INFORMATICS INSTITUTE OF TECHNOLOGY

In Collaboration with

ROBERT GORDON UNIVERSITY ABERDEEN

CM-2604

Machine Learning Coursework

Sathila Samarasinghe

IIT ID – 20210515

RGU ID – 2117535

Submitted in partial fulfillment of the requirements for the BSc (Hons) in Artificial Intelligence and Data Science degree at Robert Gordon University.

March 2023

Introduction

This is a machine learning classification problem labeling emails as spam or non-spam. The dataset used here is Spambase dataset (https://archive.ics.uci.edu/ml/datasets/Spambase) found on UCI Machine Learning Repository. The problem has been triggered successfully using the two machine learning models K Nearest Neighbors and Decision Trees.

Dataset

The spambase dataset had 4601 rows and 57 columns. However, these rows and columns had to reduce in the data preprocessing stage in order to get a good dataset.

Initial information on the dataset is shown below.

```
RangeIndex: 4601 entries, 0 to 4600
Data columns (total 58 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   word_freq_make              4601 non-null   float64
 1   word_freq_address           4601 non-null   float64
 2   word_freq_all               4601 non-null   float64
 3   word_freq_3d                4601 non-null   float64
 4   word_freq_our               4601 non-null   float64
 5   word_freq_over              4601 non-null   float64
 6   word_freq_remove            4601 non-null   float64
 7   word_freq_internet          4601 non-null   float64
 8   word_freq_order             4601 non-null   float64
 9   word_freq_mail              4601 non-null   float64
 10  word_freq_receive           4601 non-null   float64
 11  word_freq_will              4601 non-null   float64
 12  word_freq_people            4601 non-null   float64
 13  word_freq_report            4601 non-null   float64
 14  word_freq_addresses         4601 non-null   float64
 15  word_freq_free              4601 non-null   float64
 16  word_freq_business          4601 non-null   float64
 17  word_freq_email             4601 non-null   float64
 18  word_freq_you               4601 non-null   float64
 19  word_freq_credit            4601 non-null   float64
 20  word_freq_your              4601 non-null   float64
 21  word_freq_font              4601 non-null   float64
 22  word_freq_000               4601 non-null   float64
 23  word_freq_money             4601 non-null   float64
 24  word_freq_hp                4601 non-null   float64
 25  word_freq_hpl               4601 non-null   float64
 26  word_freq_george            4601 non-null   float64
 27  word_freq_650               4601 non-null   float64
 28  word_freq_lab               4601 non-null   float64
 29  word_freq_labs              4601 non-null   float64
 30  word_freq_telnet            4601 non-null   float64
 31  word_freq_857               4601 non-null   float64
 32  word_freq_data              4601 non-null   float64
 33  word_freq_415               4601 non-null   float64
 34  word_freq_85                4601 non-null   float64
 35  word_freq_technology        4601 non-null   float64
 36  word_freq_1999              4601 non-null   float64
 37  word_freq_parts             4601 non-null   float64
 38  word_freq_pm                4601 non-null   float64
 39  word_freq_direct            4601 non-null   float64
 40  word_freq_cs                4601 non-null   float64
 41  word_freq_meeting           4601 non-null   float64
 42  word_freq_original          4601 non-null   float64
 43  word_freq_project           4601 non-null   float64
 44  word_freq_re                4601 non-null   float64
 45  word_freq_edu               4601 non-null   float64
 46  word_freq_table             4601 non-null   float64
 47  word_freq_conference        4601 non-null   float64
 48  char_freq_;                 4601 non-null   float64
 49  char_freq_(                 4601 non-null   float64
 50  char_freq_[                 4601 non-null   float64
 51  char_freq_!                 4601 non-null   float64
 52  char_freq_$                 4601 non-null   float64
 53  char_freq_#                 4601 non-null   float64
 54  capital_run_length_average  4601 non-null   float64
 55  capital_run_length_longest  4601 non-null   int64
 56  capital_run_length_total    4601 non-null   int64
 57  spam                        4601 non-null   int64
dtypes: float64(55), int64(3)
```

Data Preprocessing

Data Cleaning

Data cleaning has been done by removing all the null duplicated values from the dataset.

```
Int64Index: 4210 entries, 0 to 4600
Data columns (total 58 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   word_freq_make              4210 non-null   float64
 1   word_freq_address           4210 non-null   float64
 2   word_freq_all               4210 non-null   float64
 3   word_freq_3d                4210 non-null   float64
 4   word_freq_our               4210 non-null   float64
 5   word_freq_over              4210 non-null   float64
 6   word_freq_remove            4210 non-null   float64
 7   word_freq_internet          4210 non-null   float64
 8   word_freq_order             4210 non-null   float64
 9   word_freq_mail              4210 non-null   float64
 10  word_freq_receive           4210 non-null   float64
 11  word_freq_will              4210 non-null   float64
 12  word_freq_people            4210 non-null   float64
 13  word_freq_report            4210 non-null   float64
 14  word_freq_addresses         4210 non-null   float64
 15  word_freq_free              4210 non-null   float64
 16  word_freq_business          4210 non-null   float64
 17  word_freq_email             4210 non-null   float64
 18  word_freq_you               4210 non-null   float64
 19  word_freq_credit            4210 non-null   float64
 20  word_freq_your              4210 non-null   float64
 21  word_freq_font              4210 non-null   float64
 22  word_freq_000               4210 non-null   float64
 23  word_freq_money             4210 non-null   float64
 24  word_freq_hp                4210 non-null   float64
 25  word_freq_hpl               4210 non-null   float64
 26  word_freq_george            4210 non-null   float64
 27  word_freq_650               4210 non-null   float64
 28  word_freq_lab               4210 non-null   float64
 29  word_freq_labs              4210 non-null   float64
 30  word_freq_telnet            4210 non-null   float64
 31  word_freq_857               4210 non-null   float64
 32  word_freq_data              4210 non-null   float64
 33  word_freq_415               4210 non-null   float64
 34  word_freq_85                4210 non-null   float64
 35  word_freq_technology        4210 non-null   float64
 36  word_freq_1999              4210 non-null   float64
 37  word_freq_parts             4210 non-null   float64
 38  word_freq_pm                4210 non-null   float64
 39  word_freq_direct            4210 non-null   float64
 40  word_freq_cs                4210 non-null   float64
 41  word_freq_meeting           4210 non-null   float64
 42  word_freq_original          4210 non-null   float64
 43  word_freq_project           4210 non-null   float64
 44  word_freq_re                4210 non-null   float64
 45  word_freq_edu               4210 non-null   float64
 46  word_freq_table             4210 non-null   float64
 47  word_freq_conference        4210 non-null   float64
 48  char_freq_;                 4210 non-null   float64
 49  char_freq_(                 4210 non-null   float64
 50  char_freq_[                 4210 non-null   float64
 51  char_freq_!                 4210 non-null   float64
 52  char_freq_$                 4210 non-null   float64
 53  char_freq_#                 4210 non-null   float64
 54  capital_run_length_average  4210 non-null   float64
 55  capital_run_length_longest  4210 non-null   int64
 56  capital_run_length_total    4210 non-null   int64
 57  spam                        4210 non-null   int64
dtypes: float64(55), int64(3)
memory usage: 1.9 MB
```

