: 50 mails are predicted as Spam, but it is Not Spam.

True Negative: 190 mails are predicted as Spam, and it is correct.

False Negative: 34 mails are predicted as Spam, but it is Not spam.

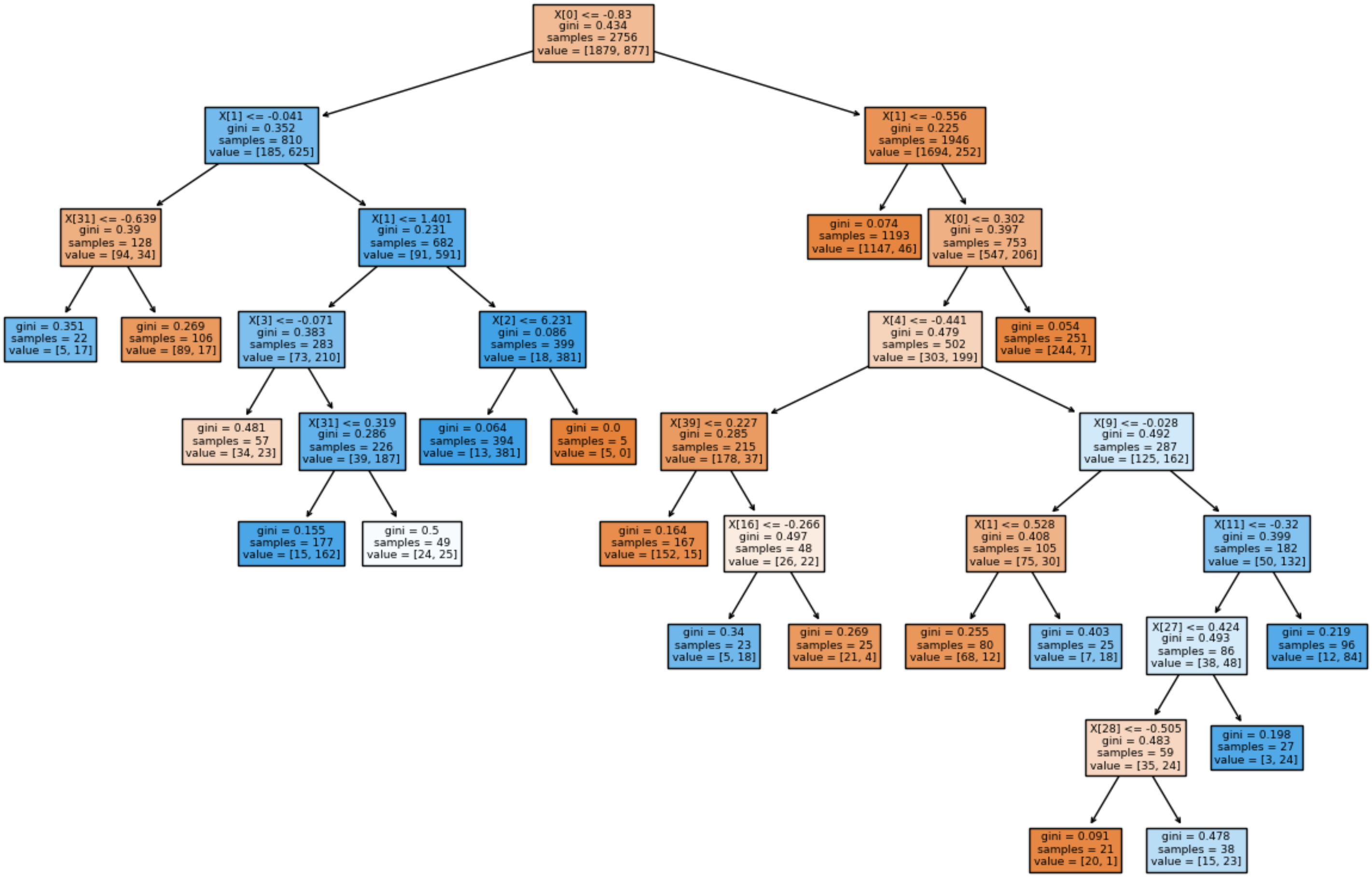
Comparisons

Accuracy of testing dataset before pruning	Accuracy of testing dataset after pruning
0.8695652173913043	0.8782608695652174

Accuracy of training dataset before pruning	Accuracy of training dataset after pruning
0.9992743105950653	0.918722786647315

Accuracy of testing dataset of KNN Algorithm	Accuracy of training dataset of KNN Algorithm
88.98550724637681	92.8156748911466

Decision tree obtained after pruning.



The link to git repository

https://github.com/DinethHasaranga/Machine-Learning-CW

Code :

```
#importing necessary libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn
import re
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_curve,auc
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier

#Loading the dataset

with open("C:\\Users\\Admin\\Desktop\\ML CW\\spambase.names") as spam:
 text = spam.read()
labels = re.findall(r'\n(\w*_?\W?):', text)

Data_set = pd.read_csv("C:\\Users\\Admin\\Desktop\\ML CW\\spambase.data", header=None,
names=labels +['spam'])

Data_array=Data_set.values
# print(Data_array)


# printing first 5 rows
Data_set.head()

print("No of rows in dataset before preprocessing : ", len(Data_set))
```

Finding the duplicates in the dataset

```
#Checking the availability of duplicates
Data_set.duplicated()
```

Dropping the duplicate values

```
Data_set.drop_duplicates(inplace=True)
print("No of rows in dataset after removing duplicates : ", len(Data_set))
```

Finding the outliers in the dataset

```
fig = plt.figure(figsize =(100, 50))
Data_set.plot.box(title='Boxplot of Spam No spam email',rot=90)
plt.show()
```

Boxplot of capital_run_length_total

```
sn.boxplot(x = Data_set['capital_run_length_total'])
```

Boxplot of capital_run_length_average

```
sn.boxplot(x = Data_set['capital_run_length_average'])
```

Boxplot of capital_run_length_longest

```
sn.boxplot(x = Data_set['capital_run_length_longest'])
```

Making all the outliers as Null values from IQR technique

```
for x in ['capital_run_length_total','capital_run_length_longest','capital_run_length_average']:
   q75,q25 = np.percentile(Data_set.loc[:,x],[75,25])
   intr_qr = q75-q25

   max = q75+(1.5*intr_qr)
   min = q25-(1.5*intr_qr)

   Data_set.loc[Data_set[x] < min,x] = np.nan
   Data_set.loc[Data_set[x] > max,x] = np.nan
```

Boxplot of capital_run_length_total without outliers

```
sn.boxplot(x = Data_set['capital_run_length_total'])
```

Boxplot of capital_run_length_longest without outliers

```
sn.boxplot(x = Data_set['capital_run_length_longest'])
```

Boxplot of capital_run_length_average without outliers

```
sn.boxplot(x = Data_set['capital_run_length_average'])
```

Finding the null values in the dataset

```
Data_set.isna().sum().any()
Data_set.isna().sum()
```

Removing all the Null values

```
# Drop all rows with NaN values
df2=Data_set.dropna()
df2=Data_set.dropna(axis=0)
```

Removing the target column

```
# Reset index after drop
df2=Data_set.dropna().reset_index(drop=True)
```

Summary of dataset before performing Standard Scaler

```
data.describe()
```

Kernel Density Plot

```
fig, ax = plt.subplots(figsize=(10,10))
sns.kdeplot(data=data, ax=ax)
ax.set_xlim(-5, 4)
plt.show()

# Calculate the standard deviation of all columns
std_dev = data.std()

# Plot the standard deviation of all columns
plt.bar(range(len(std_dev)), std_dev)
plt.title("Standard Deviation before performing Standard Scaler")
plt.xlabel("Column Index")
plt.ylabel("Standard Deviation")
plt.show()

# Calculate the mean of all columns
mean = data.mean()

# Plot the mean of all columns
plt.bar(range(len(mean)), mean)
plt.title("Mean before performing Standard Scaler")
plt.xlabel("Column Index")
plt.ylabel("Mean")
plt.show()
```

Performing Standard Scaling for the dataset

```
scaler=StandardScaler()
scaled_data=scaler.fit_transform(data)
df=pd.DataFrame(data=scaled_data, columns= data.columns)
df
```

Summmary of dataset after performing Standard Scaling

```
print("No of rows in dataset after preprocessing : ", len(Data_set))
data.describe()
```

Performing PCA to the Dataset

```
pca = PCA()
principalComponents = pca.fit_transform(df)

plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))

plt.xlabel('Number of Components')
plt.ylabel('Variance (%)') #for each component

plt.title('Explained Variance')
plt.grid(True)
plt.show()
```

KDE plot after performing Standard Scaler

```
fig, ax = plt.subplots(figsize=(10,10))
sns.kdeplot(data=df, ax=ax)
ax.set_xlim(-5, 5)
plt.show()

# Calculate the standard deviation of all columns
std_dev = df.std()

# Plot the standard deviation of all columns
plt.bar(range(len(std_dev)), std_dev)
plt.title("Standard Deviation after performing Standard Scaler")
plt.xlabel("Column Index")
plt.ylabel("Standard Deviation")
plt.show()

# Calculate the mean of all columns
mean = df.mean()

# Plot the mean of all columns
plt.bar(range(len(mean)), mean)
plt.title("Mean after performing Standard Scaler")
plt.xlabel("Column Index")
plt.ylabel("Mean")
plt.show()
```

```
pca = PCA()

principalComponents = pca.fit_transform(df)

plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))

plt.xlabel('Number of Components')
plt.ylabel('Variance (%)') #for each component

plt.title('Explained Variance')
plt.grid(True)
plt.show()
```

Introducing the PCA components

```
pca = PCA(n_components=44)
new_data = pca.fit_transform(df)

# This will be the new data fed to the algorithm.
principal_Df = pd.DataFrame(data = new_data
        , columns = ['PC1',
'PC2','PC3','PC4','PC5','PC6','PC7','PC8','PC9','PC10','PC11','PC12','PC13','PC14','PC15','PC16','PC17','PC18'
,'PC19','PC20', 'PC21',
PC22','PC23','PC24','PC25','PC26','PC27','PC28','PC29','PC30','PC31','PC32','PC33','PC34','PC35','PC36','PC
37','PC38','PC39','PC40','PC41','PC42','PC43','PC44'])
```

Dataset after performing PCA

```
principal_Df.head()
# principal_Df
print(pca.explained_variance_)
print(pca.components_)
```

Build the predictive model by appling Decision Tree algorithm

```
X = principal_Df.iloc[:,0:44].values
y = Data_set.iloc[:, 57].values
```

Splitting the dataset into the Training set and Test set

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
```

Applying KNN Algorithm

```
X = principal_Df.iloc[:,0:44].values
y = Data_set.iloc[:, 57].values
```

Splitting the dataset into the Training set and Test set

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)

# Fitting classifier to the Training set
classifier = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
classifier.fit(X_train,y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)
```

Classification Report

```
print('Classification Report : \n\n')
print(classification_report(y_test, y_pred))
```

Accuracy of testing dataset

```
print("Accuracy score of email prediction using KNN : ",accuracy_score(y_pred,y_test)*100)
```

Accuracy of training dataset

```
y_pred2 = classifier.predict(X_train)
print("Accuracy score of email prediction using KNN : ",accuracy_score(y_pred2,y_train)*100)
```

Visualization

```
# Summary of the predictions made by the classifier
mat = confusion_matrix(y_test, y_pred)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)

plt.title('Confusion matrix')
plt.xlabel('true class')
plt.ylabel('predicted class')
```

Applying Decision Tree Algorithm

Fitting classifier to the Training set

```
clf = DecisionTreeClassifier(random_state=0,criterion='gini')
clf.fit(X_train,y_train)
```

Accuracy of testing dataset

```
predictions_test=clf.predict(X_test)
accuracy_score(y_test, predictions_test)
```

Accuracy of training dataset

```
predictions_train = clf.predict(X_train)
accuracy_score(y_train,predictions_train)
```

Visualizing final decision tree

```
from sklearn import tree
plt.figure(figsize=(15,10))
tree.plot_tree(clf,filled=True)
plt.show()
```

Summary of the test dataset

```
print('Classification Report : \n\n')
print(classification_report(y_test,predictions_test))

# Summary of the predictions made by the classifier
mat = confusion_matrix(y_test, predictions_test)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)

plt.title('Confusion matrix')
plt.xlabel('true class')
plt.ylabel('predicted class')
```

Summary of the training dataset

```
print(classification_report(y_train,predictions_train))
print(confusion_matrix(y_train,predictions_train))
```

Evaluating the false positive rate and true positive rate

```
dt_probs = clf.predict_proba(X_test)[:,1]
fpr_dt, tpr_dt, thresholds_dt = roc_curve(y_test,dt_probs)
```

Plotting ROC curve for our Decision Tree

```
auc_score_dt = auc(fpr_dt,tpr_dt)
auc_score_dt

def plot_roc_curve(fpr, tpr):
```

```
    plt.figure(figsize=(10,8))
    plt.plot(fpr_dt, tpr_dt, color='orange', label='AUC = %0.2f' % auc_score_dt)
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()

plot_roc_curve(fpr_dt,tpr_dt)
```

Pruning the decision tree

```
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities

fig, ax = plt.subplots(figsize=(8,5))
ax.plot(ccp_alphas[:-1], impurities[:-1], marker='o', drawstyle="steps-post")
ax.set_xlabel("effective alpha")
ax.set_ylabel("total impurity of leaves")
ax.set_title("Total Impurity vs effective alpha for training set")

clf = DecisionTreeClassifier(random_state=0, ccp_alpha=0.003)
clf.fit(X_train,y_train)
```

Accuracy of testing dataset after pruning

```
pred=clf.predict(X_test)
accuracy_score(y_test, pred)
```

Accuracy of training dataset after pruning

```
pred_1 = clf.predict(X_train)
accuracy_score(y_train,pred_1)

print('Classification Report : \n\n')
print(classification_report(y_test,pred))

# Summary of the predictions made by the classifier
mat = confusion_matrix(y_test, pred)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False)

plt.title('Confusion matrix')
plt.xlabel('true class')
plt.ylabel('predicted class')
```

Decision tree obtained after pruning

```
plt.figure(figsize=(15,10))
tree.plot_tree(clf,filled=True)
plt.show()
```