In [1]:
```
# Classify the email using the binary classification method. Email Spam detection has tw
# a) Normal State - Not Spam
# b) Abnormal State - Spam.
# Use K-Nearest Neighbors for classification.
# Analyze their performance.
# Dataset link:
# https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv
```

In [2]:
```
import pandas as pd
```

In [3]:
```
df=pd.read_csv('emails.csv')
```

In [4]:
```
df.shape
```

Out[4]:
```
(5172, 3002)
```

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

Out[5]:

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

5 rows × 3002 columns

In [6]:
```
#input data output data sepreate
```

In [7]:
```
#INPUT
x=df.drop(['Email No.','Prediction'],axis=1)
```

In [8]:
```
#OUTPUT
y=df['Prediction']
```

In [9]:
```
#two col deteted
x.shape
```

Out[9]:
```
(5172, 3000)
```

In [10]:
```
#type of data
x.dtypes
```

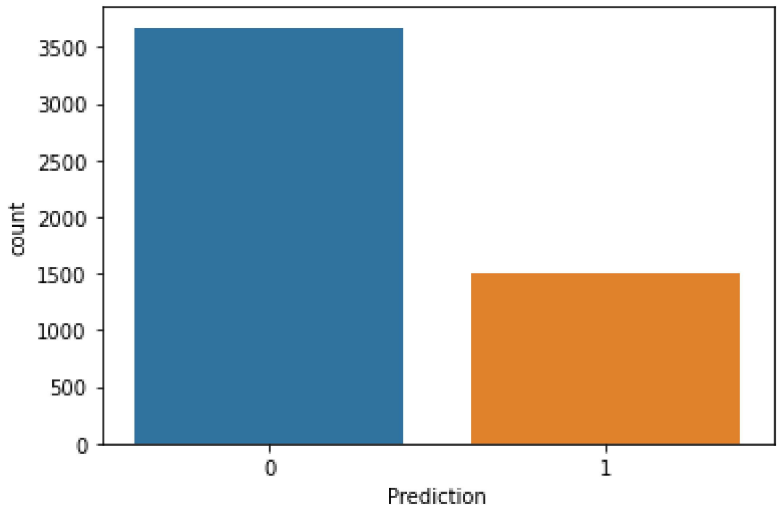
Out[10]:
```
the               int64
to                int64
ect               int64
and               int64
for               int64
                  ...
infrastructure    int64
military          int64
allowing          int64
ff                int64
dry               int64
Length: 3000, dtype: object
```

In [11]:
```
set(x.dtypes) #set : confirm that each value is in integer foem
```

Out[11]:
```
{dtype('int64')}
```

In [12]:
```
#checking output vaiable: balance of spam and not spam
import seaborn as sns
sns.countplot(x=y)
#not spam 3500 and spam 1500
```

Out[12]:
```
<AxesSubplot:xlabel='Prediction', ylabel='count'>
```



In [13]:
```
#checking output vaiable: balance of spam and not spam using value_count method
y.value_counts()
```

Out[13]:
```
0    3672
1    1500
Name: Prediction, dtype: int64
```

In [14]:
```
 #Feature Scaling---->feature are scaled in one range (0-1)
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
x_scaled=scaler.fit_transform(x)
```

In [15]:
```
x_scaled
```

Out[15]:
```
array([[0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.03809524, 0.09848485, 0.06705539, ..., 0.        , 0.00877193,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.00952381, 0.0530303 , 0.        , ..., 0.        , 0.00877193,
        0.        ],
       [0.1047619 , 0.18181818, 0.01166181, ..., 0.        , 0.        ,
        0.        ]])
```

In [16]:
```
#Cross Validation
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_scaled, y, test_size=0.25, random_
```

In [17]:
```
x_scaled.shape
```

Out[17]:
```
(5172, 3000)
```

In [18]:
```
x_train.shape
```

Out[18]:
```
(3879, 3000)
```

In [19]:
```
x_test.shape
```

Out[19]:
```
(1293, 3000)
```

K-Nearest Neighbors for classification

In [20]:
```
#KNN
#import class
from sklearn.neighbors import KNeighborsClassifier
```

In [21]:
```
#Create the object---num of neighbours 2
knn=KNeighborsClassifier(n_neighbors=1)
```

In [22]:
```
#Train the algorithm
knn.fit(x_train,y_train)
```

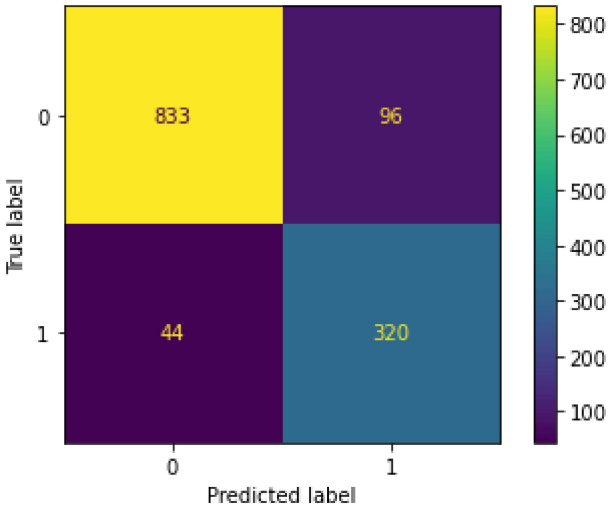
Out[22]:
```
KNeighborsClassifier(n_neighbors=1)
```

In [23]:
```
#Predict on test data
y_pred=knn.predict(x_test)
```

In [24]:
```
#import the evaluation matrix
from sklearn.metrics import ConfusionMatrixDisplay, accuracy_score, classification_repor
```

In [25]:
```
ConfusionMatrixDisplay.from_predictions(y_test,y_pred)
#from 929 -->783 value match & 146 (not spam) but pred as spam
#email 364--> 344 value match
```

Out[25]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2502f967d90>



In [26]:
```
y_test.value_counts()
```

Out[26]:
```
0    929
1    364
Name: Prediction, dtype: int64
```

In [28]:
```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of Logistic Regression: {accuracy}")
```

```
Accuracy of Logistic Regression: 0.8917246713070379
```

In [29]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.90      0.92       929
           1       0.77      0.88      0.82       364

    accuracy                           0.89      1293
   macro avg       0.86      0.89      0.87      1293
weighted avg       0.90      0.89      0.89      1293
```

Support Vector Machine for classification.

In [30]:
```
from sklearn.svm import SVC
```

In [31]:
```
svm = SVC(kernel = 'sigmoid')
#kernel = 'linear'   0.9767981438515081
#kernel = 'rbf'      0.9450889404485692
#kernel = 'poly'     0.7548337200309359
#kernel = 'sigmoid'  0.839907192575406
```

In [32]:
```
svm.fit(x_train,y_train)
```

Out[32]:
```
SVC(kernel='sigmoid')
```

In [33]:
```
y_pred = svm.predict(x_test)
```

In [35]:
```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of Logistic Regression: {accuracy}")
```

```
Accuracy of Logistic Regression: 0.839907192575406
```

In [36]:
```
# Display the classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.90      0.89       929
           1       0.72      0.70      0.71       364

    accuracy                           0.84      1293
   macro avg       0.80      0.80      0.80      1293
weighted avg       0.84      0.84      0.84      1293
```

Logistic Regression for classification.

In [37]:
```
from sklearn.linear_model import LogisticRegression
```

In [38]:
```
# Create a Logistic Regression object
logistic_regression = LogisticRegression()
```

In [39]:
```
# Train the algorithm
logistic_regression.fit(x_train, y_train)
```

Out[39]:
```
LogisticRegression()
```

In [40]:
```
# Predict on the test data
y_pred = logistic_regression.predict(x_test)
```

In [41]:
```
# Evaluate the performance of the Logistic Regression model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of Logistic Regression: {accuracy}")
```

```
Accuracy of Logistic Regression: 0.9667440061871616
```

In [42]:
```
# Display the classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       929
           1       0.97      0.91      0.94       364

    accuracy                           0.97      1293
   macro avg       0.97      0.95      0.96      1293
weighted avg       0.97      0.97      0.97      1293
```