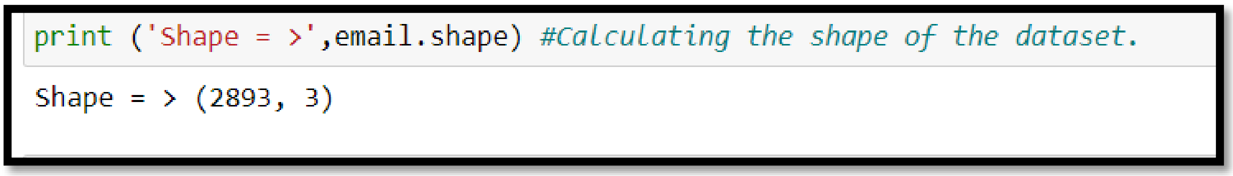
PROJECT-SPAM EMAIL CLASSIFIER

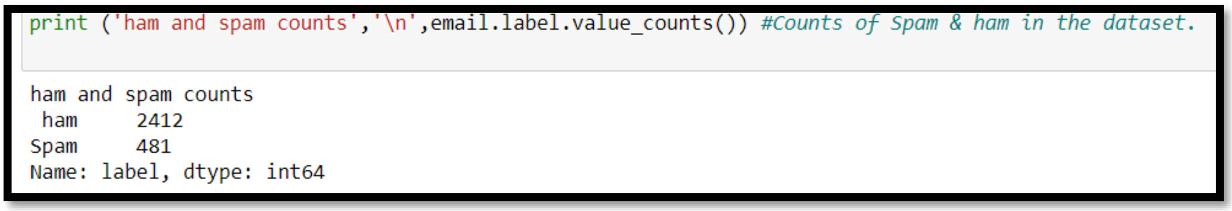
- **Data Preparation:** Spam email classifier is a process wherein an email content/documents is classified as spam/non spam. (also known as ham).Spam level in our Gmail is an perfect example of spam detection wherein emails which are coming externally, in Gmail either coming to Inbox label or going to Spam level depends on the contents/documents received from the sender.
  The Dataset used in this project has 2893 no. of rows and 3 columns.

  In the dataset label is the target variable in which model prediction is done.



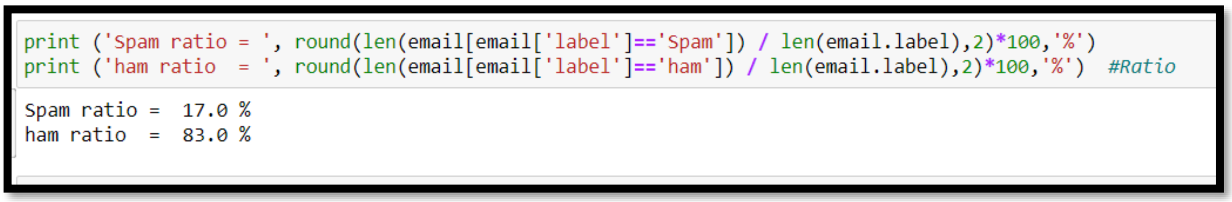
```
print ('Shape = >',email.shape) #Calculating the shape of the dataset.

Shape = > (2893, 3)
```

The shape of the dataset is (2893, 3).



```
print ('ham and spam counts','\n',email.label.value_counts()) #Counts of Spam & ham in the dataset.

ham and spam counts
 ham      2412
Spam      481
Name: label, dtype: int64
```

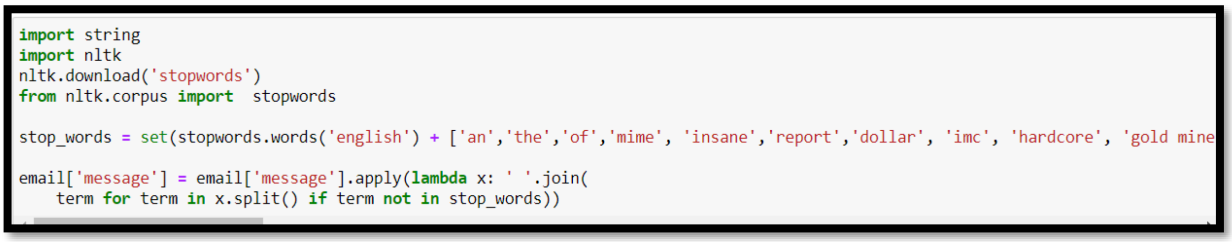
I have classified ‘0’ as ham i.e. non spam email content and ‘1’ as spam. Further I have calculated the total Spam count (481) in the dataset and ham count (2412) along with the data type.



```
print ('Spam ratio = ', round(len(email[email['label']=='Spam']) / len(email.label),2)*100,'%')
print ('ham ratio  = ', round(len(email[email['label']=='ham']) / len(email.label),2)*100,'%')  #Ratio

Spam ratio =  17.0 %
ham ratio  =  83.0 %
```

I have also calculated the percentage of spam/ham label in the dataset to understand the data in a better way.

