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

In [2]:
```
# --- Load Dataset ---
df = pd.read_csv("emails.csv")

print("✅ Dataset loaded successfully!")
print("Shape:", df.shape)
print(df.head())
```

```
✅ Dataset loaded successfully!
Shape: (5172, 3002)
  Email No.  the  to  ect  and  for  of    a  you  hou  ...  connevey  jay  \
0   Email 1    0   0    1    0    0   0    2    0    0  ...         0    0
1   Email 2    8  13   24    6    6   2  102    1   27  ...         0    0
2   Email 3    0   0    1    0    0   0    8    0    0  ...         0    0
3   Email 4    0   5   22    0    5   1   51    2   10  ...         0    0
4   Email 5    7   6   17    1    5   2   57    0    9  ...         0    0

   valued  lay  infrastructure  military  allowing  ff  dry  Prediction
0       0    0               0         0         0   0    0           0
1       0    0               0         0         0   1    0           0
2       0    0               0         0         0   0    0           0
3       0    0               0         0         0   0    0           0
4       0    0               0         0         0   1    0           0

[5 rows x 3002 columns]
```

In [3]:
```
# --- Separate features (X) and labels (y) ---
# All columns except 'Prediction' are features
X = df.drop(columns=['Prediction'])
y = df['Prediction']
```

In [4]:
```
# Drop non-numeric or irrelevant columns if any (like 'Email No.')
if 'Email No.' in X.columns:
    X = X.drop(columns=['Email No.'])
```

In [5]:
```
# --- Split Dataset ---
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42, stratify=y
)

print("\n--- Model Training and Performance Analysis ---")
```

```
--- Model Training and Performance Analysis ---
```

In [6]:
```
# --- K-Nearest Neighbors (KNN) ---
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
knn_pred = knn_model.predict(X_test)
```

In [7]:
```
# --- Support Vector Machine (SVM) ---
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train, y_train)
svm_pred = svm_model.predict(X_test)
```

In [8]:
```
# --- Evaluation Function ---
def evaluate_classifier(y_true, y_pred, model_name):
    acc = accuracy_score(y_true, y_pred)
    prec = precision_score(y_true, y_pred, zero_division=0)
    rec = recall_score(y_true, y_pred, zero_division=0)
    f1 = f1_score(y_true, y_pred, zero_division=0)
    cm = confusion_matrix(y_true, y_pred)

    print(f"\n--- {model_name} Performance ---")
    print(f"Accuracy:  {acc:.4f}")
    print(f"Precision: {prec:.4f}")
    print(f"Recall:    {rec:.4f}")
    print(f"F1-score:  {f1:.4f}")
    print(f"Confusion Matrix:\n{cm}")
    return acc, f1
```

In [9]:
```
# --- Evaluate Both Models ---
knn_acc, knn_f1 = evaluate_classifier(y_test, knn_pred, "K-Nearest Neighbors")
svm_acc, svm_f1 = evaluate_classifier(y_test, svm_pred, "Support Vector Machine"
```

```
--- K-Nearest Neighbors Performance ---
Accuracy:  0.8685
Precision: 0.7378
Recall:    0.8480
F1-score:  0.7891
Confusion Matrix:
[[805 113]
 [ 57 318]]

--- Support Vector Machine Performance ---
Accuracy:  0.9644
Precision: 0.9318
Recall:    0.9467
F1-score:  0.9392
Confusion Matrix:
[[892  26]
 [ 20 355]]
```

In [10]:
```
# --- Comparison ---
if svm_acc > knn_acc:
    print("\n✅ Conclusion: Support Vector Machine performs better overall.")
else:
    print("\n✅ Conclusion: K-Nearest Neighbors performs better overall.")

print("\nAnalysis complete — compare Accuracy and F1-score to confirm.")
```

```
✅ Conclusion: Support Vector Machine performs better overall.

Analysis complete — compare Accuracy and F1-score to confirm.
```

In [ ]: