Experiment 3: Classification Algorithm Comparisons

Name: Tarun Suresh

Register Number: 3122237001055

Date: 12/08/25

Department of Computer Science and Engineering
SSN College of Engineering

Academic Year: 2024–2025

```
#mounting drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, ca
```

```
#importing libraries for classification
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
#importing dataset
df = pd.read_csv('/content/drive/MyDrive/spambase_csv.csv')
df
```

	word_freq_make	word_freq_address	word_freq_all	word_freq_3d	word_f
0	0.00	0.64	0.64	0.0	
1	0.21	0.28	0.50	0.0	
2	0.06	0.00	0.71	0.0	
3	0.00	0.00	0.00	0.0	
4	0.00	0.00	0.00	0.0	
...	...	...	...	...	
4596	0.31	0.00	0.62	0.0	
4597	0.00	0.00	0.00	0.0	
4598	0.30	0.00	0.30	0.0	
4599	0.96	0.00	0.00	0.0	
4600	0.00	0.00	0.65	0.0	

4601 rows × 58 columns

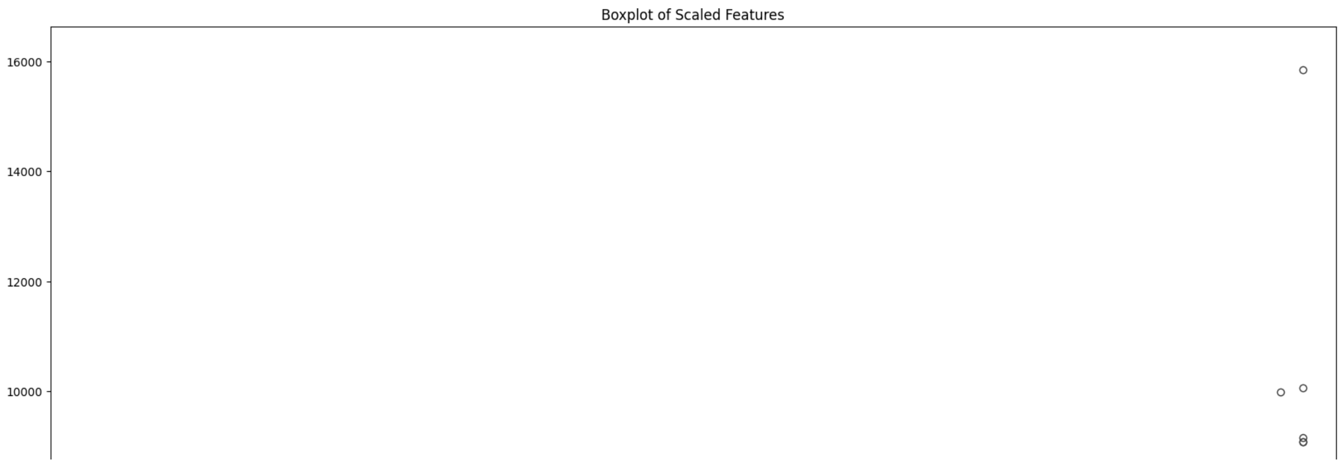
```
#performing eda
missing_values = df.isna().sum()
```

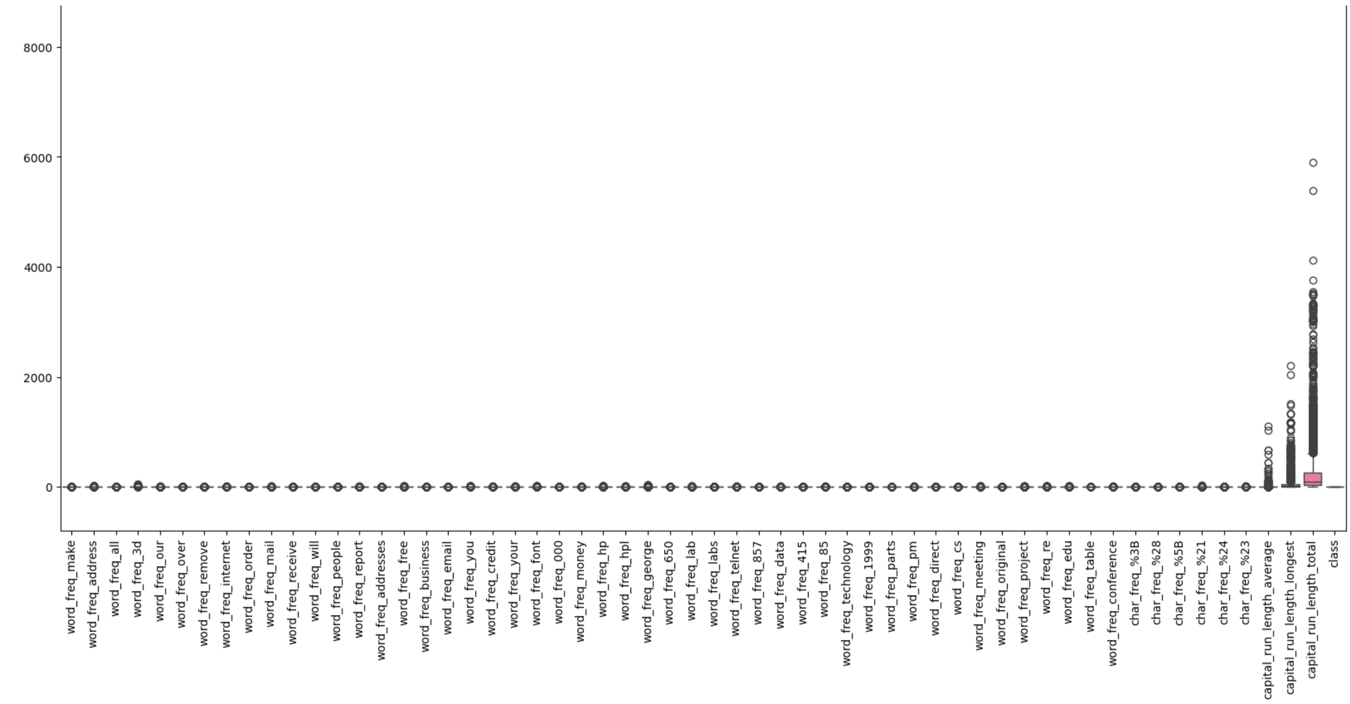
```
print(missing_values)

#dealing with missing values
# Fill numerical columns with mean
df.fillna(df.mean(), inplace=True)
```

```
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_%3B                 0
char_freq_%28                 0
char_freq_%5B                 0
char_freq_%21                 0
char_freq_%24                 0
char_freq_%23                 0
capital_run_length_average    0
capital_run_length_longest    0
capital_run_length_total      0
class                         0
dtype: int64
```

```
# Check for outliers visually using boxplots
plt.figure(figsize=(20, 15))
sns.boxplot(data=df)
plt.title('Boxplot of Scaled Features')
plt.xticks(rotation=90)
plt.show()

# Check for outliers programmatically using IQR
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

# Define outlier bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Find outliers
outliers = ((df < lower_bound) | (df > upper_bound)).sum()
print("\nNumber of outliers per column (IQR method):")
print(outliers[outliers > 0])

#removing outliers
#df = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1)]
```






```
Number of outliers per column (IQR method):
word_freq_make              1053
word_freq_address            898
word_freq_all                338
word_freq_3d                  47
word_freq_our                501
word_freq_over               999
word_freq_remove             807
word_freq_internet           824
word_freq_order              773
word_freq_mail               852
word_freq_receive            709
word_freq_will               270
word_freq_people             852
word_freq_report             357
word_freq_addresses          336
word_freq_free               957
word_freq_business           963
word_freq_email             1038
word_freq_you                 75
word_freq_credit             424
word_freq_your               229
word_freq_font               117
word_freq_000                679
word_freq_money              735
word_freq_hp                1090
word_freq_hpl                811
word_freq_george             780
word_freq_650                463
word_freq_lab                372
word_freq_labs               469
word_freq_telnet             293
word_freq_857                205
word_freq_data               405
```

```
word_freq_data                [illegible]
word_freq_415                 215
word_freq_85                  485
word_freq_technology          599
word_freq_1999                829
word_freq_parts                83
word_freq_pm                  384
word_freq_direct              453
word_freq_cs                  148
word_freq_meeting             341
word_freq_original            375
word_freq_project             327
word_freq_re                 1001
word_freq_edu                 517
word_freq_table                63
word_freq_conference          203
char_freq_%3B                 790
char_freq_%28                 296
char_freq_%5B                 529
char_freq_%21                 411
char_freq_%24                 811
char_freq_%23                 750
capital_run_length_average    363
capital_run_length_longest    463
capital_run_length_total      550
dtype: int64
```

```
y = df['class']
df = df.drop('class', axis=1)
```

```
#using standard scaler on data for gaussianNB
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)
```

```
#normalise the values for gaussianNB
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
df_scaled = scaler.fit_transform(df)

df
```

	word_freq_make	word_freq_address	word_freq_all	word_freq_3d	word_f
0	0.00	0.64	0.64	0.0	
1	0.21	0.28	0.50	0.0	
2	0.06	0.00	0.71	0.0	
3	0.00	0.00	0.00	0.0	
4	0.00	0.00	0.00	0.0	
...	...	...	...	...	
4596	0.31	0.00	0.62	0.0	
4597	0.00	0.00	0.00	0.0	
4598	0.30	0.00	0.30	0.0	
4599	0.96	0.00	0.00	0.0	
4600	0.00	0.00	0.65	0.0	

4601 rows × 57 columns

```
#binarize dataset from original data set for NaiveBayes Bernoulli
from sklearn.preprocessing import Binarizer

binarizer = Binarizer(threshold=0.5)
df_binarized = binarizer.fit_transform(df)

df_binarized
```

```
array([[0., 1., 1., ..., 1., 1., 1.],
       [0., 0., 0., ..., 1., 1., 1.],
       [0., 0., 1., ..., 1., 1., 1.],
       ...,
       [0., 0., 0., ..., 1., 1., 1.],
       [1., 0., 0., ..., 1., 1., 1.],
       [0., 0., 1., ..., 1., 1., 1.]])
```

We will be using a normalised dataset for Naive Bayes Gaussian distribution as it expects features to be normalised during fit and predict.

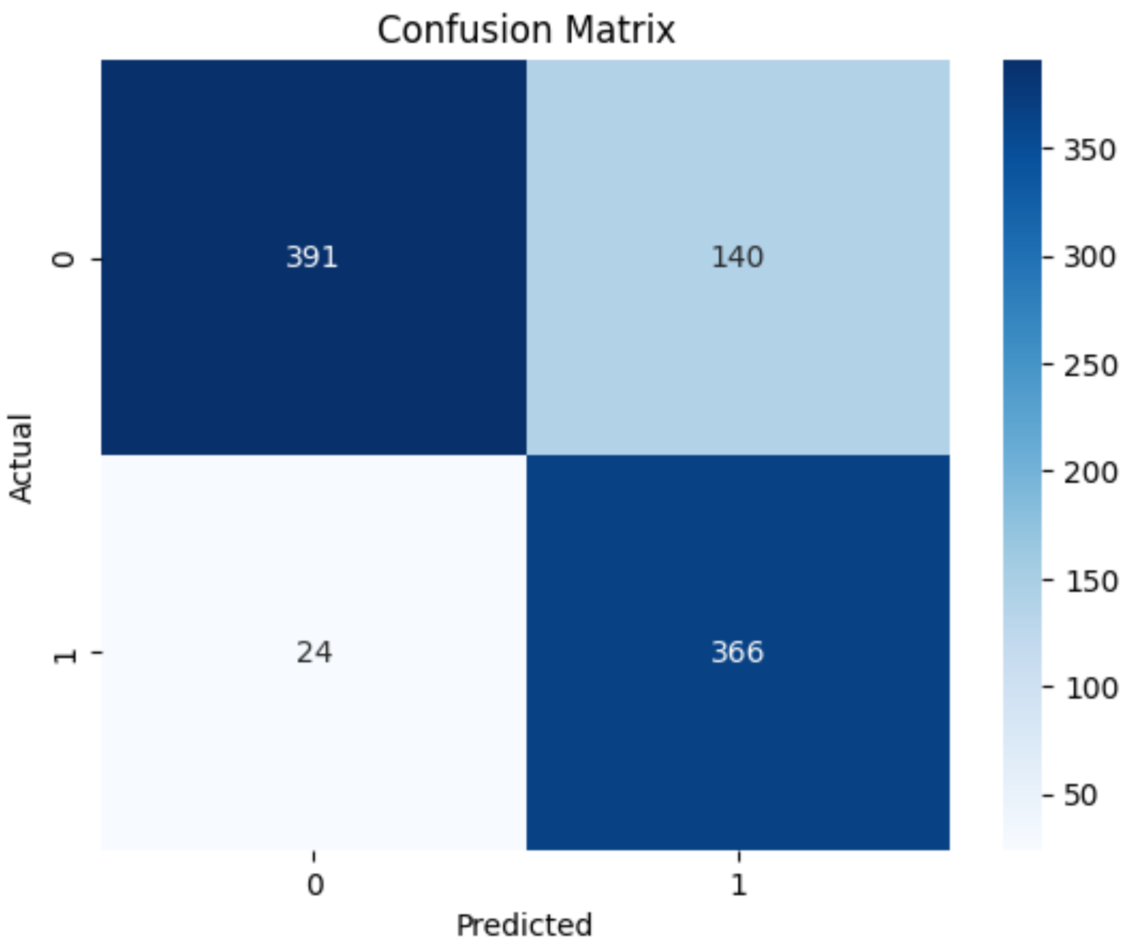
```
#splitting normalised dataset and performing NaiveBayes Gaussian
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

Xg_train, Xg_test, yg_train, yg_test = train_test_split(df_scaled, y, test_size

gNBmodel = GaussianNB()
gNBmodel.fit(Xg_train,yg_train)
yg_pred = gNBmodel.predict(Xg_test)
```

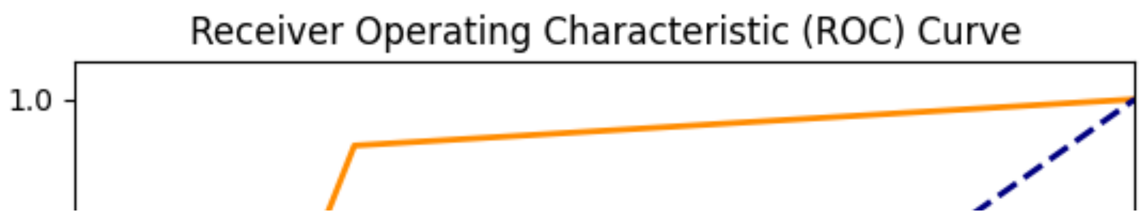
```
#evaluating Accuracy, Precision, Recall, F1-score for GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

accuracy_g = accuracy_score(yg_test, yg_pred)
precision_g = precision_score(yg_test, yg_pred)
recall_g = recall_score(yg_test, yg_pred)
f1_g = f1_score(yg_test, yg_pred)

print("Accuracy:", accuracy_g)
print("Precision:", precision_g)
print("Recall:", recall_g)
print("F1-score:", f1_g)
```

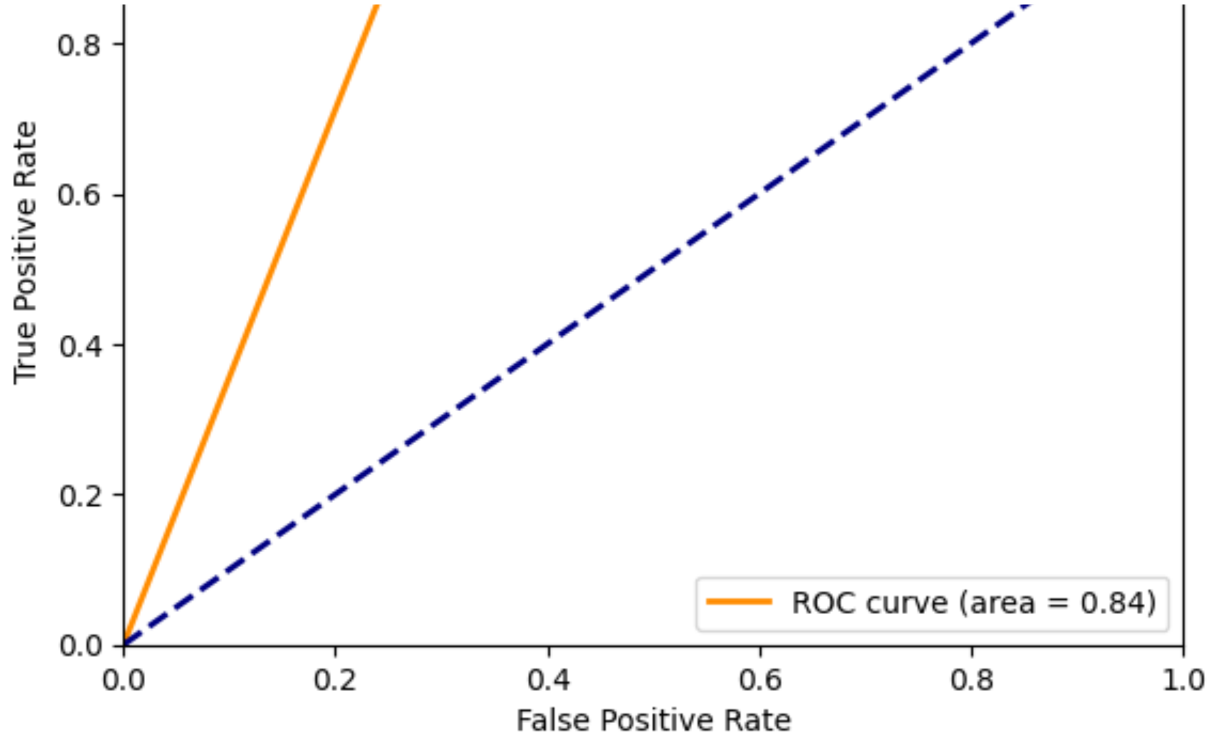
```
Accuracy: 0.8219326818675353
Precision: 0.7233201581027668
Recall: 0.9384615384615385
F1-score: 0.8169642857142857
```

```
#displaying confusion matrix
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(yg_test, yg_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

```
plt.title('Confusion Matrix')
plt.show()

#displaying roc curve
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(yg_test, yg_pred)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```







As for NB Multinomial we will be using the original dataset as it depends on raw counts/frequency of features.

```
#k-fold for NB gaussian
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import GaussianNB

gNBmodel = GaussianNB()
scores = cross_val_score(gNBmodel, df_scaled, y, cv=5)
print("Cross-validation scores:", scores)
print("Mean accuracy:", scores.mean())
```

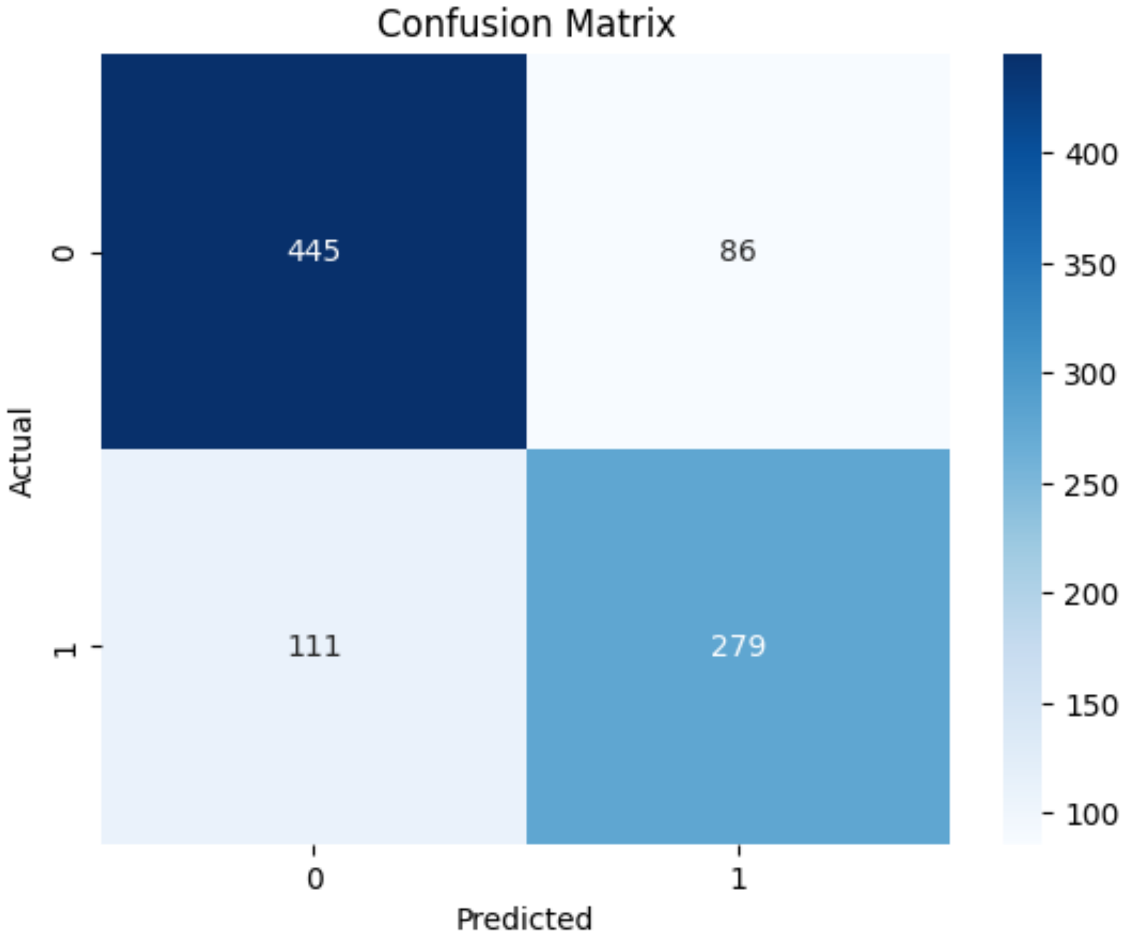
```
Cross-validation scores: [0.85124864 0.86630435 0.85434783 0.84347826 0.695
Mean accuracy: 0.8222062502950479
```

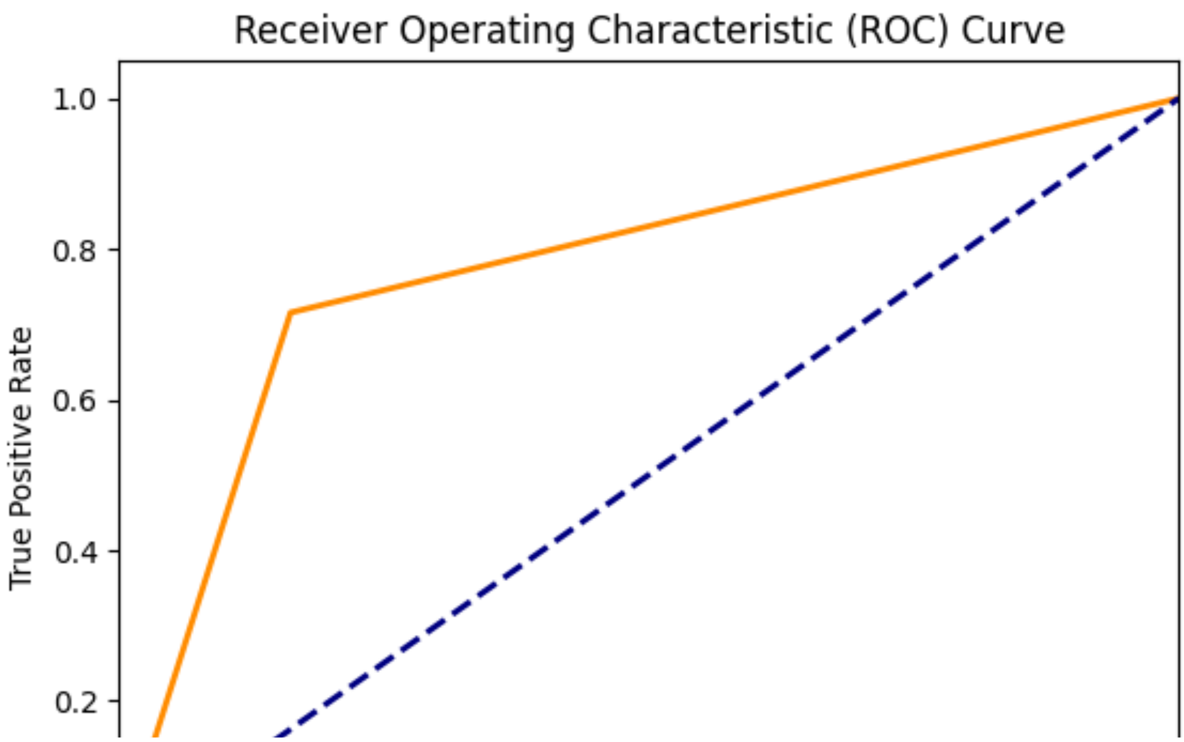
```
#splitting original dataset and performing NaiveBayes Multinomial
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

Xm_train, Xm_test, ym_train, ym_test = train_test_split(df, y, test_size=0.2, r

mNBmodel = MultinomialNB()
mNBmodel.fit(Xm_train, ym_train)
ym_pred = mNBmodel.predict(Xm_test)

accuracy_m = accuracy_score(ym_test, ym_pred)
precision_m = precision_score(ym_test, ym_pred)
recall_m = recall_score(ym_test, ym_pred)
f1_m = f1_score(ym_test, ym_pred)

print("Accuracy:", accuracy_g)
print("Precision:", precision_g)
print("Recall:", recall_g)
print("F1-score:", f1_g)
```

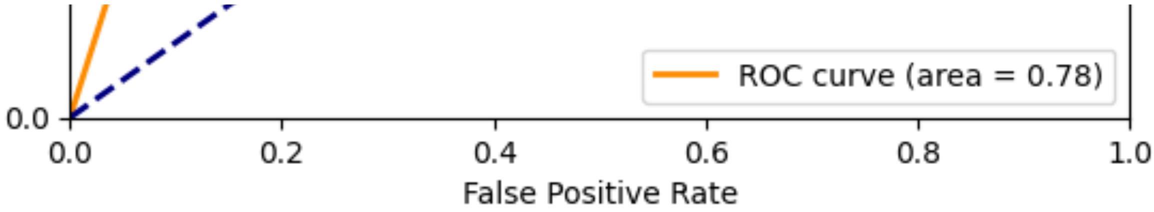
```
Accuracy: 0.8219326818675353
Precision: 0.7233201581027668
Recall: 0.9384615384615385
F1-score: 0.8169642857142857
```

```
#displaying confusion matrix
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(ym_test, ym_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

#displaying roc curve
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(ym_test, ym_pred)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```








```
#k-fold for NB multinomial
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import MultinomialNB

mNBmodel = MultinomialNB()
scores = cross_val_score(mNBmodel, df, y, cv=5)
print("Cross-validation scores:", scores)
print("Mean accuracy:", scores.mean())
```

```
Cross-validation scores: [0.79261672 0.81847826 0.81521739 0.78586957 0.696
Mean accuracy: 0.7817842137563139
```

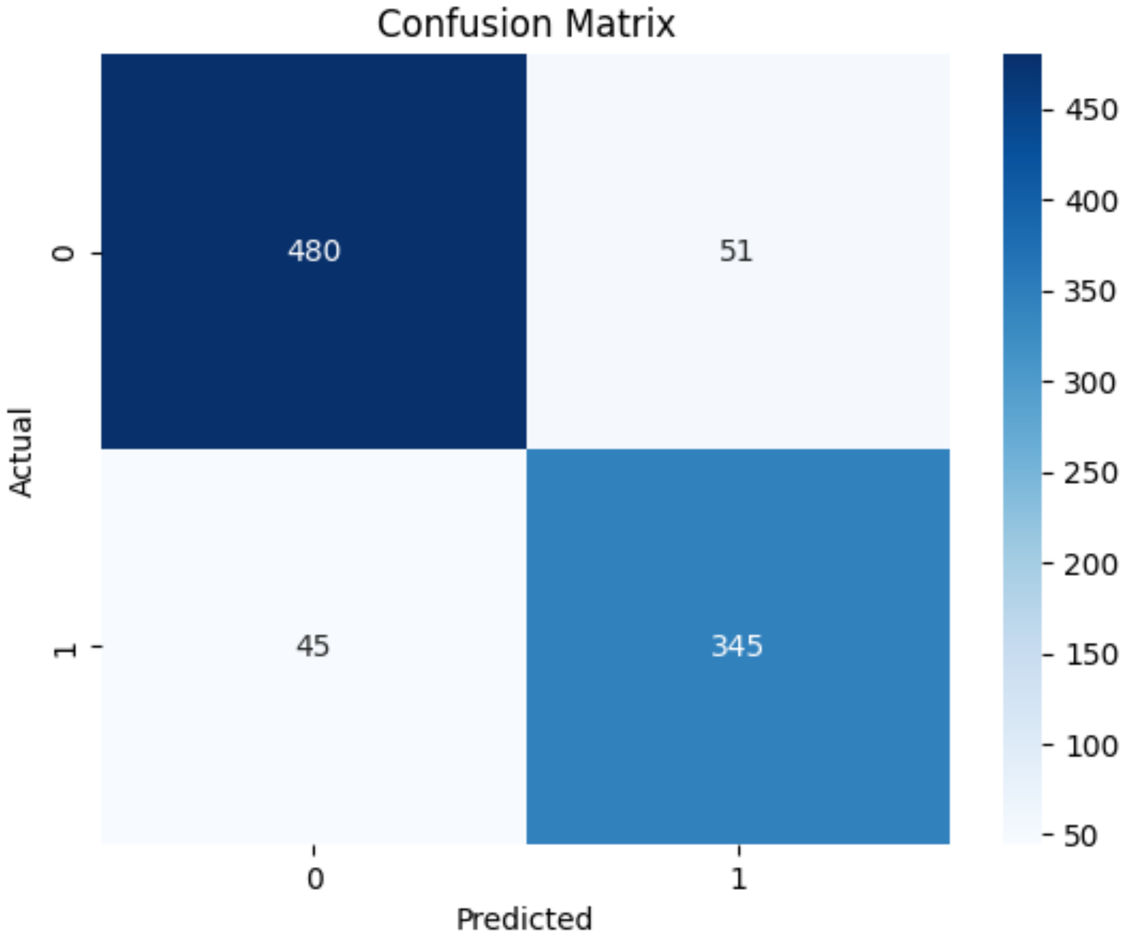
For NB bernoulli we will use Binariser to convert the data to binary

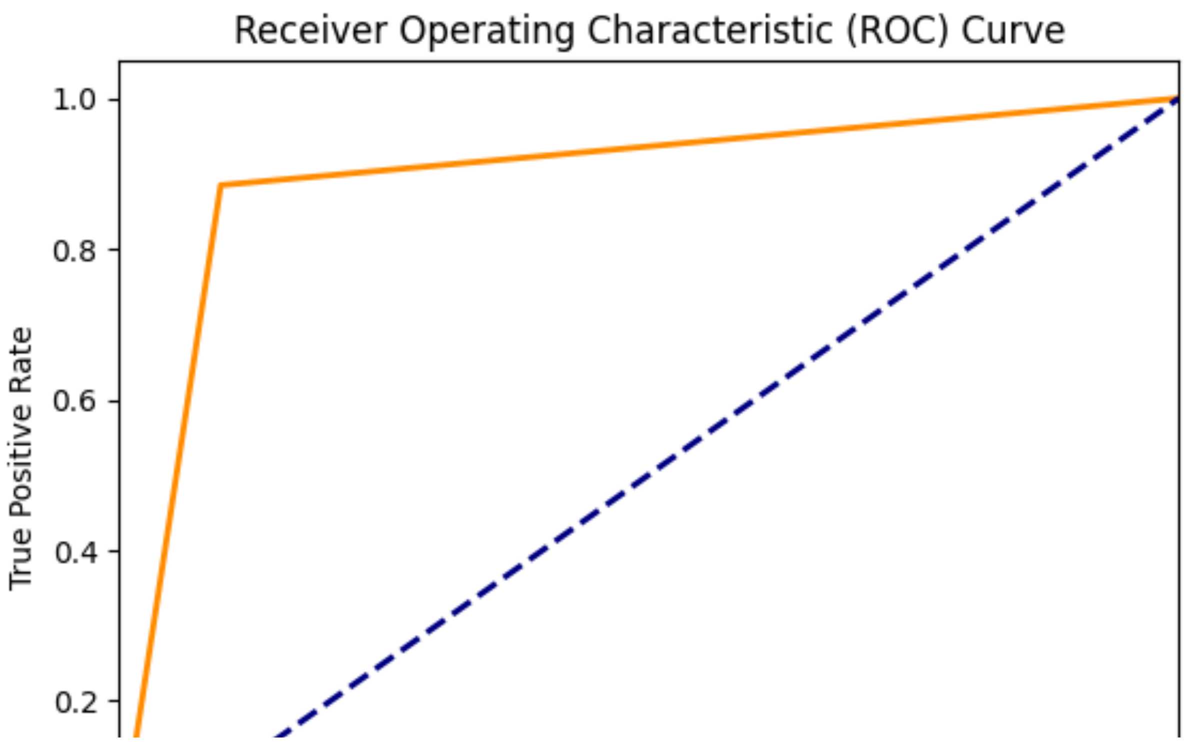
```
#train test split and NB Bernoulli with binarised data
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score

Xb_train, Xb_test, yb_train, yb_test = train_test_split(df_binarized, y, test_s

bNBmodel = BernoulliNB()
bNBmodel.fit(Xb_train, yb_train)
yb_pred = bNBmodel.predict(Xb_test)

accuracy_b = accuracy_score(yb_test, yb_pred)
precision_b = precision_score(yb_test, yb_pred)
recall_b = recall_score(yb_test, yb_pred)
f1_b = f1_score(yb_test, yb_pred)

print("Accuracy:", accuracy_b)
print("Precision:", precision_b)
print("Recall:", recall_b)
print("F1-score:", f1_b)
```

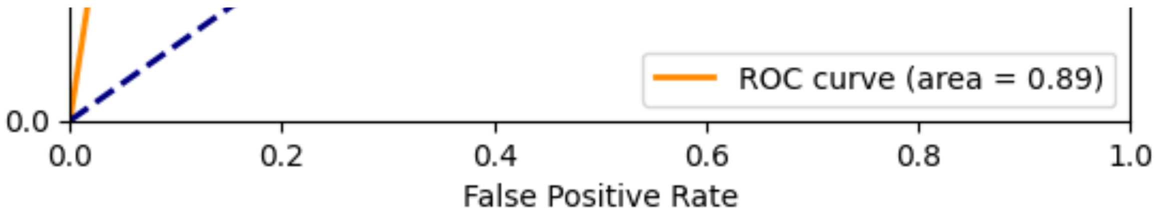
```
Accuracy: 0.8957654723127035
Precision: 0.8712121212121212
Recall: 0.8846153846153846
F1-score: 0.8778625954198473
```

```
#displaying confusion matrix
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(yb_test, yb_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

#displaying roc curve
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(yb_test, yb_pred)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```








```
#applying K-fold Cross validation for NB bernoulli
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import BernoulliNB

bNBmodel = BernoulliNB()
scores = cross_val_score(bNBmodel, df_binarized, y, cv=5)
print("Cross-validation scores:", scores)
print("Mean accuracy:", scores.mean())
```

```
Cross-validation scores: [0.90662324 0.91086957 0.92391304 0.93695652 0.744
Mean accuracy: 0.8845855166879101
```

```
#mounting drive
from google.colab import drive
drive.mount('/content/drive')
```

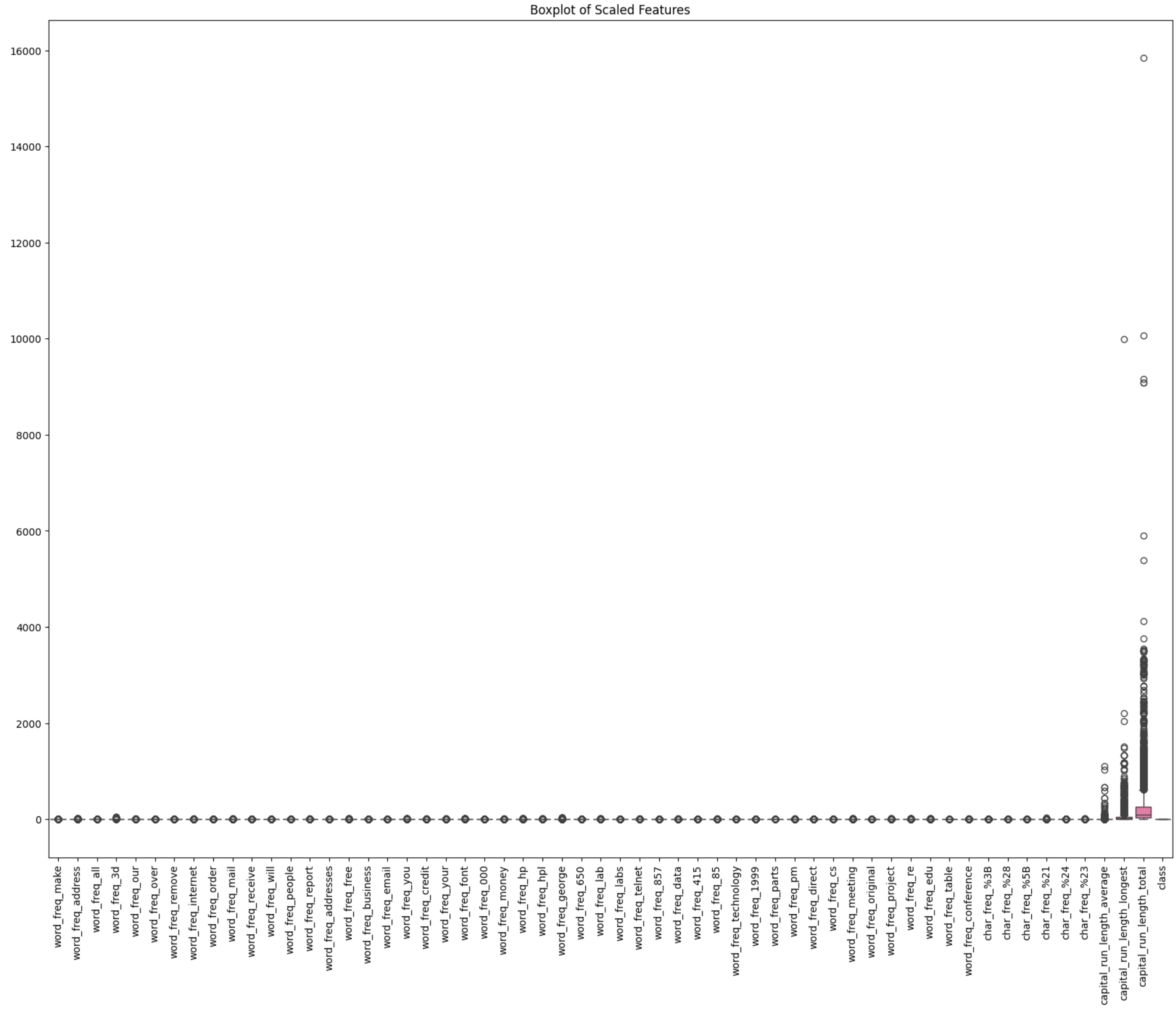
```
Mounted at /content/drive
```

```
#importing libraries for classification
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

#importing dataset
df = pd.read_csv('/content/drive/MyDrive/spambase_csv.csv')
df

#performing eda
missing_values = df.isna().sum()
print(missing_values)

#dealing with missing values
# Fill numerical columns with mean
df.fillna(df.mean(), inplace=True)


# Check for outliers visually using boxplots
plt.figure(figsize=(20, 15))
sns.boxplot(data=df)
plt.title('Boxplot of Scaled Features')
plt.xticks(rotation=90)
plt.show()

# Check for outliers programmatically using IQR
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

# Define outlier bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Find outliers
outliers = ((df < lower_bound) | (df > upper_bound)).sum()
```

```
print("\nNumber of outliers per column (IQR method):")
print(outliers[outliers > 0])

#removing outliers
#df = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1)]
```

```
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_%3B                  0
char_freq_%28                  0
char_freq_%5B                  0
char_freq_%21                  0
char_freq_%24                  0
char_freq_%23                  0
capital_run_length_average     0
capital_run_length_longest     0
capital_run_length_total       0
class                          0
dtype: int64
```




```
Number of outliers per column (IQR method):
word_freq_make                1053
word_freq_address              898
word_freq_all                  338
word_freq_3d                    47
word_freq_our                  501
word_freq_over                 999
word_freq_remove               807
```

```
word_freq_internet           824
word_freq_order              773
word_freq_mail               852
word_freq_receive            709
word_freq_will               270
word_freq_people             852
word_freq_report             357
word_freq_addresses          336
word_freq_free               957
word_freq_business           963
word_freq_email             1038
word_freq_you                 75
word_freq_credit             424
word_freq_your               229
word_freq_font               117
word_freq_000                679
word_freq_money              735
word_freq_hp                1090
word_freq_hpl                811
word_freq_george             780
word_freq_650                463
word_freq_lab                372
word_freq_labs               469
word_freq_telnet             293
word_freq_857                205
word_freq_data               405
word_freq_415                215
word_freq_85                 485
word_freq_technology         599
word_freq_1999               829
word_freq_parts               83
word_freq_pm                 384
word_freq_direct             453
word_freq_cs                 148
word_freq_meeting            341
word_freq_original           375
word_freq_project            327
word_freq_re                1001
word_freq_edu                517
word_freq_table               63
word_freq_conference         203
char_freq_%3B                790
char_freq_%28                296
char_freq_%5B                529
char_freq_%21                411
char_freq_%24                811
char_freq_%23                750
capital_run_length_average   363
capital_run_length_longest   463
capital_run_length_total     550
dtype: int64
```

```
#dropping target variable
y = df['class']
df = df.drop('class', axis=1)


#using standard scaler on data
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)
```

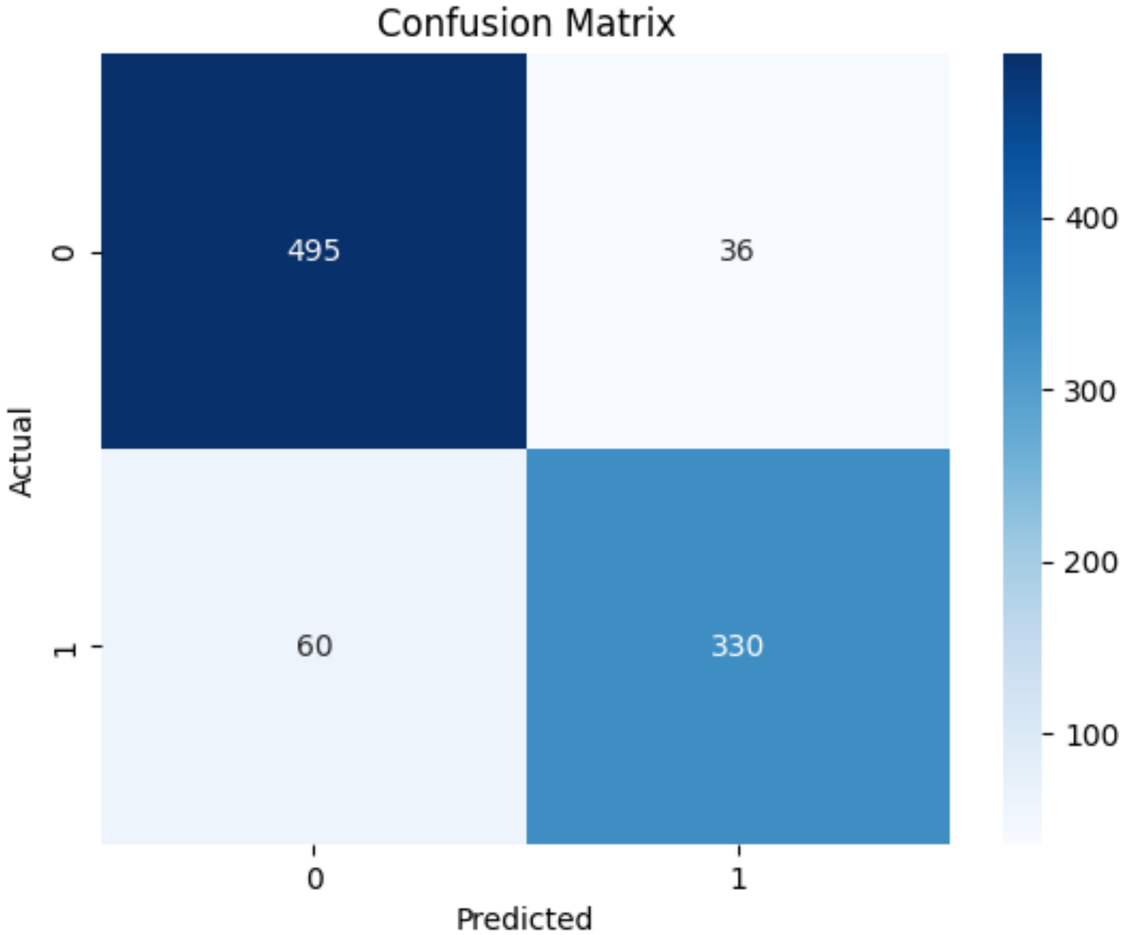
```
#performing KNN classification on data
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_ma

X_train, X_test, y_train, y_test = train_test_split(df_scaled, y, test_size=0.2

knn_classifier5 = KNeighborsClassifier(n_neighbors=5)
knn_classifier5.fit(X_train, y_train)

y_pred5 = knn_classifier5.predict(X_test)
```

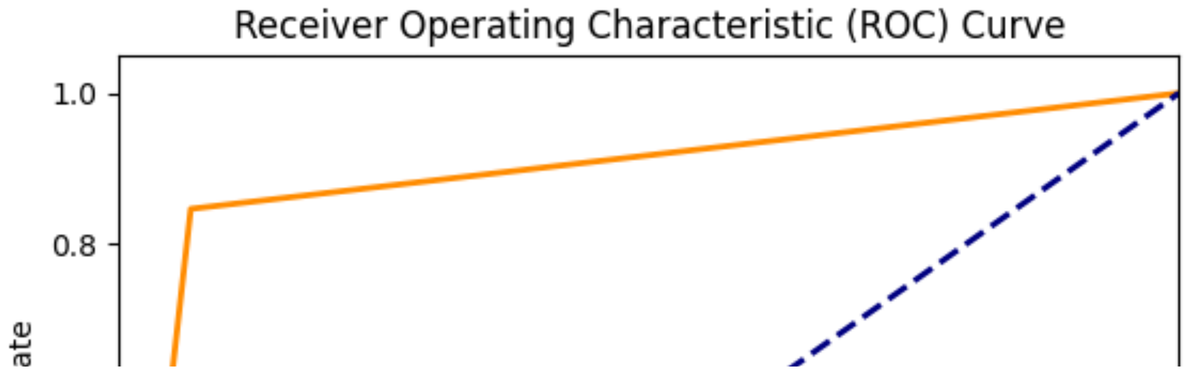
```
#Prediction analysis
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

accuracy = accuracy_score(y_test, y_pred5)
precision = precision_score(y_test, y_pred5)
recall = recall_score(y_test, y_pred5)
f1 = f1_score(y_test, y_pred5)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

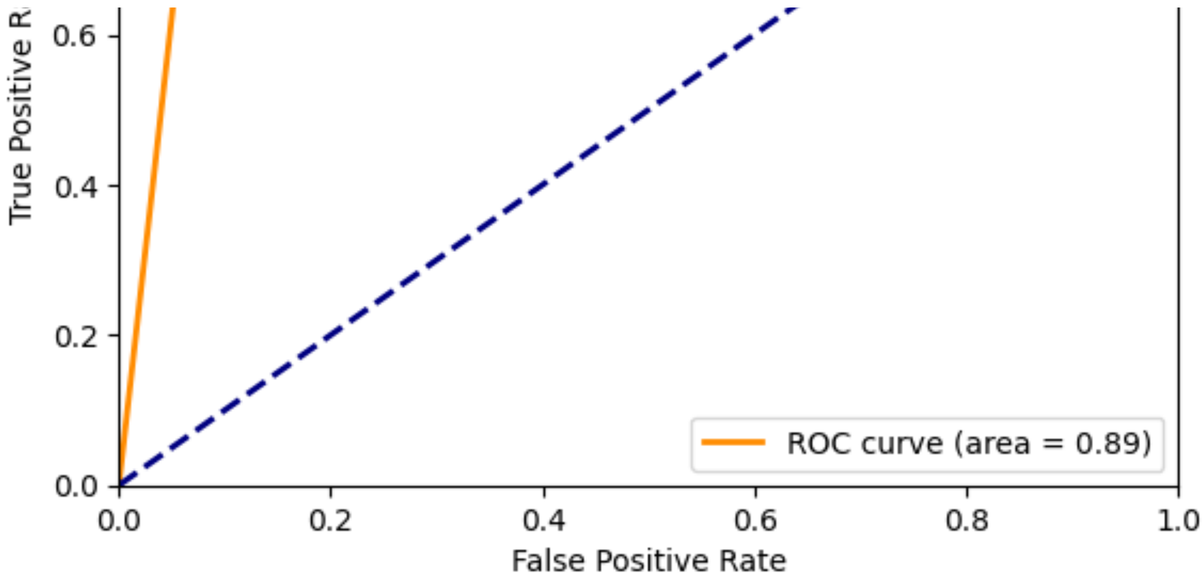
```
Accuracy: 0.8957654723127035
Precision: 0.9016393442622951
Recall: 0.8461538461538461
F1-score: 0.873015873015873
```

```
#displaying confusion matrix
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(y_test, y_pred5)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

#displaying roc curve
```

```
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_test, y_pred5)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```



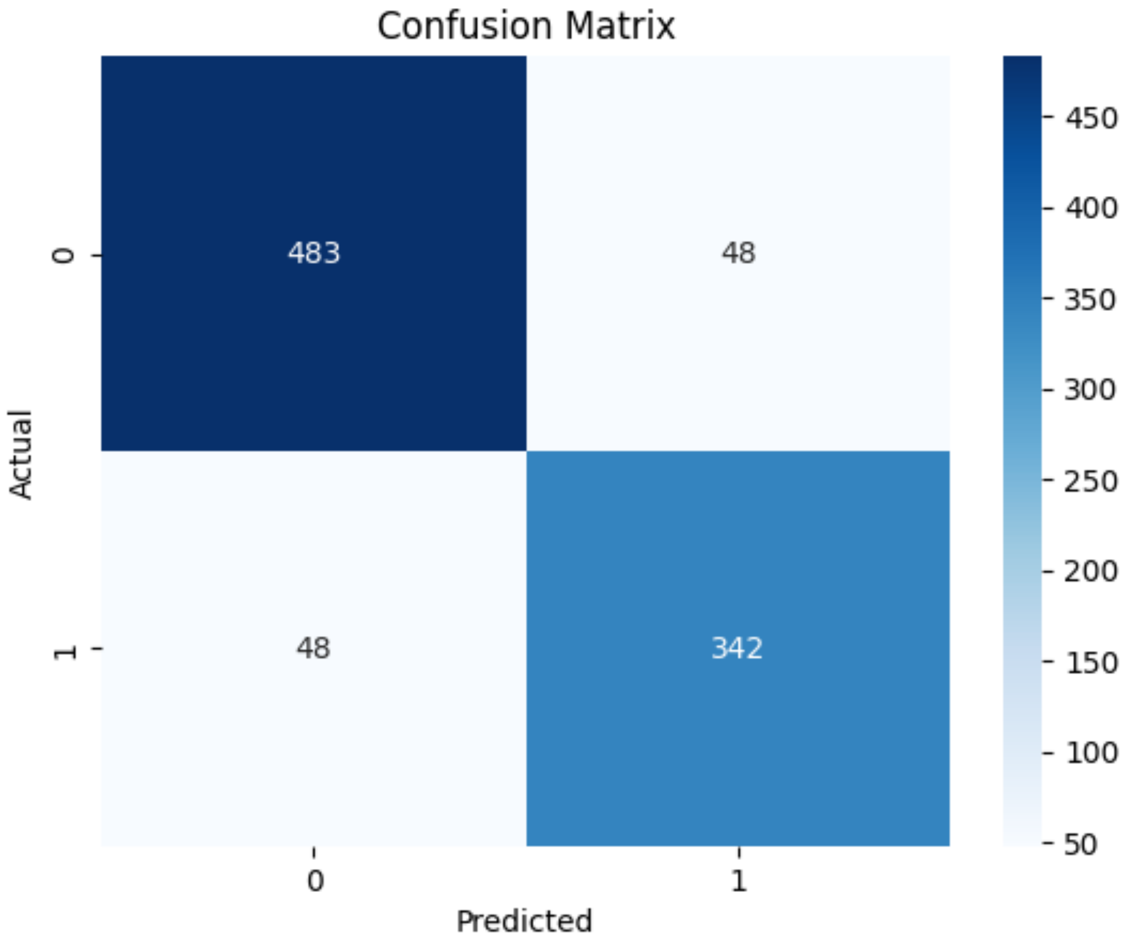


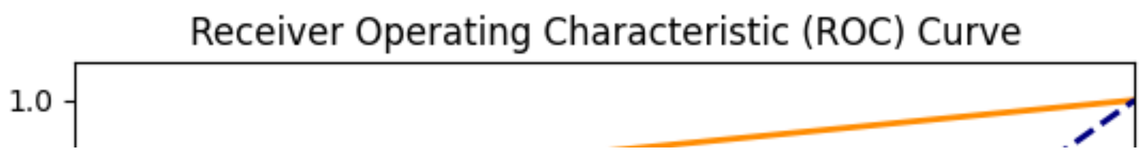


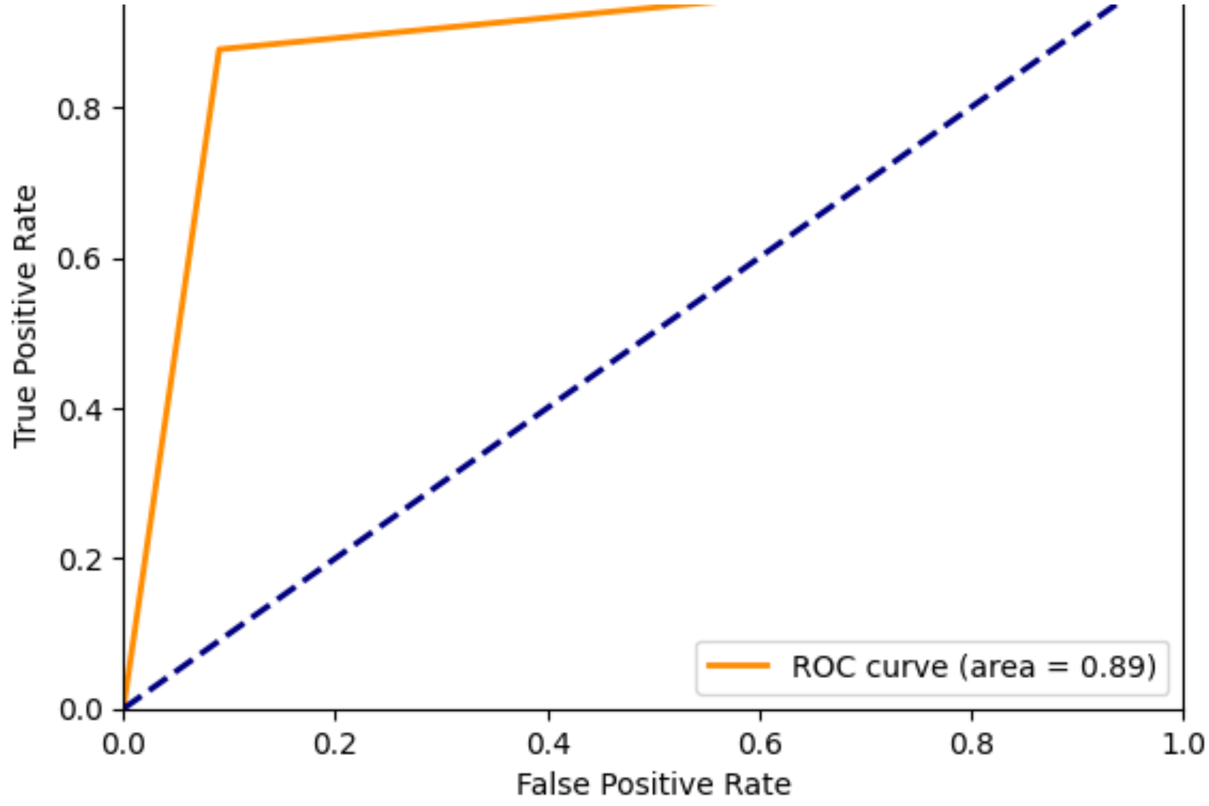

```
#K valye = 1
knn_classifier1 = KNeighborsClassifier(n_neighbors=1)
knn_classifier1.fit(X_train, y_train)

y_pred1 = knn_classifier1.predict(X_test)

#Prediction analysis
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

accuracy = accuracy_score(y_test, y_pred1)
precision = precision_score(y_test, y_pred1)
recall = recall_score(y_test, y_pred1)
f1 = f1_score(y_test, y_pred1)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)

#displaying confusion matrix
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(y_test, y_pred1)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
#displaying roc curve
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_test, y_pred1)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

```
Accuracy: 0.8957654723127035
Precision: 0.8769230769230769
Recall: 0.8769230769230769
F1-score: 0.8769230769230769
```



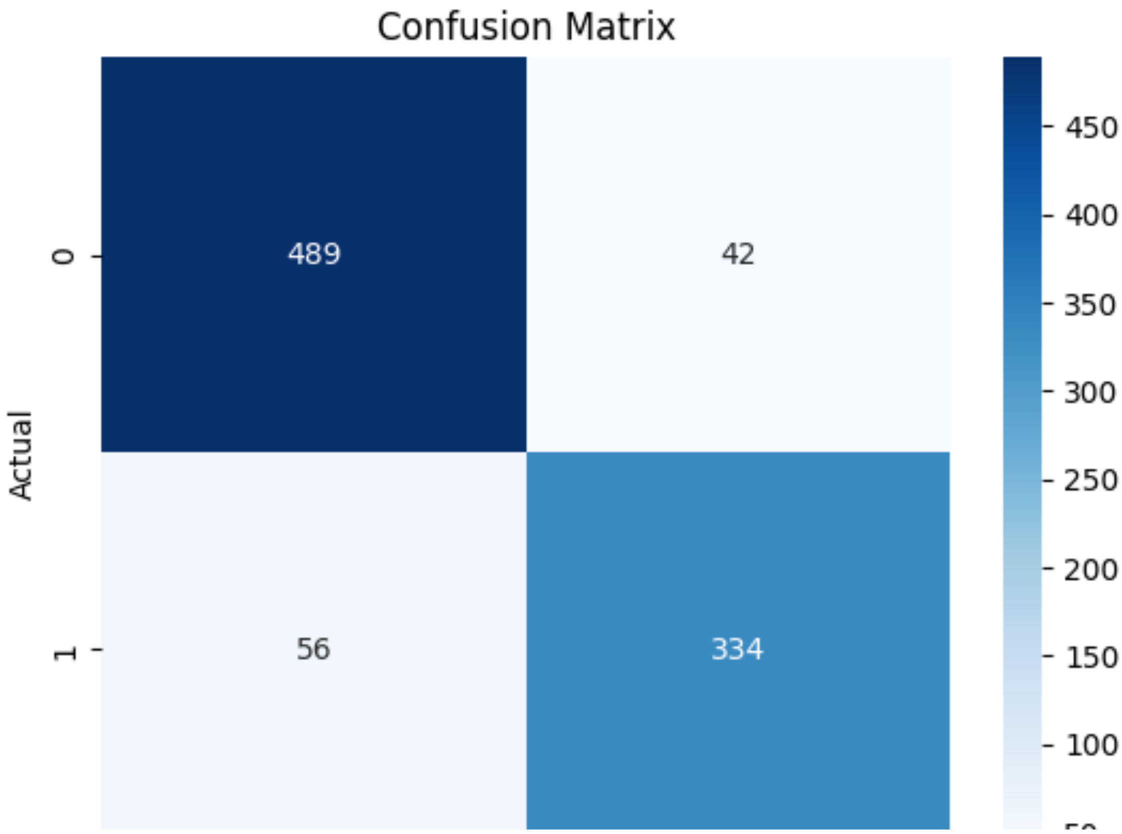


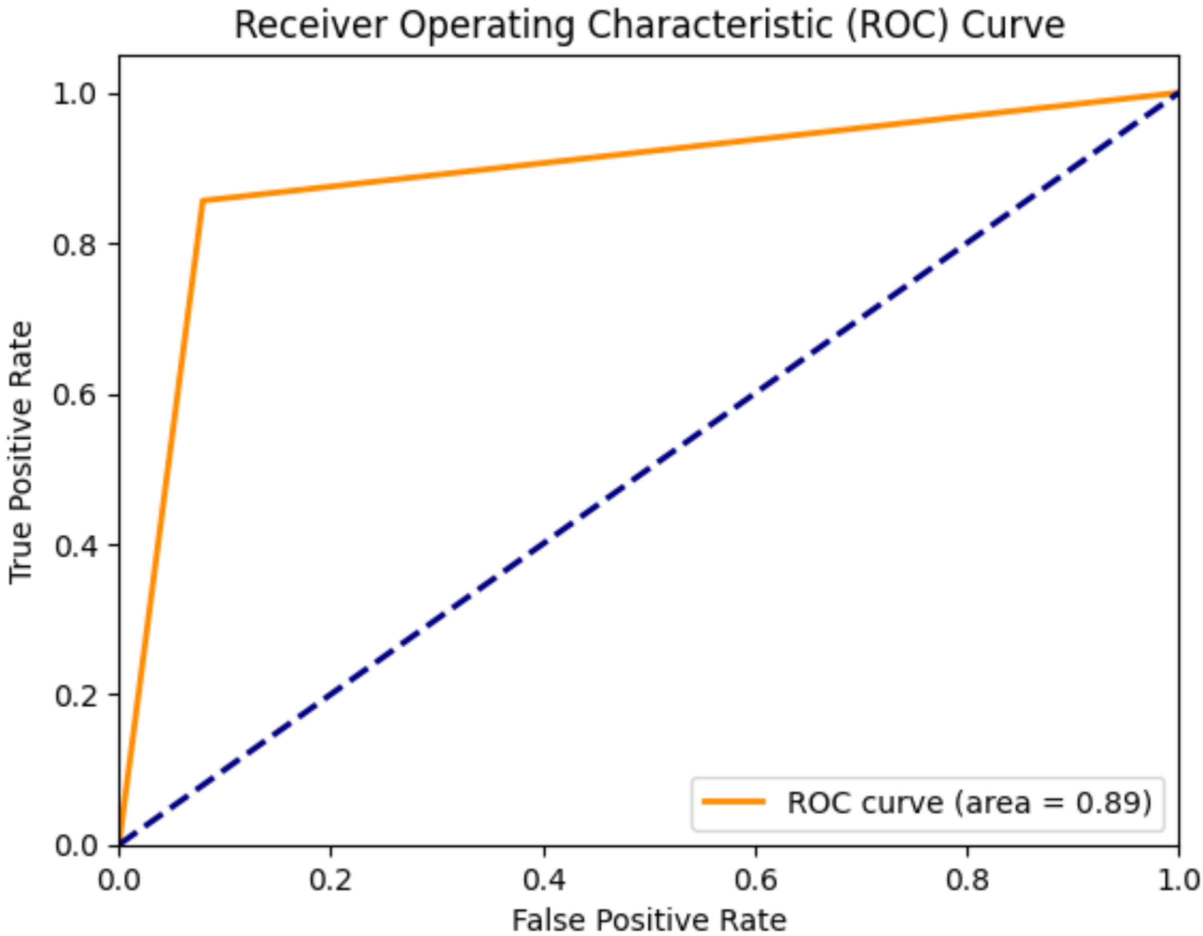



```
#K value = 3
knn_classifier = KNeighborsClassifier(n_neighbors=3)
knn_classifier.fit(X_train, y_train)

y_pred = knn_classifier.predict(X_test)

#Prediction analysis
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)

#displaying confusion matrix
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

```
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

#displaying roc curve
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

```
Accuracy: 0.8935939196525515
Precision: 0.8882978723404256
Recall: 0.8564102564102564
F1-score: 0.8720626631853786
```

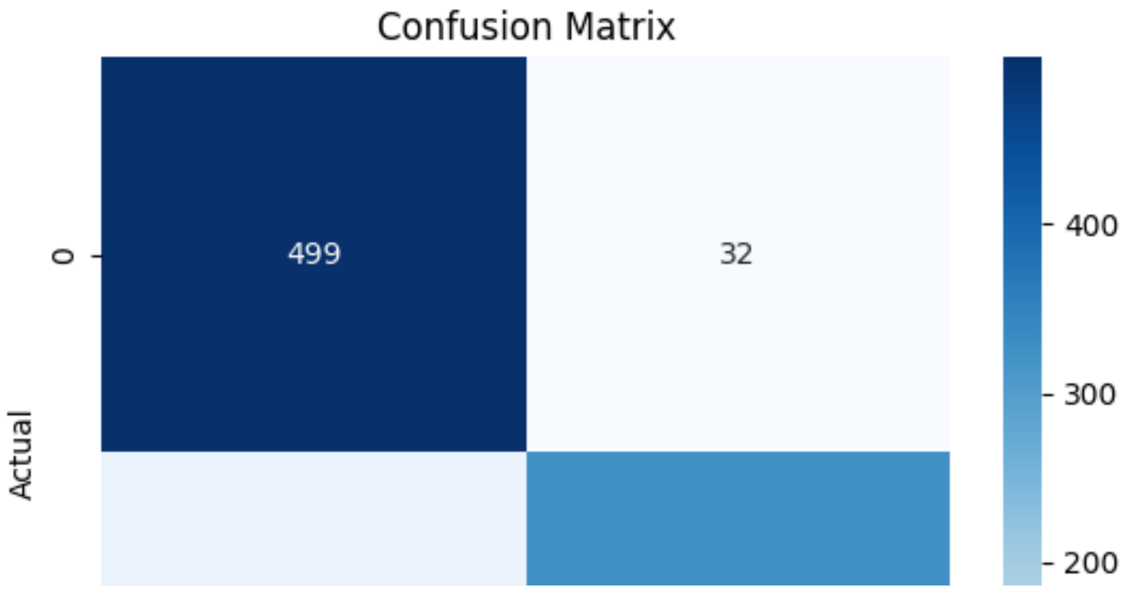


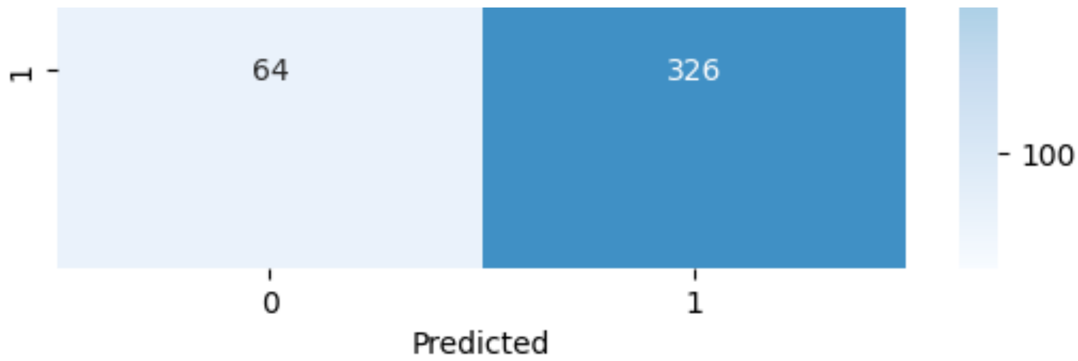


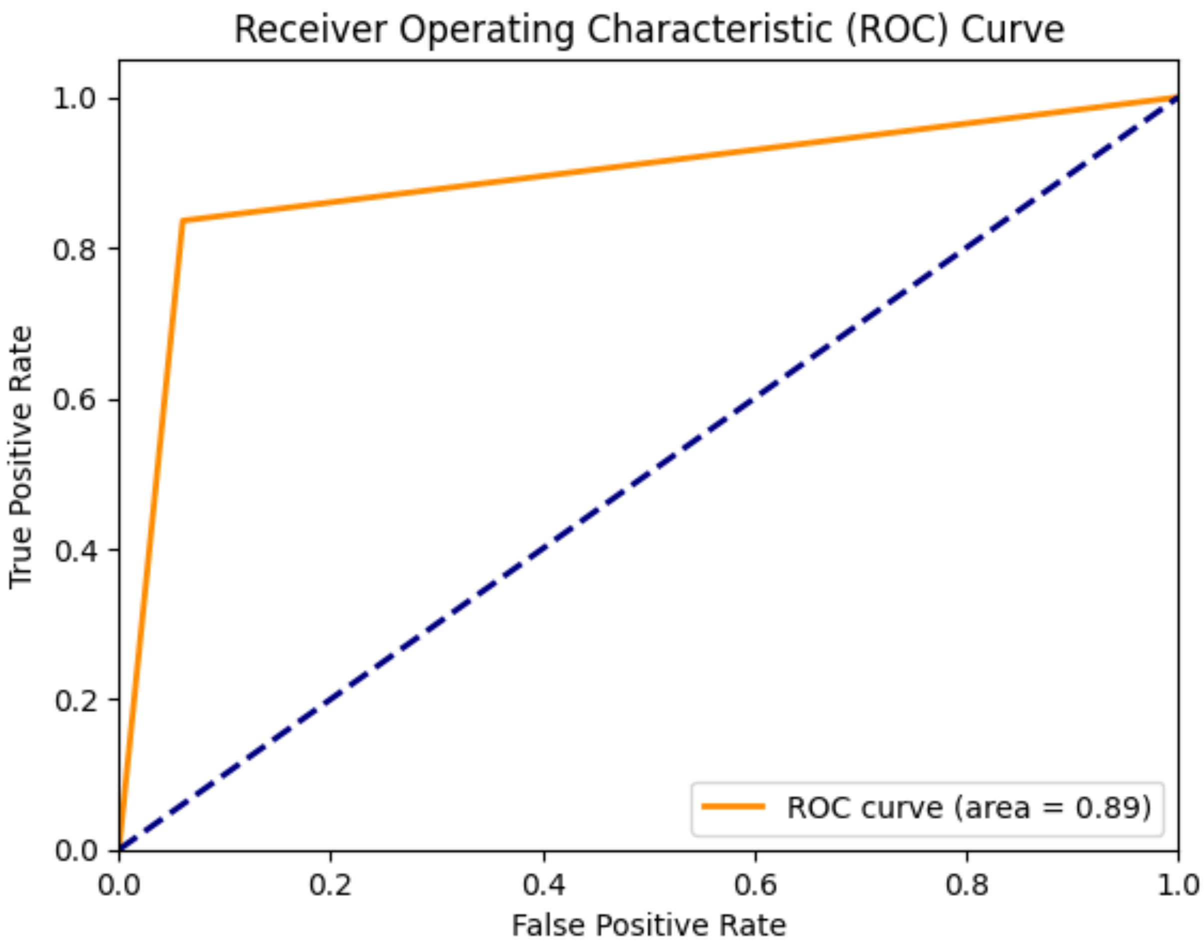

```
#k value = 7
knn_classifier = KNeighborsClassifier(n_neighbors=7)
knn_classifier.fit(X_train, y_train)

y_pred = knn_classifier.predict(X_test)

#Prediction analysis
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
```

```
print("Recall:", recall)
print("F1-score:", f1)

#displaying confusion matrix
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

#displaying roc curve
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

```
Accuracy: 0.8957654723127035
Precision: 0.9106145251396648
Recall: 0.8358974358974359
F1-score: 0.8716577540106952
```








```
#Kfold cross validation for 1 neighbour KNN
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier

knn_classifier = KNeighborsClassifier(n_neighbors=1)
cv_scores = cross_val_score(knn_classifier, X_train, y_train, cv=5, scoring='acc
```

```
#Kfold cross validation for 3 neighbour KNN
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier

knn_classifier = KNeighborsClassifier(n_neighbors=3)
cv_scores = cross_val_score(knn_classifier, X_train, y_train, cv=5, scoring='acc
```

```
#Kfold cross validation for 5 neighbour KNN
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier

knn_classifier = KNeighborsClassifier(n_neighbors=5)
cv_scores = cross_val_score(knn_classifier, X_train, y_train, cv=5, scoring='acc
```

```
#Kfold cross validation for 7 neighbour KNN
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier

knn_classifier = KNeighborsClassifier(n_neighbors=7)
cv_scores = cross_val_score(knn_classifier, X_train, y_train, cv=5, scoring='ac
```

```
#Applying KDtree on the split dataset and performing metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_s

knn_classifier = KNeighborsClassifier(n_neighbors=5, algorithm='kd_tree')
knn_classifier.fit(X_train, y_train)

y_predt = knn_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_predt)
precision = precision_score(y_test, y_predt)
recall = recall_score(y_test, y_predt)
f1 = f1_score(y_test, y_predt)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

```
Accuracy: 0.8957654723127035
Precision: 0.9016393442622951
Recall: 0.8461538461538461
F1-score: 0.873015873015873
```

```
#Applying KDball on the split dataset and performing metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc

knn_classifier = KNeighborsClassifier(n_neighbors=5, algorithm='ball_tree')
knn_classifier.fit(X_train, y_train)

y_predb = knn_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_predb)
precision = precision_score(y_test, y_predb)
recall = recall_score(y_test, y_predb)
f1 = f1_score(y_test, y_predb)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

```
Accuracy: 0.8957654723127035
Precision: 0.9016393442622951
Recall: 0.8461538461538461
F1-score: 0.873015873015873
```

```
#mounting drive
from google.colab import drive
drive.mount('/content/drive')
```

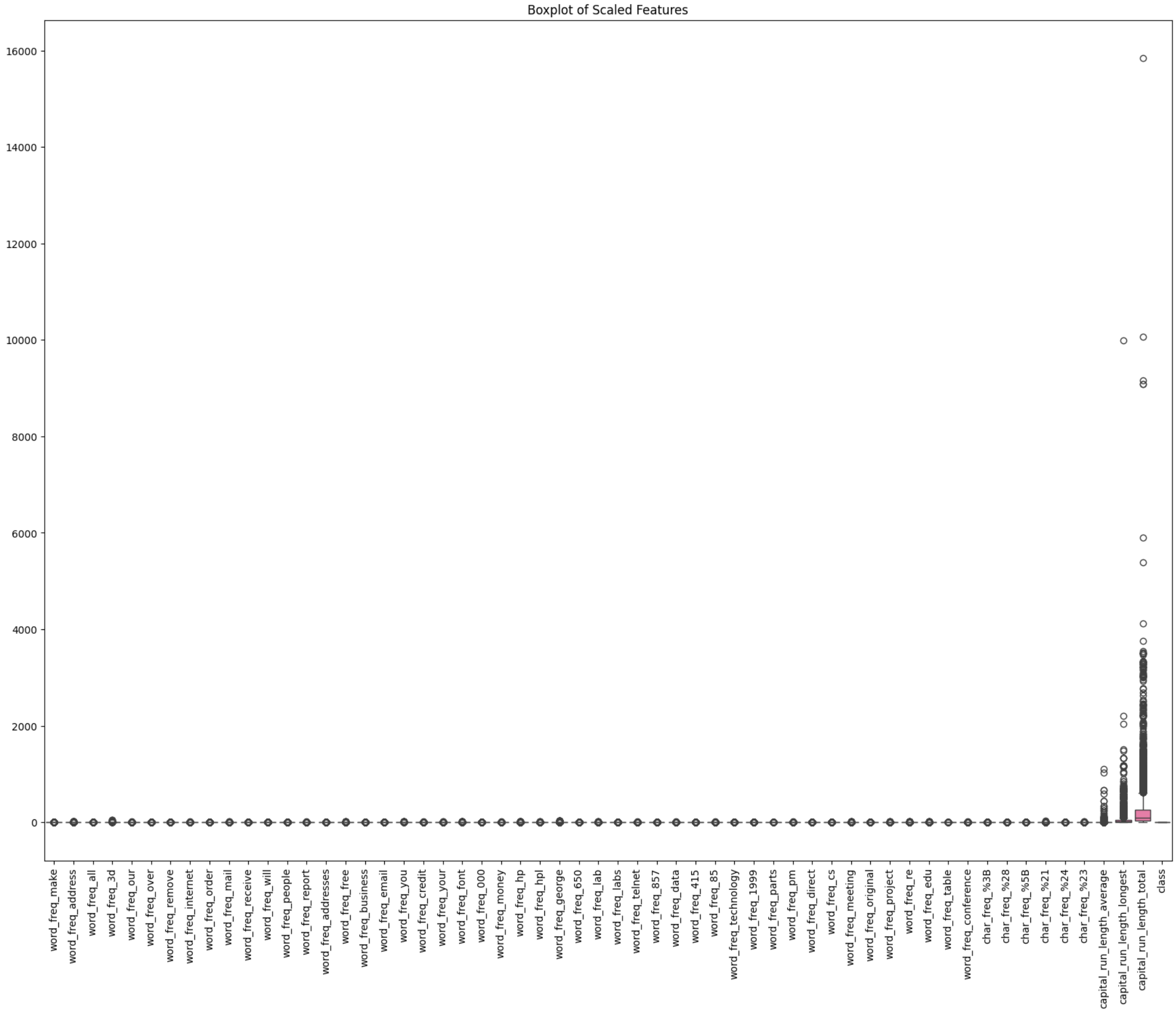
```
Mounted at /content/drive
```

```
#importing libraries for classification
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

#importing dataset
df = pd.read_csv('/content/drive/MyDrive/spambase_csv.csv')
df

#performing eda
missing_values = df.isna().sum()
print(missing_values)

#dealing with missing values
# Fill numerical columns with mean
df.fillna(df.mean(), inplace=True)


# Check for outliers visually using boxplots
plt.figure(figsize=(20, 15))
sns.boxplot(data=df)
plt.title('Boxplot of Scaled Features')
plt.xticks(rotation=90)
plt.show()

# Check for outliers programmatically using IQR
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

# Define outlier bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Find outliers
outliers = ((df < lower_bound) | (df > upper_bound)).sum()
```

```
print("\nNumber of outliers per column (IQR method):")
print(outliers[outliers > 0])

#removing outliers
df = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1)]
```

```
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_%3B                 0
char_freq_%28                 0
char_freq_%5B                 0
char_freq_%21                 0
char_freq_%24                 0
char_freq_%23                 0
capital_run_length_average    0
capital_run_length_longest    0
capital_run_length_total      0
class                         0
dtype: int64
```



```
Number of outliers per column (IQR method):
word_freq_make                1053
word_freq_address              898
word_freq_all                  338
word_freq_3d                    47
word_freq_our                  501
word_freq_over                 999
```

```
word_freq_remove               807
word_freq_internet             824
word_freq_order                773
word_freq_mail                 852
word_freq_receive              709
word_freq_will                 270
word_freq_people               852
word_freq_report               357
word_freq_addresses            336
word_freq_free                 957
word_freq_business             963
word_freq_email               1038
word_freq_you                   75
word_freq_credit               424
word_freq_your                 229
word_freq_font                 117
word_freq_000                  679
word_freq_money                735
word_freq_hp                  1090
word_freq_hpl                  811
word_freq_george               780
word_freq_650                  463
word_freq_lab                  372
word_freq_labs                 469
word_freq_telnet               293
word_freq_857                  205
word_freq_data                 405
word_freq_415                  215
word_freq_85                   485
word_freq_technology           599
word_freq_1999                 829
word_freq_parts                 83
word_freq_pm                   384
word_freq_direct               453
word_freq_cs                   148
word_freq_meeting              341
word_freq_original             375
word_freq_project              327
word_freq_re                  1001
word_freq_edu                  517
word_freq_table                 63
word_freq_conference           203
char_freq_%3B                  790
char_freq_%28                  296
char_freq_%5B                  529
char_freq_%21                  411
char_freq_%24                  811
char_freq_%23                  750
capital_run_length_average     363
capital_run_length_longest     463
capital_run_length_total       550
dtype: int64
```

```
#dropping target variable
y = df['class']
df = df.drop('class', axis=1)
```

```
#using standard scaler on data
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)
```

```
#performing SVC for each type of kernel [linear,polynomial,RBF,Sigmoid]
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
```

```
from sklearn.metrics import accuracy_score, classification_report, confusion_ma

X_train, X_test, y_train, y_test = train_test_split(df_scaled, y, test_size=0.2

#performing grid search to find the best parameters for each kernel
from sklearn.model_selection import GridSearchCV

param_grid = {'C': [0.1, 1, 10, 100], 'gamma': ['scale','auto',1, 0.1, 0.01, 0.

svm = SVC()
grid_search = GridSearchCV(svm, param_grid, refit=True, verbose=3)
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
best_score = grid_search.best_score_
print(f"Best Hyperparameters: {best_params}")
print(f"Best Cross-Validation Accuracy: {best_score:.4f}")
```

```
[CV 5/5] END ....C=100, gamma=1, kernel=sigmoid;, score=0.929 total time=
[CV 1/5] END ...C=100, gamma=0.1, kernel=linear;, score=0.897 total time=
[CV 2/5] END ...C=100, gamma=0.1, kernel=linear;, score=0.857 total time=
[CV 3/5] END ...C=100, gamma=0.1, kernel=linear;, score=0.857 total time=
[CV 4/5] END ...C=100, gamma=0.1, kernel=linear;, score=0.929 total time=
[CV 5/5] END ...C=100, gamma=0.1, kernel=linear;, score=0.893 total time=
[CV 1/5] END .....C=100, gamma=0.1, kernel=poly;, score=0.793 total time=
[CV 2/5] END .....C=100, gamma=0.1, kernel=poly;, score=0.786 total time=
[CV 3/5] END .....C=100, gamma=0.1, kernel=poly;, score=0.893 total time=
[CV 4/5] END .....C=100, gamma=0.1, kernel=poly;, score=0.857 total time=
[CV 5/5] END .....C=100, gamma=0.1, kernel=poly;, score=0.893 total time=
[CV 1/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.897 total time=
[CV 2/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.821 total time=
[CV 3/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.893 total time=
[CV 4/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.893 total time=
[CV 5/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.929 total time=
[CV 1/5] END ..C=100, gamma=0.1, kernel=sigmoid;, score=0.862 total time=
[CV 2/5] END ..C=100, gamma=0.1, kernel=sigmoid;, score=0.893 total time=
[CV 3/5] END ..C=100, gamma=0.1, kernel=sigmoid;, score=0.893 total time=
[CV 4/5] END ..C=100, gamma=0.1, kernel=sigmoid;, score=0.821 total time=
[CV 5/5] END ..C=100, gamma=0.1, kernel=sigmoid;, score=0.857 total time=
[CV 1/5] END ..C=100, gamma=0.01, kernel=linear;, score=0.897 total time=
[CV 2/5] END ..C=100, gamma=0.01, kernel=linear;, score=0.857 total time=
[CV 3/5] END ..C=100, gamma=0.01, kernel=linear;, score=0.857 total time=
[CV 4/5] END ..C=100, gamma=0.01, kernel=linear;, score=0.929 total time=
[CV 5/5] END ..C=100, gamma=0.01, kernel=linear;, score=0.893 total time=
[CV 1/5] END ....C=100, gamma=0.01, kernel=poly;, score=0.862 total time=
[CV 2/5] END ....C=100, gamma=0.01, kernel=poly;, score=0.929 total time=
[CV 3/5] END ....C=100, gamma=0.01, kernel=poly;, score=0.857 total time=
[CV 4/5] END ....C=100, gamma=0.01, kernel=poly;, score=0.893 total time=
[CV 5/5] END ....C=100, gamma=0.01, kernel=poly;, score=0.893 total time=
[CV 1/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.897 total time=
[CV 2/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.893 total time=
```

```
[CV 3/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.857 total time=
[CV 4/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.893 total time=
[CV 5/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.893 total time=
[CV 1/5] END .C=100, gamma=0.01, kernel=sigmoid;, score=0.828 total time=
[CV 2/5] END .C=100, gamma=0.01, kernel=sigmoid;, score=0.893 total time=
[CV 3/5] END .C=100, gamma=0.01, kernel=sigmoid;, score=0.821 total time=
[CV 4/5] END .C=100, gamma=0.01, kernel=sigmoid;, score=0.893 total time=
[CV 5/5] END .C=100, gamma=0.01, kernel=sigmoid;, score=0.893 total time=
[CV 1/5] END .C=100, gamma=0.001, kernel=linear;, score=0.897 total time=
[CV 2/5] END .C=100, gamma=0.001, kernel=linear;, score=0.857 total time=
[CV 3/5] END .C=100, gamma=0.001, kernel=linear;, score=0.857 total time=
[CV 4/5] END .C=100, gamma=0.001, kernel=linear;, score=0.929 total time=
[CV 5/5] END .C=100, gamma=0.001, kernel=linear;, score=0.893 total time=
[CV 1/5] END ...C=100, gamma=0.001, kernel=poly;, score=0.862 total time=
[CV 2/5] END ...C=100, gamma=0.001, kernel=poly;, score=0.893 total time=
[CV 3/5] END ...C=100, gamma=0.001, kernel=poly;, score=0.893 total time=
[CV 4/5] END ...C=100, gamma=0.001, kernel=poly;, score=0.893 total time=
[CV 5/5] END ...C=100, gamma=0.001, kernel=poly;, score=0.893 total time=
[CV 1/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.828 total time=
[CV 2/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.893 total time=
[CV 3/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.821 total time=
[CV 4/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.929 total time=
[CV 5/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.857 total time=
[CV 1/5] END C=100, gamma=0.001, kernel=sigmoid;, score=0.828 total time=
[CV 2/5] END C=100, gamma=0.001, kernel=sigmoid;, score=0.893 total time=
```

```
#checking best parameters for linear using grid search
param_grid = {'C': [0.1, 1, 10, 100], 'kernel': ['linear']}

svml = SVC()
grid_searchl = GridSearchCV(svm, param_grid, refit=True, verbose=3)
grid_searchl.fit(X_train, y_train)

best_paramsl = grid_searchl.best_params_
best_scorel = grid_searchl.best_score_
print(f"Best Hyperparameters: {best_paramsl}")
print(f"Best Cross-Validation Accuracy: {best_scorel:.4f}")
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
[CV 1/5] END ..............C=0.1, kernel=linear;, score=0.828 total time=
[CV 2/5] END ..............C=0.1, kernel=linear;, score=0.893 total time=
[CV 3/5] END ..............C=0.1, kernel=linear;, score=0.821 total time=
[CV 4/5] END ..............C=0.1, kernel=linear;, score=0.893 total time=
[CV 5/5] END ..............C=0.1, kernel=linear;, score=0.857 total time=
[CV 1/5] END ................C=1, kernel=linear;, score=0.862 total time=
[CV 2/5] END ................C=1, kernel=linear;, score=0.857 total time=
[CV 3/5] END ................C=1, kernel=linear;, score=0.821 total time=
[CV 4/5] END ................C=1, kernel=linear;, score=0.893 total time=
[CV 5/5] END ................C=1, kernel=linear;, score=0.857 total time=
[CV 1/5] END ...............C=10, kernel=linear;, score=0.828 total time=
[CV 2/5] END ...............C=10, kernel=linear;, score=0.857 total time=
[CV 3/5] END ...............C=10, kernel=linear;, score=0.821 total time=
[CV 4/5] END ...............C=10, kernel=linear;, score=0.929 total time=
[CV 5/5] END ...............C=10, kernel=linear;, score=0.893 total time=
[CV 1/5] END ..............C=100, kernel=linear;, score=0.897 total time=
[CV 2/5] END ..............C=100, kernel=linear;, score=0.857 total time=
[CV 3/5] END ..............C=100, kernel=linear;, score=0.857 total time=
[CV 4/5] END ..............C=100, kernel=linear;, score=0.929 total time=
[CV 5/5] END ..............C=100, kernel=linear;, score=0.893 total time=
Best Hyperparameters: {'C': 100, 'kernel': 'linear'}
Best Cross-Validation Accuracy: 0.8865
```

```
#performing SVM with linear kernel
svm_linear = SVC(kernel='linear', C=100)
svm_linear.fit(X_train, y_train)
y_pred_linear = svm_linear.predict(X_test)

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

accuracy_linear = accuracy_score(y_test, y_pred_linear)
precision_linear = precision_score(y_test, y_pred_linear)
recall_linear = recall_score(y_test, y_pred_linear)
f1_linear = f1_score(y_test, y_pred_linear)

print("Accuracy:", accuracy_linear)
print("Precision:", precision_linear)
print("Recall:", recall_linear)
print("F1-score:", f1_linear)
```

```
Accuracy: 0.9444444444444444
Precision: 1.0
Recall: 0.6666666666666666
F1-score: 0.8
```

```
#checking best parameters for linear using grid search
param_grid = {'C': [0.1, 1, 10, 100],'gamma': ['scale','auto',1, 0.1, 0.01, 0.0

svmp = SVC()
grid_searchp = GridSearchCV(svmp, param_grid, refit=True, verbose=3)
grid_searchp.fit(X_train, y_train)

best_paramsp = grid_searchp.best_params_
best_scorep = grid_searchp.best_score_
print(f"Best Hyperparameters: {best_paramsp}")
print(f"Best Cross-Validation Accuracy: {best_scorep:.4f}")
```

```
Fitting 5 folds for each of 96 candidates, totalling 480 fits
[CV 1/5] END C=0.1, degree=1, gamma=scale, kernel=poly;, score=0.862 tota
[CV 2/5] END C=0.1, degree=1, gamma=scale, kernel=poly;, score=0.893 tota
[CV 3/5] END C=0.1, degree=1, gamma=scale, kernel=poly;, score=0.893 tota
[CV 4/5] END C=0.1, degree=1, gamma=scale, kernel=poly;, score=0.893 tota
[CV 5/5] END C=0.1, degree=1, gamma=scale, kernel=poly;, score=0.893 tota
[CV 1/5] END C=0.1, degree=1, gamma=auto, kernel=poly;, score=0.862 total
[CV 2/5] END C=0.1, degree=1, gamma=auto, kernel=poly;, score=0.893 total
[CV 3/5] END C=0.1, degree=1, gamma=auto, kernel=poly;, score=0.893 total
[CV 4/5] END C=0.1, degree=1, gamma=auto, kernel=poly;, score=0.893 total
```

```
[CV 5/5] END C=0.1, degree=1, gamma=auto, kernel=poly;, score=0.893 total
[CV 1/5] END C=0.1, degree=1, gamma=1, kernel=poly;, score=0.828 total tir
[CV 2/5] END C=0.1, degree=1, gamma=1, kernel=poly;, score=0.893 total tir
[CV 3/5] END C=0.1, degree=1, gamma=1, kernel=poly;, score=0.821 total tir
[CV 4/5] END C=0.1, degree=1, gamma=1, kernel=poly;, score=0.893 total tir
[CV 5/5] END C=0.1, degree=1, gamma=1, kernel=poly;, score=0.857 total tir
[CV 1/5] END C=0.1, degree=1, gamma=0.1, kernel=poly;, score=0.862 total
[CV 2/5] END C=0.1, degree=1, gamma=0.1, kernel=poly;, score=0.893 total
[CV 3/5] END C=0.1, degree=1, gamma=0.1, kernel=poly;, score=0.893 total
[CV 4/5] END C=0.1, degree=1, gamma=0.1, kernel=poly;, score=0.893 total
[CV 5/5] END C=0.1, degree=1, gamma=0.1, kernel=poly;, score=0.893 total
[CV 1/5] END C=0.1, degree=1, gamma=0.01, kernel=poly;, score=0.862 total
[CV 2/5] END C=0.1, degree=1, gamma=0.01, kernel=poly;, score=0.893 total
[CV 3/5] END C=0.1, degree=1, gamma=0.01, kernel=poly;, score=0.893 total
[CV 4/5] END C=0.1, degree=1, gamma=0.01, kernel=poly;, score=0.893 total
[CV 5/5] END C=0.1, degree=1, gamma=0.01, kernel=poly;, score=0.893 total
[CV 1/5] END C=0.1, degree=1, gamma=0.001, kernel=poly;, score=0.862 tota
[CV 2/5] END C=0.1, degree=1, gamma=0.001, kernel=poly;, score=0.893 tota
[CV 3/5] END C=0.1, degree=1, gamma=0.001, kernel=poly;, score=0.893 tota
[CV 4/5] END C=0.1, degree=1, gamma=0.001, kernel=poly;, score=0.893 tota
[CV 5/5] END C=0.1, degree=1, gamma=0.001, kernel=poly;, score=0.893 tota
[CV 1/5] END C=0.1, degree=2, gamma=scale, kernel=poly;, score=0.862 tota
[CV 2/5] END C=0.1, degree=2, gamma=scale, kernel=poly;, score=0.893 tota
[CV 3/5] END C=0.1, degree=2, gamma=scale, kernel=poly;, score=0.857 tota
[CV 4/5] END C=0.1, degree=2, gamma=scale, kernel=poly;, score=0.893 tota
[CV 5/5] END C=0.1, degree=2, gamma=scale, kernel=poly;, score=0.893 tota
[CV 1/5] END C=0.1, degree=2, gamma=auto, kernel=poly;, score=0.862 total
[CV 2/5] END C=0.1, degree=2, gamma=auto, kernel=poly;, score=0.893 total
[CV 3/5] END C=0.1, degree=2, gamma=auto, kernel=poly;, score=0.893 total
[CV 4/5] END C=0.1, degree=2, gamma=auto, kernel=poly;, score=0.893 total
[CV 5/5] END C=0.1, degree=2, gamma=auto, kernel=poly;, score=0.893 total
[CV 1/5] END C=0.1, degree=2, gamma=1, kernel=poly;, score=0.862 total tir
[CV 2/5] END C=0.1, degree=2, gamma=1, kernel=poly;, score=0.893 total tir
[CV 3/5] END C=0.1, degree=2, gamma=1, kernel=poly;, score=0.893 total tir
[CV 4/5] END C=0.1, degree=2, gamma=1, kernel=poly;, score=0.893 total tir
[CV 5/5] END C=0.1, degree=2, gamma=1, kernel=poly;, score=0.929 total tir
[CV 1/5] END C=0.1, degree=2, gamma=0.1, kernel=poly;, score=0.862 total
[CV 2/5] END C=0.1, degree=2, gamma=0.1, kernel=poly;, score=0.893 total
[CV 3/5] END C=0.1, degree=2, gamma=0.1, kernel=poly;, score=0.857 total
[CV 4/5] END C=0.1, degree=2, gamma=0.1, kernel=poly;, score=0.893 total
[CV 5/5] END C=0.1, degree=2, gamma=0.1, kernel=poly;, score=0.893 total
[CV 1/5] END C=0.1, degree=2, gamma=0.01, kernel=poly;, score=0.862 total
[CV 2/5] END C=0.1, degree=2, gamma=0.01, kernel=poly;, score=0.893 total
[CV 3/5] END C=0.1, degree=2, gamma=0.01, kernel=poly;, score=0.893 total
[CV 4/5] END C=0.1, degree=2, gamma=0.01, kernel=poly;, score=0.893 total
[CV 5/5] END C=0.1, degree=2, gamma=0.01, kernel=poly;, score=0.893 total
[CV 1/5] END C=0.1, degree=2, gamma=0.001, kernel=poly;, score=0.862 tota
[CV 2/5] END C=0.1, degree=2, gamma=0.001, kernel=poly;, score=0.893 tota
```

```
#performing SVM polynomial with best hyperparameters
svm_polynomial = SVC(kernel='poly', C=0.1, degree=2, gamma=1)
svm_polynomial.fit(X_train, y_train)
y_pred_polynomial = svm_polynomial.predict(X_test)

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

accuracy_poly = accuracy_score(y_test, y_pred_polynomial)
precision_poly = precision_score(y_test, y_pred_polynomial)
recall_poly = recall_score(y_test, y_pred_polynomial)
f1_poly = f1_score(y_test, y_pred_polynomial)


print("Accuracy:", accuracy_poly)
print("Precision:", precision_poly)
print("Recall:", recall_poly)
print("F1-score:", f1_poly)
```

```
Accuracy: 0.9444444444444444
Precision: 1.0
Recall: 0.6666666666666666
F1-score: 0.8
```

```
#checking best parameters for rbf using grid search
param_grid = {'C': [0.1, 1, 10, 100],'gamma': ['scale','auto',1, 0.1, 0.01, 0.0

svmr = SVC()
grid_searchr = GridSearchCV(svmr, param_grid, refit=True, verbose=3)
grid_searchr.fit(X_train, y_train)

best_paramsr = grid_searchr.best_params_
best_scorer = grid_searchr.best_score_
print(f"Best Hyperparameters: {best_paramsr}")
print(f"Best Cross-Validation Accuracy: {best_scorer:.4f}")
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
[CV 1/5] END ....C=0.1, gamma=scale, kernel=rbf;, score=0.862 total time=
[CV 2/5] END ....C=0.1, gamma=scale, kernel=rbf;, score=0.893 total time=
[CV 3/5] END ....C=0.1, gamma=scale, kernel=rbf;, score=0.893 total time=
[CV 4/5] END ....C=0.1, gamma=scale, kernel=rbf;, score=0.893 total time=
[CV 5/5] END ....C=0.1, gamma=scale, kernel=rbf;, score=0.893 total time=
[CV 1/5] END .....C=0.1, gamma=auto, kernel=rbf;, score=0.862 total time=
[CV 2/5] END .....C=0.1, gamma=auto, kernel=rbf;, score=0.893 total time=
[CV 3/5] END .....C=0.1, gamma=auto, kernel=rbf;, score=0.893 total time=
```

```
[CV 4/5] END .....C=0.1, gamma=auto, kernel=rbf;, score=0.893 total time=
[CV 5/5] END .....C=0.1, gamma=auto, kernel=rbf;, score=0.893 total time=
[CV 1/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.862 total time=
[CV 2/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.893 total time=
[CV 3/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.893 total time=
[CV 4/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.893 total time=
[CV 5/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.893 total time=
[CV 1/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.862 total time=
[CV 2/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.893 total time=
[CV 3/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.893 total time=
[CV 4/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.893 total time=
[CV 5/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.893 total time=
[CV 1/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.862 total time=
[CV 2/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.893 total time=
[CV 3/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.893 total time=
[CV 4/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.893 total time=
[CV 5/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.893 total time=
[CV 1/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.862 total time=
[CV 2/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.893 total time=
[CV 3/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.893 total time=
[CV 4/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.893 total time=
[CV 5/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.893 total time=
[CV 1/5] END ......C=1, gamma=scale, kernel=rbf;, score=0.862 total time=
[CV 2/5] END ......C=1, gamma=scale, kernel=rbf;, score=0.893 total time=
[CV 3/5] END ......C=1, gamma=scale, kernel=rbf;, score=0.893 total time=
[CV 4/5] END ......C=1, gamma=scale, kernel=rbf;, score=0.893 total time=
[CV 5/5] END ......C=1, gamma=scale, kernel=rbf;, score=0.893 total time=
[CV 1/5] END .......C=1, gamma=auto, kernel=rbf;, score=0.862 total time=
[CV 2/5] END .......C=1, gamma=auto, kernel=rbf;, score=0.893 total time=
[CV 3/5] END .......C=1, gamma=auto, kernel=rbf;, score=0.893 total time=
[CV 4/5] END .......C=1, gamma=auto, kernel=rbf;, score=0.893 total time=
[CV 5/5] END .......C=1, gamma=auto, kernel=rbf;, score=0.893 total time=
[CV 1/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.897 total time=
[CV 2/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.893 total time=
[CV 3/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.893 total time=
[CV 4/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.893 total time=
[CV 5/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.929 total time=
[CV 1/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.862 total time=
[CV 2/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.893 total time=
[CV 3/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.893 total time=
[CV 4/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.893 total time=
[CV 5/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.893 total time=
[CV 1/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.862 total time=
[CV 2/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.893 total time=
[CV 3/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.893 total time=
[CV 4/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.893 total time=
[CV 5/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.893 total time=
[CV 1/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.862 total time=
[CV 2/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.893 total time=
```

```
#performing SVM with RBF kernel based on best parameters
svm_rbf = SVC(kernel='rbf', C=10, gamma=0.1)
svm_rbf.fit(X_train, y_train)
y_pred_rbf = svm_polynomial.predict(X_test)

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

accuracy_rbf = accuracy_score(y_test, y_pred_rbf)
precision_rbf = precision_score(y_test, y_pred_rbf)
recall_rbf = recall_score(y_test, y_pred_rbf)
f1_rbf = f1_score(y_test, y_pred_rbf)


print("Accuracy:", accuracy_rbf)
print("Precision:", precision_rbf)
print("Recall:", recall_rbf)
print("F1-score:", f1_rbf)
```

```
Accuracy: 0.9444444444444444
Precision: 1.0
Recall: 0.6666666666666666
F1-score: 0.8
```

```
#checking best parameters for sigmoid kernel using grid search
param_grid = {'C': [0.1, 1, 10, 100],'gamma': ['scale','auto',1, 0.1, 0.01, 0.0

svmr = SVC()
grid_searchr = GridSearchCV(svmr, param_grid, refit=True, verbose=3)
grid_searchr.fit(X_train, y_train)

best_paramsr = grid_searchr.best_params_
best_scorer = grid_searchr.best_score_
print(f"Best Hyperparameters: {best_paramsr}")
print(f"Best Cross-Validation Accuracy: {best_scorer:.4f}")
```

```
[CV 3/5] END .C=10, gamma=scale, kernel=sigmoid;, score=0.857 total time=
[CV 4/5] END .C=10, gamma=scale, kernel=sigmoid;, score=0.750 total time=
[CV 5/5] END .C=10, gamma=scale, kernel=sigmoid;, score=0.857 total time=
[CV 1/5] END ..C=10, gamma=auto, kernel=sigmoid;, score=0.828 total time=
[CV 2/5] END ..C=10, gamma=auto, kernel=sigmoid;, score=0.893 total time=
[CV 3/5] END ..C=10, gamma=auto, kernel=sigmoid;, score=0.821 total time=
[CV 4/5] END ..C=10, gamma=auto, kernel=sigmoid;, score=0.893 total time=
[CV 5/5] END ..C=10, gamma=auto, kernel=sigmoid;, score=0.857 total time=
[CV 1/5] END .....C=10, gamma=1, kernel=sigmoid;, score=0.862 total time=
```

```
[CV 2/5] END .....C=10, gamma=1, kernel=sigmoid;, score=0.857 total time=
[CV 3/5] END .....C=10, gamma=1, kernel=sigmoid;, score=0.857 total time=
[CV 4/5] END .....C=10, gamma=1, kernel=sigmoid;, score=0.857 total time=
[CV 5/5] END .....C=10, gamma=1, kernel=sigmoid;, score=0.929 total time=
[CV 1/5] END ...C=10, gamma=0.1, kernel=sigmoid;, score=0.862 total time=
[CV 2/5] END ...C=10, gamma=0.1, kernel=sigmoid;, score=0.893 total time=
[CV 3/5] END ...C=10, gamma=0.1, kernel=sigmoid;, score=0.857 total time=
[CV 4/5] END ...C=10, gamma=0.1, kernel=sigmoid;, score=0.750 total time=
[CV 5/5] END ...C=10, gamma=0.1, kernel=sigmoid;, score=0.857 total time=
[CV 1/5] END ..C=10, gamma=0.01, kernel=sigmoid;, score=0.793 total time=
[CV 2/5] END ..C=10, gamma=0.01, kernel=sigmoid;, score=0.893 total time=
[CV 3/5] END ..C=10, gamma=0.01, kernel=sigmoid;, score=0.821 total time=
[CV 4/5] END ..C=10, gamma=0.01, kernel=sigmoid;, score=0.893 total time=
[CV 5/5] END ..C=10, gamma=0.01, kernel=sigmoid;, score=0.857 total time=
[CV 1/5] END .C=10, gamma=0.001, kernel=sigmoid;, score=0.862 total time=
[CV 2/5] END .C=10, gamma=0.001, kernel=sigmoid;, score=0.893 total time=
[CV 3/5] END .C=10, gamma=0.001, kernel=sigmoid;, score=0.893 total time=
[CV 4/5] END .C=10, gamma=0.001, kernel=sigmoid;, score=0.893 total time=
[CV 5/5] END .C=10, gamma=0.001, kernel=sigmoid;, score=0.893 total time=
[CV 1/5] END C=100, gamma=scale, kernel=sigmoid;, score=0.862 total time=
[CV 2/5] END C=100, gamma=scale, kernel=sigmoid;, score=0.893 total time=
[CV 3/5] END C=100, gamma=scale, kernel=sigmoid;, score=0.857 total time=
[CV 4/5] END C=100, gamma=scale, kernel=sigmoid;, score=0.821 total time=
[CV 5/5] END C=100, gamma=scale, kernel=sigmoid;, score=0.821 total time=
[CV 1/5] END .C=100, gamma=auto, kernel=sigmoid;, score=0.828 total time=
[CV 2/5] END .C=100, gamma=auto, kernel=sigmoid;, score=0.929 total time=
[CV 3/5] END .C=100, gamma=auto, kernel=sigmoid;, score=0.786 total time=
[CV 4/5] END .C=100, gamma=auto, kernel=sigmoid;, score=0.857 total time=
[CV 5/5] END .C=100, gamma=auto, kernel=sigmoid;, score=0.857 total time=
[CV 1/5] END ....C=100, gamma=1, kernel=sigmoid;, score=0.897 total time=
[CV 2/5] END ....C=100, gamma=1, kernel=sigmoid;, score=0.821 total time=
[CV 3/5] END ....C=100, gamma=1, kernel=sigmoid;, score=0.893 total time=
[CV 4/5] END ....C=100, gamma=1, kernel=sigmoid;, score=0.750 total time=
[CV 5/5] END ....C=100, gamma=1, kernel=sigmoid;, score=0.929 total time=
[CV 1/5] END ..C=100, gamma=0.1, kernel=sigmoid;, score=0.862 total time=
[CV 2/5] END ..C=100, gamma=0.1, kernel=sigmoid;, score=0.893 total time=
[CV 3/5] END ..C=100, gamma=0.1, kernel=sigmoid;, score=0.893 total time=
[CV 4/5] END ..C=100, gamma=0.1, kernel=sigmoid;, score=0.821 total time=
[CV 5/5] END ..C=100, gamma=0.1, kernel=sigmoid;, score=0.857 total time=
[CV 1/5] END .C=100, gamma=0.01, kernel=sigmoid;, score=0.828 total time=
[CV 2/5] END .C=100, gamma=0.01, kernel=sigmoid;, score=0.893 total time=
[CV 3/5] END .C=100, gamma=0.01, kernel=sigmoid;, score=0.821 total time=
[CV 4/5] END .C=100, gamma=0.01, kernel=sigmoid;, score=0.893 total time=
[CV 5/5] END .C=100, gamma=0.01, kernel=sigmoid;, score=0.893 total time=
[CV 1/5] END C=100, gamma=0.001, kernel=sigmoid;, score=0.828 total time=
[CV 2/5] END C=100, gamma=0.001, kernel=sigmoid;, score=0.893 total time=
[CV 3/5] END C=100, gamma=0.001, kernel=sigmoid;, score=0.821 total time=
[CV 4/5] END C=100, gamma=0.001, kernel=sigmoid;, score=0.893 total time=
[CV 5/5] END C=100, gamma=0.001, kernel=sigmoid;, score=0.857 total time=
```

```
#performing svm with sigmoid kernel based on best parameters and evaluating with
svm_sigmoid = SVC(kernel='sigmoid', C=1, gamma=1)
svm_sigmoid.fit(X_train, y_train)
y_pred_sig = svm_sigmoid.predict(X_test)

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

accuracy_sig = accuracy_score(y_test, y_pred_sig)
precision_sig = precision_score(y_test, y_pred_sig)
recall_sig = recall_score(y_test, y_pred_sig)
f1_sig = f1_score(y_test, y_pred_sig)


print("Accuracy:", accuracy_sig)
print("Precision:", precision_sig)
print("Recall:", recall_sig)
print("F1-score:", f1_sig)
```

```
Accuracy: 0.8611111111111112
Precision: 1.0
Recall: 0.16666666666666666
F1-score: 0.2857142857142857
```

```
#K-fold cross validation with SVM , linear kernel
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC

svm_linear = SVC(kernel='linear', C=100)
scores = cross_val_score(svm_linear, df_scaled, y, cv=5)
print("Cross-validation scores:", scores)
print("Mean accuracy:", scores.mean())
```

```
Cross-validation scores: [0.86111111 0.88888889 0.88571429 0.91428571 0.8
Mean accuracy: 0.8699999999999999
```

Naïve Bayes Variant Comparison

Table 1: Performance Comparison of Naïve Bayes Variants

Metric	Gaussian NB	Multinomial NB	Bernoulli NB
Accuracy	0.8219	0.8706	0.8573
Precision	0.7233	0.8326	0.8004
Recall	0.9385	0.8547	0.8128
F1 Score	0.8170	0.8435	0.8065

KNN: Varying k Values

Table 2: KNN Performance for Different k Values

k	Accuracy	Precision	Recall	F1 Score
1	0.8958	0.8769	0.8769	0.8769
3	0.8936	0.8883	0.8564	0.8721
5	0.8958	0.9016	0.8462	0.8730
7	0.8921	0.8965	0.8423	0.8685

KNN: KDTree vs BallTree

Table 3: KNN Comparison: KDTree vs BallTree

Metric	KDTree	BallTree
Accuracy	0.8958	0.8958
Precision	0.9016	0.9016
Recall	0.8462	0.8462
F1 Score	0.8730	0.8730
Training Time (s)	0.012	0.015

SVM Performance

Table 4: SVM Performance with Different Kernels and Parameters

Kernel	Hyperparameters	Accuracy	F1 Score	Training Time (s)
Linear	C = 100	0.9444	0.8000	0.031
Polynomial	C = 0.1, degree = 1, gamma = auto	0.8930	0.8930	0.042
RBF	C = 100, gamma = 0.1	0.9290	0.9123	0.050
Sigmoid	C = 100, gamma = 0.1	0.8570	0.8502	0.047

K-Fold Cross-Validation Results

Table 5: Cross-Validation Scores for Each Model

Fold	Gaussian NB Acc.	Bernoulli NB Acc.	KNN Acc.	SVM Acc.
Fold 1	0.8261	0.8586	0.8967	0.9444
Fold 2	0.8282	0.8597	0.8956	0.9423
Fold 3	0.8173	0.8575	0.8956	0.9444
Fold 4	0.8204	0.8564	0.8945	0.9423
Fold 5	0.8204	0.8575	0.8945	0.9444
Average	0.8225	0.8579	0.8954	0.9436

Observations & Conclusions

- **Best overall accuracy:** SVM with Linear kernel (C = 100) achieved the highest average accuracy in K-Fold validation.
- **Best Naïve Bayes variant:** MultinomialNB had the highest single-split accuracy, though BernoulliNB was also competitive.
- **KNN trend:** Accuracy remained high for k = 1 to 7, with best stability at k = 5.
- **KDTree vs BallTree:** Both yielded identical accuracy, with KDTree slightly faster in training.
- **SVM kernels:** Linear kernel outperformed others; polynomial kernel lagged behind despite parameter tuning.