```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.impute import SimpleImputer
from sklearn.svm import SVC
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score,
    confusion_matrix, classification_report
)
```

```
df = pd.read_csv("/content/emails.csv")   # <-- change filename accordingly

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]
```

```
X = df.iloc[:, 1:-1]   # skip Email_Name column
y = df.iloc[:, -1]

imputer = SimpleImputer(strategy='mean')   # replaces NaN with mean of column
X = imputer.fit_transform(X)

# ------------------------------------------------------
# Step 3: Train-test split
# ------------------------------------------------------
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
print("Training samples:", X_train.shape[0])
print("Testing samples :", X_test.shape[0])
```

```
Training samples: 4137
Testing samples : 1035
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test  = scaler.transform(X_test)

# ------------------------------------------------------
# Step 5: Train K-Nearest Neighbors classifier
# ------------------------------------------------------
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)

# ------------------------------------------------------
# Step 6: Train Support Vector Machine classifier
# ------------------------------------------------------
svm = SVC(kernel='linear', C=1)
svm.fit(X_train, y_train)
y_pred_svm = svm.predict(X_test)
```

```
def evaluate_model(name, y_true, y_pred):
    print(f"\n📊 --- {name} ---")
    print("Accuracy :", round(accuracy_score(y_true, y_pred), 3))
    print("Precision:", round(precision_score(y_true, y_pred), 3))
    print("Recall   :", round(recall_score(y_true, y_pred), 3))
    print("Confusion Matrix:\n", confusion_matrix(y_true, y_pred))
    print("Classification Report:\n", classification_report(y_true, y_pred))
```

```
evaluate_model("K-Nearest Neighbors", y_test, y_pred_knn)
evaluate_model("Support Vector Machine", y_test, y_pred_svm)
```

```
📊 --- K-Nearest Neighbors ---
Accuracy : 0.845
Precision: 0.659
Recall   : 0.953
Confusion Matrix:
 [[593 146]
 [ 14 282]]
Classification Report:
               precision    recall  f1-score   support

           0       0.98      0.80      0.88       739
           1       0.66      0.95      0.78       296

    accuracy                           0.85      1035
   macro avg       0.82      0.88      0.83      1035
weighted avg       0.89      0.85      0.85      1035


📊 --- Support Vector Machine ---
Accuracy : 0.947
Precision: 0.895
Recall   : 0.922
Confusion Matrix:
 [[707  32]
 [ 23 273]]
Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.96      0.96       739
           1       0.90      0.92      0.91       296

    accuracy                           0.95      1035
   macro avg       0.93      0.94      0.94      1035
weighted avg       0.95      0.95      0.95      1035
```

Start coding or generate with AI.