Number of rows before removing duplicates – 4601

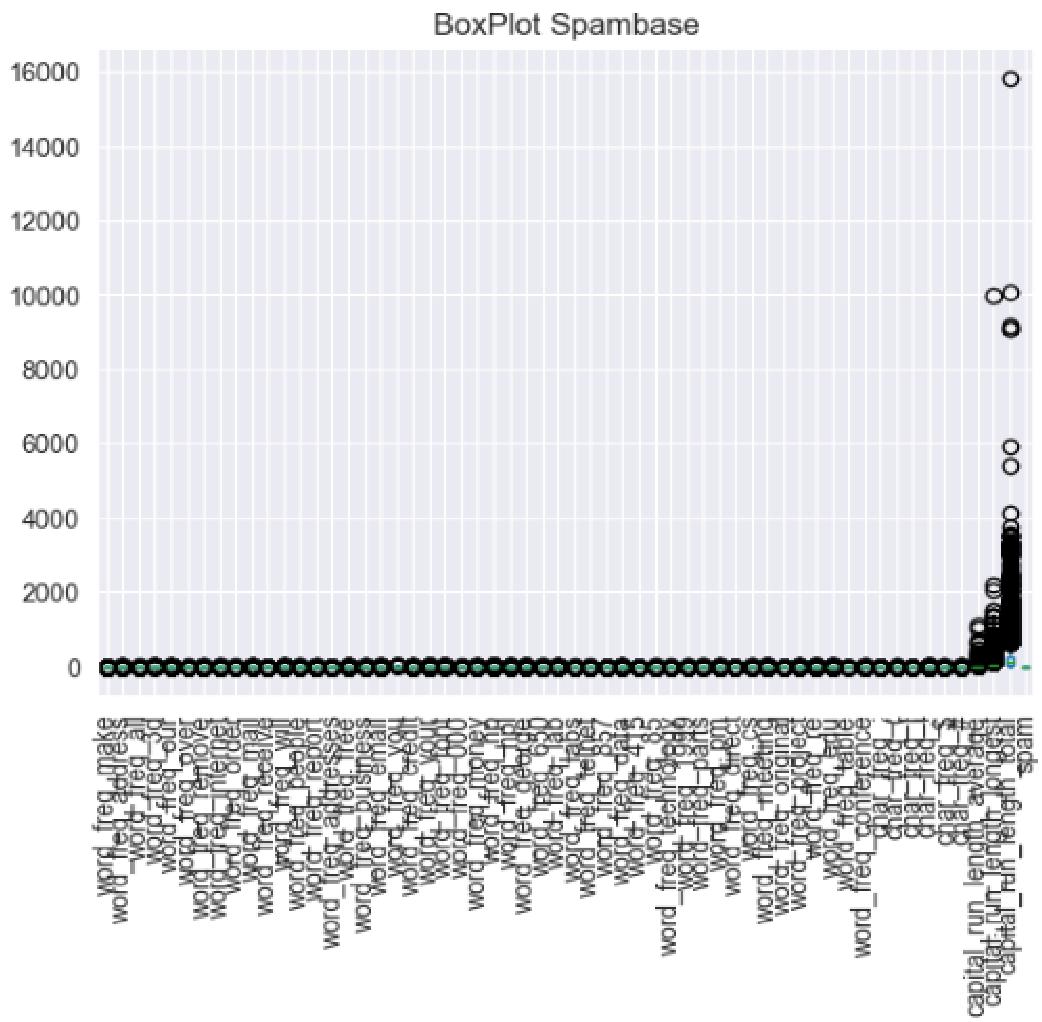
Number of rows after removing duplicates – 4210

Data Transformation

Data transformation has been done using the technique of removing outliers and performing a standard scaler.

Identifying Outliers

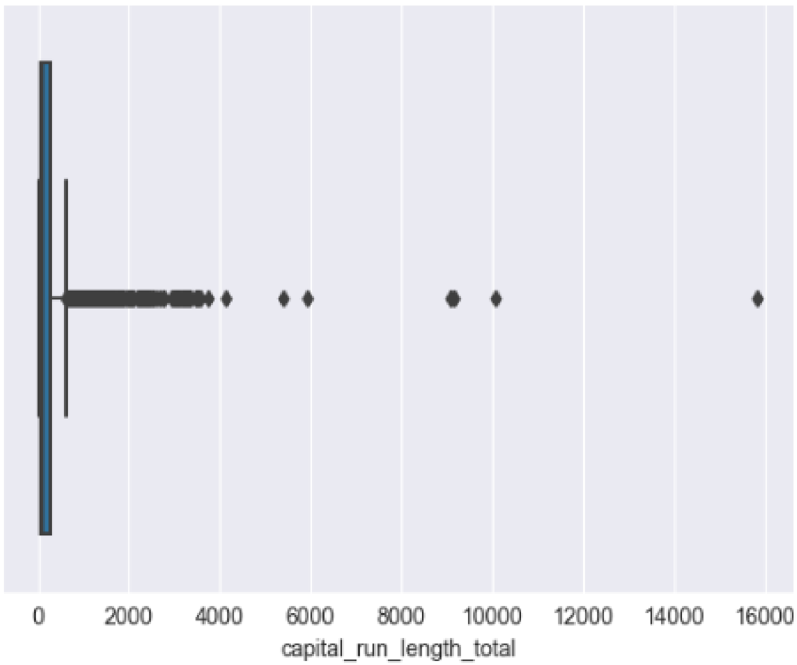
A boxplot has been made to identify the outliers.



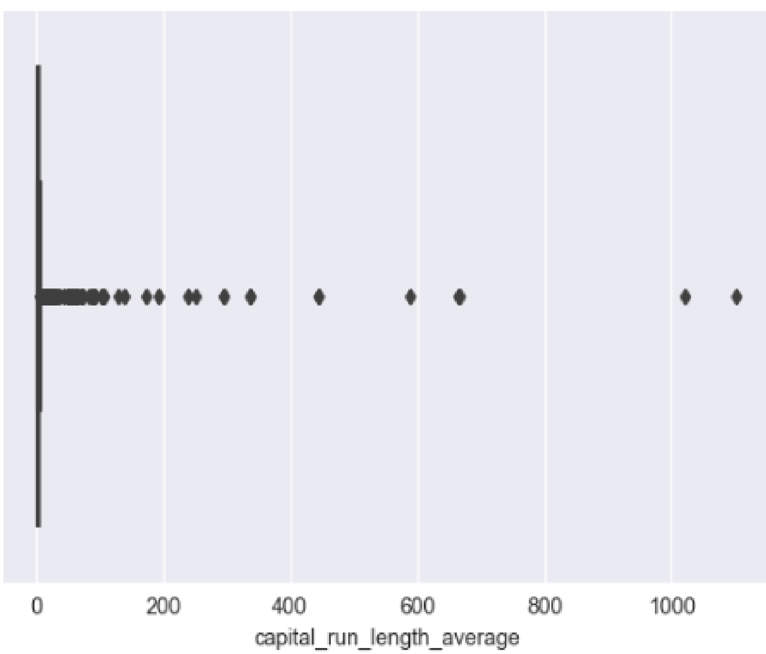
From the above boxplot, it has been found there are outliers in the following three columns of the dataset.

- capital_run_length_total
- capital_run_length_average
- capital_run_length_longest

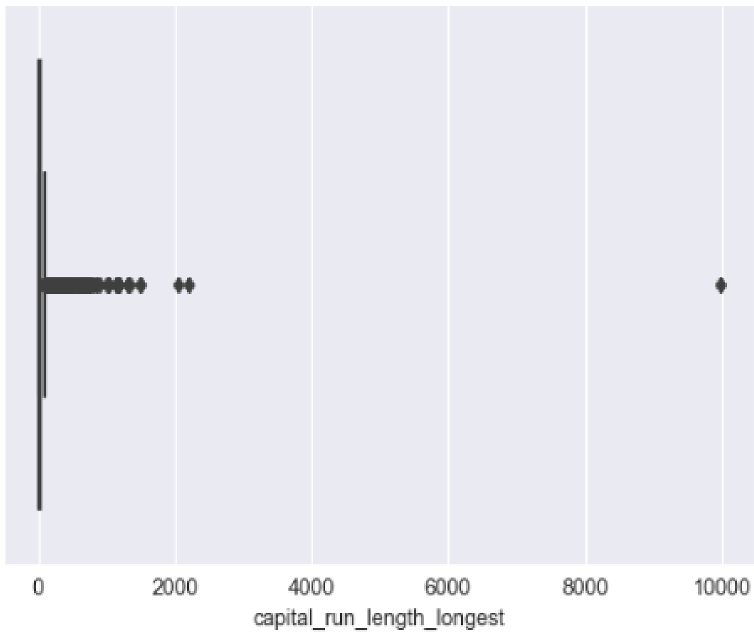
Outlier boxplot of capital_run_length_total:



Outlier boxplot of capital_run_length_average:



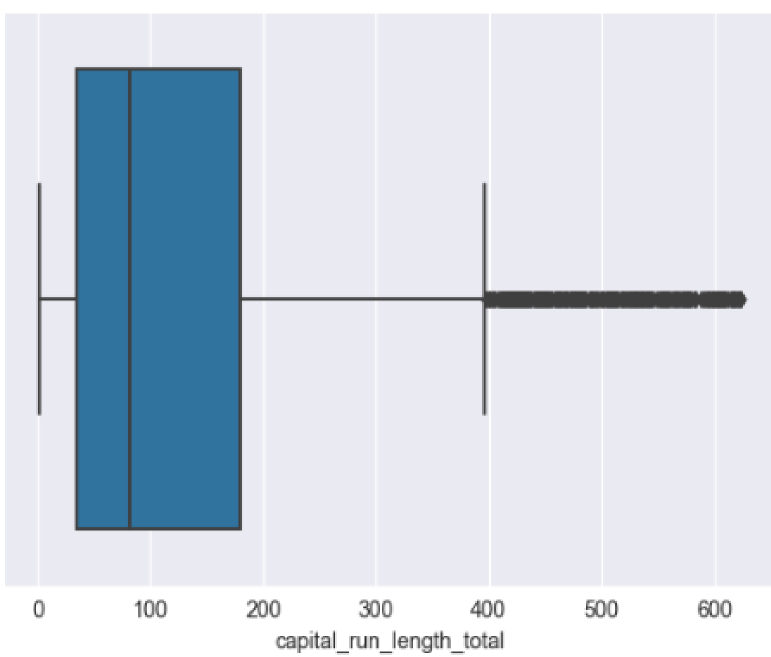
Outlier boxplot of capital_run_length_longest



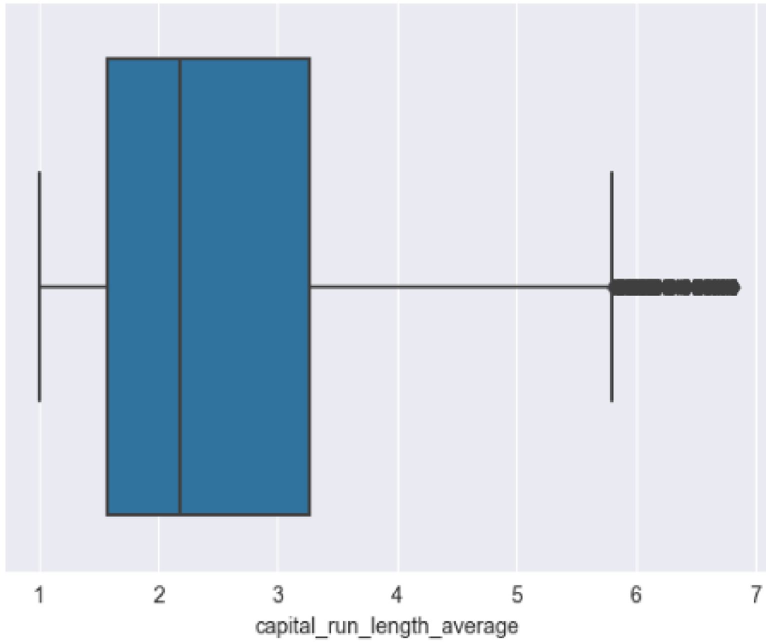
Removing Outliers

The identified outliers have been removed from the dataset by converting them to null values using the Inter Quartile Range (IQR) technique.

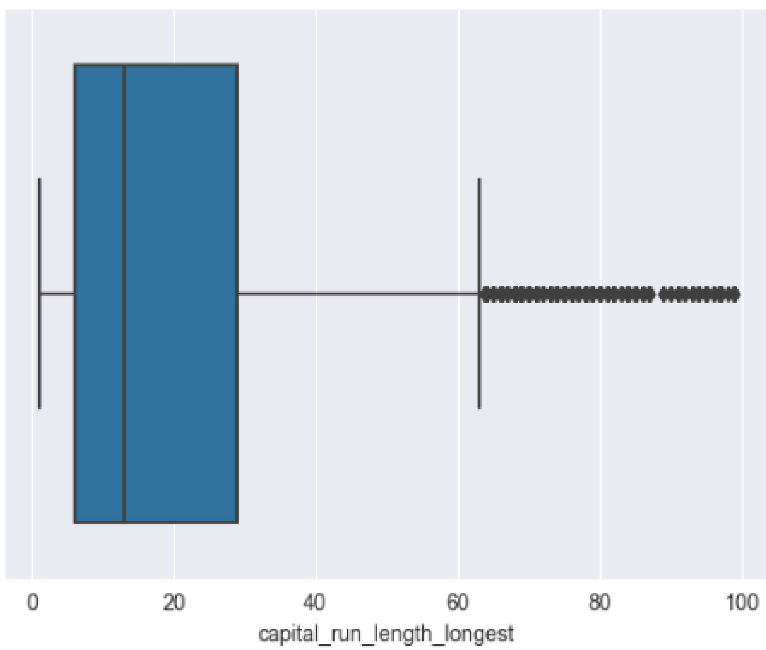
Boxplot of capital_run_length_total after removing outliers:



Boxplot of capital_run_length_avarage after removing outliers:



Boxplot of capital_run_length_longest after removing outliers:



Number of rows before removing outliers – 4210

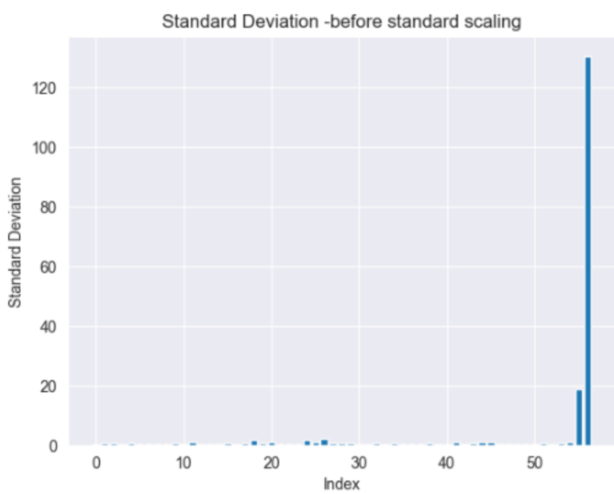
Number of rows after removing outliers – 3446

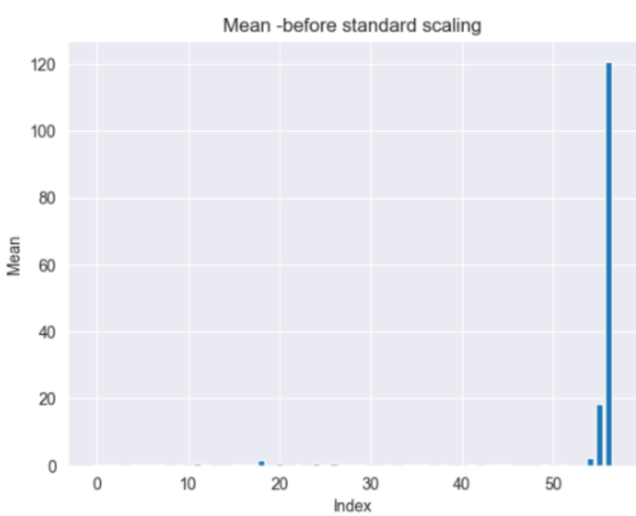
Standard Scaling

Performing a standard scaler was to normalize the range of values of each column to reduce the mean to zero and the standard deviation to one.

Before Standard Scaling:

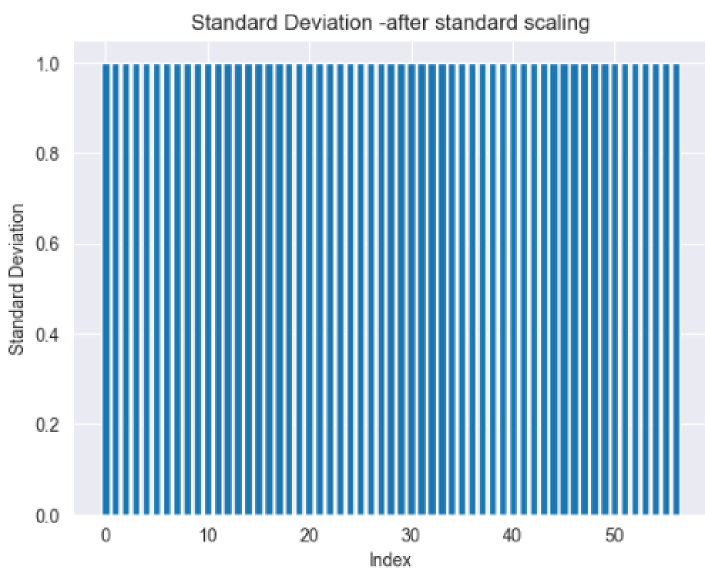
	word_freq_make	word_freq_address	word_freq_all	word_freq_3d	word_freq_our	word_freq_over	word_freq_remove	word_freq_internet	word_freq_order	word_freq_mail
count	3446.000000	3446.000000	3446.000000	3446.000000	3446.000000	3446.000000	3446.000000	3446.000000	3446.000000	3446.000000
mean	0.094779	0.092716	0.268056	0.005818	0.308175	0.085267	0.093688	0.096164	0.047841	0.201033
std	0.309801	0.474629	0.529981	0.134848	0.701947	0.281174	0.356036	0.420321	0.222685	0.581086
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.360000	0.000000	0.360000	0.000000	0.000000	0.000000	0.000000	0.000000
max	4.540000	14.280000	5.100000	7.070000	10.000000	5.880000	7.270000	11.110000	5.260000	11.110000

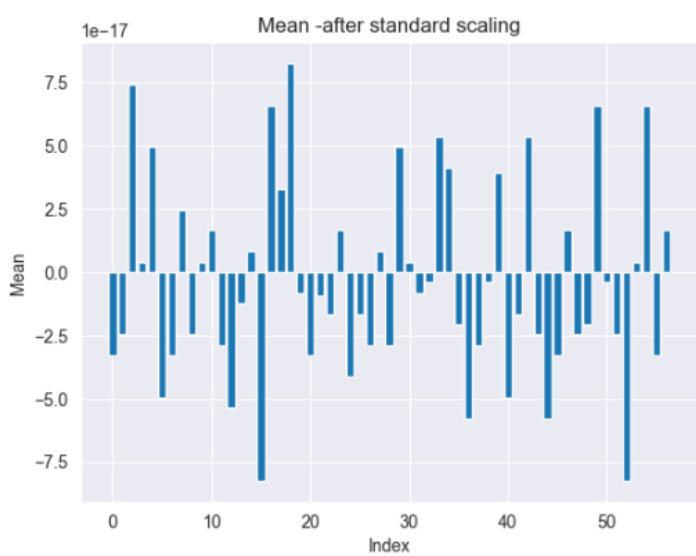




After Standard Scaling:

	word_freq_make	word_freq_address	word_freq_all	word_freq_3d	word_freq_our	word_freq_over	word_freq_remove	word_freq_internet	word_freq_order	word_freq_mail
count	3.446000e+03	3.446000e+03	3.446000e+03	3.446000e+03	3.446000e+03	3.446000e+03	3.446000e+03	3.446000e+03	3.446000e+03	3.446000e+03
mean	-3.299096e-17	-2.474322e-17	7.422965e-17	4.123870e-18	4.948644e-17	-4.948644e-17	-3.299096e-17	2.474322e-17	-2.474322e-17	4.123870e-18
std	1.000145e+00	1.000145e+00	1.000145e+00	1.000145e+00	1.000145e+00	1.000145e+00	1.000145e+00	1.000145e+00	1.000145e+00	1.000145e+00
min	-3.059813e-01	-1.953730e-01	-5.058574e-01	-4.315372e-02	-4.390922e-01	-3.032971e-01	-2.631810e-01	-2.288195e-01	-2.148683e-01	-3.460115e-01
25%	-3.059813e-01	-1.953730e-01	-5.058574e-01	-4.315372e-02	-4.390922e-01	-3.032971e-01	-2.631810e-01	-2.288195e-01	-2.148683e-01	-3.460115e-01
50%	-3.059813e-01	-1.953730e-01	-5.058574e-01	-4.315372e-02	-4.390922e-01	-3.032971e-01	-2.631810e-01	-2.288195e-01	-2.148683e-01	-3.460115e-01
75%	-3.059813e-01	-1.953730e-01	1.735113e-01	-4.315372e-02	7.384151e-02	-3.032971e-01	-2.631810e-01	-2.288195e-01	-2.148683e-01	-3.460115e-01
max	1.435073e+01	2.989567e+01	9.118532e+00	5.239393e+01	1.380907e+01	2.061203e+01	2.015906e+01	2.620720e+01	2.340938e+01	1.877615e+01

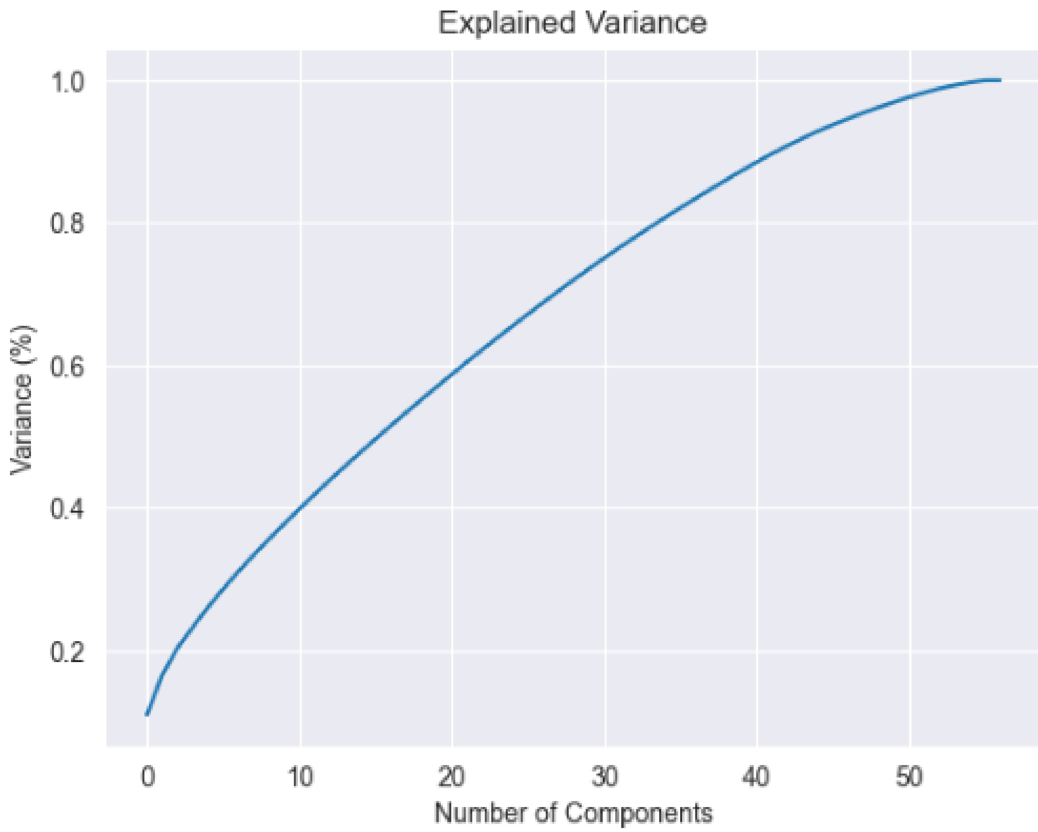




Dimensional Reduction

Reducing the number of features and retaining the most useful features in the dataset have been done to simplify analysis and make the data more manageable.

The most common feature extraction technique used here is principal component analysis (PCA). Principal components are designed to capture more variance in data while reducing the dimensionality.



It was identified as more than 44 principal components have more than 90% variance of data. So, for the newly created data frame, 44 principal components have been used.

K Nearest Neighbors (KNN) Classification

The splitting of the dataset has been done as follows.

70% - Training data

30% - Testing data

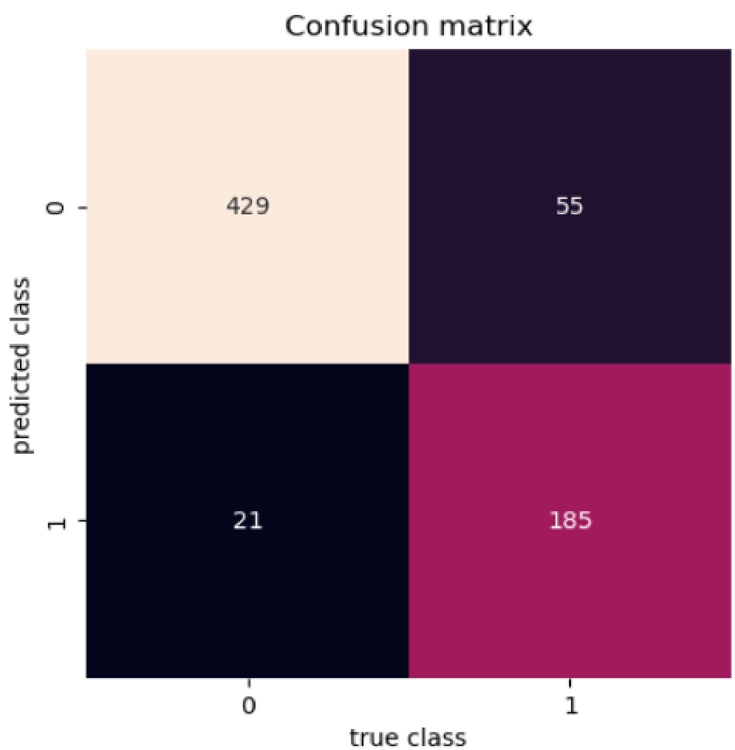
Random state – 0

Through a grid search optimal k value was found as 7.

Accuracy of training data achieved: 92.24709784411277 %

Accuracy of testing data achieved: 90.13539651837525 %

Confusion Matrix



Spam = 1 (positives)

Non-spam = 0 (negatives)

Accuracy = (True Positives + True Negatives)/(True Positives + False Positives + True Negatives + False Negatives)

Recall = (True Positives)/(True Positives + False Negatives)

Precision = (True Positives)/(True Positives + False Positives)

F1 Score = 2* (Precision* Recall)/(Precision+ Recall)

- True Positives - instances that are positive and have been correctly classified as positive by the classification model.
- True Negatives - instances that are negative and have been correctly classified as negative by the classification model.
- False Positives - instances that are negative but have been incorrectly classified as positive by the classification model.
- False Negatives - instances that are positive but have been incorrectly classified as negative by the classification model.

```
Classification Report :

              precision    recall  f1-score   support

           0       0.90      0.96      0.93       676
           1       0.91      0.79      0.85       358

    accuracy                           0.90      1034
   macro avg       0.90      0.88      0.89      1034
weighted avg       0.90      0.90      0.90      1034
```

Decision Tree Classification

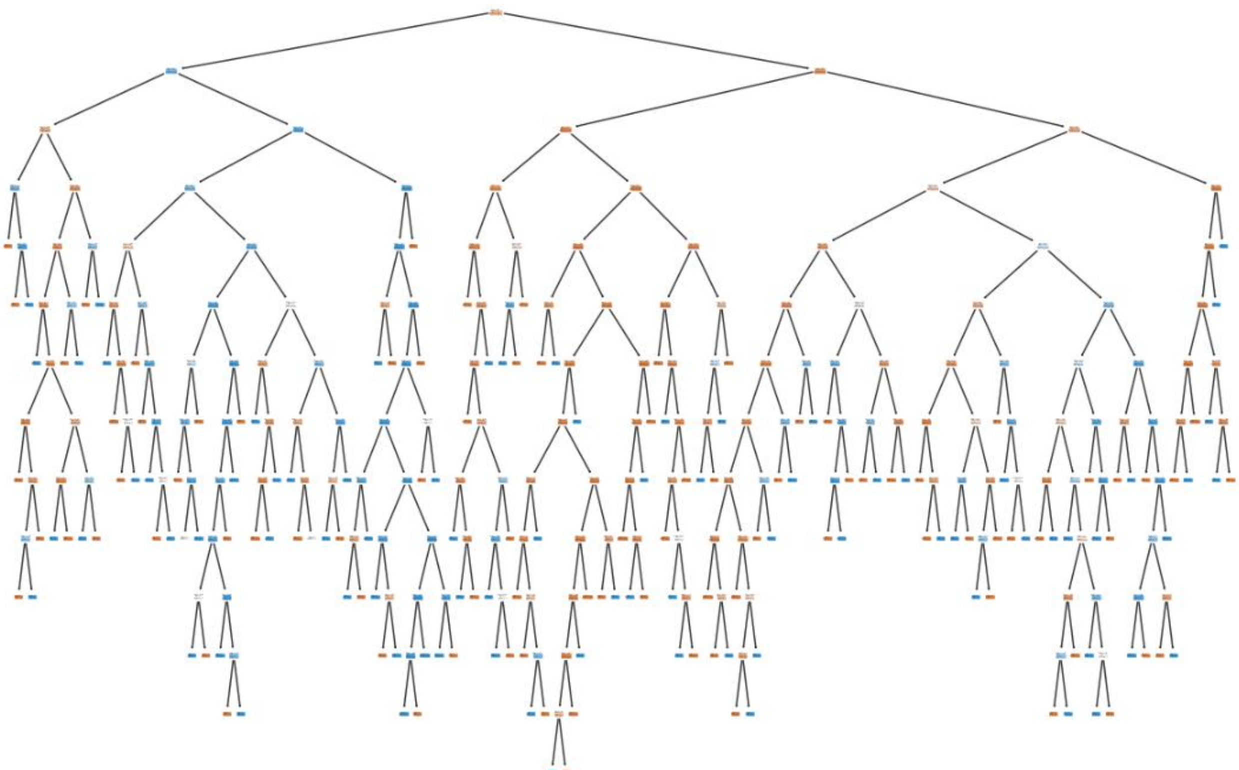
The splitting of the dataset has been done as follows.

80% - Training data

20% - Testing data

Random state – 0

The resulting decision tree before pruning:

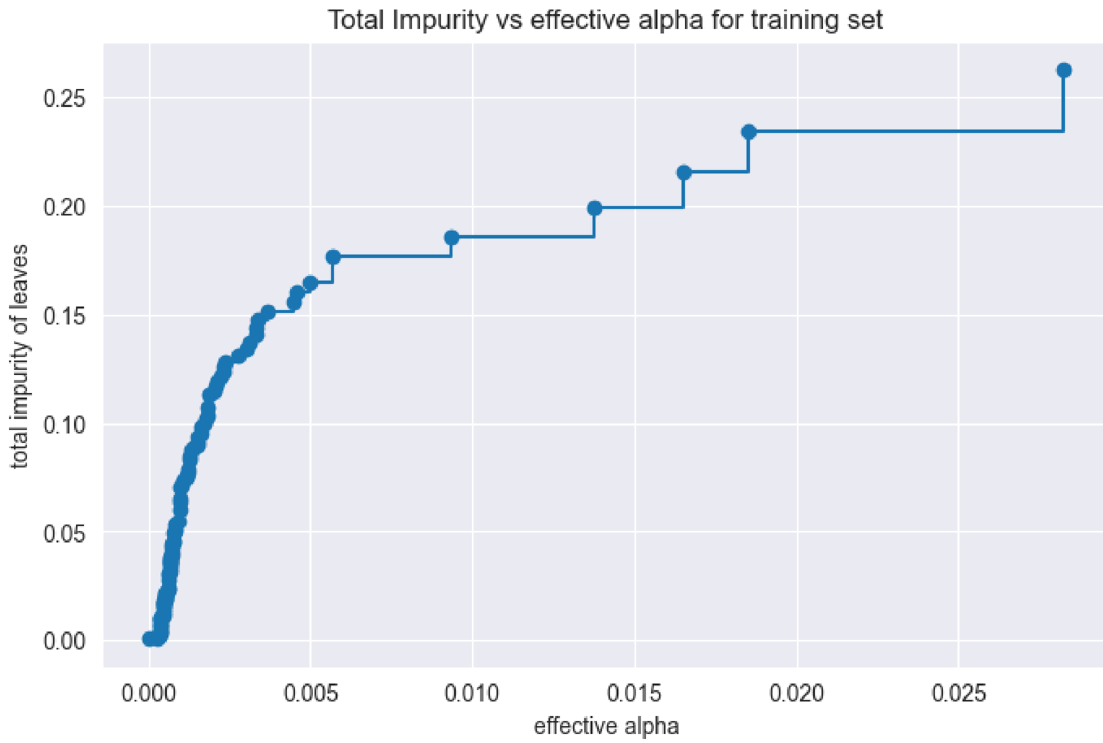


Accuracy of training data achieved: 99.92743105950653 %

Accuracy of testing data achieved: 87.10144927536231 %

Since there is a considerable gap between testing accuracy and training accuracy, the model has to be called overfitted. Pruning the decision tree has solved this issue.

Pruning the Decision Tree



The first elbow point (0.003) was taken as the optimal ccp_alpha value.

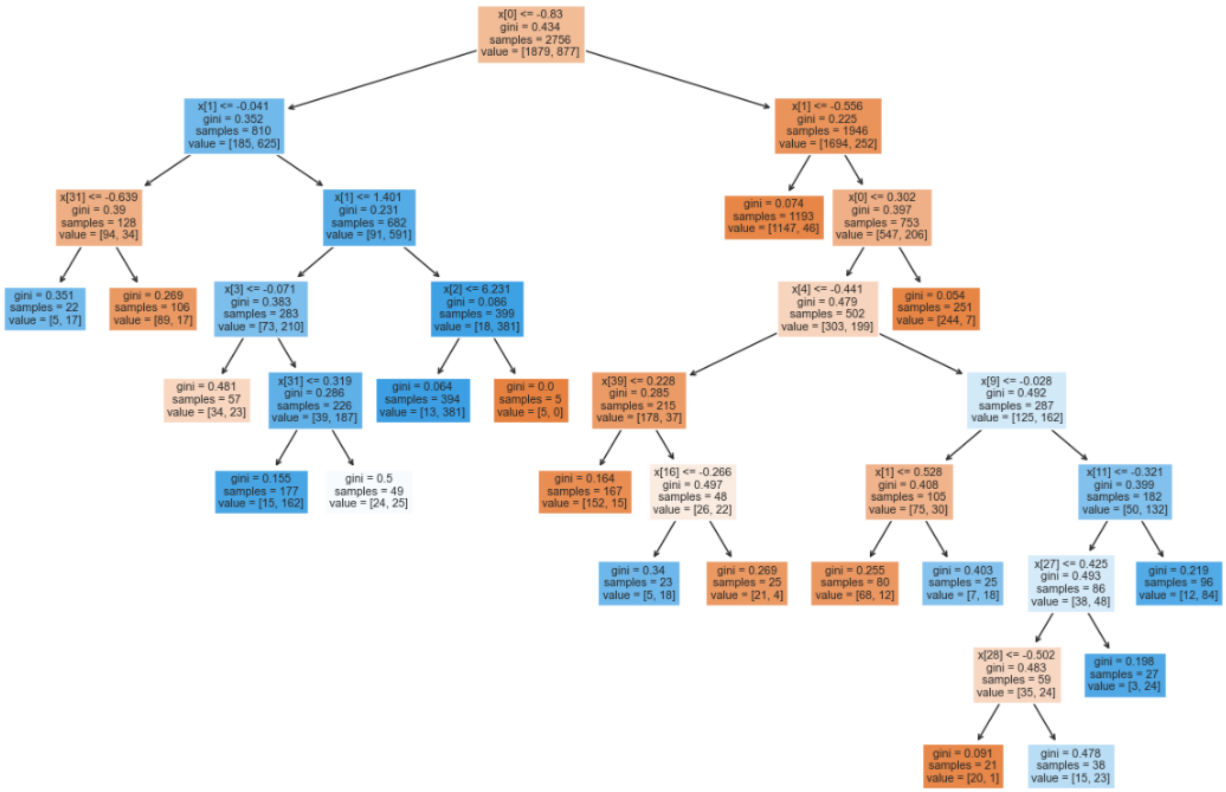
Accuracy of training and testing data after pruning:

Accuracy of training data achieved: 91.8722786647315 %

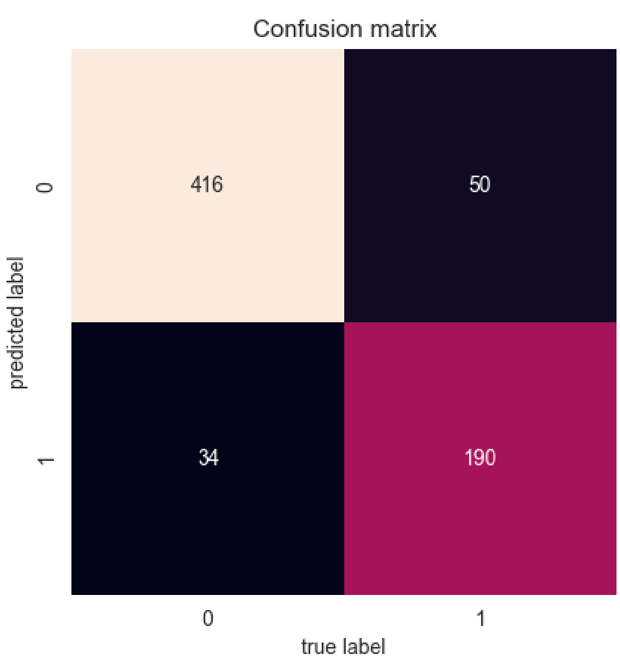
Accuracy of testing data achieved: 87.82608695652175 %

The above accuracy values of training and testing data confirmed the overfitting was treated successfully.

Pruned Decision Tree:



Confusion Matrix



Spam = 1 (positives)

Non-spam = 0 (negatives)

Accuracy = (True Positives + True Negatives)/(True Positives + False Positives + True Negatives + False Negatives)

Recall = (True Positives)/(True Positives + False Negatives)

Precision = (True Positives)/(True Positives + False Positives)

F1 Score = 2* (Precision* Recall)/(Precision+ Recall)

- True Positives - instances that are positive and have been correctly classified as positive by the classification model.
- True Negatives - instances that are negative and have been correctly classified as negative by the classification model.
- False Positives - instances that are negative but have been incorrectly classified as positive by the classification model.
- False Negatives - instances that are positive but have been incorrectly classified as negative by the classification model.

```
Classification Report :

              precision    recall  f1-score   support

           0       0.89      0.92      0.91       450
           1       0.85      0.79      0.82       240

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

Comparison of results in KNN and Decision Tree Classifiers

	KNN	Decision Tree
Accuracy of Training data	92.24709784411277	91.8722786647315
Accuracy of Testing data	90.13539651837525	87.82608695652175

Limitations

- The dataset is old (created in 1999). So, the resulting predictions may not be helpful in present-day emails.
- The dataset had 4601 email data where 1813 were spam and 2788 were non-spam. This could affect the predicted result because more data is biased toward non-spam.
- Fewer email data (4601) which is not sufficient for a project like this. Because there are billions of data collected with emails in the present day.
- When dropping the duplicates and null values, lots of rows had to drop. It lessens the amount of data further.
- The decision tree has to be pruned because of overfitting.

Future Enhancements

- Can use machine learning models such as SVM and Random Forests and see for better results.
- Use data augmentation to generate more data for the trained model. Techniques like data synthesis could create more similar data to the original data.
- Going with a recently updated dataset with a large volume can be used to make good predictions in real-time emails.
- Scaling techniques such as min-max scaling and chi-squared test can be applied to improve the performance of the model.
- Use hyperparameter tuning techniques other than grid search.
  e.x: Random search, Bayesian optimization, genetic algorithms etc.

Source Code

Link to GitHub repository: https://github.com/Sathila01/Machine-Learning-CourseWork.git

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import re

%matplotlib inline
with open("spambase/spambase.names") as spam:
 text = spam.read()
labels = re.findall(r'\n(\w*_?\W?):', text)
#labels.append('Class')
df = pd.read_csv("spambase/spambase.data", header=None, names=labels
+['spam'])

df.head()

df.info()

df.duplicated()

#Dropping duplicates

df.drop_duplicates(inplace=True)
df.info()

#Finding Outliers

fig = plt.figure(figsize =(200, 100))
df.plot.box(title='BoxPlot Spambase',rot=90) #rot = axis rotation

plt.show()

import seaborn as sn
sn.boxplot(x = df['capital_run_length_total'])
sn.boxplot(x = df['capital_run_length_average'])
sn.boxplot(x = df['capital_run_length_longest'])

#Using IQR technique to make all outliers to null values

for x in
['capital_run_length_total','capital_run_length_longest','capital_run_length_
average']:
    q75,q25 = np.percentile(df.loc[:,x],[75,25])
    intr_qr = q75-q25

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

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

```
sn.boxplot(x = df['capital_run_length_total'])
sn.boxplot(x = df['capital_run_length_average'])
sn.boxplot(x = df['capital_run_length_longest'])

df.isnull().sum()


# Drop all rows with NaN values
preprocessed_df=df.dropna(axis=0)

# Reset index after drop
preprocessed_df=df.dropna().reset_index(drop=True)
preprocessed_df

#Removing the target column

target_dropped_df = preprocessed_df.drop(labels=['spam'], axis=1)
target_dropped_df.head()

target_dropped_df.describe()

#Standard Deviation before performing Standard Scaling

# Calculate the standard deviation of all columns
sd = target_dropped_df.std()

# Plot the standard deviation of all columns
plt.bar(range(len(sd)), sd)
plt.title("Standard Deviation -before standard scaling")
plt.xlabel("Index")
plt.ylabel("Standard Deviation")
plt.show()

#Mean before performing Standard Scaling

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

# Plot the mean of all columns
plt.bar(range(len(mean)), mean)
plt.title("Mean -before standard scaling")
plt.xlabel("Index")
plt.ylabel("Mean")
plt.show()

#Performing Standard Scaling

from sklearn.preprocessing import StandardScaler

scaler=StandardScaler()
scaled_data=scaler.fit_transform(target_dropped_df)
standardized_df=pd.DataFrame(data=scaled_data, columns=
target_dropped_df.columns)

standardized_df
```

```
standardized_df.describe()

#Standard Deviation after performing Standard Scaling

# Calculate the standard deviation of all columns
sd = standardized_df.std()

# Plot the standard deviation of all columns
plt.bar(range(len(sd)), sd)
plt.title("Standard Deviation -after standard scaling")
plt.xlabel("Index")
plt.ylabel("Standard Deviation")
plt.show()


#Mean after performing Standard Scaling

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

# Plot the mean of all columns
plt.bar(range(len(mean)), mean)
plt.title("Mean -after standard scaling")
plt.xlabel("Index")
plt.ylabel("Mean")
plt.show()

#Performing PCA to dataset

from sklearn.decomposition import PCA
pca = PCA()

principalComponents = pca.fit_transform(standardized_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()

pca = PCA(n_components=44)
new_data = pca.fit_transform(standardized_df)

# This will be the new data fed to the algorithm.
pca_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','PC37','PC38','PC39','PC40','PC41','PC42','PC43',
'PC44'])
```

```
pca_df

print(pca.explained_variance_)

print(pca.components_)

# Defining Independent variable and Dependant variable

X = pca_df.iloc[:,0:44].values
y = preprocessed_df.iloc[:, 57].values
```

KNN

```
# Splitting the dataset into the Training set and Testing set

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30,
random_state = 0)

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

# Perform grid search to find optimal value of k
param_grid = {'n_neighbors': [1, 3, 5, 7, 9]}
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Print optimal value of k
optimal_k_value = grid_search.best_params_['n_neighbors']
print('Optimal value of k:', optimal_k_value)

# Fitting classifier to the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=optimal_k_value,
metric='minkowski', p=2)

classifier.fit(X_train,y_train)
# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm

from sklearn.metrics import confusion_matrix
import seaborn as sns

# 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 label')
plt.ylabel('Predicted label')


from sklearn.metrics import classification_report

print('Classification Report : \n')
print(classification_report(y_test, y_pred)) #support - no. of samples in the test set


from sklearn.metrics import accuracy_score
print("KNN accuracy of testing dataset : ",accuracy_score(y_pred,y_test)*100)


y_pred2 = classifier.predict(X_train)
print("KNN accuracy of training dataset : ", accuracy_score(y_pred2, y_train) * 100)
```

Decision Tree

```
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)


from sklearn.tree import DecisionTreeClassifier

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

from sklearn.metrics import accuracy_score

predictions_test=clf.predict(X_test)
print("Decision Tree accuracy of testing dataset : ",accuracy_score(y_test,predictions_test)*100)

predictions_train = clf.predict(X_train)
print("Decision Tree accuracy of training dataset : ",accuracy_score(y_train,predictions_train)*100)

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

# 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)#elbow point
clf.fit(X_train,y_train)

predictions_test=clf.predict(X_test)
print("Accuracy of testing data after pruning :
",accuracy_score(y_test,predictions_test)*100)

predictions_train = clf.predict(X_train)
print("Accuracy of training data after pruning :
",accuracy_score(y_train,predictions_train)*100)

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

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 label')
plt.ylabel('predicted label')

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