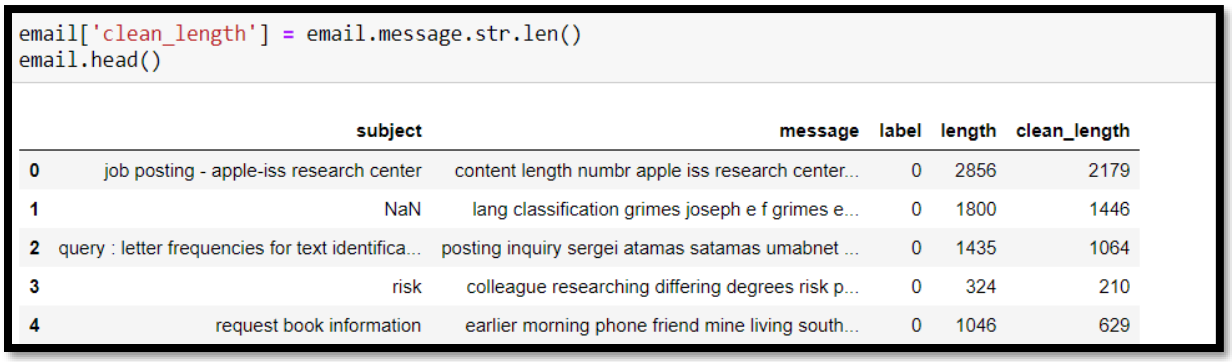
- **Creating Word Dictionary:** In the first step, I have created a dictionary of words and. I have defined a python function creates the dictionary for you.
  Once the dictionary is created I have added a few lines of code written below to the above function to remove stop-words after importing the NLTK library.



```python
import string
import nltk
nltk.download('stopwords')
from nltk.corpus import  stopwords

stop_words = set(stopwords.words('english') + ['an','the','of','mime', 'insane','report','dollar', 'imc', 'hardcore', 'gold mine

email['message'] = email['message'].apply(lambda x: ' '.join(
    term for term in x.split() if term not in stop_words))
```

Removal of stop words – Stop words are like “and”, “the”, “of” are vastly common in all English sentences and are not much significant in determining spam or ha, status, so these words have been filtered out from the emails.

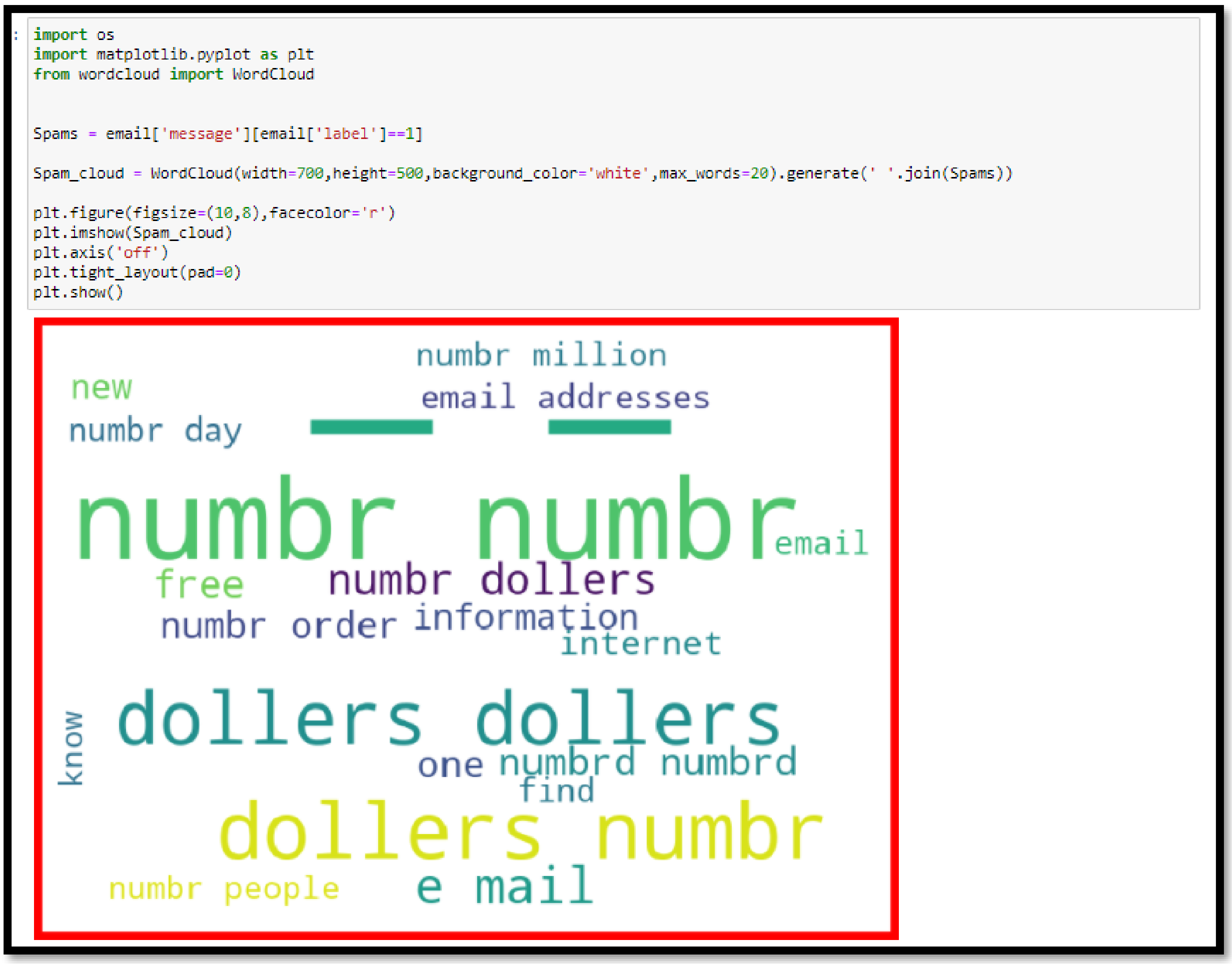
Feature extraction: For the feature extraction process I have removed the punctuation, whitespace in the email message. Then I have find out the actual length of message, before I have converted the string earlier to lower case.



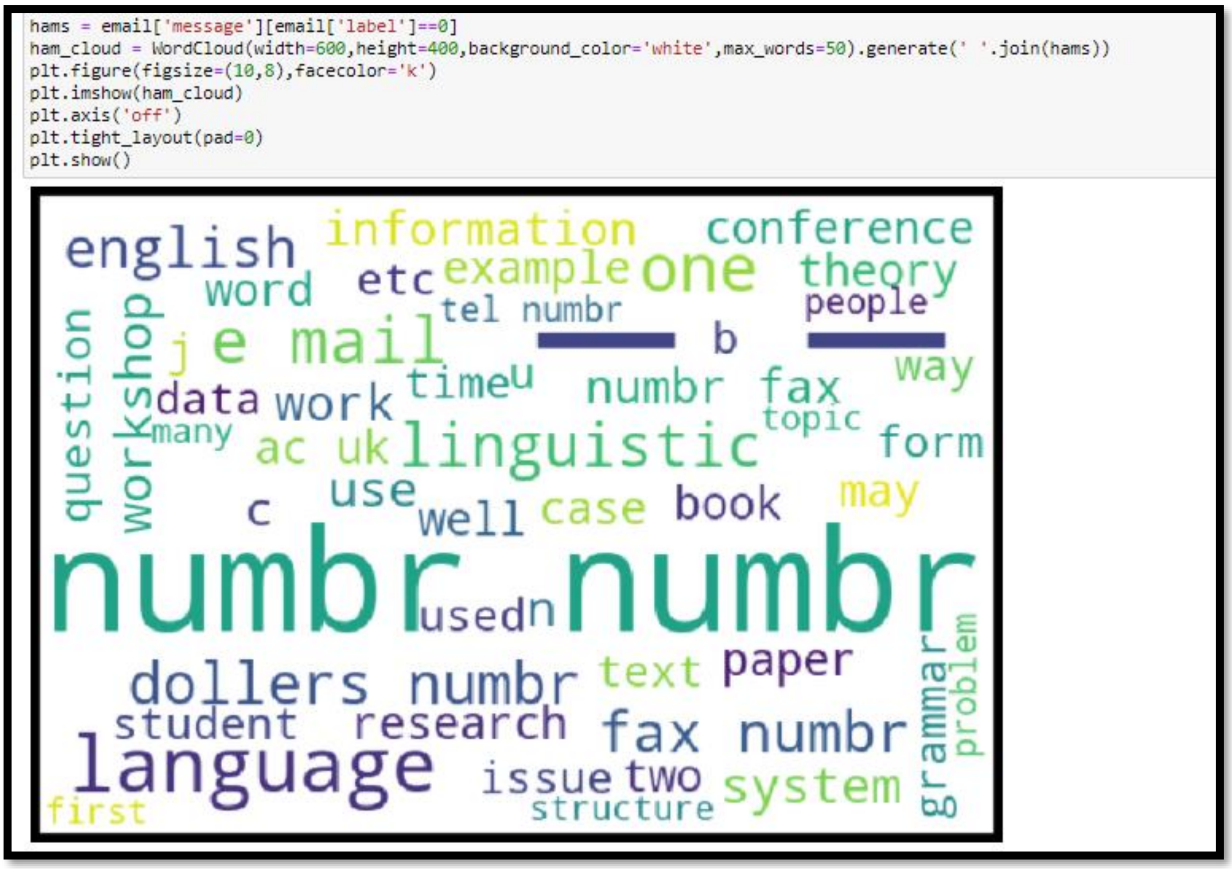
```python
email['clean_length'] = email.message.str.len()
email.head()
```

	subject	message	label	length	clean_length
0	job posting - apple-iss research center	content length numbr apple iss research center...	0	2856	2179
1	NaN	lang classification grimes joseph e f grimes e...	0	1800	1446
2	query : letter frequencies for text identifica...	posting inquiry sergei atamas satamas umabnet ...	0	1435	1064
3	risk	colleague researching differing degrees risk p...	0	324	210
4	request book information	earlier morning phone friend mine living south...	0	1046	629

Also I have found out the most frequent words using Word cloud libraries in both Spam & non spam email dataset by using matplotlib libraries visualization tool.



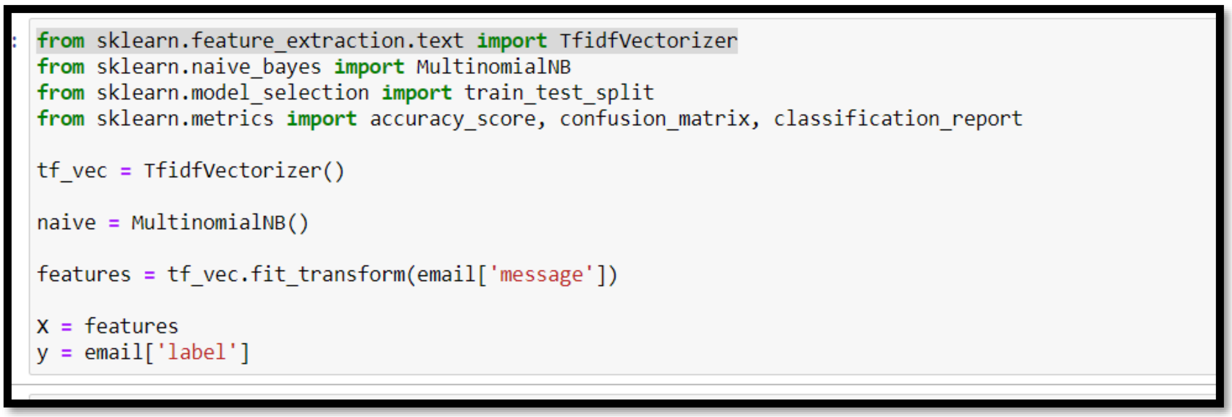
```python
import os
import matplotlib.pyplot as plt
from wordcloud import WordCloud


Spams = email['message'][email['label']==1]

Spam_cloud = WordCloud(width=700,height=500,background_color='white',max_words=20).generate(' '.join(Spams))

plt.figure(figsize=(10,8),facecolor='r')
plt.imshow(Spam_cloud)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```



```
hams = email['message'][email['label']==0]
ham_cloud = WordCloud(width=600,height=400,background_color='white',max_words=50).generate(' '.join(hams))
plt.figure(figsize=(10,8),facecolor='k')
plt.imshow(ham_cloud)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```

- **Training classifiers:** For this project to train the dataset I have used various scikit-learn libraries I have Naive Bayes classifier model which is mainly an accepted method for document classification model. It is a supervised probabilistic classifier based on Bayes theorem assuming independence between every pair of features in the dataset.
  I have used **train_test_classfier** to split the training and testing dataset using model prediction.

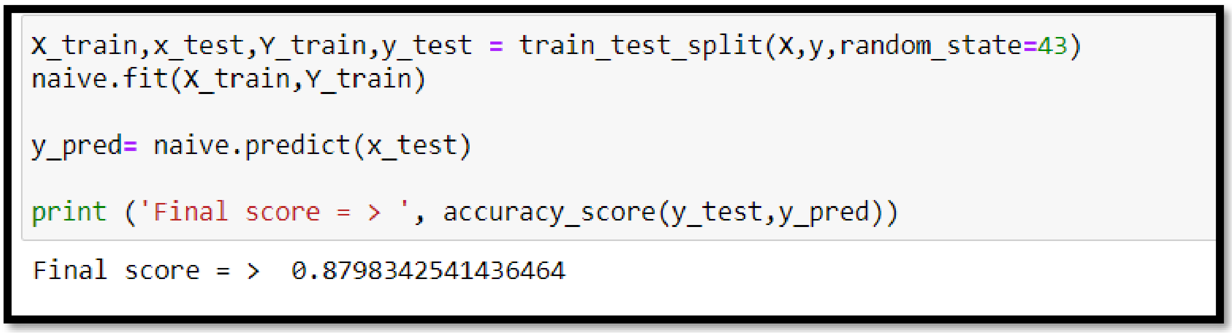
  Also, for each word in our dataset, I have used a measure called Term Frequency, Inverse Document Frequency, abbreviated to tf-idf. I have used from **sklearn.feature_extraction.text import TfidfVectorizer** this is used for text classification algorithm and also Convert a collection of email documents to a matrix of TF-IDF features.



```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

tf_vec = TfidfVectorizer()

naive = MultinomialNB()

features = tf_vec.fit_transform(email['message'])

X = features
y = email['label']
```

- **Testing:** Once the classifiers are trained, I have check the performance of the models on test-set with the trained NB classifier model and predicted the accuracy score.

  We got an accuracy score of 87.98% for this classification model.

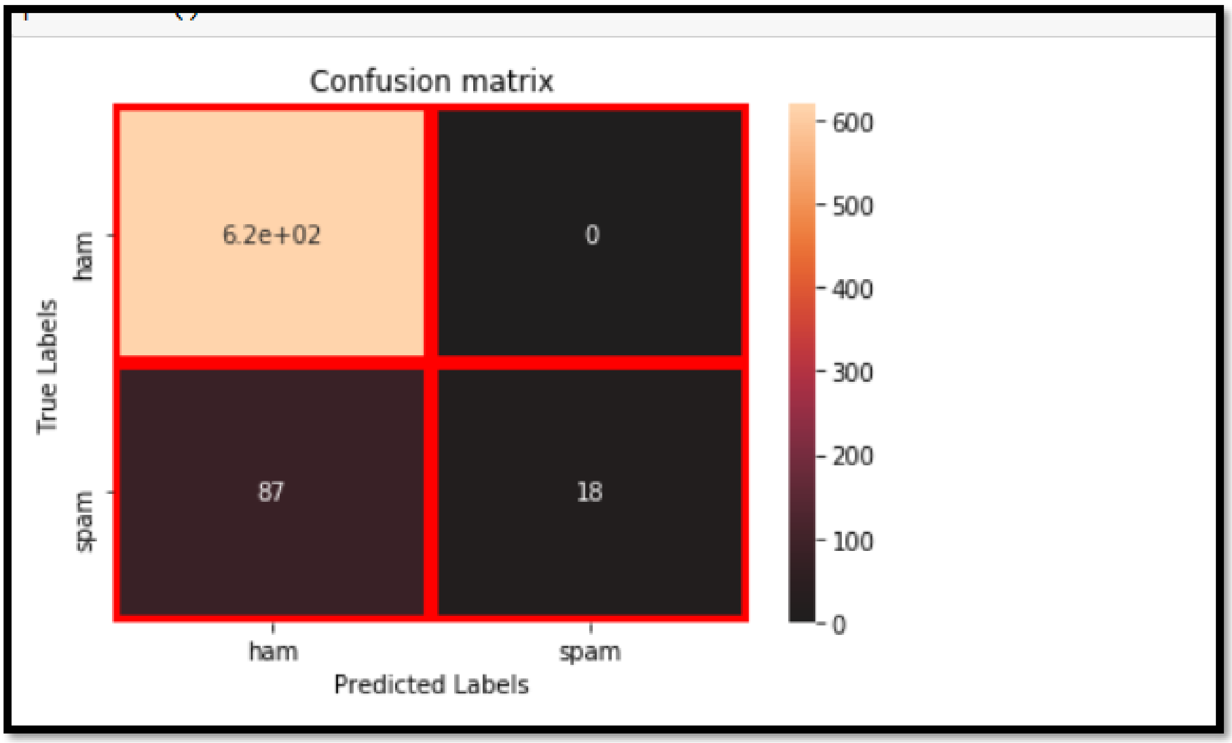


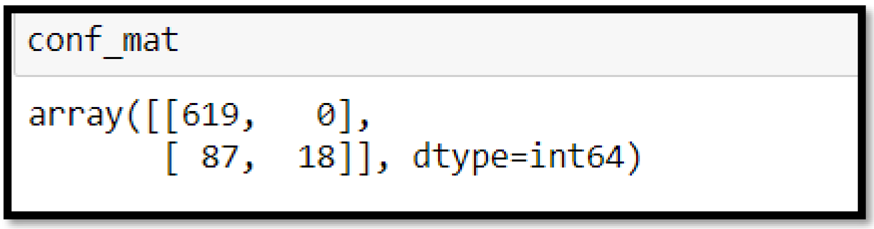
```
X_train,x_test,Y_train,y_test = train_test_split(X,y,random_state=43)
naive.fit(X_train,Y_train)

y_pred= naive.predict(x_test)

print ('Final score = > ', accuracy_score(y_test,y_pred))
```

```
Final score = >  0.8798342541436464
```

- **Performance evaluation using multiple metrics:** For the performance evaluation of this model I have used confusion matrix and classification report.

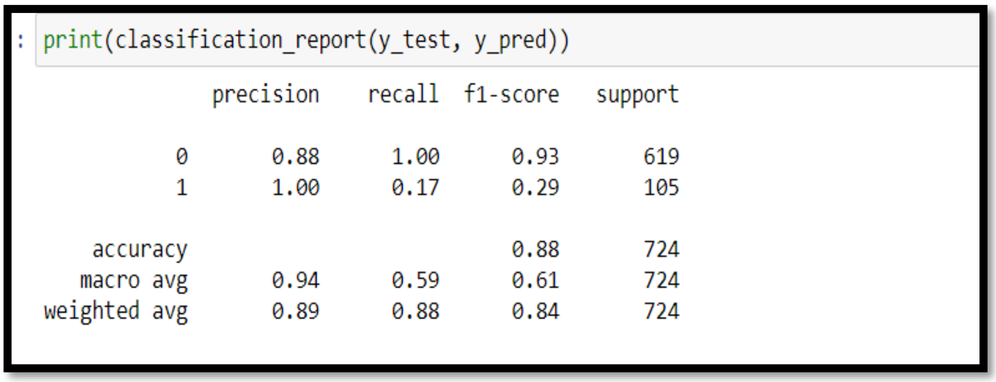
Confusion matrix	
TP(True Positive)	FP(False Positive)
FN(False Positive)	TN(True Negative)





```
conf_mat

array([[619,   0],
       [ 87,  18]], dtype=int64)
```

Validation number for Classification model	
Accuracy	=(TP+TN)/TOTAL
Precision	=TP/(TP+FP)
Recall	=TP/(TP+FN)



```
: print(classification_report(y_test, y_pred))

              precision    recall  f1-score   support

           0       0.88      1.00      0.93       619
           1       1.00      0.17      0.29       105

    accuracy                           0.88       724
   macro avg       0.94      0.59      0.61       724
weighted avg       0.89      0.88      0.84       724
```