In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, accuracy_score
```

In [4]:
```
data = pd.read_csv('emails.csv')
```

In [5]:
```
data.head()
```

Out[5]:

	Email No.	the	to	ect	and	for	of	a	you	hou	...	connevey	jay	valued	lay	infrastructure	military	allowing	ff	dry	Predic
0	Email 1	0	0	1	0	0	0	2	0	0	...	0	0	0	0	0	0	0	0	0	
1	Email 2	8	13	24	6	6	2	102	1	27	...	0	0	0	0	0	0	0	1	0	
2	Email 3	0	0	1	0	0	0	8	0	0	...	0	0	0	0	0	0	0	0	0	
3	Email 4	0	5	22	0	5	1	51	2	10	...	0	0	0	0	0	0	0	0	0	
4	Email 5	7	6	17	1	5	2	57	0	9	...	0	0	0	0	0	0	0	1	0	

5 rows × 3002 columns

In [6]:
```
data.isnull().sum()
```

Out[6]:

```
Email No.     0
the           0
to            0
ect           0
and           0
             ..
military      0
allowing      0
ff            0
dry           0
Prediction    0
Length: 3002, dtype: int64
```

In [7]:

```python
# Drop the unnecessary columns such as 'Email No.' and 'Prediction'
# Assuming 'Prediction' is the target and the rest are features.
X = data.drop(columns=['Email No.', 'Prediction'])
y = data['Prediction']
```

In [8]:

```python
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [9]:

```python
# Standardize the data (important for K-NN and SVM)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

b) Apply K-Nearest Neighbors (K-NN)

In [10]:

```python
# Initialize the K-NN model
knn = KNeighborsClassifier(n_neighbors=5)  # You can experiment with the number of neighbors

# Train the model
knn.fit(X_train_scaled, y_train)

# Make predictions
y_pred_knn = knn.predict(X_test_scaled)

# Calculate confusion matrix and accuracy score
cm_knn = confusion_matrix(y_test, y_pred_knn)
```

```
accuracy_knn = accuracy_score(y_test, y_pred_knn)

print("K-Nearest Neighbors:")
print("Confusion Matrix:\n", cm_knn)
print("Accuracy Score:", accuracy_knn)
```

```
K-Nearest Neighbors:
Confusion Matrix:
 [[846 251]
 [ 20 435]]
Accuracy Score: 0.8253865979381443
```

Apply Support Vector Machine (SVM)

In [11]:

```
# Initialize the SVM model
svm = SVC(kernel='linear')  # You can experiment with other kernels like 'rbf', 'poly', etc.

# Train the model
svm.fit(X_train_scaled, y_train)

# Make predictions
y_pred_svm = svm.predict(X_test_scaled)

# Calculate confusion matrix and accuracy score
cm_svm = confusion_matrix(y_test, y_pred_svm)
accuracy_svm = accuracy_score(y_test, y_pred_svm)

print("Support Vector Machine:")
print("Confusion Matrix:\n", cm_svm)
print("Accuracy Score:", accuracy_svm)
```

```
Support Vector Machine:
Confusion Matrix:
 [[1043   54]
 [  39  416]]
Accuracy Score: 0.9400773195876289
```

Compare the Performance Finally, you can compare the performance of the two classifiers based on their accuracy and confusion matrix. A good comparison can help you decide which algorithm performs better on your dataset.

In [12]:

```
print("Performance Comparison:")

# Print the results for K-NN
print("K-Nearest Neighbors - Accuracy:", accuracy_knn)
print("Confusion Matrix:\n", cm_knn)

# Print the results for SVM
print("Support Vector Machine - Accuracy:", accuracy_svm)
print("Confusion Matrix:\n", cm_svm)
```

```
Performance Comparison:
K-Nearest Neighbors - Accuracy: 0.8253865979381443
Confusion Matrix:
 [[846 251]
 [ 20 435]]
Support Vector Machine - Accuracy: 0.9400773195876289
Confusion Matrix:
 [[1043   54]
 [  39  416]]
```

In [ ]: