Day 21/100 of Data Science

Naïve Bayes Classifier Algorithm

- The Naive Bayes Classifier algorithm is a popular and simple supervised learning algorithm for classification tasks.
- It works based on Bayes' theorem to calculate probabilities and predict the class an unseen data point belongs to.
- Naive Bayes calculates the probability of a data point being in each class based on the probabilities of its individual features.

Why is it called Naïve Bayes?

- Naïve: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features.
- Bayes: It is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes' Theorem

- Bayes' theorem is also known as Bayes' Rule or Bayes' law, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as: $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$

Where,

- P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.
- P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.
- P(A) is Prior Probability: Probability of hypothesis before observing the evidence.
- P(B) is Marginal Probability: Probability of Evidence.

In [ ]:

Implementation

In [2]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # for data visualization purposes
import seaborn as sns # for statistical data visualization
%matplotlib inline

import warnings

warnings.filterwarnings('ignore')
```

In [11]:
```
data = 'adult.csv'

#df = pd.read_csv(data, header=None, sep=',\s')
df = pd.read_csv(data)
```

In [12]:
```
df.head()
```

Out[12]:

	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	U
0	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	U
1	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	U
2	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	U
3	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	
4	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	0	40	U

In [13]:
```
# view dimensions of dataset

df.shape
```

Out[13]:
```
(32560, 15)
```

In [14]:
```
# preview the dataset

df.head()
```

Out[14]:

	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	U
0	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	U
1	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	U
2	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	U
3	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	
4	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	0	40	U

In [16]:

```
#Rename column names
col_names = ['age', 'workclass', 'fnlwgt', 'education', 'education_num', 'marital_stat
             'race', 'sex', 'capital_gain', 'capital_loss', 'hours_per_week', 'native_

df.columns = col_names

df.columns
```

Out[16]:

```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education_num',
       'marital_status', 'occupation', 'relationship', 'race', 'sex',
       'capital_gain', 'capital_loss', 'hours_per_week', 'native_country',
       'income'],
      dtype='object')
```

In [17]:

```
# let's again preview the dataset

df.head()
```

Out[17]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race
0	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White
1	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White
2	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black
3	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black
4	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White

In [18]:
```
# view summary of dataset

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32560 entries, 0 to 32559
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             32560 non-null  int64 
 1   workclass       32560 non-null  object
 2   fnlwgt          32560 non-null  int64 
 3   education       32560 non-null  object
 4   education_num   32560 non-null  int64 
 5   marital_status  32560 non-null  object
 6   occupation      32560 non-null  object
 7   relationship    32560 non-null  object
 8   race            32560 non-null  object
 9   sex             32560 non-null  object
 10  capital_gain    32560 non-null  int64 
 11  capital_loss    32560 non-null  int64 
 12  hours_per_week  32560 non-null  int64 
 13  native_country  32560 non-null  object
 14  income          32560 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

In [19]:
```
# find categorical variables

categorical = [var for var in df.columns if df[var].dtype=='O']

print('There are {} categorical variables\n'.format(len(categorical)))

print('The categorical variables are :\n\n', categorical)
```

```
There are 9 categorical variables

The categorical variables are :

 ['workclass', 'education', 'marital_status', 'occupation', 'relationship', 'race', 'sex', 'native_country', 'income']
```

In [20]:
```
# view the categorical variables

df[categorical].head()
```

Out[20]:

	workclass	education	marital_status	occupation	relationship	race	sex	native_country	incor
0	Self-emp-not-inc	Bachelors	Married-civ-spouse	Exec-managerial	Husband	White	Male	United-States	<=5
1	Private	HS-grad	Divorced	Handlers-cleaners	Not-in-family	White	Male	United-States	<=5
2	Private	11th	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	United-States	<=5
3	Private	Bachelors	Married-civ-spouse	Prof-specialty	Wife	Black	Female	Cuba	<=5
4	Private	Masters	Married-civ-spouse	Exec-managerial	Wife	White	Female	United-States	<=5

In [21]:

```
# check missing values in categorical variables

df[categorical].isnull().sum()
```

Out[21]:

```
workclass         0
education         0
marital_status    0
occupation        0
relationship      0
race              0
sex               0
native_country    0
income            0
dtype: int64
```

In [22]:

```
# view frequency counts of values in categorical variables

for var in categorical:

    print(df[var].value_counts())
```

```
workclass
 Private             22696
 Self-emp-not-inc     2541
 Local-gov            2093
 ?                    1836
 State-gov            1297
 Self-emp-inc         1116
 Federal-gov           960
 Without-pay            14
 Never-worked            7
Name: count, dtype: int64
education
 HS-grad         10501
 Some-college     7291
 Bachelors        5354
 Masters          1723
 Assoc-voc        1382
 11th             1175
 Assoc-acdm       1067
 10th              933
 7th-8th           646
 Prof-school       576
 9th               514
 12th              433
 Doctorate         413
 5th-6th           333
 1st-4th           168
 Preschool          51
Name: count, dtype: int64
marital_status
 Married-civ-spouse       14976
 Never-married            10682
 Divorced                  4443
 Separated                 1025
 Widowed                    993
 Married-spouse-absent      418
 Married-AF-spouse           23
Name: count, dtype: int64
occupation
 Prof-specialty       4140
 Craft-repair         4099
 Exec-managerial      4066
 Adm-clerical         3769
 Sales                3650
 Other-service        3295
 Machine-op-inspct    2002
 ?                    1843
 Transport-moving     1597
 Handlers-cleaners    1370
 Farming-fishing       994
 Tech-support          928
 Protective-serv       649
 Priv-house-serv       149
 Armed-Forces            9
Name: count, dtype: int64
relationship
 Husband           13193
 Not-in-family      8304
 Own-child          5068
 Unmarried          3446
```

```
 Wife               1568
 Other-relative      981
Name: count, dtype: int64
race
 White                 27815
 Black                  3124
 Asian-Pac-Islander     1039
 Amer-Indian-Eskimo      311
 Other                   271
Name: count, dtype: int64
sex
 Male      21789
 Female    10771
Name: count, dtype: int64
native_country
 United-States                 29169
 Mexico                          643
 ?                               583
 Philippines                     198
 Germany                         137
 Canada                          121
 Puerto-Rico                     114
 El-Salvador                     106
 India                           100
 Cuba                             95
 England                          90
 Jamaica                          81
 South                            80
 China                            75
 Italy                            73
 Dominican-Republic               70
 Vietnam                          67
 Guatemala                        64
 Japan                            62
 Poland                           60
 Columbia                         59
 Taiwan                           51
 Haiti                            44
 Iran                             43
 Portugal                         37
 Nicaragua                        34
 Peru                             31
 France                           29
 Greece                           29
 Ecuador                          28
 Ireland                          24
 Hong                             20
 Cambodia                         19
 Trinadad&Tobago                  19
 Laos                             18
 Thailand                         18
 Yugoslavia                       16
 Outlying-US(Guam-USVI-etc)       14
 Honduras                         13
 Hungary                          13
 Scotland                         12
 Holand-Netherlands                1
Name: count, dtype: int64
income
 <=50K    24719
```

```
 >50K      7841
Name: count, dtype: int64
```

In [23]:
```
# check labels in workclass variable

df.workclass.unique()
```

Out[23]:
```
array([' Self-emp-not-inc', ' Private', ' State-gov', ' Federal-gov',
       ' Local-gov', ' ?', ' Self-emp-inc', ' Without-pay',
       ' Never-worked'], dtype=object)
```

In [29]:
```
# check frequency distribution of values in workclass variable

df.workclass.value_counts()
```

Out[29]:
```
workclass
 Private             22696
 Self-emp-not-inc     2541
 Local-gov            2093
 ?                    1836
 State-gov            1297
 Self-emp-inc         1116
 Federal-gov           960
 Without-pay            14
 Never-worked            7
Name: count, dtype: int64
```

In [32]:
```
# replace '?' values in workclass variable with `NaN`


df['workclass'].replace(' ?', np.NaN, inplace=True)
```

In [33]:
```
# again check the frequency distribution of values in workclass variable

df.workclass.value_counts()
```

Out[33]:
```
workclass
 Private             22696
 Self-emp-not-inc     2541
 Local-gov            2093
 State-gov            1297
 Self-emp-inc         1116
 Federal-gov           960
 Without-pay            14
 Never-worked            7
Name: count, dtype: int64
```

In [34]:
```
# check labels in occupation variable

df.occupation.unique()
```

Out[34]:
```
array([' Exec-managerial', ' Handlers-cleaners', ' Prof-specialty',
       ' Other-service', ' Adm-clerical', ' Sales', ' Craft-repair',
       ' Transport-moving', ' Farming-fishing', ' Machine-op-inspct',
       ' Tech-support', ' ?', ' Protective-serv', ' Armed-Forces',
       ' Priv-house-serv'], dtype=object)
```

In [35]:
```
# check frequency distribution of values in occupation variable

df.occupation.value_counts()
```

Out[35]:
```
occupation
 Prof-specialty       4140
 Craft-repair         4099
 Exec-managerial      4066
 Adm-clerical         3769
 Sales                3650
 Other-service        3295
 Machine-op-inspct    2002
 ?                    1843
 Transport-moving     1597
 Handlers-cleaners    1370
 Farming-fishing       994
 Tech-support          928
 Protective-serv       649
 Priv-house-serv       149
 Armed-Forces            9
Name: count, dtype: int64
```

In [36]:
```python
# replace '?' values in occupation variable with `NaN`

df['occupation'].replace(' ?', np.NaN, inplace=True)
```

In [37]:
```python
# again check the frequency distribution of values in occupation variable

df.occupation.value_counts()
```

Out[37]:
```
occupation
 Prof-specialty       4140
 Craft-repair         4099
 Exec-managerial      4066
 Adm-clerical         3769
 Sales                3650
 Other-service        3295
 Machine-op-inspct    2002
 Transport-moving     1597
 Handlers-cleaners    1370
 Farming-fishing       994
 Tech-support          928
 Protective-serv       649
 Priv-house-serv       149
 Armed-Forces            9
Name: count, dtype: int64
```

In [38]:
```python
X = df.drop(['income'], axis=1)

y = df['income']
```

In [39]:
```python
# split X and y into training and testing sets

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_stat
```

In [40]:
```python
# check the shape of X_train and X_test

X_train.shape, X_test.shape
```

Out[40]:
```
((22792, 14), (9768, 14))
```

In [41]:
```
# display categorical variables

categorical = [col for col in X_train.columns if X_train[col].dtypes == 'O']

categorical
```

Out[41]:
```
['workclass',
 'education',
 'marital_status',
 'occupation',
 'relationship',
 'race',
 'sex',
 'native_country']
```

In [42]:
```
# display numerical variables

numerical = [col for col in X_train.columns if X_train[col].dtypes != 'O']

numerical
```

Out[42]:
```
['age',
 'fnlwgt',
 'education_num',
 'capital_gain',
 'capital_loss',
 'hours_per_week']
```

In [43]:
```
# print percentage of missing values in the categorical variables in training set

X_train[categorical].isnull().mean()
```

Out[43]:
```
workclass         0.057213
education         0.000000
marital_status    0.000000
occupation        0.057389
relationship      0.000000
race              0.000000
sex               0.000000
native_country    0.000000
dtype: float64
```

In [44]:
```
# print categorical variables with missing data

for col in categorical:
    if X_train[col].isnull().mean()>0:
        print(col, (X_train[col].isnull().mean()))
```

```
workclass 0.057213057213057215
occupation 0.05738855738855739
```

In [45]:
```
# impute missing categorical variables with most frequent value

for df2 in [X_train, X_test]:
    df2['workclass'].fillna(X_train['workclass'].mode()[0], inplace=True)
    df2['occupation'].fillna(X_train['occupation'].mode()[0], inplace=True)
    df2['native_country'].fillna(X_train['native_country'].mode()[0], inplace=True)
```

In [46]:
```
# check missing values in categorical variables in X_train
```

```
X_train[categorical].isnull().sum()
```

Out[46]:

```
workclass         0
education         0
marital_status    0
occupation        0
relationship      0
race              0
sex               0
native_country    0
dtype: int64
```

In [47]:

```
# check missing values in categorical variables in X_test

X_test[categorical].isnull().sum()
```

Out[47]:

```
workclass         0
education         0
marital_status    0
occupation        0
relationship      0
race              0
sex               0
native_country    0
dtype: int64
```

In [48]:

```
# check missing values in X_train

X_train.isnull().sum()
```

Out[48]:

```
age               0
workclass         0
fnlwgt            0
education         0
education_num     0
marital_status    0
occupation        0
relationship      0
race              0
sex               0
capital_gain      0
capital_loss      0
hours_per_week    0
native_country    0
dtype: int64
```

In [49]:

```
# check missing values in X_test

X_test.isnull().sum()
```

Out[49]:
```
age               0
workclass         0
fnlwgt            0
education         0
education_num     0
marital_status    0
occupation        0
relationship      0
race              0
sex               0
capital_gain      0
capital_loss      0
hours_per_week    0
native_country    0
dtype: int64
```

In [50]:
```
# print categorical variables

categorical
```

Out[50]:
```
['workclass',
 'education',
 'marital_status',
 'occupation',
 'relationship',
 'race',
 'sex',
 'native_country']
```

In [51]:
```
X_train[categorical].head()
```

Out[51]:

	workclass	education	marital_status	occupation	relationship	race	sex	native_country
20721	Self-emp-inc	Some-college	Married-civ-spouse	Transport-moving	Husband	Black	Male	Haiti
32097	Private	HS-grad	Married-civ-spouse	Craft-repair	Husband	White	Male	United-States
25205	State-gov	HS-grad	Divorced	Adm-clerical	Unmarried	White	Female	United-States
23491	Private	Bachelors	Never-married	Farming-fishing	Not-in-family	White	Male	United-States
12367	Private	Some-college	Never-married	Adm-clerical	Own-child	White	Male	India

In [52]:
```
# import category encoders

import category_encoders as ce
```

In [53]:
```
# encode remaining variables with one-hot encoding

encoder = ce.OneHotEncoder(cols=['workclass', 'education', 'marital_status', 'occupati
                                 'race', 'sex', 'native_country'])

X_train = encoder.fit_transform(X_train)
```

```
X_test = encoder.transform(X_test)
```

In [54]:
```
X_train.head()
```

Out[54]:

	age	workclass_1	workclass_2	workclass_3	workclass_4	workclass_5	workclass_6	workclass_7
20721	32	1	0	0	0	0	0	0
32097	45	0	1	0	0	0	0	0
25205	47	0	0	1	0	0	0	0
23491	37	0	1	0	0	0	0	0
12367	24	0	1	0	0	0	0	0

5 rows × 106 columns

In [55]:
```
X_train.shape
```

Out[55]:
```
(22792, 106)
```

In [56]:
```
X_test.head()
```

Out[56]:

	age	workclass_1	workclass_2	workclass_3	workclass_4	workclass_5	workclass_6	workclass_7
22278	40	1	0	0	0	0	0	0
8950	46	0	1	0	0	0	0	0
7838	33	0	1	0	0	0	0	0
16505	21	0	1	0	0	0	0	0
19140	59	0	1	0	0	0	0	0

5 rows × 106 columns

In [57]:
```
X_test.shape
```

Out[57]:
```
(9768, 106)
```

In [58]:
```
cols = X_train.columns
```

In [59]:
```
from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

In [60]:
```
X_train = pd.DataFrame(X_train, columns=[cols])
```

In [61]:
```
X_test = pd.DataFrame(X_test, columns=[cols])
```

In [62]:
```
X_train.head()
```

Out[62]:

	age	workclass_1	workclass_2	workclass_3	workclass_4	workclass_5	workclass_6	workclass_7	w
0	-0.25	1.0	-1.0	0.0	0.0	0.0	0.0	0.0	
1	0.40	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.50	0.0	-1.0	1.0	0.0	0.0	0.0	0.0	
3	0.00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	-0.65	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

5 rows × 106 columns

In [63]:
```
# train a Gaussian Naive Bayes classifier on the training set
from sklearn.naive_bayes import GaussianNB


# instantiate the model
gnb = GaussianNB()


# fit the model
gnb.fit(X_train, y_train)
```

Out[63]:
```
▾ GaussianNB
GaussianNB()
```

In [64]:
```
y_pred = gnb.predict(X_test)

y_pred
```

Out[64]:
```
array([' >50K', ' <=50K', ' <=50K', ..., ' >50K', ' <=50K', ' <=50K'],
      dtype='<U6')
```

In [70]:
```
from sklearn.metrics import accuracy_score

print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))

#{0:0.4f}: This is a string formatting syntax.
#It specifies that the value to be printed (accuracy_score(y_test, y_pred))
#should be displayed as a floating-point number (f) with four decimal places (0.4f).
#The {0} is a placeholder for the first argument passed to the format method.
```

```
Model accuracy score: 0.8163
```

Here, y_test are the true class labels and y_pred are the predicted class labels in the test-set.

In [71]:
```
#Compare the train-set and test-set accuracy
y_pred_train = gnb.predict(X_train)
```

```
y_pred_train
```

Out[71]:
```
array([' <=50K', ' >50K', ' <=50K', ..., ' <=50K', ' <=50K', ' <=50K'],
      dtype='<U6')
```

In [72]:
```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_t
```

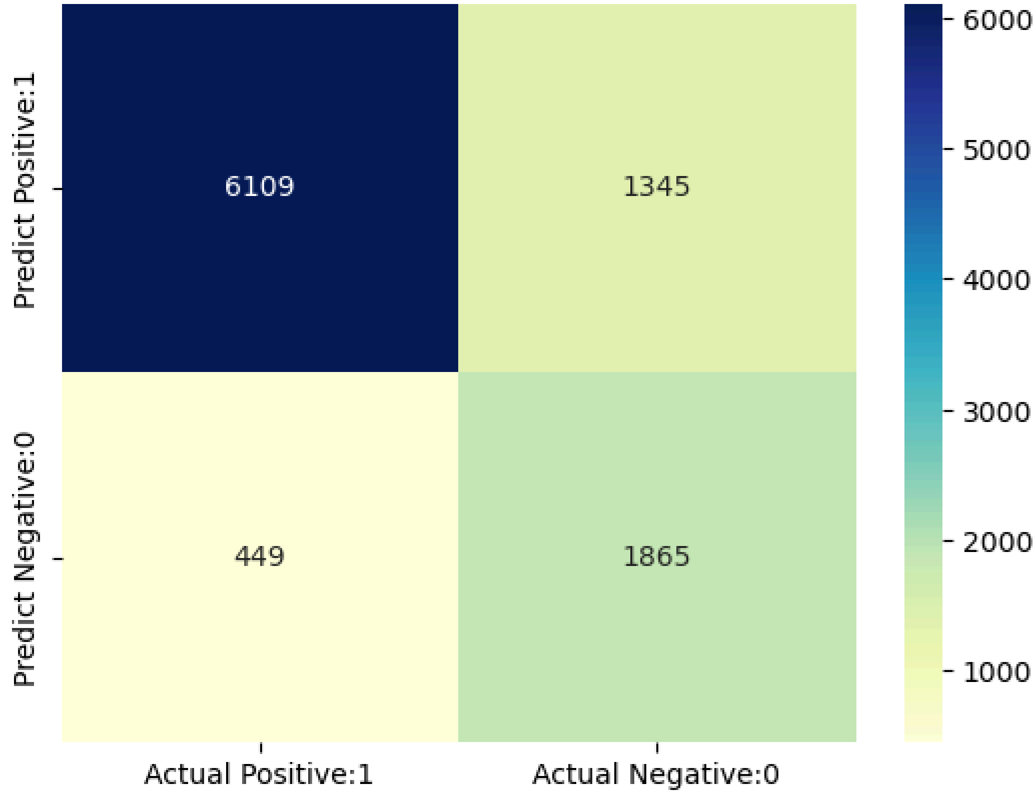
```
Training-set accuracy score: 0.8094
```

In [73]:
```
# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
Confusion matrix

 [[6109 1345]
 [ 449 1865]]

True Positives(TP) =  6109

True Negatives(TN) =  1865

False Positives(FP) =  1345

False Negatives(FN) =  449
```

In [74]:
```
# visualize confusion matrix with seaborn heatmap

cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'],
                                 index=['Predict Positive:1', 'Predict Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

Out[74]:
```
<Axes: >
```



In [75]:
```python
from sklearn.metrics import classification_report

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

```
              precision    recall  f1-score   support

       <=50K       0.93      0.82      0.87      7454
        >50K       0.58      0.81      0.68      2314

    accuracy                           0.82      9768
   macro avg       0.76      0.81      0.77      9768
weighted avg       0.85      0.82      0.83      9768
```

In [ ]:

In [ ]:

Follow for more

Linkedin: https://www.linkedin.com/company/eternaltek/about/?viewAsMember=true

Medium: https://medium.com/@eternaltek.info

WhatsApp Channel: https://whatsapp.com/channel/0029Va5onCbDjiOTi6D1vU36

Github: https://github.com/Vamsi-2203

In [ ]:

```

```