In [1]:

```
# This Python 3 environment comes with many helpful analytics librari
# It is defined by the kaggle/python Docker image: https://github.com
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv

# Input data files are available in the read-only "../input/" directo
# For example, running this (by clicking run or pressing Shift+Enter,

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/
# You can also write temporary files to /kaggle/temp/, but they won't
```

In [2]:

```
# importing required libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [4]:

```
# loading the dataset into pandas dataframe
df = pd.read_csv("/home/s6ad2/Downloads/archive/1.csv")
```

In [5]:

```
df
```

Out[5]:

```
         step      type      amount     nameOrig  oldbalanceOrg  newbalanceOrig    nar
      0     1   PAYMENT     9839.64  C1231006815      170136.00       160296.36  M1979
      1     1   PAYMENT     1864.28  C1666544295       21249.00        19384.72  M2044
      2     1  TRANSFER      181.00  C1305486145         181.00            0.00   C553
      3     1  CASH_OUT      181.00   C840083671         181.00            0.00    C38
      4     1   PAYMENT    11668.14  C2048537720       41554.00        29885.86  M1230
    ...   ...       ...         ...          ...            ...             ...
6362615   743  CASH_OUT   339682.13   C786484425      339682.13            0.00   C776
6362616   743  TRANSFER  6311409.28  C1529008245     6311409.28            0.00  C1881
6362617   743  CASH_OUT  6311409.28  C1162922333     6311409.28            0.00  C1365
6362618   743  TRANSFER   850002.52  C1685995037      850002.52            0.00  C2080
6362619   743  CASH_OUT   850002.52  C1280323807      850002.52            0.00   C873

6362620 rows × 11 columns
```

In [6]:

```
# Retain the 6 features and the target variable
df = df[['amount','oldbalanceOrg','newbalanceOrig','oldbalanceDest',
```

In [7]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 7 columns):
amount            float64
oldbalanceOrg     float64
newbalanceOrig    float64
oldbalanceDest    float64
newbalanceDest    float64
isFraud           int64
isFlaggedFraud    int64
dtypes: float64(5), int64(2)
memory usage: 339.8 MB
```

In [8]:

```
df.isnull().sum()
```

Out[8]:

```
amount            0
oldbalanceOrg     0
newbalanceOrig    0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
isFlaggedFraud    0
dtype: int64
```

In [9]:

```
df['isFraud'].value_counts()
```

Out[9]:

```
0    6354407
1       8213
Name: isFraud, dtype: int64
```

In [10]:

```
# Load the features to a variable X
# X is created by simply dropping the diagnosis column and retaining
X = df.drop('isFraud',axis=1)


#Load the target variable to y
y = df['isFraud']
```

In [11]:

```
# Do the train/test split

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.30,
```

In [12]:

```
# Train the Logistic Regression Model
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression (solver='liblinear')
classifier.fit(X_train,y_train)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/svm/_base.py:1225: C
onvergenceWarning: Liblinear failed to converge, increase the numbe
r of iterations.
  warnings.warn(
```

Out[12]:

```
▾          LogisticRegression

LogisticRegression(solver='liblinear')
```

In [13]:

```
# Prediction with the test set
y_predict = classifier.predict(X_test)
```

In [14]:

```
Results = pd.DataFrame({'A':y_test,'P':y_predict})
Results.head(10)
```

Out[14]:

	A	P
6322570	0	0
3621196	0	0
1226256	0	0
2803274	0	0
3201247	0	0
3681019	0	0
1351584	0	0
5422829	0	0
5870912	0	0
2400263	0	0

In [15]:

```
# Compute Model Accuracy.

from sklearn.metrics import accuracy_score
print(accuracy_score(y_test,y_predict))
```

```
0.9992524044078278
```

In [16]:

```
# Model accuracy on training set.

# Get the predictions from the model for the training set.

y_train_pred = classifier.predict(X_train)

print(accuracy_score(y_train,y_train_pred))
```

```
0.9992803952729267
```

In [17]:

```
# Getting probability predictions from the model.
y_test_proba = classifier.predict_proba(X_test)

print(y_test_proba.shape)
```

```
(1908786, 2)
```

In [18]:

```
y_test_proba[0:5,:]
```

Out[18]:

```
array([[9.90914469e-01, 9.08553137e-03],
       [5.78209055e-01, 4.21790945e-01],
       [6.83086498e-01, 3.16913502e-01],
       [9.99999508e-01, 4.91585075e-07],
       [9.99693434e-01, 3.06566184e-04]])
```

In [19]:

```
import numpy as np

# Given array in scientific notation
array_in_scientific = y_test_proba[0:5,:]

# Convert to normal number format
array_in_normal = np.vectorize(lambda x: format(x, '.16f'))(array_in_

print(array_in_normal)
```

```
[['0.9909144686259501' '0.0090855313740498']
 ['0.5782090546663730' '0.4217909453336271']
 ['0.6830864978197835' '0.3169135021802165']
 ['0.9999995084149252' '0.0000004915850747']
 ['0.9996934338164003' '0.0003065661835997']]
```

In [20]:

```
T = y_test_proba[:,1]
array_in_one = T

# Convert to normal number format
S = np.vectorize(lambda x: format(x, '.16f'))(array_in_one)

print(S)
```

```
['0.0090855313740498' '0.4217909453336271' '0.3169135021802165' ...
 '0.0000000000000000' '0.0000000000000000' '0.3897303181872133']
```

In [21]:

```
Results = pd.DataFrame({'Actual':y_test,'Predictions':y_predict,'Prob

Results.head(5)
```

Out[21]:

	Actual	Predictions	Prob(Class = 1)
6322570	0	0	0.0090855313740498
3621196	0	0	0.4217909453336271
1226256	0	0	0.3169135021802165
2803274	0	0	0.0000004915850747
3201247	0	0	0.0003065661835997

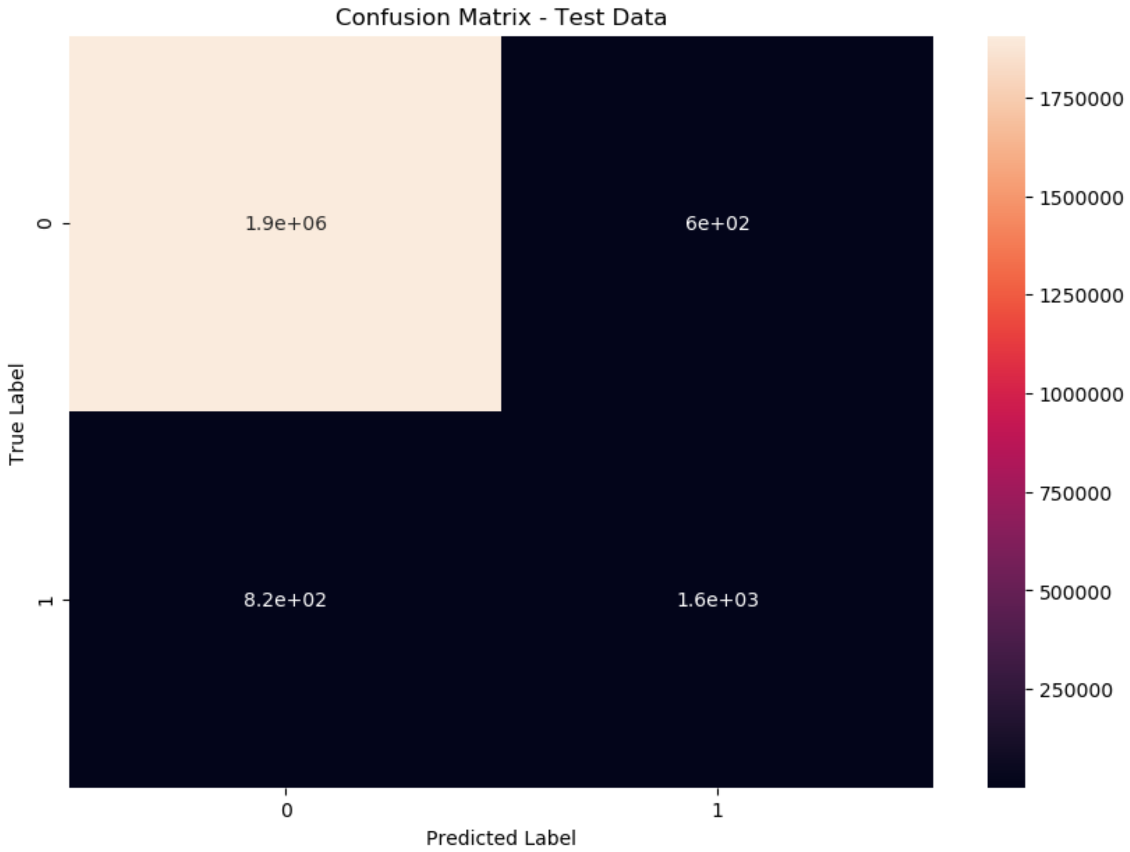
In [22]:

```
#Generate the Confusion Matrix

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test,y_predict)
print(cm)
```

```
[[1905716     602]
 [    825    1643]]
```

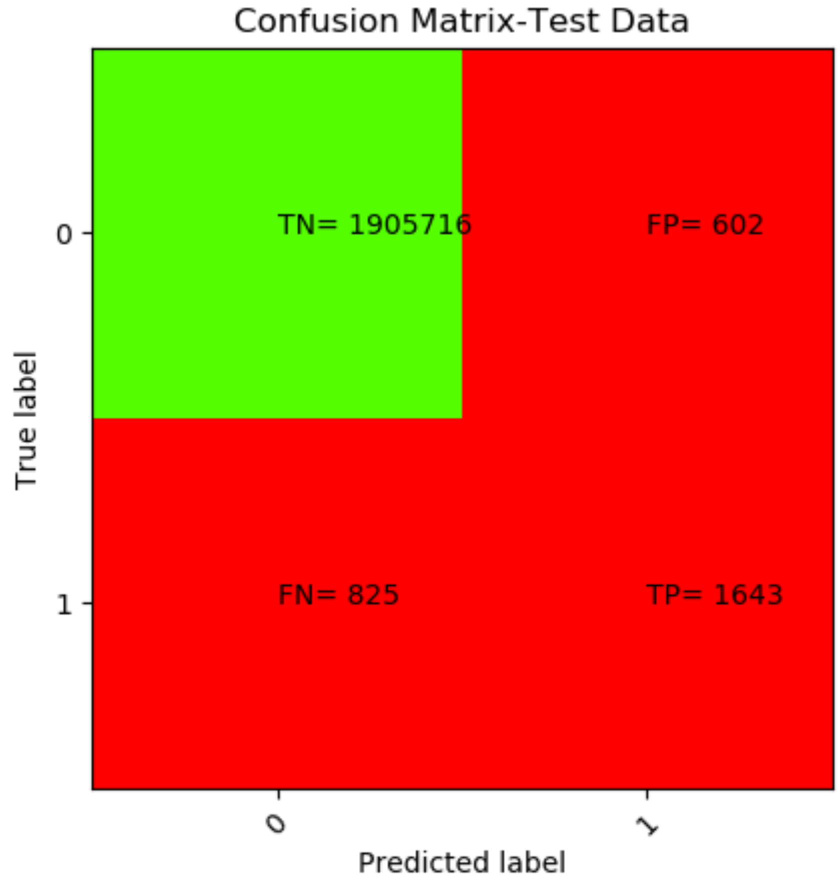
In [23]:
```
import seaborn as sn
plt.figure(figsize = (10,7))
sn.heatmap(cm, annot=True)
plt.title('Confusion Matrix - Test Data')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
```

Out[23]:
```
Text(95.72222222222221, 0.5, 'True Label')
```



In [24]:

```
plt.clf()
plt.imshow(cm,interpolation='nearest',cmap=plt.cm.prism)
classNames = ['0','1']
plt.title('Confusion Matrix-Test Data')
plt.ylabel('True label')
plt.xlabel('Predicted label')
tick_marks = np.arange(2)
plt.xticks(tick_marks,classNames,rotation=45)
plt.yticks(tick_marks,classNames)
s = [['TN','FP'],['FN','TP']]
for i in range(2):
    for j in range(2):
        plt.text(j,i,str(s[i][j])+"= "+str(cm[i][j]))
plt.show()
```



In [25]:

```
#Calculate common error metrics for a 2-class classifier

from sklearn.metrics import classification_report
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   1906318
           1       0.73      0.67      0.70      2468

    accuracy                           1.00   1908786
   macro avg       0.87      0.83      0.85   1908786
weighted avg       1.00      1.00      1.00   1908786
```

In [26]:

```
# Calculate metrics values individually

# Assigning Variables for convinience

TN = cm[0][0]
FP = cm[0][1]
FN = cm[1][0]
TP = cm[1][1]
```

In [27]:

```
recall = TP / (TP + FN )
print("Recall= ",recall)
```

```
Recall=  0.6657212317666127
```

In [28]:

```
precision = TP / (TP + FP)
print("Precision=",precision)
```

```
Precision= 0.7318485523385301
```

In [29]:

```
specificity = TN /  (TN + FP)
print("Specificity = ", specificity)
```

```
Specificity =  0.9996842079862857
```

In [30]:

```
accuracy = ( TP + TN ) / ( TP + TN + FP + FN)
print("Accuracy =" , accuracy)
```

```
Accuracy = 0.9992524044078278
```

In [ ]: