In [93]:
```python
# Import necessary libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
```

In [94]:
```python
#load the dataset
spambase = pd.read_csv('../BAN 230/spambase_raw.csv')
spambase.head()
```

Out[94]:

	0	0.64	0.64.1	0.1	0.32	0.2	0.3	0.4	0.5	0.6	...	0.41	0.42	0.43	0.778	0.44	0.45	3.756	61	278	1
0	0.21	0.28	0.50	0.0	0.14	0.28	0.21	0.07	0.00	0.94	...	0.00	0.132	0.0	0.372	0.180	0.048	5.114	101	1028	1
1	0.06	0.00	0.71	0.0	1.23	0.19	0.19	0.12	0.64	0.25	...	0.01	0.143	0.0	0.276	0.184	0.010	9.821	485	2259	1
2	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.00	0.137	0.0	0.137	0.000	0.000	3.537	40	191	1
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.00	0.135	0.0	0.135	0.000	0.000	3.537	40	191	1
4	0.00	0.00	0.00	0.0	1.85	0.00	0.00	1.85	0.00	0.00	...	0.00	0.223	0.0	0.000	0.000	0.000	3.000	15	54	1

5 rows × 58 columns

In [95]:
```python
spambase.shape
```

Out[95]:
```
(4600, 58)
```

In [96]:
```python
spambase.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4600 entries, 0 to 4599
Data columns (total 58 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   0       4600 non-null   float64
 1   0.64    4600 non-null   float64
 2   0.64.1  4600 non-null   float64
 3   0.1     4600 non-null   float64
 4   0.32    4600 non-null   float64
 5   0.2     4600 non-null   float64
 6   0.3     4600 non-null   float64
 7   0.4     4600 non-null   float64
 8   0.5     4600 non-null   float64
 9   0.6     4600 non-null   float64
 10  0.7     4600 non-null   float64
 11  0.64.2  4600 non-null   float64
 12  0.8     4600 non-null   float64
 13  0.9     4600 non-null   float64
 14  0.10    4600 non-null   float64
 15  0.32.1  4600 non-null   float64
 16  0.11    4600 non-null   float64
 17  1.29    4600 non-null   float64
 18  1.93    4600 non-null   float64
 19  0.12    4600 non-null   float64
 20  0.96    4600 non-null   float64
 21  0.13    4600 non-null   float64
 22  0.14    4600 non-null   float64
 23  0.15    4600 non-null   float64
 24  0.16    4600 non-null   float64
 25  0.17    4600 non-null   float64
 26  0.18    4600 non-null   float64
 27  0.19    4600 non-null   float64
 28  0.20    4600 non-null   float64
 29  0.21    4600 non-null   float64
 30  0.22    4600 non-null   float64
 31  0.23    4600 non-null   float64
 32  0.24    4600 non-null   float64
 33  0.25    4600 non-null   float64
 34  0.26    4600 non-null   float64
 35  0.27    4600 non-null   float64
 36  0.28    4600 non-null   float64
 37  0.29    4600 non-null   float64
 38  0.30    4600 non-null   float64
 39  0.31    4600 non-null   float64
 40  0.33    4600 non-null   float64
 41  0.34    4600 non-null   float64
 42  0.35    4600 non-null   float64
```

```
 43  0.36    4600 non-null   float64
 44  0.37    4600 non-null   float64
 45  0.38    4600 non-null   float64
 46  0.39    4600 non-null   float64
 47  0.40    4600 non-null   float64
 48  0.41    4600 non-null   float64
 49  0.42    4600 non-null   float64
 50  0.43    4600 non-null   float64
 51  0.778   4600 non-null   float64
 52  0.44    4600 non-null   float64
 53  0.45    4600 non-null   float64
 54  3.756   4600 non-null   float64
 55  61      4600 non-null   int64
 56  278     4600 non-null   int64
 57  1       4600 non-null   int64
dtypes: float64(55), int64(3)
memory usage: 2.0 MB
```

In [97]:

```
spambase.isnull().sum()
# there is no null in data
```

```
Out[97]: 0         0
0.64      0
0.64.1    0
0.1       0
0.32      0
0.2       0
0.3       0
0.4       0
0.5       0
0.6       0
0.7       0
0.64.2    0
0.8       0
0.9       0
0.10      0
0.32.1    0
0.11      0
1.29      0
1.93      0
0.12      0
0.96      0
0.13      0
0.14      0
0.15      0
0.16      0
0.17      0
0.18      0
0.19      0
0.20      0
0.21      0
0.22      0
0.23      0
0.24      0
0.25      0
0.26      0
0.27      0
0.28      0
0.29      0
0.30      0
0.31      0
0.33      0
0.34      0
0.35      0
0.36      0
0.37      0
0.38      0
0.39      0
```

```
0.40      0
0.41      0
0.42      0
0.43      0
0.778     0
0.44      0
0.45      0
3.756     0
61        0
278       0
1         0
dtype: int64
```

In [98]:
```
#splitting feature X  and target Y
X = spambase.iloc[:, :-1]
y = spambase.iloc[:, -1]
```

In [99]:
```
# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled
```

Out[99]:
```
array([[ 3.45251829e-01,  5.19760953e-02,  4.35261246e-01, ...,
        -2.45283780e-03,  2.50545504e-01,  1.22818869e+00],
       [-1.45981828e-01, -1.64984012e-01,  8.51832832e-01, ...,
         1.45895187e-01,  2.22087495e+00,  3.25837649e+00],
       [-3.42475291e-01, -1.64984012e-01, -5.56575862e-01, ...,
        -5.21543111e-02, -6.24495382e-02, -1.52207080e-01],
       ...,
       [ 6.39992023e-01, -1.64984012e-01,  3.85264032e-02, ...,
        -1.19378942e-01, -2.36905791e-01, -2.72600020e-01],
       [ 2.80142011e+00, -1.64984012e-01, -5.56575862e-01, ...,
        -1.27478675e-01, -2.42036857e-01, -3.38568754e-01],
       [-3.42475291e-01, -1.64984012e-01,  7.32812379e-01, ...,
        -1.24232478e-01, -2.42036857e-01, -4.01239052e-01]])
```

In [100…
```
# Split into train and test sets (70/30)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)
```

In [101…
```
# Train the SVM model
model = SVC(kernel='linear')
model.fit(X_train, y_train)
```

Out[101...

SVC

```
SVC(kernel='linear')
```

In [102...

```python
#predict and model
y_pred = model.predict(X_test)
```

In [103...

```python
print(classification_report(y_test, y_pred))
```

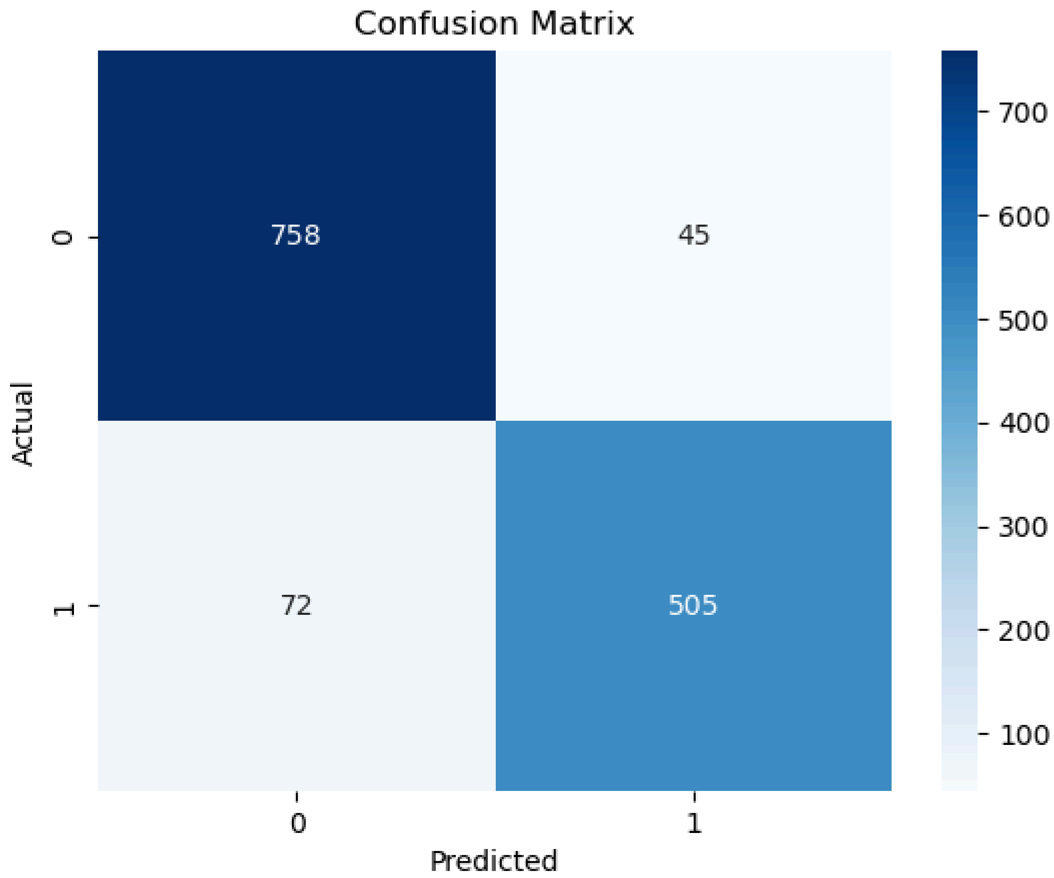
```
              precision    recall  f1-score   support

           0       0.91      0.94      0.93       803
           1       0.92      0.88      0.90       577

    accuracy                           0.92      1380
   macro avg       0.92      0.91      0.91      1380
weighted avg       0.92      0.92      0.91      1380
```

In [104...

```python
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()
```



Interpret

The SVM model with a linear kernel demonstrated strong performance in classifying emails as spam or not spam. It achieved an overall accuracy of 92%, as shown in the classification report.

From the confusion matrix, we observe: 758 legitimate (non-spam) emails were correctly identified. 45 legitimate emails were incorrectly classified as spam (false positives). 505 spam emails were correctly detected. 72 spam emails were missed and classified as non-spam (false negatives).

The model achieved a precision of 0.92 and recall of 0.88 for spam emails (label 1). This indicates it is highly accurate in labeling spam correctly but still misses some spam messages (the 72 false negatives). The F1-score of 0.90 for spam confirms a solid balance between "precision" and "recall". The false positives may arise from legitimate emails containing words or patterns resembling spam, while false negatives may occur when spam messages are too subtle or resemble normal communication. Overall, the model effectively minimizes both types of errors, making it a reliable spam filter with room for slight recall improvements on spam detection.

In [106…

```
#80/20 train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

In [107…

```
# Train the SVM model
model = SVC(kernel='linear')
model.fit(X_train, y_train)
#predict and model
y_pred = model.predict(X_test)
#print the classification report
print(classification_report(y_test, y_pred))
```

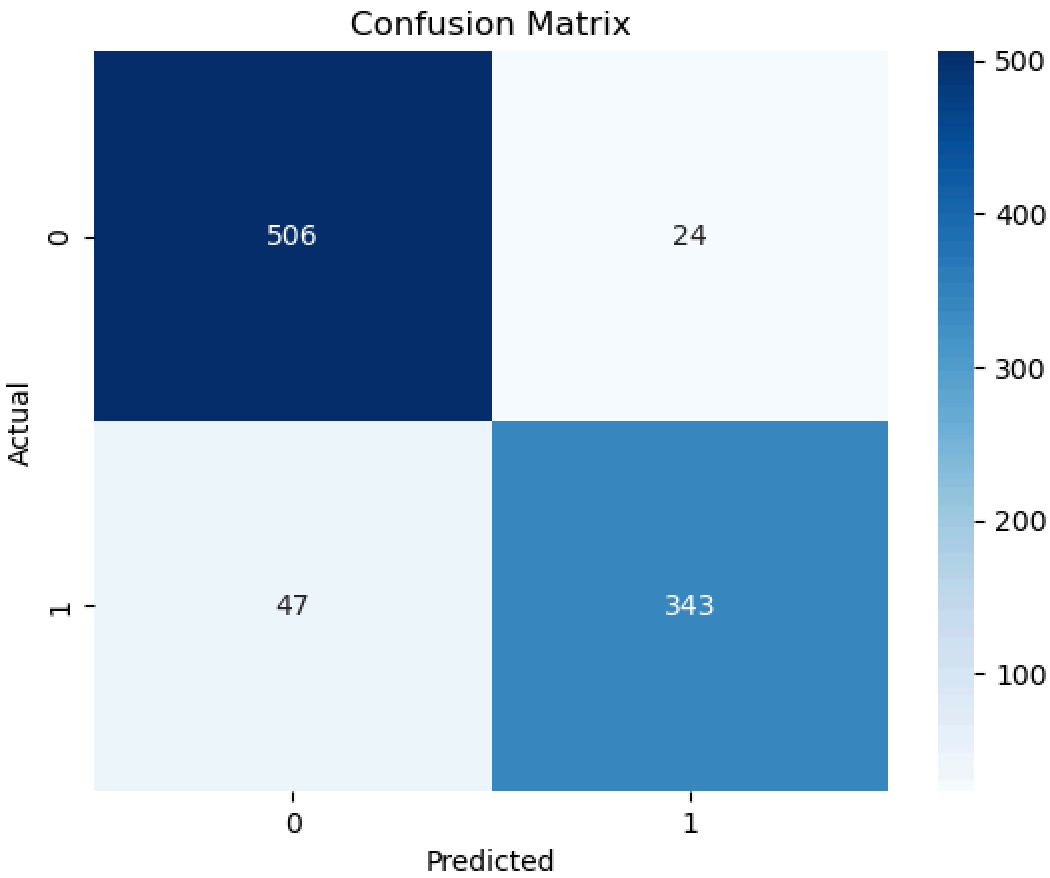
```
              precision    recall  f1-score   support

           0       0.92      0.95      0.93       530
           1       0.93      0.88      0.91       390

    accuracy                           0.92       920
   macro avg       0.92      0.92      0.92       920
weighted avg       0.92      0.92      0.92       920
```

In [108…

```
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()
```



Compare the train-test split between 80/20 and 70/30:

In the 80/20 split scenario, the model trained on a larger portion of the data, with 80% used for training and 20% for testing. The model achieved the same overall accuracy of 92%, consistent with the 70/30 split. However, there were small but meaningful differences in the precision and recall values.

From the classification report: Spam (label 1) detection showed slightly higher precision (0.93 vs. 0.92) and same recall (0.88), leading to a small boost in F1-score (0.91 vs. 0.90). Non-spam (label 0) recall also improved (0.95 vs. 0.94), and precision increased to 0.92.

From the confusion matrix: False positives (non-spam predicted as spam) decreased from 45 to 24. False negatives (spam missed) slightly decreased from 72 to 47. The 80/20 split yielded slightly better classification performance, particularly in reducing misclassified legitimate emails (false positives). The model benefited from having more training data, improving generalization. The results suggest that, for this dataset size, using more data for training can lead to marginal gains in precision without sacrificing overall accuracy. This makes the 80/20 split a favorable option for deployment scenarios where spam precision is critical.

In [ ]: