NAIVE BAYES CLASSIFIER

In [ ]:

In [ ]:

Name : Rahul Prasanth D

Roll Number : 2020506070

In [ ]:

Aim:

The aim of this notebook is to perform Naive Bayes classifier on the given adult income Dataset

In [ ]:

Dataset Information

DataSet name : **Adult dataset**
Description : This dataset contains the information about the income i.e <=50K or >50K. With the help of some attributes we can determine the target

In [ ]:

Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df=pd.read_csv("dataset/adult_new.csv")
```

In [3]:
```
# df=df.head(30000)
df
```

Out[3]:

	Age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black
...	...	...	...	...	...	...	...	...	...
22275	71	?	287372	Doctorate	16	Married-civ-spouse	?	Husband	White
22276	39	Local-gov	111499	Assoc-acdm	12	Married-civ-spouse	Adm-clerical	Wife	White
22277	53	Private	321865	Masters	14	Married-civ-spouse	Exec-managerial	Husband	White
22278	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White
22279	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White

22280 rows × 15 columns

In [4]:
```
df=df.drop(["fnlwgt","education","marital-status","relationship","capital-gain"],
```

In [ ]:

Finding the unique values in the following attributes for label encoding

In [5]:
```
val=np.unique(np.array(df['workclass']))
workclass_dict=dict()
for i in range(len(val)):
    workclass_dict[i+1]=val[i]
workclass_dict
```

Out[5]:
```
{1: ' ?',
 2: ' Federal-gov',
 3: ' Local-gov',
 4: ' Never-worked',
 5: ' Private',
 6: ' Self-emp-inc',
 7: ' Self-emp-not-inc',
 8: ' State-gov',
 9: ' Without-pay'}
```

In [6]:
```
val2=np.unique(np.array(df['occupation']))
occupation_dict=dict()
for i in range(len(val2)):
    occupation_dict[i+1]=val2[i]
occupation_dict
```

Out[6]:
```
{1: ' ?',
 2: ' Adm-clerical',
 3: ' Armed-Forces',
 4: ' Craft-repair',
 5: ' Exec-managerial',
 6: ' Farming-fishing',
 7: ' Handlers-cleaners',
 8: ' Machine-op-inspct',
 9: ' Other-service',
 10: ' Priv-house-serv',
 11: ' Prof-specialty',
 12: ' Protective-serv',
 13: ' Sales',
 14: ' Tech-support',
 15: ' Transport-moving'}
```

In [7]:
```
val3=np.unique(np.array(df['race']))
race_dict=dict()
for i in range(len(val3)):
    race_dict[i+1]=val3[i]
race_dict
```

Out[7]:
```
{1: ' Amer-Indian-Eskimo',
 2: ' Asian-Pac-Islander',
 3: ' Black',
 4: ' Other',
 5: ' White'}
```

In [8]:
```
val4=np.unique(np.array(df['gender']))
gender_dict=dict()
for i in range(len(val4)):
    gender_dict[i+1]=val4[i]
gender_dict
```

Out[8]:
```
{1: ' Female', 2: ' Male'}
```

In [9]:
```
val5=np.unique(np.array(df['native-country']))
native_dict=dict()
for i in range(len(val5)):
    native_dict[i+1]=val5[i]
native_dict
```

Out[9]:
```
{1: ' ?',
 2: ' Cambodia',
 3: ' Canada',
 4: ' China',
 5: ' Columbia',
 6: ' Cuba',
 7: ' Dominican-Republic',
 8: ' Ecuador',
 9: ' El-Salvador',
 10: ' England',
 11: ' France',
 12: ' Germany',
 13: ' Greece',
 14: ' Guatemala',
 15: ' Haiti',
 16: ' Holand-Netherlands',
 17: ' Honduras',
 18: ' Hong',
 19: ' Hungary',
 20: ' India',
 21: ' Iran',
 22: ' Ireland',
 23: ' Italy',
 24: ' Jamaica',
 25: ' Japan',
 26: ' Laos',
 27: ' Mexico',
 28: ' Nicaragua',
 29: ' Outlying-US(Guam-USVI-etc)',
 30: ' Peru',
 31: ' Philippines',
 32: ' Poland',
 33: ' Portugal',
 34: ' Puerto-Rico',
 35: ' Scotland',
 36: ' South',
 37: ' Taiwan',
 38: ' Thailand',
 39: ' Trinadad&Tobago',
 40: ' United-States',
 41: ' Vietnam',
 42: ' Yugoslavia'}
```

In [ ]:
```

```

In [ ]:
```

```

Convert string attributes to integers using

LabelEncoder

In [10]:
```
from sklearn.preprocessing import LabelEncoder
```

In [11]:
```
Lr=LabelEncoder()
```

In [12]:
```
Lr
```

Out[12]:
```
LabelEncoder()
```

In [13]:
```
temp=df[['workclass', 'occupation', 'race', 'gender', 'native-country', 'income']
```

In [ ]:
```
```

In [14]:
```
df.columns
```

Out[14]:
```
Index(['Age', 'workclass', 'educational-num', 'occupation', 'race', 'gender',
       'capital-loss', 'hours-per-week', 'native-country', 'income'],
      dtype='object')
```

In [15]:
```
for i in temp.columns:
    df[i]=Lr.fit_transform(df[i])
```

In [16]:
```
df
```

Out[16]:

	Age	workclass	educational-num	occupation	race	gender	capital-loss	hours-per-week	native-country	income
0	39	7	13	1	4	1	0	40	39	0
1	50	6	13	4	4	1	0	13	39	0
2	38	4	9	6	4	1	0	40	39	0
3	53	4	7	6	2	1	0	40	39	0
4	28	4	13	10	2	0	0	40	5	0
...	...	...	...	...	...	...	...	...	...	...
22275	71	0	16	0	4	1	0	10	39	1
22276	39	2	12	1	4	0	0	20	39	1
22277	53	4	14	4	4	1	0	40	39	1
22278	40	4	9	7	4	1	0	40	39	1
22279	52	5	9	4	4	0	0	40	39	1

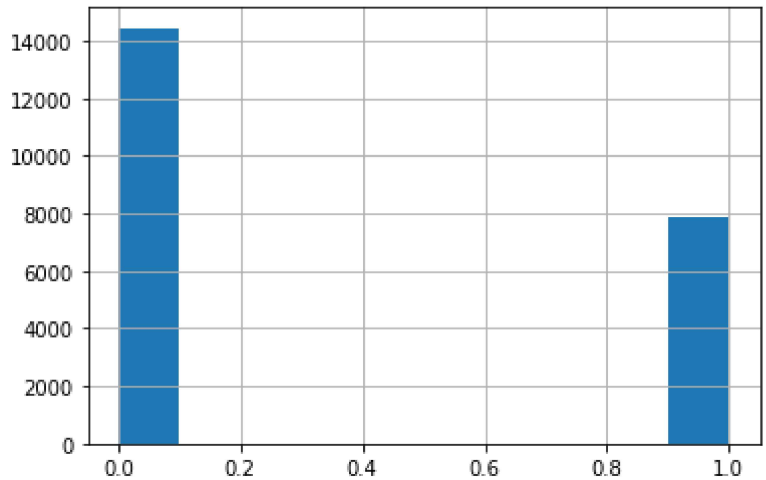
22280 rows × 10 columns

In [17]:
```
df.columns
```

Out[17]:
```
Index(['Age', 'workclass', 'educational-num', 'occupation', 'race', 'gender',
       'capital-loss', 'hours-per-week', 'native-country', 'income'],
      dtype='object')
```

In [18]:
```
df['income'].hist()
```

Out[18]:
```
<AxesSubplot:>
```



In [19]:
```
df[df['income']==1].count()
```

Out[19]:
```
Age                7841
workclass          7841
educational-num    7841
occupation         7841
race               7841
gender             7841
capital-loss       7841
hours-per-week     7841
native-country     7841
income             7841
dtype: int64
```

In [20]:
```
df[df['income']==0].count()
```

Out[20]:
```
Age                14439
workclass          14439
educational-num    14439
occupation         14439
race               14439
gender             14439
capital-loss       14439
hours-per-week     14439
native-country     14439
income             14439
dtype: int64
```

In [ ]:

```

```

In [ ]:

```

```

Calculate the Prior probability

In [21]:

```
def calculate_prior(df, Y):
    classes = sorted(list(df[Y].unique()))
    prior = []
    for i in classes:
        prior.append(len(df[df[Y] == i]) / len(df))
    return prior
```

Calculate the Likelihood using Gaussian Distribution

In [22]:

```
def calculate_likelihood_gaussian(df, feat_name, feat_val, Y, Label):
    feat = list(df.columns)
    df = df[df[Y] == Label]
    mean, std = df[feat_name].mean() ,df[feat_name].std()
    if std!=0:
        p_x_given_y = (1/(np.sqrt(2*np.pi)*std)) * np.exp(- ((feat_val - mean) **
        return p_x_given_y
    else:
        return 1
```

In [23]:

```
df[df.isnull()].count()
```

Out[23]:

```
Age                0
workclass          0
educational-num    0
occupation         0
race               0
gender             0
capital-loss       0
hours-per-week     0
native-country     0
income             0
dtype: int64
```

Apply the followings to find the posterior probability

In [24]:
```
def naivebayes(df, X, Y):
    features = list(df.columns)[:-1]
    prior = calculate_prior(df, Y)
    Y_pred = []
    for x in X:
        labels = sorted(list(df[Y].unique()))
        likelihood = [1]*len(labels)
        for j in range(len(labels)):
            for i in range(len(features)):
                likelihood[j] += np.log(calculate_likelihood_gaussian(df, feature

        post_prob = [1]*len(labels)
        for j in range(len(labels)):
            post_prob[j] = likelihood[j] + np.log(prior[j])
        Y_pred.append(np.argmax((post_prob)))
    return np.array(Y_pred)
```

In [ ]:

In [ ]:

Using SKLEARN train_test_split to split the train and test data

In [25]:
```
from sklearn.model_selection import train_test_split
train, test = train_test_split(df, test_size=.2, random_state=40)
```

In [26]:
```


X_test = test.iloc[:,:-1].values
Y_test = test.iloc[:,-1].values

Y_pred = naivebayes(train, X=X_test, Y="income")

from sklearn.metrics import accuracy_score

accuracy_score(Y_test, Y_pred,normalize=True)
```

Out[26]:
```
0.7529174147217235
```

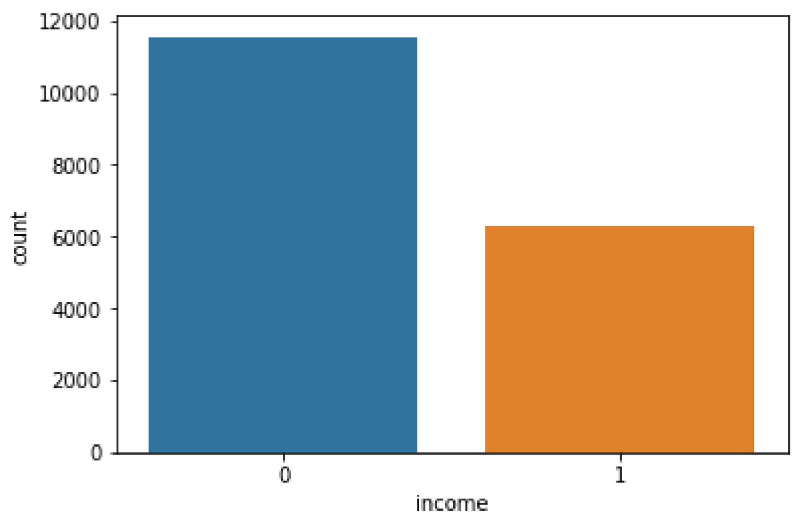
In [27]:
```
a=0
b=0
for i in range(Y_pred.size):
    if(Y_pred[i]==0):
        a+=1
    else:
        b+=1
print(a)
print(b)
```

```
3274
1182
```

In [28]:
```
sns.countplot(train["income"])
```

```
i:\python\python system files\lib\site-packages\seaborn\_decorators.py:36: Futu
reWarning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other arguments
without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[28]:
```
<AxesSubplot:xlabel='income', ylabel='count'>
```



In [ ]:

In [ ]:

As we have imbalanced dataset we have to rebalance the dataset to prevent biasing of dataset

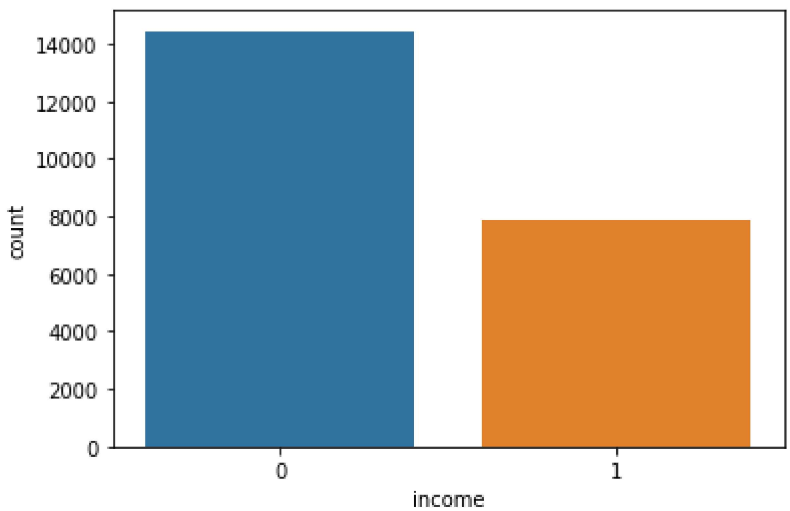
RESAMPLING METHOD

This technique is used to upsample or downsample the minority or majority class. When we are using an imbalanced dataset, we can oversample the minority class using replacement. This technique is called oversampling. Similarly, we can randomly delete rows from the majority class to match them with the minority class which is called undersampling. After sampling the data we can get a balanced dataset for both majority and minority classes. So, when both classes have a similar number of records present in the dataset, we can assume that the classifier will give equal importance to both classes.

In [29]:
```
sns.countplot(df["income"])
```

```
i:\python\python system files\lib\site-packages\seaborn\_decorators.py:36: Futu
reWarning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other arguments
without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[29]:
```
<AxesSubplot:xlabel='income', ylabel='count'>
```



In [30]:
```
from sklearn.utils import resample
```

Now we have to create two classes i.e major and minor

In [31]:
```
train_major=train[(train["income"]==0)]
train_minor=train[(train["income"]==1)]
```

In [32]:
```
train_major.count()
```

Out[32]:
```
Age                  11552
workclass            11552
educational-num      11552
occupation           11552
race                 11552
gender               11552
capital-loss         11552
hours-per-week       11552
native-country       11552
income               11552
dtype: int64
```

In [33]:
```
train_minor.count()
```

Out[33]:
```
Age                  6272
workclass            6272
educational-num      6272
occupation           6272
race                 6272
gender               6272
capital-loss         6272
hours-per-week       6272
native-country       6272
income               6272
dtype: int64
```

In [34]:
```
print("Size before concatenating the new data = \n",train.count())
```

```
Size before concatenating the new data =
 Age                  17824
workclass            17824
educational-num      17824
occupation           17824
race                 17824
gender               17824
capital-loss         17824
hours-per-week       17824
native-country       17824
income               17824
dtype: int64
```

In [35]:
```
train_minor_up=resample(
          train_minor,
    replace=True,
    n_samples=2000,
    random_state=42
)

#cancat with the old data

train=pd.concat([train_minor_up,train_major])
```

In [36]:
```
train
```

Out[36]:

	Age	workclass	educational-num	occupation	race	gender	capital-loss	hours-per-week	native-country	income
18861	29	2	11	13	4	0	0	35	27	1
15598	58	4	9	6	2	1	0	20	39	1
18914	54	6	9	5	4	1	0	60	39	1
16451	51	4	13	10	4	1	0	40	39	1
15758	55	4	9	10	4	1	0	40	39	1
...	...	...	...	...	...	...	...	...	...	...
3603	37	4	9	1	2	0	0	40	39	0
12914	23	4	3	3	4	1	0	40	26	0
5959	40	2	14	10	4	0	0	44	39	0
11532	19	4	10	12	4	1	0	25	39	0
11590	36	4	9	4	4	1	0	52	39	0

13552 rows × 10 columns

In [37]:
```
train_major.count()
```

Out[37]:
```
Age               11552
workclass         11552
educational-num   11552
occupation        11552
race              11552
gender            11552
capital-loss      11552
hours-per-week    11552
native-country    11552
income            11552
dtype: int64
```

In [38]:
```
train_minor.count()
```

Out[38]:
```
Age               6272
workclass         6272
educational-num   6272
occupation        6272
race              6272
gender            6272
capital-loss      6272
hours-per-week    6272
native-country    6272
income            6272
dtype: int64
```

In [39]:
```
print("Size After concatenating the new data = \n",train.count())
```

```
Size After concatenating the new data =
 Age                  13552
workclass            13552
educational-num      13552
occupation           13552
race                 13552
gender               13552
capital-loss         13552
hours-per-week       13552
native-country       13552
income               13552
dtype: int64
```

In [ ]:

In [65]:
```

X_test = test.iloc[:,:-1].values
Y_test = test.iloc[:,-1].values
```

In [ ]:

ACCURACY OF THE MODEL:

In [66]:
```
Y_pred = naivebayes(train, X=X_test, Y="income")

from sklearn.metrics import accuracy_score

print(accuracy_score(Y_test, Y_pred,normalize=True))
```

```
0.7264362657091562
```

In [67]:
```
Y_pred
```

Out[67]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [68]:
```
#for user input
```

In [69]:
```
X_test = test.iloc[:,:-1].values
Y_test = test.iloc[:,-1].values
```

In [70]:
```
Y_pred
```

Out[70]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [71]:
```
test.columns
```

Out[71]:
```
Index(['Age', 'workclass', 'educational-num', 'occupation', 'race', 'gender',
       'capital-loss', 'hours-per-week', 'native-country', 'income'],
      dtype='object')
```

ENTER IN THIS FORMAT

In [72]:
```
workclass_dict
```

Out[72]:
```
{1: ' ?',
 2: ' Federal-gov',
 3: ' Local-gov',
 4: ' Never-worked',
 5: ' Private',
 6: ' Self-emp-inc',
 7: ' Self-emp-not-inc',
 8: ' State-gov',
 9: ' Without-pay'}
```

In [73]:
```
occupation_dict
```

Out[73]:
```
{1: ' ?',
 2: ' Adm-clerical',
 3: ' Armed-Forces',
 4: ' Craft-repair',
 5: ' Exec-managerial',
 6: ' Farming-fishing',
 7: ' Handlers-cleaners',
 8: ' Machine-op-inspct',
 9: ' Other-service',
 10: ' Priv-house-serv',
 11: ' Prof-specialty',
 12: ' Protective-serv',
 13: ' Sales',
 14: ' Tech-support',
 15: ' Transport-moving'}
```

In [74]:
```
race_dict
```

Out[74]:
```
{1: ' Amer-Indian-Eskimo',
 2: ' Asian-Pac-Islander',
 3: ' Black',
 4: ' Other',
 5: ' White'}
```

In [75]:
```
gender_dict
```

Out[75]:
```
{1: ' Female', 2: ' Male'}
```

In [76]:
```
native_dict
```

Out[76]:
```
{1: ' ?',
 2: ' Cambodia',
 3: ' Canada',
 4: ' China',
 5: ' Columbia',
 6: ' Cuba',
 7: ' Dominican-Republic',
 8: ' Ecuador',
 9: ' El-Salvador',
 10: ' England',
 11: ' France',
 12: ' Germany',
 13: ' Greece',
 14: ' Guatemala',
 15: ' Haiti',
 16: ' Holand-Netherlands',
 17: ' Honduras',
 18: ' Hong',
 19: ' Hungary',
 20: ' India',
 21: ' Iran',
 22: ' Ireland',
 23: ' Italy',
 24: ' Jamaica',
 25: ' Japan',
 26: ' Laos',
 27: ' Mexico',
 28: ' Nicaragua',
 29: ' Outlying-US(Guam-USVI-etc)',
 30: ' Peru',
 31: ' Philippines',
 32: ' Poland',
 33: ' Portugal',
 34: ' Puerto-Rico',
 35: ' Scotland',
 36: ' South',
 37: ' Taiwan',
 38: ' Thailand',
 39: ' Trinadad&Tobago',
 40: ' United-States',
 41: ' Vietnam',
 42: ' Yugoslavia'}
```

In [ ]:
```

```

In [77]:
```
for i in range(Y_pred.size):
    if(Y_pred[i]==1):
        print(i)
```

```
25
34
36
49
50
52
54
75
77
81
103
105
107
114
119
127
128
136
144
```

In [78]:
```
X_test[25]
```

Out[78]:
```
array([  48,    4,   13,   12,    4,    0, 2472,   70,   39], dtype=int64)
```

In [ ]:

In [ ]:

In [79]:
```
age=int(input("Enter the age = "))
workclass=int(input("Enter the workclas = "))
educational_num=int(input("Enter the educational number = "))
occupation=int(input("Enter the occupation = "))
race=int(input("Enter the race = "))
gender=int(input("Enter the gender = "))
capital_loss=int(input("Enter the capital-loss = "))
hours=int(input("Enter the hours = "))
native=int(input("Enter the native = "))

X_t=np.array([[age,workclass,educational_num,occupation,race,gender,capital_loss,
```

```
Enter the age = 48
Enter the workclas = 4
Enter the educational number = 13
Enter the occupation = 12
Enter the race = 4
Enter the gender = 0
Enter the capital-loss = 2472
Enter the hours = 70
Enter the native = 39
```

In [ ]:

In [81]:
```
Y_pre = naivebayes(train, X=X_t, Y="income")

print(Y_pre,"\n")
```

```
[1]
```

In [ ]:

In [ ]:

Result

Thus, Naive Bayes Classifier has been performed on the adult income dataset.

In [ ]: