Informatics Institute of Technology
Department of Computing

Bsc(Hons) Artificial Intelligence and Data Science

Module: CM2604 Machine Learning

Module Coordinator: Mr. Prasan Yapa

Coursework Report

Vinuwara Ronath Jayasuriya RGU
ID – 2119942

IIT Student No. – 20210167

Git repository: https://github.com/vinuwara/Machine-Learning-CW-Vinuwara.git

Spam dataset is done using the Decision Trees Classification and K Nearest Neighbors Classification (KNN)
Preprocessing techniques

Corpus Preparation

Data Cleaning: This was done by removing duplicates and null values from the dataset.

Removing duplicate in dataset.

Getting the duplicates in the dataset
Data_set.duplicated()
0 False 1 False 2 False 3 False 4 False ... 4596 False 4597 False 4598 False 4599 False 4600 False Length: 4601, dtype: bool
removing the duplicate values
Data_set.drop_duplicates(inplace=True)

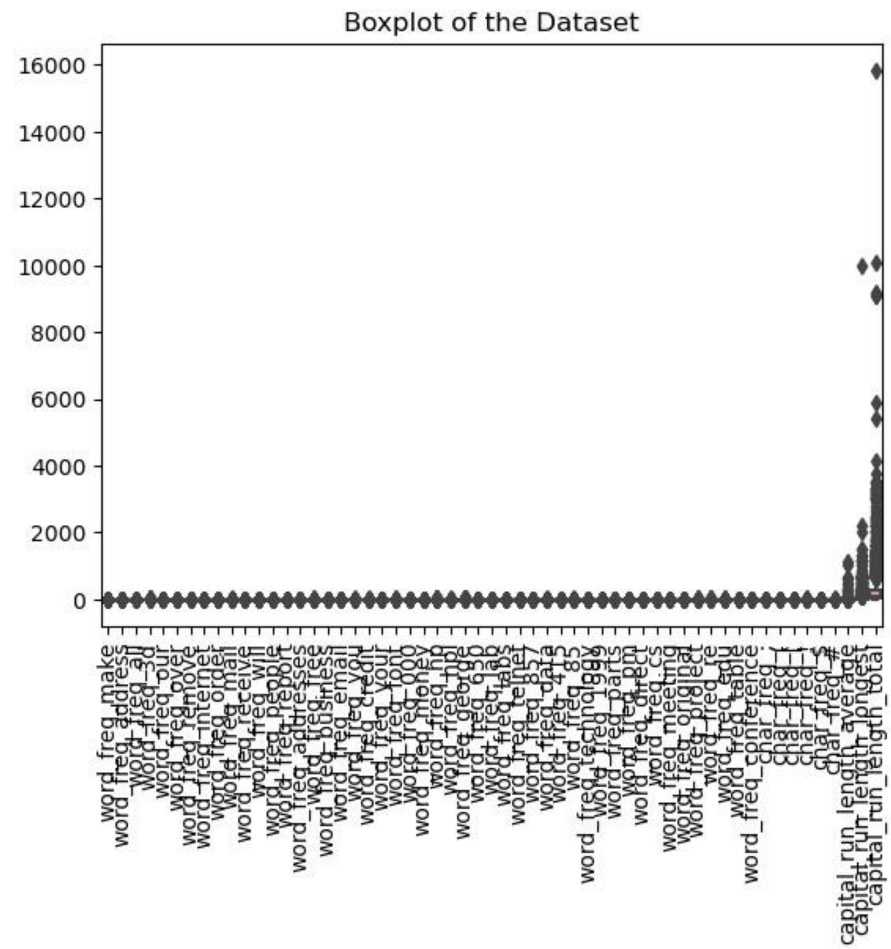
Removing null values

Data_set.isna().sum()

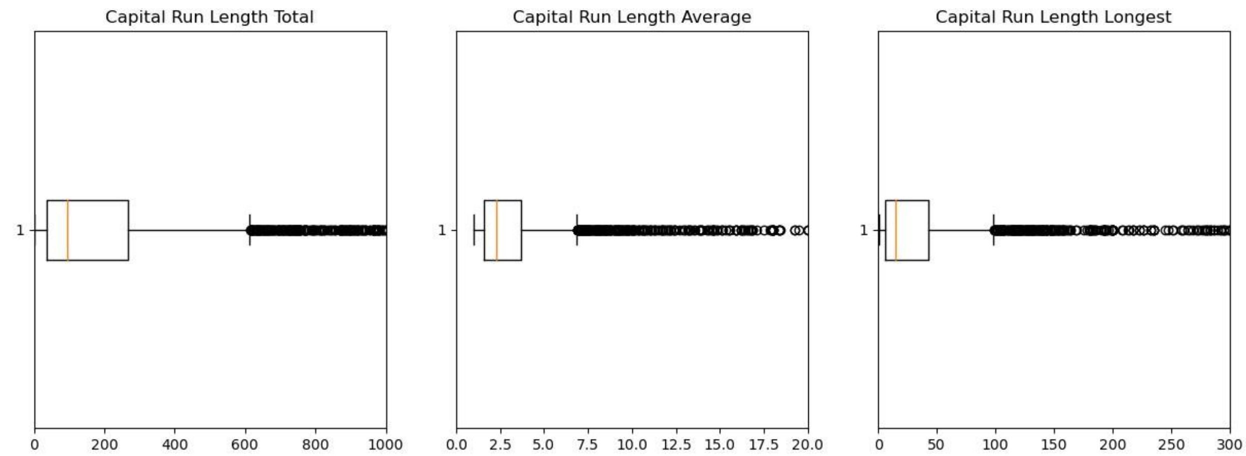
word_freq_make	0	word_freq_address	0
word_freq_all	0	word_freq_3d	0
word_freq_our	0	word_freq_over	0
word_freq_remove	0	word_freq_internet	0
word_freq_order	0	word_freq_mail	0
word_freq_receive	0	word_freq_will	0
word_freq_people	0	word_freq_report	0
word_freq_addresses	0	word_freq_free	0
word_freq_business	0		

word_freq_email 0 word_freq_you 0 word_freq_credit 0 word_freq_your 0 word_freq_font 0 word_freq_000 0 word_freq_money 0 word_freq_hp 0 word_freq_hpl 0 word_freq_george 0 word_freq_650 0 word_freq_lab 0 word_freq_labs 0 word_freq_telnet 0 word_freq_857 0 word_freq_data 0 word_freq_415 0 word_freq_85 0 word_freq_technology 0 word_freq_1999 0 word_freq_parts 0 word_freq_pm 0 word_freq_direct 0 word_freq_cs 0 word_freq_meeting 0 word_freq_original 0 word_freq_project 0 word_freq_re 0 word_freq_edu 0 word_freq_table 0 word_freq_conference 0 char_freq_; 0 char_freq_(0 char_freq_[0 char_freq_! 0 char_freq_$ 0 char_freq_# 0 capital_run_length_average 0 capital_run_length_longest 0 capital_run_length_total 0 spam 0 dtype: int64
Removing the target column
data=Data_set.drop(labels=['spam'], axis=1) data.head()

Data Transformation: outliers were removed using standard scaling.

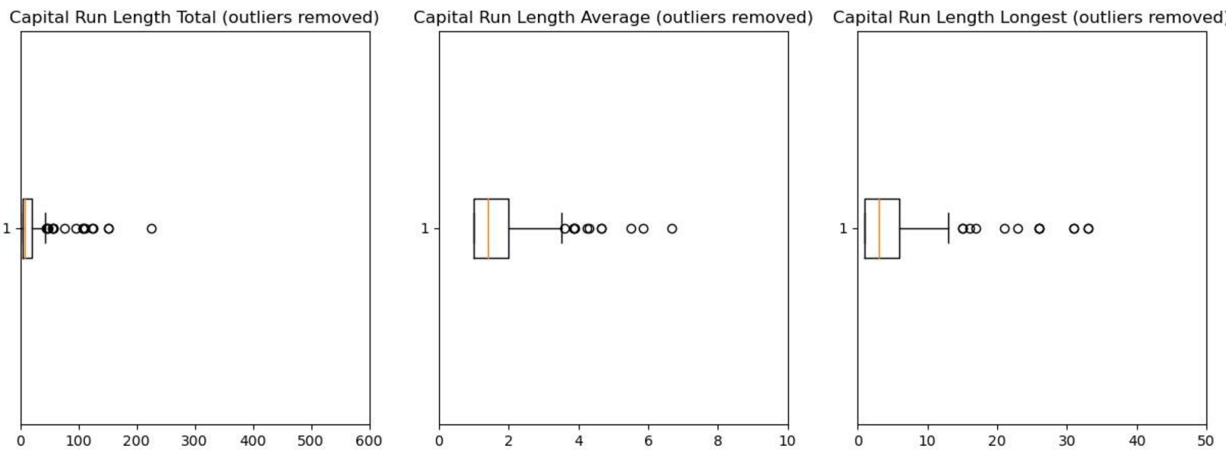


Since capital_run_lenghth_avarage,capital_run_length_longest and Capital_run_lenghth_total are outliers



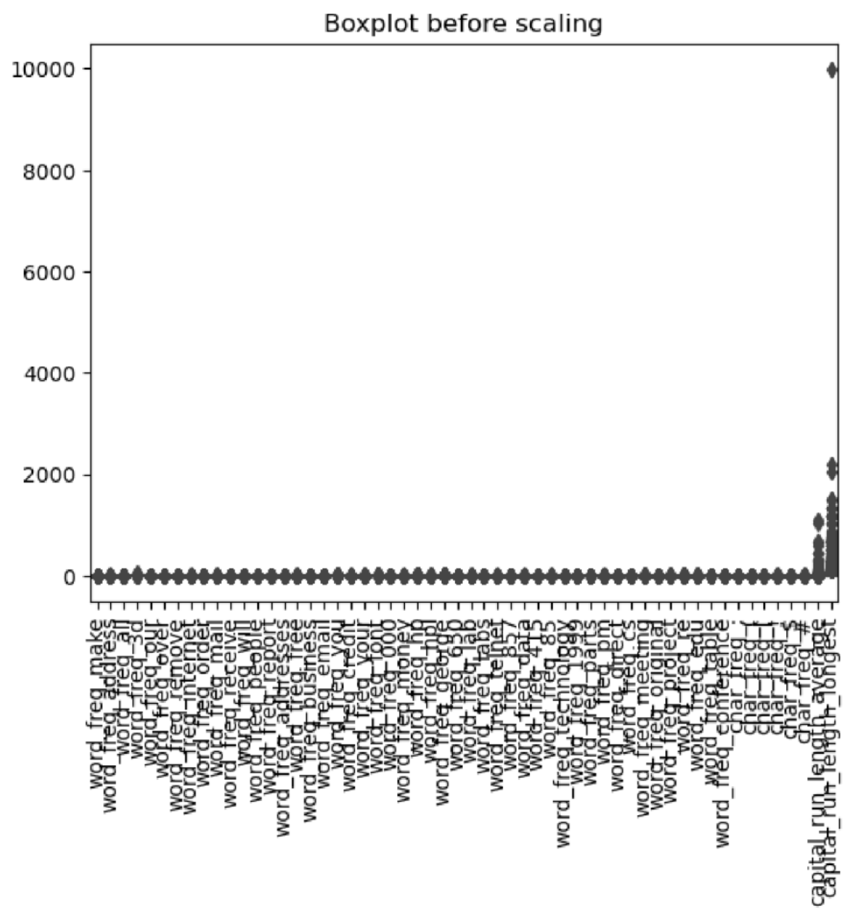
Remove the outliers using the IQR method code

```
import numpy as np

# Calculate the interquartile range for each column
Q1 = Data_set.quantile(0.25)
Q3 = Data_set.quantile(0.75)
IQR = Q3 - Q1

# Remove the outliers using the IQR method
Data_set_outliers_removed = Data_set[~((Data_set < (Q1 - 1.5 * IQR)) | (Data_set > (Q3 + 1.5 *
IQR))).any(axis=1)]

# Create individual box plots for the columns with outliers removed fig,
axs = plt.subplots(1, 3, figsize=(15, 5))

axs[0].boxplot(Data_set_outliers_removed['capital_run_length_total'], vert=False)
axs[0].set_title('Capital Run Length Total (outliers removed)') axs[0].set_xlabel('')
axs[0].set_xlim([0, 600])

axs[1].boxplot(Data_set_outliers_removed['capital_run_length_average'], vert=False)
axs[1].set_title('Capital Run Length Average (outliers removed)') axs[1].set_xlabel('')
axs[1].set_xlim([0, 10])

axs[2].boxplot(Data_set_outliers_removed['capital_run_length_longest'], vert=False)
axs[2].set_title('Capital Run Length Longest (outliers removed)') axs[2].set_xlabel('')
axs[2].set_xlim([0, 50])

plt.show()
```

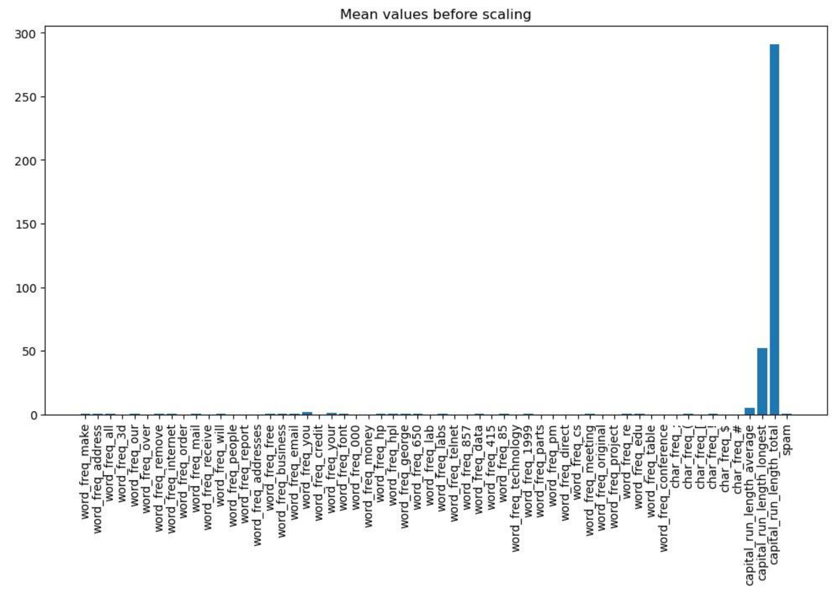


Used Standard Scaler on the dataset

Before standard scaler



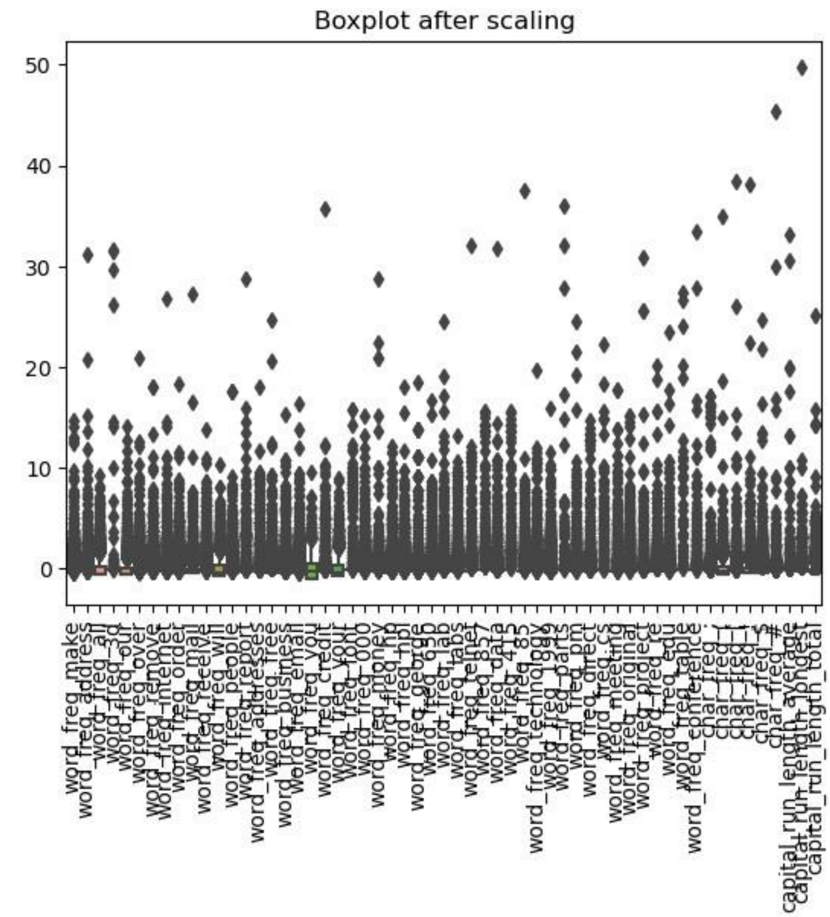
Mean



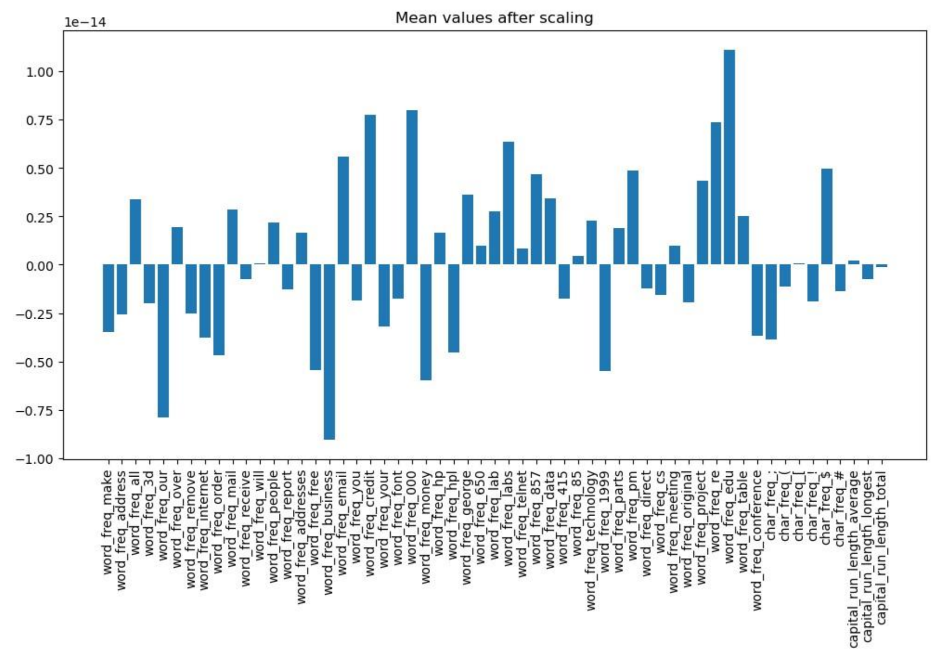
Code – performing standard scaler on the dataset

```
# Perform Standard Scaling on the data scaler =
StandardScaler() scaled_data =
scaler.fit_transform(Data_set.iloc[:, :-1])
Data_scaled = pd.DataFrame(data=scaled_data, columns=Data_set.columns[:-
1])
```

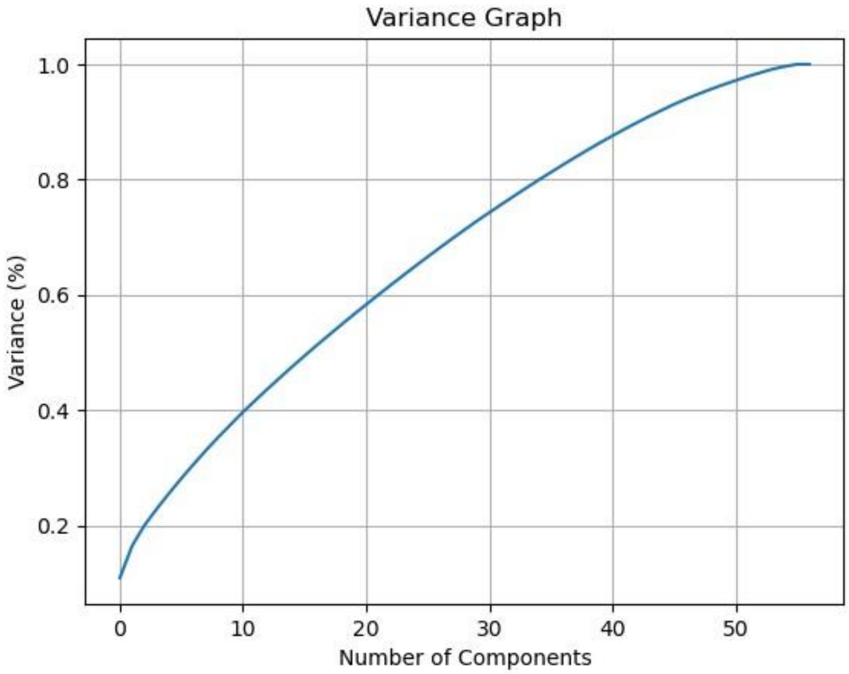
Boxplot after performing the standard scaler on the dataset


Boxplot after scaling
50
40
30
20
10
0

Mean


Mean values after scaling
1e−14
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
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
word_freq_receive
word_freq_will
word_freq_people
word_freq_report
word_freq_addresses
word_freq_free
word_freq_business
word_freq_email
word_freq_you
word_freq_credit
word_freq_your
word_freq_font
word_freq_000
word_freq_money
word_freq_hp
word_freq_hpl
word_freq_george
word_freq_650
word_freq_lab
word_freq_labs
word_freq_telnet
word_freq_857
word_freq_data
word_freq_415
word_freq_85
word_freq_technology
word_freq_1999
word_freq_parts
word_freq_pm
word_freq_direct
word_freq_cs
word_freq_meeting
word_freq_original
word_freq_project
word_freq_re
word_freq_edu
word_freq_table
word_freq_conference
char_freq_;
char_freq_(
char_freq_[
char_freq_!
char_freq_$
char_freq_#
capital_run_length_average
capital_run_length_longest
capital_run_length_total

Variance graph



Performing PCA to the Dataset

Code

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

# 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','PC37','PC38','PC39','PC40',
    'PC41','PC42','PC43','PC44'
])
```

Splitting the dataset into the Training set and Test set

- 20 percent of the dataset was used to test the dataset.
- 80 percent of the dataset was used to train the dataset

Code

```
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)
```

Decision Tree Classification

Accuracy of testing dataset : 0.9560570071258907

Summary of the dataset code

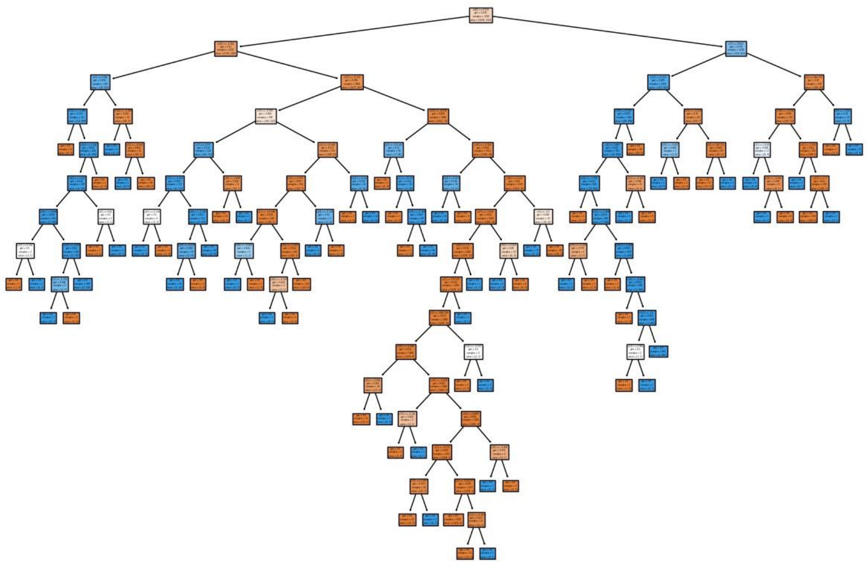
```
from sklearn.metrics import classification_report predicted_labels
= clf.predict(X_test)
classification_report = classification_report(y_test, predicted_labels)

print(classification_report)
```

Classification report

	precision	recall	f1-score	support
0	0.96	0.96	0.96	495
1	0.94	0.94	0.94	347
accuracy			0.95	842
macro avg	0.95	0.95	0.95	842
weighted avg	0.95	0.95	0.95	842

Accuracy of the training dataset of the decision tree: 1.0

Decision tree



Confusion Matrix before pruning

Equations

True positive rate: measures the proportion of positive instances correctly identified as positive.

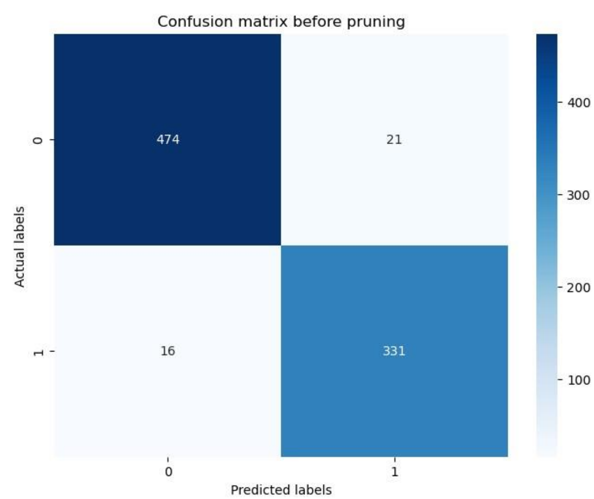
- FN = False Negative : Number of positive instances that are incorrectly classified as positive.

$$TPR = TP / (TP + FN)$$

False positive rate : measures the proportion of negative instances incorrectly identified as positive.

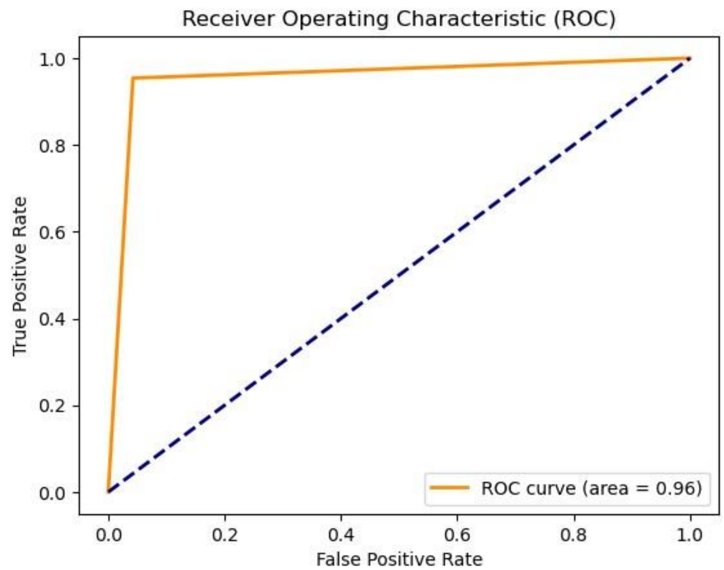
- TN = true negative : The number of negative instances that are correctly classified as negative FPR = FP / (FP + TN

Experiments



- TP :474 mails are correctly identified as not spam.
- FP : 21 mails are incorrectly identified as spam as not spam.
- TN :331 mails are correctly identified as spam.
- FN : 16 mails are incorrectly identified as not spam as spam.

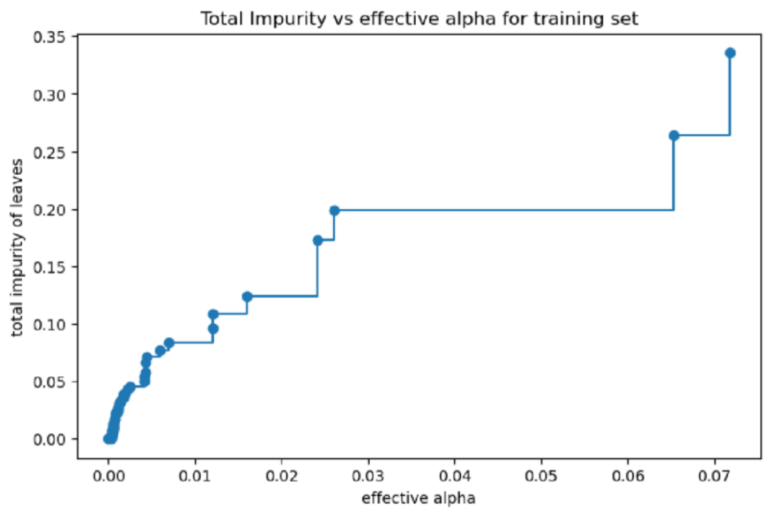
Receiver Operating Characteristics (ROC) : A plot to test sensitivity as the y coordinate versus its 1-specificity or false positive rate (FPR) as the x coordinate, is an effective method of evaluating the performance of diagnostic tests.

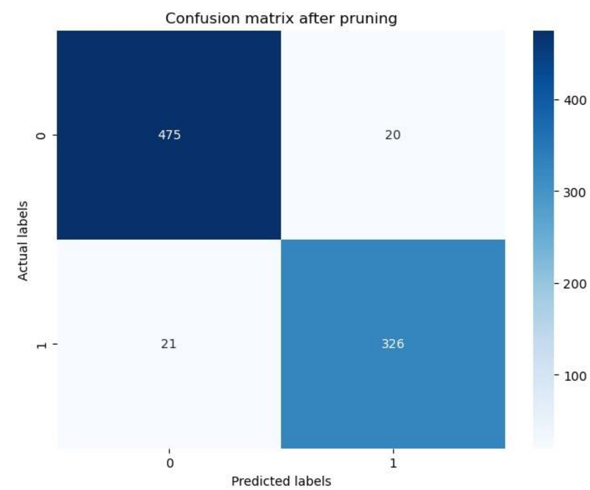


AUC score for the code: 0.96

Pruning the Decision tree : Pruning is a technique that removes the parts of the Decision Tree which prevent it from growing to its full depth. The parts that it removes from the tree are the parts that do not provide the power to classify instances.

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.

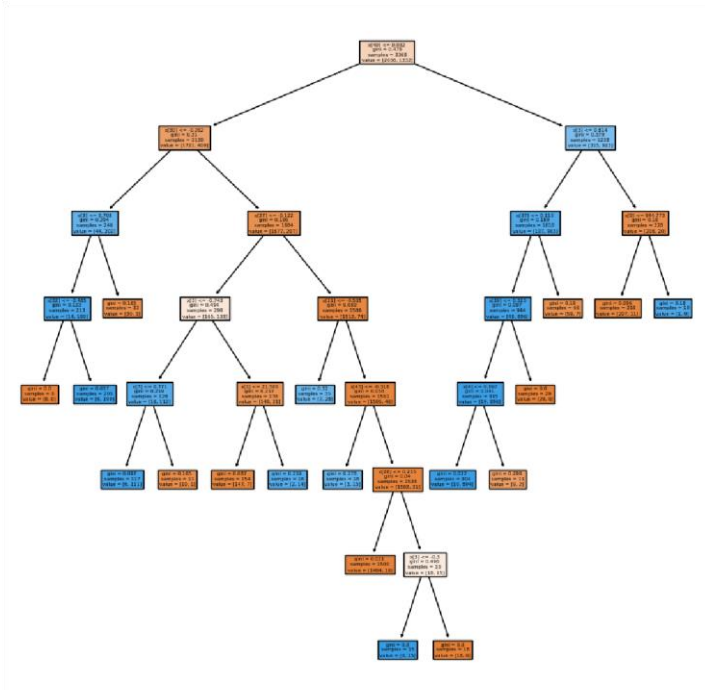




- TP :475 mails are correctly identified as not spam.
- FP : 20 mails are incorrectly identified as spam as not spam.
- TN :326 mails are correctly identified as spam.
- FN : 21 mails are incorrectly identified as not spam as spam.

	precision	recall	f1-score	support
0	0.96	0.96	0.96	495
1	0.94	0.94	0.94	347
accuracy			0.95	842
macro avg	0.95	0.95	0.95	842
weighted avg	0.95	0.95	0.95	842

Decision Tree obtaining after pruning.



K Nearest Neighbors (KNN) Classification

I have removed outliers , duplicates , null values , scalling and pca to the knn classification same as in the decision tree.

Find optimal K value

Code

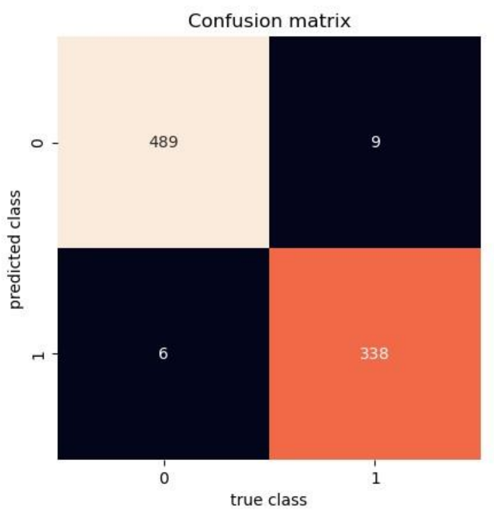
```
from sklearn.model_selection import GridSearchCV from
sklearn.neighbors import KNeighborsClassifier
# Grid search to find optimal value of k param_grid
= {'n_neighbors': [1, 3, 5, 7, 9]}
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5)
grid_search.fit(X_train, y_train)

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

Perform 10-fold cross validation

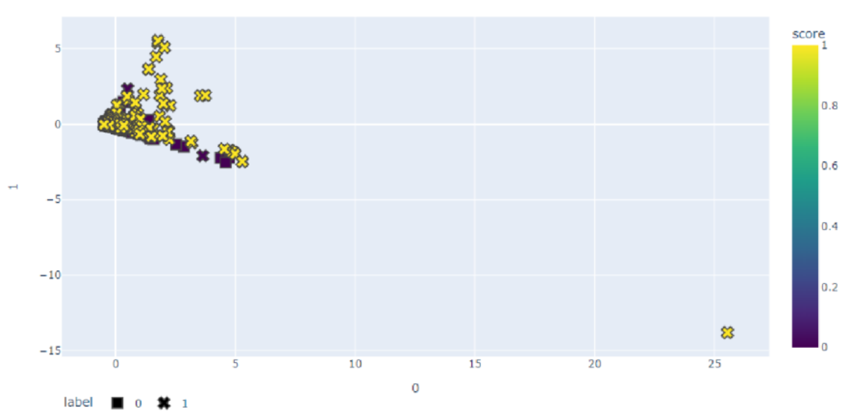
Code

Feature Scaling sc = StandardScaler() X_train = sc.fit_transform(X_train) X_test = sc.transform(X_test)
Fitting classifier to the Training set #k =5 has been taken classifier = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2) classifier.fit(X_train,y_train)

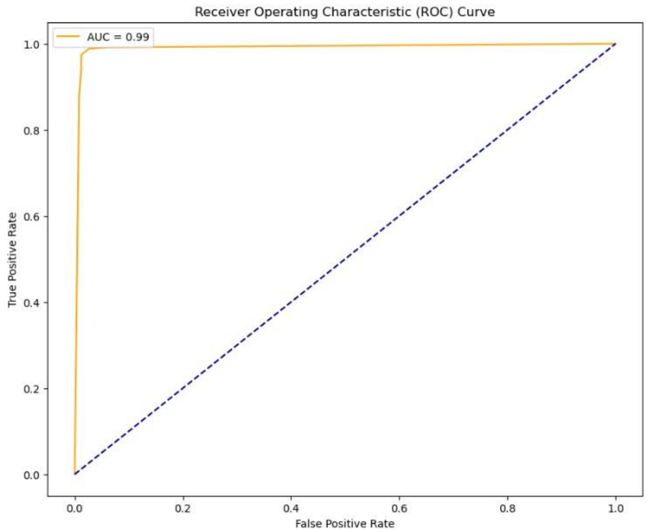
```
Accuracy score of email prediction using KNN :  98.2185273159145
```



- TP :489 mails are correctly identified as not spam.
- FP : 9 mails are incorrectly identified as spam as not spam.
- TN :338 mails are correctly identified as spam.
- FN : 6 mails are incorrectly identified as not spam as spam.



ROC curve for the KNN classifer



AUC score for the code: 0.99

Limitations & Enhancements

Limitations

- Due to the dataset being small the accuracy of the model would be effected.
- Feature engineering step may not have included all the relevant information, making the model performance suboptimal.
- Dataset is old.
- Lots of rows were dropped when dropping

Future Enhancements

- Training the model with a larger dataset would increase the accuracy of it
- Utilizing other machine learning algorithms ○ Support vector machines - SVM ○ Random Forest
- Experiment with different feature engineering techniques

Appendix

Decision Tree code

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

```python
import seaborn as sns
import re
from sklearn import tree
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier


# In[2]:


with open("spambase.names") as spam:
    text = spam.read()
    labels = re.findall(r'\n(\w*_?\W?):', text)
#labels.append('Class')
Data_set = pd.read_csv("spambase.data", header=None, names=labels +['spam'])
# spam = spamData.pop('spam')
Data_array=Data_set.values
# print(Data_array)


# In[3]:


Data_set.head()


# In[4]:


print("Number of rows before preprocessing : ", len(Data_set))


# Boxplot of spam no spam email

# In[38]:


sns.boxplot(data=Data_set.iloc[:, :-1])
plt.xticks(rotation=90)
plt.title("Boxplot of the Dataset")
plt.show()


# Individual box plots of the above columns before removing the outliers

# In[6]:


import matplotlib.pyplot as plt

# Create individual box plots for the columns with outliers
fig, axs = plt.subplots(1, 3, figsize=(15, 5))

axs[0].boxplot(Data_set['capital_run_length_total'], vert=False)
```

```
axs[0].set_title('Capital Run Length Total')
axs[0].set_xlabel('')
axs[0].set_xlim([0, 1000])

axs[1].boxplot(Data_set['capital_run_length_average'], vert=False)
axs[1].set_title('Capital Run Length Average')
axs[1].set_xlabel('')
axs[1].set_xlim([0, 20])

axs[2].boxplot(Data_set['capital_run_length_longest'], vert=False)
axs[2].set_title('Capital Run Length Longest')
axs[2].set_xlabel('')
axs[2].set_xlim([0, 300])

plt.show()


# In[7]:


import numpy as np

# Calculate the interquartile range for each column
Q1 = Data_set.quantile(0.25)
Q3 = Data_set.quantile(0.75)
IQR = Q3 - Q1

# Remove the outliers using the IQR method
Data_set_outliers_removed = Data_set[~((Data_set < (Q1 - 1.5 * IQR)) |
(Data_set > (Q3 + 1.5 * IQR))).any(axis=1)]

# Create individual box plots for the columns with outliers removed
fig, axs = plt.subplots(1, 3, figsize=(15, 5))

axs[0].boxplot(Data_set_outliers_removed['capital_run_length_total'],
vert=False)
axs[0].set_title('Capital Run Length Total (outliers removed)')
axs[0].set_xlabel('')
axs[0].set_xlim([0, 600])

axs[1].boxplot(Data_set_outliers_removed['capital_run_length_average'],
vert=False)
axs[1].set_title('Capital Run Length Average (outliers removed)')
axs[1].set_xlabel('')
axs[1].set_xlim([0, 10])

axs[2].boxplot(Data_set_outliers_removed['capital_run_length_longest'],
vert=False)
axs[2].set_title('Capital Run Length Longest (outliers removed)')
axs[2].set_xlabel('')
axs[2].set_xlim([0, 50])

plt.show()


# getting the duplicates in the dataset
```

```
# In[8]:


Data_set.duplicated()


# removing the duplicate values

# In[9]:


Data_set.drop_duplicates(inplace=True)


# In[10]:


print("Number of rows after removing duplicates : ", len(Data_set))


# All null values in the dataset

# individual box plots of the above columns before removing the outliers

# In[11]:


Data_set.isna().sum()


# Removing the target column

# In[12]:


data=Data_set.drop(labels=['spam'], axis=1)
data.head()


# In[13]:


print("Number of rows after preprocessing : ", len(Data_set))


# Summary of dataset before performing Standard Scaler

# In[14]:


data.describe()


#  Box plot before scaling
#

# In[15]:
```

```python
import matplotlib.pyplot as plt
import seaborn as sns

sns.boxplot(data=data.iloc[:, :-1])
plt.xticks(rotation=90)
plt.title("Boxplot before scaling")

# Save the plot as a PNG file
plt.savefig('boxplot_before_scaling.png')


# Standard Scaling for the dataset

# In[16]:


# Compute mean values before scaling
mean_before = Data_set.mean()

# Create a bar plot of mean values before scaling
plt.figure(figsize=(12, 6))
plt.bar(mean_before.index, mean_before.values)
plt.xticks(rotation=90)
plt.title('Mean values before scaling')

# Perform Standard Scaling on the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(Data_set.iloc[:, :-1])
Data_scaled = pd.DataFrame(data=scaled_data, columns=Data_set.columns[:-1])

# Compute mean values after scaling
mean_after = Data_scaled.mean()

# Create a bar plot of mean values after scaling
plt.figure(figsize=(12, 6))
plt.bar(mean_after.index, mean_after.values)
plt.xticks(rotation=90)
plt.title('Mean values after scaling')

plt.show()
plt.savefig('boxplot_before_scaling.png')


# In[39]:


import matplotlib.pyplot as plt
import seaborn as sns

# Create box plot after scaling
plt.xticks(rotation=90)
plt.title("Boxplot after scaling")


sns.boxplot(data=Data_scaled, orient='v')
```

```python
plt.show()


# Summmary of dataset after performing Standard Scaling

# In[18]:


Data_set.describe()


# Performing PCA to the Dataset

# In[19]:


pca = PCA()

principalComponents = pca.fit_transform(Data_scaled)

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

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

plt.title('Variance Graph')
plt.grid(True)
plt.show()


# introducing the PCA components

# In[20]:


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

# 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','PC37','PC38','PC39','PC40',
    'PC41','PC42','PC43','PC44'
])


# After performing PCA

# In[21]:


principal_Df.head()
```

```
# In[22]:


pca.explained_variance_


# In[23]:


pca.components_


# predictive model by appling K-Nearest Neighbors (KNN) algorithm

# In[24]:


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


#     Splitting the dataset into the Training set and Test set

# In[25]:


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

# In[26]:


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


# checking Accuracy of testing dataset

# In[27]:


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


# Checking accuracy of training dataset
#

# In[28]:


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

```
# Visualizing final decision tree
#

# In[29]:


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


# confusion matrix before pruning

# In[30]:


import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Assuming you have already trained and tested your decision tree model, and
have obtained the predicted labels and actual labels
predicted_labels = clf.predict(X_test)
cm = confusion_matrix(y_test, predicted_labels)

# Plotting the confusion matrix as a heatmap
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted labels')
plt.ylabel('Actual labels')
plt.title('Confusion matrix before pruning')
plt.show()


# In[31]:


from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Assuming you have predicted scores or probabilities for binary
classification
y_pred_scores = clf.predict_proba(X_test)[:,1]

# Calculate ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred_scores)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area =
%0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
```

```python
plt.show()


# In[42]:


roc_auc = auc(fpr, tpr)
print("AUC score: {:.2f}".format(roc_auc))


# Pruning desision tree

# In[32]:


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


# In[33]:


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")


# In[34]:


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


# In[44]:


import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Assuming you have already trained and tested your decision tree model, and
have obtained the predicted labels and actual labels
predicted_labels = clf.predict(X_test)
cm = confusion_matrix(y_test, predicted_labels)

# Plotting the confusion matrix as a heatmap
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted labels')
plt.ylabel('Actual labels')
plt.title('Confusion matrix after pruning')
plt.show()
```

```
# In[36]:


from sklearn.metrics import classification_report
predicted_labels = clf.predict(X_test)
classification_report = classification_report(y_test, predicted_labels)

print(classification_report)


# In[48]:


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


# In[ ]:
```

KNN – code

```
#!/usr/bin/env python
# coding: utf-8

# ## Decision Tree Classifier

# In[1]:


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
from sklearn import tree
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier


# In[2]:


with open("spambase.names") as spam:
    text = spam.read()
```

```
    labels = re.findall(r'\n(\w*_?\W?):', text)
#labels.append('Class')
Data_set = pd.read_csv("spambase.data", header=None, names=labels +['spam'])
# spam = spamData.pop('spam')
Data_array=Data_set.values
# print(Data_array)


# In[3]:


Data_set.head()


# In[4]:


print("Number of rows before preprocessing : ", len(Data_set))


# Boxplot of spam no spam email

# In[5]:


sns.boxplot(data=Data_set.iloc[:, :-1])
plt.xticks(rotation=90)
plt.title("Boxplot of the Dataset")
plt.show()


# Individual box plots of the above columns before removing the outliers

# In[6]:


import matplotlib.pyplot as plt

# Create individual box plots for the columns with outliers
fig, axs = plt.subplots(1, 3, figsize=(15, 5))

axs[0].boxplot(Data_set['capital_run_length_total'], vert=False)
axs[0].set_title('Capital Run Length Total')
axs[0].set_xlabel('')
axs[0].set_xlim([0, 1000])

axs[1].boxplot(Data_set['capital_run_length_average'], vert=False)
axs[1].set_title('Capital Run Length Average')
axs[1].set_xlabel('')
axs[1].set_xlim([0, 20])

axs[2].boxplot(Data_set['capital_run_length_longest'], vert=False)
axs[2].set_title('Capital Run Length Longest')
axs[2].set_xlabel('')
axs[2].set_xlim([0, 300])

plt.show()
```

```python
# In[7]:


import numpy as np

# Calculate the interquartile range for each column
Q1 = Data_set.quantile(0.25)
Q3 = Data_set.quantile(0.75)
IQR = Q3 - Q1

# Remove the outliers using the IQR method
Data_set_outliers_removed = Data_set[~((Data_set < (Q1 - 1.5 * IQR)) |
(Data_set > (Q3 + 1.5 * IQR))).any(axis=1)]

# Create individual box plots for the columns with outliers removed
fig, axs = plt.subplots(1, 3, figsize=(15, 5))

axs[0].boxplot(Data_set_outliers_removed['capital_run_length_total'],
vert=False)
axs[0].set_title('Capital Run Length Total (outliers removed)')
axs[0].set_xlabel('')
axs[0].set_xlim([0, 600])

axs[1].boxplot(Data_set_outliers_removed['capital_run_length_average'],
vert=False)
axs[1].set_title('Capital Run Length Average (outliers removed)')
axs[1].set_xlabel('')
axs[1].set_xlim([0, 10])

axs[2].boxplot(Data_set_outliers_removed['capital_run_length_longest'],
vert=False)
axs[2].set_title('Capital Run Length Longest (outliers removed)')
axs[2].set_xlabel('')
axs[2].set_xlim([0, 50])

plt.show()


# getting the duplicates in the dataset

# In[8]:


Data_set.duplicated()


# removing the duplicate values

# In[9]:


Data_set.drop_duplicates(inplace=True)


# In[10]:
```

```
print("Number of rows after removing duplicates : ", len(Data_set))


# All null values in the dataset

# individual box plots of the above columns before removing the outliers

# In[11]:


Data_set.isna().sum()


# Removing the target column

# In[12]:


data=Data_set.drop(labels=['spam'], axis=1)
data.head()


# In[13]:


print("Number of rows after preprocessing : ", len(Data_set))


# Summary of dataset before performing Standard Scaler

# In[14]:


data.describe()


#  Box plot before scaling
#

# In[15]:


import matplotlib.pyplot as plt
import seaborn as sns

sns.boxplot(data=data.iloc[:, :-1])
plt.xticks(rotation=90)
plt.title("Boxplot before scaling")

# Save the plot as a PNG file
plt.savefig('boxplot_before_scaling.png')


# Standard Scaling for the dataset
```

```
# In[16]:


# Compute mean values before scaling
mean_before = Data_set.mean()

# Create a bar plot of mean values before scaling
plt.figure(figsize=(12, 6))
plt.bar(mean_before.index, mean_before.values)
plt.xticks(rotation=90)
plt.title('Mean values before scaling')

# Perform Standard Scaling on the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(Data_set.iloc[:, :-1])
Data_scaled = pd.DataFrame(data=scaled_data, columns=Data_set.columns[:-1])

# Compute mean values after scaling
mean_after = Data_scaled.mean()

# Create a bar plot of mean values after scaling
plt.figure(figsize=(12, 6))
plt.bar(mean_after.index, mean_after.values)
plt.xticks(rotation=90)
plt.title('Mean values after scaling')

plt.show()
plt.savefig('boxplot_before_scaling.png')


# In[17]:


import matplotlib.pyplot as plt
import seaborn as sns

# Create box plot after scaling
plt.xticks(rotation=90)
plt.title("Boxplot after scaling")


sns.boxplot(data=Data_scaled, orient='v')
plt.show()


# Summmary of dataset after performing Standard Scaling

# In[18]:


Data_set.describe()


# Performing PCA to the Dataset

# In[19]:
```

```
pca = PCA()

principalComponents = pca.fit_transform(Data_scaled)

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

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

plt.title('Variance Graph')
plt.grid(True)
plt.show()


# introducing the PCA components

# In[20]:


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

# 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','PC37','PC38','PC39','PC40',
    'PC41','PC42','PC43','PC44'
])


# After performing PCA

# In[21]:


principal_Df.head()


# In[22]:


pca.explained_variance_


# In[23]:


pca.components_


# Build the predictive model by appling K-Nearest Neighbors (KNN) algorithm
#
```

```
# In[24]:


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


# Splitting the dataset into the Training set and Test set

# In[25]:


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


# Find optimal K value

# In[26]:


from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
# 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
print('Optimal value of k:', grid_search.best_params_['n_neighbors'])


# Perform 10-fold cross validation

# In[27]:


# Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)


# In[28]:


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


# In[29]:


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

```
# In[30]:


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


# In[31]:


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


# Visualization

# In[32]:


# based on test values generate the 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 class')
plt.ylabel('predicted class')


# In[33]:


import plotly.express as px
fig = px.scatter(
    X_test, x=0, y=1,
    color=y_pred, color_continuous_scale='Viridis',
    symbol=y_test, symbol_map={'0': 'square-dot', '1': 'circle-dot'},
    labels={'symbol': 'label', 'color': 'score'}
)
fig.update_traces(marker_size=12, marker_line_width=1.5)
fig.update_layout(legend_orientation='h')
fig.show()


# Plotting ROC curve for KNN

# In[34]:


from sklearn.metrics import roc_curve, auc
```

```
k = 5  # number of nearest neighbors
clf = KNeighborsClassifier(n_neighbors=k)
clf.fit(X_train, y_train)

knn_probs = clf.predict_proba(X_test)[:, 1]
fpr_knn, tpr_knn, thresholds_knn = roc_curve(y_test, knn_probs)
auc_score_knn = auc(fpr_knn, tpr_knn)


# Plotting ROC curve for KNN

# In[35]:


k = 5  # number of nearest neighbors
clf = KNeighborsClassifier(n_neighbors=k)
clf.fit(X_train, y_train)

knn_probs = clf.predict_proba(X_test)[:, 1]
fpr_knn, tpr_knn, thresholds_knn = roc_curve(y_test, knn_probs)
auc_score_knn = auc(fpr_knn, tpr_knn)


# In[36]:


def plot_roc_curve(fpr, tpr):
    plt.figure(figsize=(10,8))
    plt.plot(fpr_knn, tpr_knn, color='orange', label='AUC = %0.2f' %
auc_score_knn)
    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_knn,tpr_knn)


# In[ ]:
```