Practical No 2

2) Classify the email using the binary classification method. Email Spam detection has two states: a) Normal State – Not Spam, b) Abnormal State – Spam. Use K-Nearest Neighbors and Support Vector Machine for classification. Analyze their performance.

Dataset link: The emails.csv dataset on the Kaggle https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv

[15]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_score, confusion_matrix, ConfusionMatrixDisplay, classification_report
```

[16]:
```
df=pd.read_csv('F:/11 ANJALI VILAD COLLEGE/11 Prof Anjali Phaltane/MACHINE LEARNING/ML LAB LP-III/LP-III ML CODE/PRACTICAL NO 2/emails.csv')
df.head()
```

[16]:

	Email No.	the	to	ect	and	for	of	a	you	hou	...	connevey	jay	valued	lay	infrastructure	military	allowing	ff	dry	Prediction
0	Email 1	0	0	1	0	0	0	2	0	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	0
2	Email 3	0	0	1	0	0	0	8	0	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	0
4	Email 5	7	6	17	1	5	2	57	0	9	...	0	0	0	0	0	0	0	1	0	0

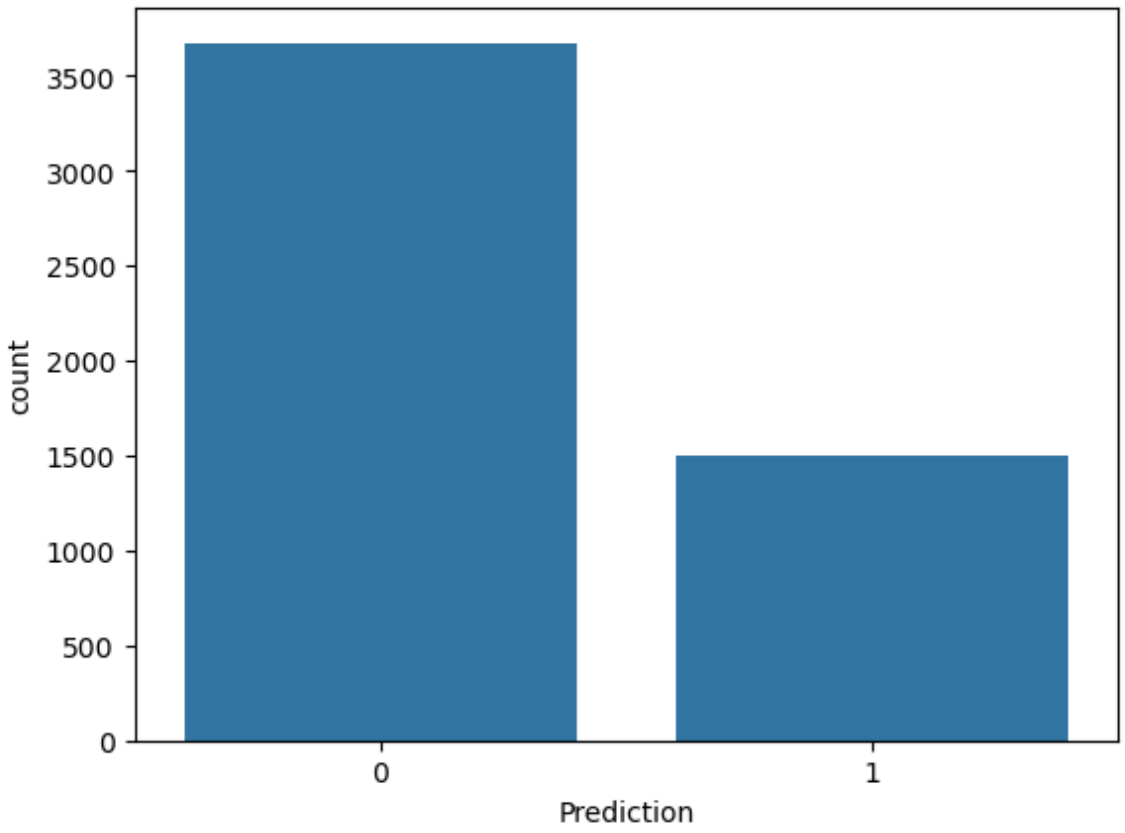
[17]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5172 entries, 0 to 5171
Columns: 3002 entries, Email No. to Prediction
dtypes: int64(3001), object(1)
memory usage: 118.5+ MB
```

[18]:
```
df.describe()
```

[18]:

	the	to	ect	and	for	of	a	you	hou	in	...	connevey	jay
count	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	...	5172.000000	5172.000000
mean	6.640565	6.188128	5.143852	3.075599	3.124710	2.627030	55.517401	2.466551	2.024362	10.600155	...	0.005027	0.012568
std	11.745009	9.534576	14.101142	6.045970	4.680522	6.229845	87.574172	4.314444	6.967878	19.281892	...	0.105788	0.199682
min	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000
25%	0.000000	1.000000	1.000000	0.000000	1.000000	0.000000	12.000000	0.000000	0.000000	1.000000	...	0.000000	0.000000
50%	3.000000	3.000000	1.000000	1.000000	2.000000	1.000000	28.000000	1.000000	0.000000	5.000000	...	0.000000	0.000000
75%	8.000000	7.000000	4.000000	3.000000	4.000000	2.000000	62.250000	3.000000	1.000000	12.000000	...	0.000000	0.000000
max	210.000000	132.000000	344.000000	89.000000	47.000000	77.000000	1898.000000	70.000000	167.000000	223.000000	...	4.000000	7.000000



[22]:
```
y = df.iloc[:, -1]
y
```

[22]:
```
0       0
1       0
2       0
3       0
4       0
       ..
5167    0
5168    0
5169    1
5170    1
5171    0
Name: Prediction, Length: 5172, dtype: int64
```

[23]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=8)
```

[23]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=8)
```

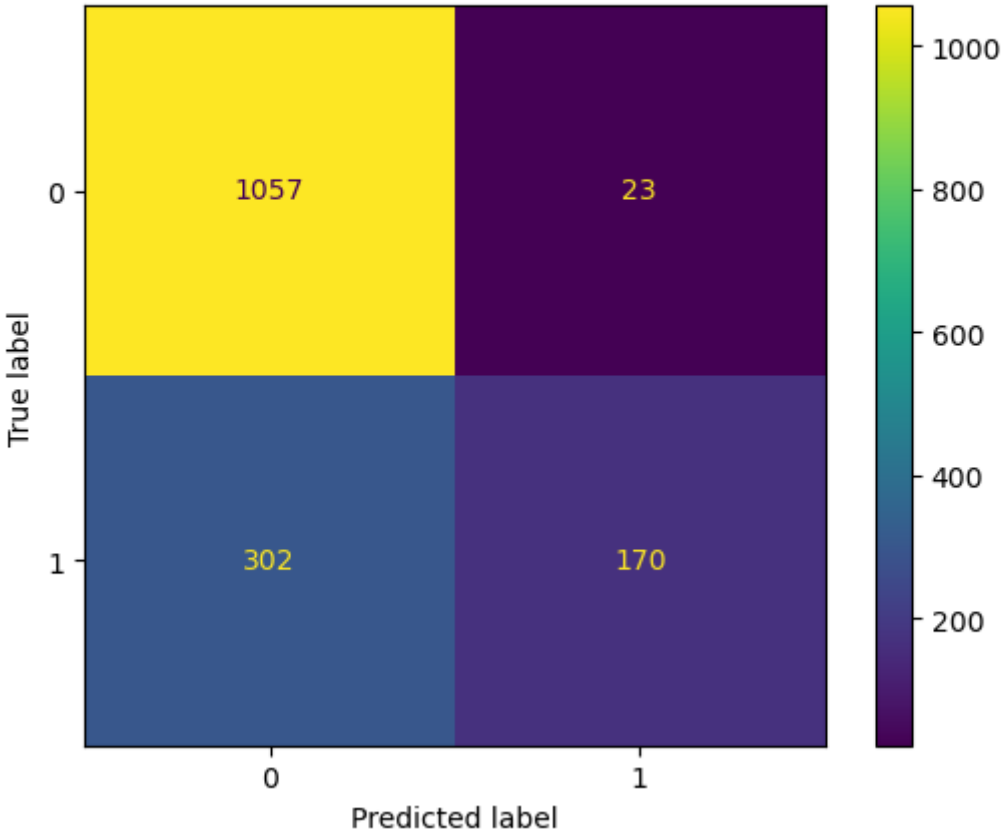
[28]:
```
def perform(y_pred):
    print("Precision : ", precision_score(y_test, y_pred))
    print("Recall : ", recall_score(y_test, y_pred))
    print("Accuracy : ", accuracy_score(y_test, y_pred))
    print("F1 Score : ", f1_score(y_test, y_pred))
    print('')
    print(confusion_matrix(y_test, y_pred), '\n')
    cm = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix(y_test, y_pred))
    cm.plot()
```

[25]:
```
model_SVC = SVC()
model_SVC.fit(X_train, y_train)
y_pred_SVC = model_SVC.predict(X_test)
perform(y_pred_SVC)
```

```
Precision :  0.8808290155440415
Recall :  0.3601694915254237
Accuracy :  0.7905927835051546
F1 Score :  0.5112781954887218

[[1057   23]
 [ 302  170]]
```

```
[[1057   23]
 [ 302  170]]
```

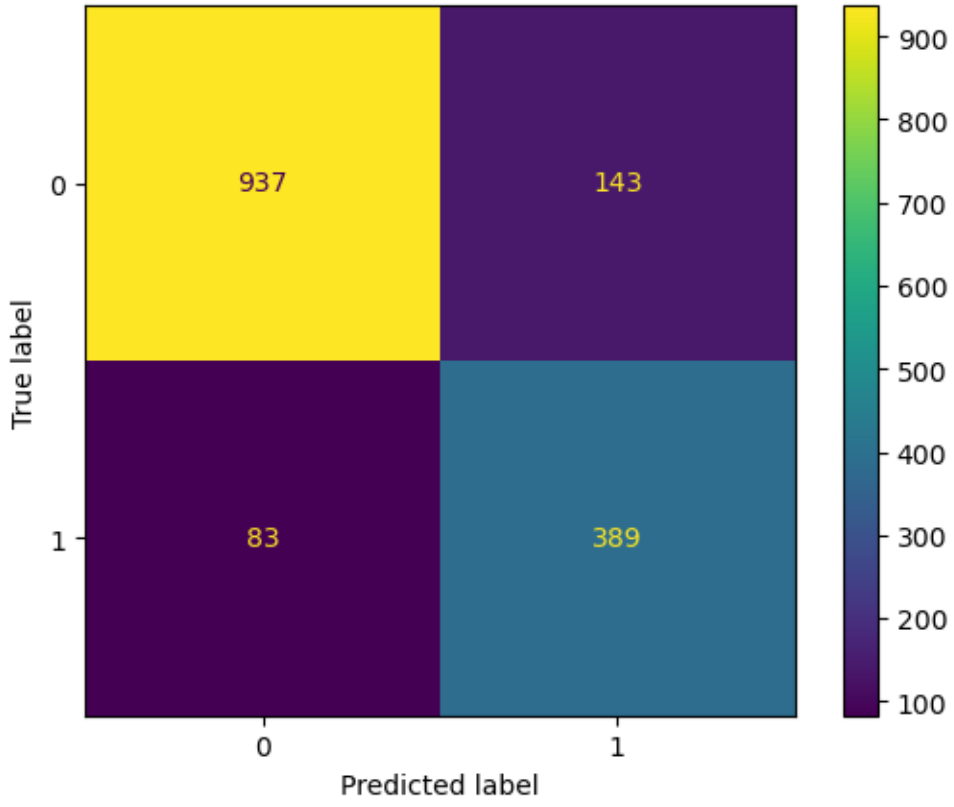


[26]:
```
knn_model=KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train,y_train)
y_pred_knn = knn_model.predict(X_test)
perform(y_pred_knn)
```

```
Precision :  0.731203007518797
Recall :  0.8241525423728814
Accuracy :  0.854381443298969
F1 Score :  0.7749003984063745

[[937 143]
 [ 83 389]]
```

```
 [ 83 389]]
```


937
143
83
389
0
1
True label
0
1
Predicted label
900
800
700
600
500
400
300
200
100