Naive Bayes Classifier in Python

Import Libraries

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import os
for dirname, _, filenames in os.walk(r"C:\Users\Prachi\Documents\VS Code Files\Mach
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
c:\Users\Prachi\anaconda3\Lib\site-packages\pandas\core\arrays\masked.py:61: UserW
arning: Pandas requires version '1.3.6' or newer of 'bottleneck' (version '1.3.5'
currently installed).
  from pandas.core import (
C:\Users\Prachi\Documents\VS Code Files\Machine Learning\Naive Bayes Algorithm\adu
lt.csv
C:\Users\Prachi\Documents\VS Code Files\Machine Learning\Naive Bayes Algorithm\nai
ve_bayes_algo.ipynb
```

In [3]:

```
import warnings
warnings.filterwarnings('ignore')
```

Import dataset

In [4]:

```
data = 'adult.csv'
df = pd.read_csv(data)
```

In [5]:

```
df
```

Out[5]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-family
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family
2	66	?	186061	Some-college	10	Widowed	?	Unmarried
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child
...	...	...	...	...	...	...	...	...
32556	22	Private	310152	Some-college	10	Never-married	Protective-serv	Not-in-family
32557	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife
32558	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband
32559	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried
32560	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child

32561 rows × 15 columns

Exploratory data analysis

In [6]:

```
df.shape
```

Out[6]:

```
(32561, 15)
```

In [7]:

```
# preview the dataset

df.head()
```

Out[7]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	ra
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-family	Wh
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	Wh
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	Bl
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	Wh
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	Wh

Rename column names

In [8]:

```
col_names = ['age', 'workclass', 'fnlwgt', 'education', 'education_num', 'marital_s
             'race', 'sex', 'capital_gain', 'capital_loss', 'hours_per_week', 'nati

df.columns = col_names

df.columns
```

Out[8]:

```
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 [9]:

```
df.head()
```

Out[9]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	r
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-family	W
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	W
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	B
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	W
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	W

In [10]:

```
# view summary of dataset

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             32561 non-null  int64 
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64 
 3   education       32561 non-null  object
 4   education_num   32561 non-null  int64 
 5   marital_status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital_gain    32561 non-null  int64 
 11  capital_loss    32561 non-null  int64 
 12  hours_per_week  32561 non-null  int64 
 13  native_country  32561 non-null  object
 14  income          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

Types of variables

Explore categorical variables

In [11]:

```
# 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', 'rac
e', 'sex', 'native_country', 'income']
```

In [12]:

```
# view the categorical variables
df[categorical].head()
```

Out[12]:

	workclass	education	marital_status	occupation	relationship	race	sex	native_country	i
0	?	HS-grad	Widowed	?	Not-in-family	White	Female	United-States	
1	Private	HS-grad	Widowed	Exec-managerial	Not-in-family	White	Female	United-States	
2	?	Some-college	Widowed	?	Unmarried	Black	Female	United-States	
3	Private	7th-8th	Divorced	Machine-op-inspct	Unmarried	White	Female	United-States	
4	Private	Some-college	Separated	Prof-specialty	Own-child	White	Female	United-States	

Explore problems within categorical variables

Missing values in categorical Variables

In [13]:

```
# check missing values in categorical variables

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

Out[13]:

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

Frequency counts of categorical variables

In [14]:

```
# 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            1298
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        5355
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            10683
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         3770
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      8305
Own-child          5068
Unmarried          3446
Wife               1568
Other-relative      981
Name: count, dtype: int64
race
```

```
White                  27816
Black                   3124
Asian-Pac-Islander      1039
Amer-Indian-Eskimo       311
Other                    271
Name: count, dtype: int64
sex
Male      21790
Female    10771
Name: count, dtype: int64
native_country
United-States                 29170
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
Greece                           29
France                           29
Ecuador                          28
Ireland                          24
Hong                             20
Cambodia                         19
Trinadad&Tobago                  19
Laos                             18
Thailand                         18
Yugoslavia                       16
Outlying-US(Guam-USVI-etc)       14
Hungary                          13
Honduras                         13
Scotland                         12
Holand-Netherlands                1
Name: count, dtype: int64
income
<=50K    24720
>50K      7841
Name: count, dtype: int64
```

In [15]:

```
# view frequency distribution of categorical variables

for var in categorical:
     print(df[var].value_counts()/float(len(df)))
```

```
workclass
Private             0.697030
Self-emp-not-inc    0.078038
Local-gov           0.064279
?                   0.056386
State-gov           0.039864
Self-emp-inc        0.034274
Federal-gov         0.029483
Without-pay         0.000430
Never-worked        0.000215
Name: count, dtype: float64
education
HS-grad         0.322502
Some-college    0.223918
Bachelors       0.164461
Masters         0.052916
Assoc-voc       0.042443
11th            0.036086
Assoc-acdm      0.032769
10th            0.028654
7th-8th         0.019840
Prof-school     0.017690
9th             0.015786
12th            0.013298
Doctorate       0.012684
5th-6th         0.010227
1st-4th         0.005160
Preschool       0.001566
Name: count, dtype: float64
marital_status
Married-civ-spouse       0.459937
Never-married            0.328092
Divorced                 0.136452
Separated                0.031479
Widowed                  0.030497
Married-spouse-absent    0.012837
Married-AF-spouse        0.000706
Name: count, dtype: float64
occupation
Prof-specialty       0.127146
Craft-repair         0.125887
Exec-managerial      0.124873
Adm-clerical         0.115783
Sales                0.112097
Other-service        0.101195
Machine-op-inspct    0.061485
?                    0.056601
Transport-moving     0.049046
Handlers-cleaners    0.042075
Farming-fishing      0.030527
Tech-support         0.028500
Protective-serv      0.019932
Priv-house-serv      0.004576
Armed-Forces         0.000276
Name: count, dtype: float64
relationship
Husband           0.405178
Not-in-family     0.255060
Own-child         0.155646
Unmarried         0.105832
Wife              0.048156
Other-relative    0.030128
Name: count, dtype: float64
race
```

```
White                 0.854274
Black                 0.095943
Asian-Pac-Islander    0.031909
Amer-Indian-Eskimo    0.009551
Other                 0.008323
Name: count, dtype: float64
sex
Male      0.669205
Female    0.330795
Name: count, dtype: float64
native_country
United-States                 0.895857
Mexico                        0.019748
?                             0.017905
Philippines                   0.006081
Germany                       0.004207
Canada                        0.003716
Puerto-Rico                   0.003501
El-Salvador                   0.003255
India                         0.003071
Cuba                          0.002918
England                       0.002764
Jamaica                       0.002488
South                         0.002457
China                         0.002303
Italy                         0.002242
Dominican-Republic            0.002150
Vietnam                       0.002058
Guatemala                     0.001966
Japan                         0.001904
Poland                        0.001843
Columbia                      0.001812
Taiwan                        0.001566
Haiti                         0.001351
Iran                          0.001321
Portugal                      0.001136
Nicaragua                     0.001044
Peru                          0.000952
Greece                        0.000891
France                        0.000891
Ecuador                       0.000860
Ireland                       0.000737
Hong                          0.000614
Cambodia                      0.000584
Trinadad&Tobago               0.000584
Laos                          0.000553
Thailand                      0.000553
Yugoslavia                    0.000491
Outlying-US(Guam-USVI-etc)    0.000430
Hungary                       0.000399
Honduras                      0.000399
Scotland                      0.000369
Holand-Netherlands            0.000031
Name: count, dtype: float64
income
<=50K    0.75919
>50K     0.24081
Name: count, dtype: float64
```

Explore workclass variable

In [16]:
```
# check labels in workclass variable
df.workclass.unique()
```

Out[16]:
```
array(['?', 'Private', 'State-gov', 'Federal-gov', 'Self-emp-not-inc',
       'Self-emp-inc', 'Local-gov', 'Without-pay', 'Never-worked'],
      dtype=object)
```

In [17]:
```
# check frequency distribution of values in workclass variable

df.workclass.value_counts()
```

Out[17]:
```
workclass
Private             22696
Self-emp-not-inc     2541
Local-gov            2093
?                    1836
State-gov            1298
Self-emp-inc         1116
Federal-gov           960
Without-pay            14
Never-worked            7
Name: count, dtype: int64
```

In [18]:
```
# replace '?' values in workclass variable with 'Nan'

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

In [19]:
```
# again check the frequency distribution of values in workclass variable

df.workclass.value_counts()
```

Out[19]:
```
workclass
Private             22696
Self-emp-not-inc     2541
Local-gov            2093
State-gov            1298
Self-emp-inc         1116
Federal-gov           960
Without-pay            14
Never-worked            7
Name: count, dtype: int64
```

Explore occupation variable

In [20]:
```
# check labels in occupation variable

df.occupation.unique()
```

Out[20]:
```
array(['?', 'Exec-managerial', 'Machine-op-inspct', 'Prof-specialty',
       'Other-service', 'Adm-clerical', 'Craft-repair',
       'Transport-moving', 'Handlers-cleaners', 'Sales',
       'Farming-fishing', 'Tech-support', 'Protective-serv',
       'Armed-Forces', 'Priv-house-serv'], dtype=object)
```

In [21]:
```
# check frequency distribution of values in occupation variable

df.occupation.value_counts()
```

Out[21]:
```
occupation
Prof-specialty       4140
Craft-repair         4099
Exec-managerial      4066
Adm-clerical         3770
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 [22]:
```
# replace '?' values in occupation variable with 'NaN'ArithmeticError

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

In [23]:
```
# again check the frequency distribution of values in occupation variable

df.occupation.value_counts()
```

Out[23]:
```
occupation
Prof-specialty       4140
Craft-repair         4099
Exec-managerial      4066
Adm-clerical         3770
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
```

Explore native_country variable

In [24]:
```
# check labels in native_country variable

df.native_country.unique()
```

Out[24]:
```
array(['United-States', '?', 'Mexico', 'Greece', 'Vietnam', 'China',
       'Taiwan', 'India', 'Philippines', 'Trinadad&Tobago', 'Canada',
       'South', 'Holand-Netherlands', 'Puerto-Rico', 'Poland', 'Iran',
       'England', 'Germany', 'Italy', 'Japan', 'Hong', 'Honduras', 'Cuba',
       'Ireland', 'Cambodia', 'Peru', 'Nicaragua', 'Dominican-Republic',
       'Haiti', 'El-Salvador', 'Hungary', 'Columbia', 'Guatemala',
       'Jamaica', 'Ecuador', 'France', 'Yugoslavia', 'Scotland',
       'Portugal', 'Laos', 'Thailand', 'Outlying-US(Guam-USVI-etc)'],
      dtype=object)
```

In [25]:
```
# check frequency distribution of values in native_country variable

df.native_country.value_counts()
```

Out[25]:
```
native_country
United-States                 29170
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
Greece                           29
France                           29
Ecuador                          28
Ireland                          24
Hong                             20
Cambodia                         19
Trinadad&Tobago                  19
Laos                             18
Thailand                         18
Yugoslavia                       16
Outlying-US(Guam-USVI-etc)       14
Hungary                          13
Honduras                         13
Scotland                         12
Holand-Netherlands                1
Name: count, dtype: int64
```

In [26]:
```
# replace '?' values in native_country variable with `NaN`

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

In [27]:
```
# again check the frequency distribution of values in native_country variable

df.native_country.value_counts()
```

Out[27]:
```
native_country
United-States                 29170
Mexico                          643
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
Greece                           29
France                           29
Ecuador                          28
Ireland                          24
Hong                             20
Trinadad&Tobago                  19
Cambodia                         19
Thailand                         18
Laos                             18
Yugoslavia                       16
Outlying-US(Guam-USVI-etc)       14
Hungary                          13
Honduras                         13
Scotland                         12
Holand-Netherlands                1
Name: count, dtype: int64
```

Check missing values in categorical variables again

In [28]:
```
df[categorical].isnull().sum()
```

Out[28]:
```
workclass         1836
education            0
marital_status       0
occupation        1843
relationship         0
race                 0
sex                  0
native_country     583
income               0
dtype: int64
```

Number of Labels : cardinality

In [29]:
```
# check for cardinality in categorical variables

for var in categorical:
    print(var, 'contains', len(df[var].unique()), 'labels')
```

```
workclass contains 9 labels
education contains 16 labels
marital_status contains 7 labels
occupation contains 15 labels
relationship contains 6 labels
race contains 5 labels
sex contains 2 labels
native_country contains 42 labels
income contains 2 labels
```

Expplore Numerical Variables

In [30]:
```
# find numerical variables

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

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

print('The numerical variables are :', numerical)
```

```
There are 6 numerical variables

The numerical variables are : ['age', 'fnlwgt', 'education_num', 'capital_gain', 'capital_loss', 'hours_per_week']
```

In [31]:
```
# View the numerical variables
df[numerical].head()
```

Out[31]:

	age	fnlwgt	education_num	capital_gain	capital_loss	hours_per_week
0	90	77053	9	0	4356	40
1	82	132870	9	0	4356	18
2	66	186061	10	0	4356	40
3	54	140359	4	0	3900	40
4	41	264663	10	0	3900	40

Explore problems within numerical variables

In [32]:
```
# check missing values in numerical variables

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

Out[32]:
```
age               0
fnlwgt            0
education_num     0
capital_gain      0
capital_loss      0
hours_per_week    0
dtype: int64
```

Declare feature vector and target variable

In [33]:
```
X = df.drop(['income'],axis=1)
y = df['income']
```

Split data into separate training and test set

In [34]:
```
# 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_s
```

In [35]:
```
# check the shape of X_train and X_test

X_train.shape, X_test.shape
```

Out[35]:
```
((22792, 14), (9769, 14))
```

Feature Engineering

In [36]:
```
# check data types in X_train

X_train.dtypes
```

Out[36]:
```
age                int64
workclass         object
fnlwgt             int64
education         object
education_num      int64
marital_status    object
occupation        object
relationship      object
race              object
sex               object
capital_gain       int64
capital_loss       int64
hours_per_week     int64
native_country    object
dtype: object
```

In [37]:
```
# display categorical variables

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

categorical
```

Out[37]:
```
['workclass',
 'education',
 'marital_status',
 'occupation',
 'relationship',
 'race',
 'sex',
 'native_country']
```

In [38]:
```
# display numerical variables

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

numerical
```

Out[38]:
```
['age',
 'fnlwgt',
 'education_num',
 'capital_gain',
 'capital_loss',
 'hours_per_week']
```

Engineering missing values in categorical variables

In [39]:
```
# print percentage of missing values in the categorical variables in training set

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

Out[39]:
```
workclass         0.056774
education         0.000000
marital_status    0.000000
occupation        0.057038
relationship      0.000000
race              0.000000
sex               0.000000
native_country    0.018208
dtype: float64
```

In [40]:
```
# 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 0.056774306774306775
occupation 0.05703757037557036
native_country 0.018208143208143207
```

In [41]:
```
# 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 [42]:
```
# check missing values in categorical variables in X_train

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

Out[42]:
```
workclass         0
education         0
marital_status    0
occupation        0
relationship      0
race              0
sex               0
native_country    0
dtype: int64
```

In [43]:
```
# check missing values in categorical variables in X_test

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

Out[43]:
```
workclass         0
education         0
marital_status    0
occupation        0
relationship      0
race              0
sex               0
native_country    0
dtype: int64
```

In [44]:
```
# check missing values in X_train

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

Out[44]:
```
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 [45]:
```
# check missing values in X_test

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

Out[45]:
```
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
```

Encode categorical Variables

In [46]:
```
# print categorical variables
categorical
```

Out[46]:
```
['workclass',
 'education',
 'marital_status',
 'occupation',
 'relationship',
 'race',
 'sex',
 'native_country']
```

In [47]:
```
X_train[categorical].head()
```

Out[47]:

	workclass	education	marital_status	occupation	relationship	race	sex	native_count
32098	State-gov	Bachelors	Married-civ-spouse	Exec-managerial	Wife	White	Female	United-Stat
25206	Local-gov	HS-grad	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	United-Stat
23491	Private	Some-college	Never-married	Exec-managerial	Not-in-family	White	Female	United-Stat
12367	Local-gov	HS-grad	Never-married	Farming-fishing	Own-child	White	Male	United-Stat
7054	Federal-gov	Masters	Married-civ-spouse	Exec-managerial	Husband	White	Male	United-Stat

In [48]:
```
# import category encoders

import category_encoders as ce
```

In [49]:
```
# encode remaining variables with one-hot encoding

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

X_train = encoder.fit_transform(X_train)

X_test = encoder.transform(X_test)
```

In [50]:
```
X_train.head()
```

Out[50]:

	age	workclass_1	workclass_2	workclass_3	workclass_4	workclass_5	workclass_6	workcla
32098	40	1	0	0	0	0	0	
25206	39	0	1	0	0	0	0	
23491	42	0	0	1	0	0	0	
12367	27	0	1	0	0	0	0	
7054	38	0	0	0	1	0	0	

5 rows × 105 columns

In [51]:
```
X_train.shape
```
Out[51]:
```
(22792, 105)
```

In [52]:
```
X_test.head()
```
Out[52]:

	age	workclass_1	workclass_2	workclass_3	workclass_4	workclass_5	workclass_6	workcla
22278	56	0	0	1	0	0	0	
8950	19	0	0	1	0	0	0	
7838	23	0	0	1	0	0	0	
16505	37	0	0	0	1	0	0	
19140	49	0	0	1	0	0	0	

5 rows × 105 columns

In [53]:
```
X_test.shape
```
Out[53]:
```
(9769, 105)
```

Feature Scaling

In [54]:
```
cols = X_train.columns
```
In [55]:
```
from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```
In [56]:
```
X_train = pd.DataFrame(X_train, columns=[cols])
```
In [57]:
```
X_train.head()
```

Out[57]:

	age	workclass_1	workclass_2	workclass_3	workclass_4	workclass_5	workclass_6	workclass_7
0	0.15	1.0	0.0	-1.0	0.0	0.0	0.0	0.0
1	0.10	0.0	1.0	-1.0	0.0	0.0	0.0	0.0
2	0.25	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	-0.50	0.0	1.0	-1.0	0.0	0.0	0.0	0.0
4	0.05	0.0	0.0	-1.0	1.0	0.0	0.0	0.0

5 rows × 105 columns

Model Training

In [58]:

```
# 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[58]:

```
GaussianNB
GaussianNB()
```

Predict the results

In [59]:

```
y_pred = gnb.predict(X_test)
y_pred
```

Out[59]:

```
array(['<=50K', '<=50K', '<=50K', ..., '<=50K', '<=50K', '>50K'],
      dtype='<U5')
```

check the accuracy score

In [60]:

```
from sklearn.metrics import accuracy_score

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

```
Model accuracy score: 0.8031
```

compare the train-set and test-set accuracy

In [61]:

```
y_pred_train = gnb.predict(X_train)
y_pred_train
```

Out[61]:
```
array(['>50K', '<=50K', '<=50K', ..., '<=50K', '>50K', '>50K'],
      dtype='<U5')
```

In [62]:
```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pre
```

```
Training-set accuracy score: 0.8009
```

Check for overfitting and underfitting

In [63]:
```
#  print the scores on training and test set

print('Training set score : {:.4f}'.format(gnb.score(X_train, y_train)))

print('Test set score : {:.4f}'.format(gnb.score(X_test, y_test)))
```

```
Training set score : 0.8009
Test set score : 0.8031
```

compare model accuracy with null accuracy

In [64]:
```
#check class distribution in test set

y_test.value_counts()
```

Out[64]:
```
income
<=50K    7410
>50K     2359
Name: count, dtype: int64
```

In [65]:
```
# check null accuracy score

null_accuracy = (7410/(7410+2359))

print('Null accuracy score: {0:0.4f}'. format(null_accuracy))
```

```
Null accuracy score: 0.7585
```

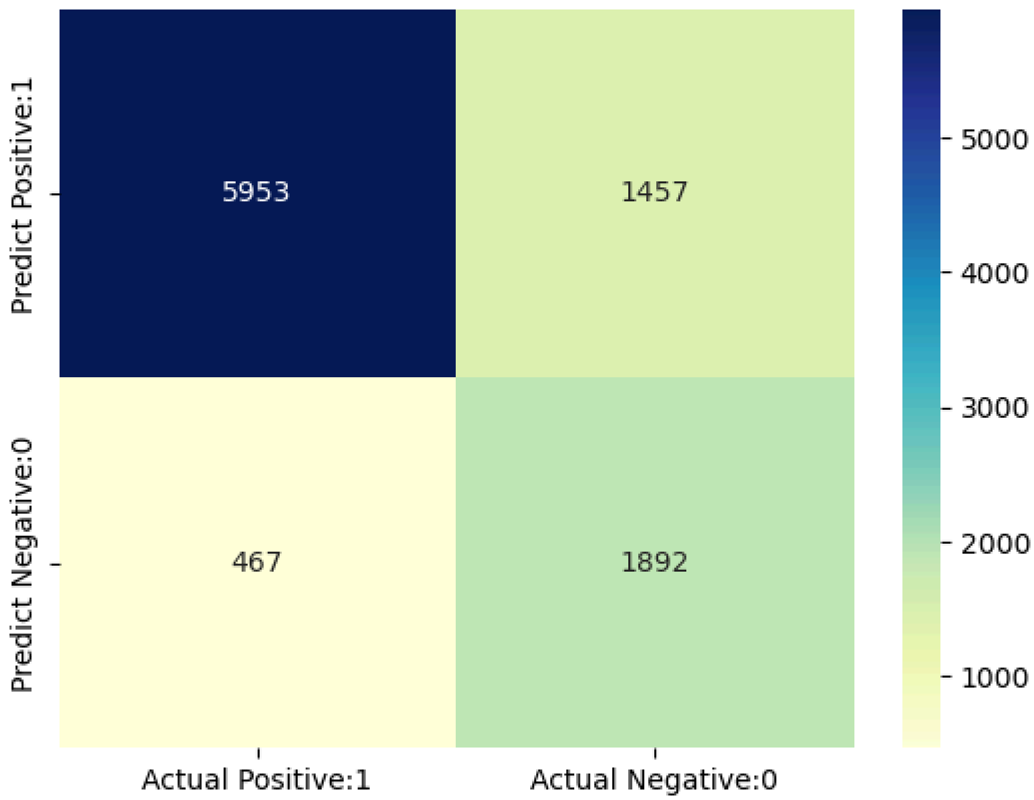
confusion matrix

In [66]:
```
# 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

 [[5953 1457]
 [ 467 1892]]

True Positives(TP) =  5953

True Negatives(TN) =  1892

False Positives(FP) =  1457

False Negatives(FN) =  467
```

In [67]:
```python
# 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[67]:
```
<Axes: >
```



Classification metrices

Classification Report

In [68]:
```python
from sklearn.metrics import classification_report

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

```
              precision    recall  f1-score   support

       <=50K       0.93      0.80      0.86      7410
        >50K       0.56      0.80      0.66      2359

    accuracy                           0.80      9769
   macro avg       0.75      0.80      0.76      9769
weighted avg       0.84      0.80      0.81      9769
```

Classification Accuracy

In [69]:

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

Classifiaction error

In [70]:

```
# print classification error

classification_error = (FP + FN) / float(TP + TN + FP + FN)

print('Classification error : {0:0.4f}'.format(classification_error))
```

```
Classification error : 0.1969
```

Precision

In [71]:

```
# print precision score

precision = TP/ float(TP+FP)

print('Precision: {0:0.4f}'.format(precision))
```

```
Precision: 0.8034
```

Recall

In [73]:

```
recall = TP/float(TP+FN)

print('Recall or Sensitivity : {0:0.4f}'.format(recall))
```

```
Recall or Sensitivity : 0.9273
```

True Positive Rate

In [74]:

```
true_positive_rate = TP/float(TP+FN)
print('True Positive Rate: {0:0.4f}'.format(true_positive_rate))
```

```
True Positive Rate: 0.9273
```

False Positive Rate

In [75]:
```
false_positive_rate = FP/float(FP+TN)

print('False Positive Rate: {0:0.4f}'.format(false_positive_rate))
```

```
False Positive Rate: 0.4351
```

Specificity

In [ ]:
```
specificity = TN / (TN + FP)

print('Specificity : {0:0.4f}'.format(specificity))
```

F1 -Score

Calculate class probabilities

In [76]:
```
# print the first 10 predicted probabilities of two classes- 0 and 1

y_pred_prob = gnb.predict_proba(X_test)[0:10]

y_pred_prob
```

Out[76]:
```
array([[9.99999693e-01, 3.06618197e-07],
       [1.00000000e+00, 1.02355439e-10],
       [9.99999997e-01, 3.02850706e-09],
       [8.78002299e-04, 9.99121998e-01],
       [7.55021219e-04, 9.99244979e-01],
       [9.99505992e-01, 4.94008099e-04],
       [9.99999697e-01, 3.03376335e-07],
       [9.63760637e-01, 3.62393626e-02],
       [9.99999937e-01, 6.31028512e-08],
       [1.41650243e-03, 9.98583498e-01]])
```

In [77]:
```
# store the probabilities in dataframe

y_pred_prob_df = pd.DataFrame(data=y_pred_prob, columns=['Prob of - <=50K', 'Prob c

y_pred_prob_df
```

Out[77]:

	Prob of - <=50K	Prob of - >50K
0	1.000000	3.066182e-07
1	1.000000	1.023554e-10
2	1.000000	3.028507e-09
3	0.000878	9.991220e-01
4	0.000755	9.992450e-01
5	0.999506	4.940081e-04
6	1.000000	3.033763e-07
7	0.963761	3.623936e-02
8	1.000000	6.310285e-08
9	0.001417	9.985835e-01

In [78]:

```
# print the first 10 predicted probabilities for class 1 - Probability of >50K

gnb.predict_proba(X_test)[0:10, 1]
```

Out[78]:

```
array([3.06618197e-07, 1.02355439e-10, 3.02850706e-09, 9.99121998e-01,
       9.99244979e-01, 4.94008099e-04, 3.03376335e-07, 3.62393626e-02,
       6.31028512e-08, 9.98583498e-01])
```

In [79]:

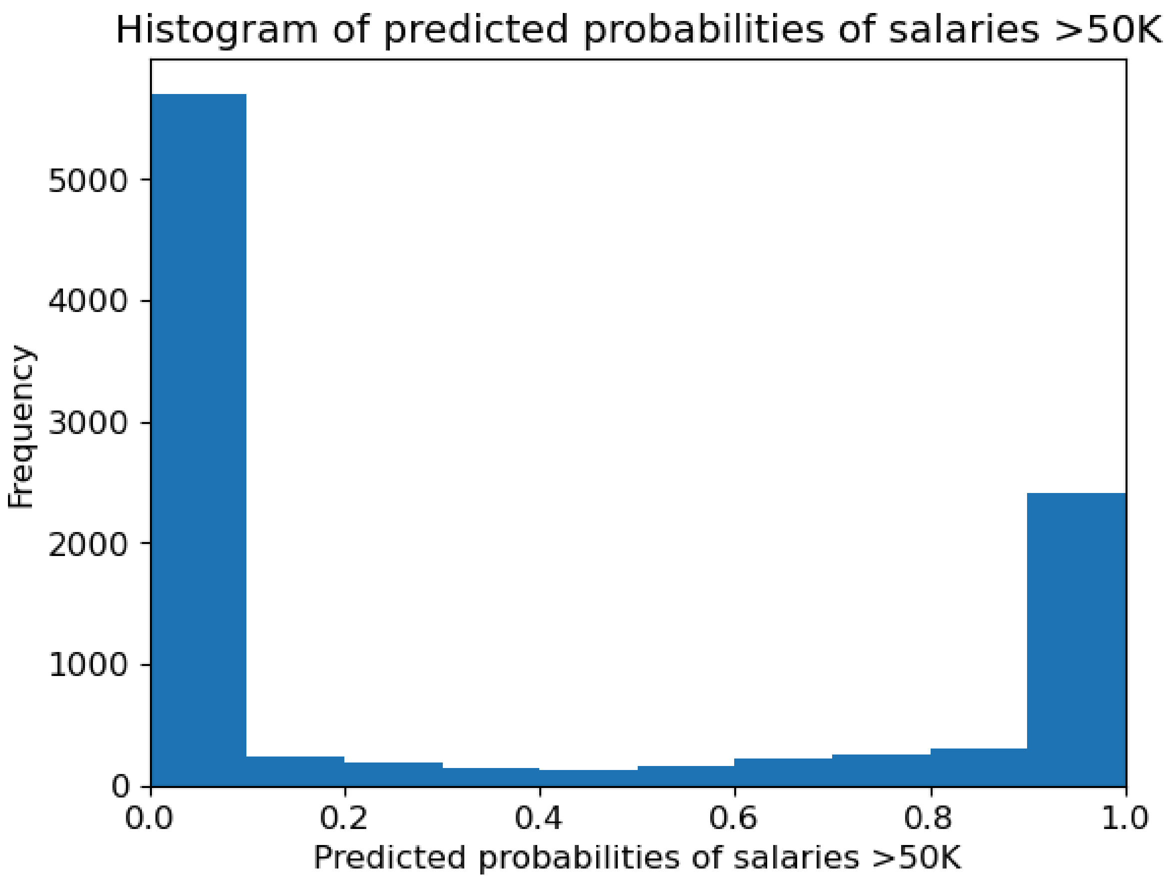
```
# store the predicted probabilities for class 1 - Probability of >50K

y_pred1 = gnb.predict_proba(X_test)[:, 1]
```

In [80]:

```
# plot histogram of predicted probabilities


# adjust the font size
plt.rcParams['font.size'] = 12


# plot histogram with 10 bins
plt.hist(y_pred1, bins = 10)


# set the title of predicted probabilities
plt.title('Histogram of predicted probabilities of salaries >50K')


# set the x-axis limit
plt.xlim(0,1)


# set the title
plt.xlabel('Predicted probabilities of salaries >50K')
plt.ylabel('Frequency')
```

Out[80]:

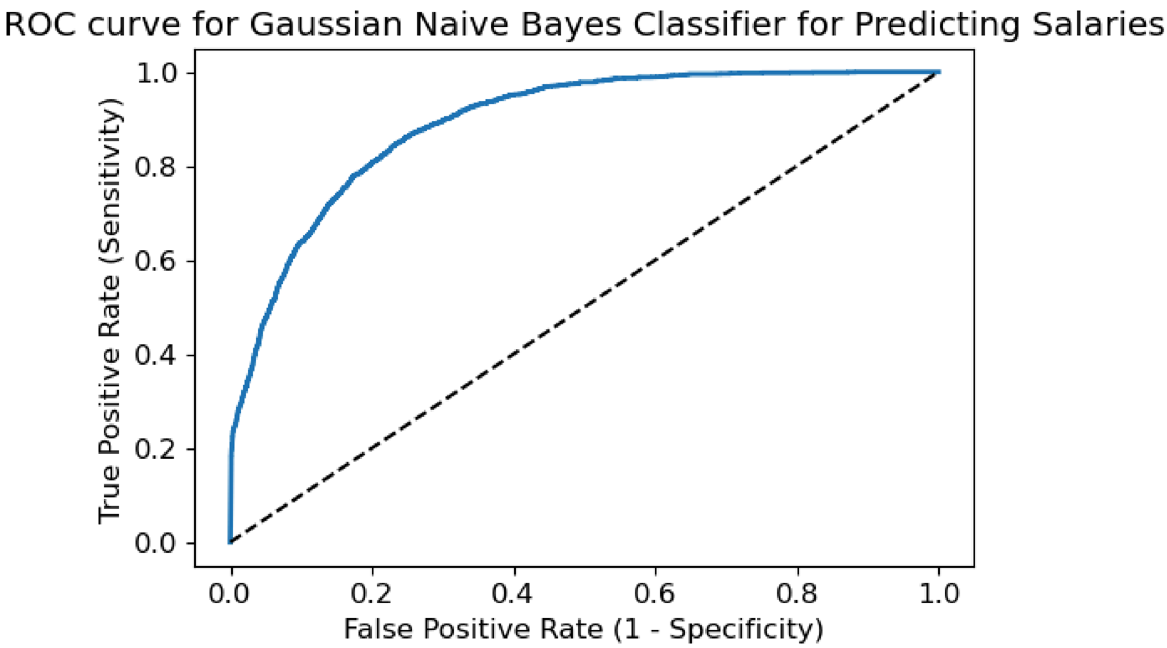
```
Text(0, 0.5, 'Frequency')
```



ROC AUC

In [81]:

```
# plot ROC Curve

from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, y_pred1, pos_label = '>50K')

plt.figure(figsize=(6,4))

plt.plot(fpr, tpr, linewidth=2)

plt.plot([0,1], [0,1], 'k--' )

plt.rcParams['font.size'] = 12

plt.title('ROC curve for Gaussian Naive Bayes Classifier for Predicting Salaries')

plt.xlabel('False Positive Rate (1 - Specificity)')

plt.ylabel('True Positive Rate (Sensitivity)')

plt.show()
```



In [82]:
```
# compute ROC AUC

from sklearn.metrics import roc_auc_score

ROC_AUC = roc_auc_score(y_test, y_pred1)

print('ROC AUC : {:.4f}'.format(ROC_AUC))
```

```
ROC AUC : 0.8909
```

In [83]:
```
# calculate cross-validated ROC AUC

from sklearn.model_selection import cross_val_score

Cross_validated_ROC_AUC = cross_val_score(gnb, X_train, y_train, cv=5, scoring='roc

print('Cross validated ROC AUC : {:.4f}'.format(Cross_validated_ROC_AUC))
```

```
Cross validated ROC AUC : 0.8936
```

K-fold Cross validation

In [84]:
```
# Applying 10-Fold Cross Validation

from sklearn.model_selection import cross_val_score

scores = cross_val_score(gnb, X_train, y_train, cv = 10, scoring='accuracy')

print('Cross-validation scores:{}'.format(scores))
```

```
Cross-validation scores:[0.80701754 0.7877193  0.79947345 0.81439228 0.785871   0.
81526986
 0.78894252 0.79420799 0.80122861 0.8056165 ]
```

In [85]:
```
# compute Average cross-validation score

print('Average cross-validation score: {:.4f}'.format(scores.mean()))
```

```
Average cross-validation score: 0.8000
```

Results and Conclusion

1. In this project, I build a Gaussian Naïve Bayes Classifier model to predict whether a person makes over 50K a year. The model yields a very good performance as indicated by the model accuracy which was found to be 0.8083.
2. The training-set accuracy score is 0.8067 while the test-set accuracy to be 0.8083. These two values are quite comparable. So, there is no sign of overfitting.
3. I have compared the model accuracy score which is 0.8083 with null accuracy score which is 0.7582. So, we can conclude that our Gaussian Naïve Bayes classifier model is doing a very good job in predicting the class labels.
4. ROC AUC of our model approaches towards 1. So, we can conclude that our classifier does a very good job in predicting whether a person makes over 50K a year.
5. Using the mean cross-validation, we can conclude that we expect the model to be around 80.63% accurate on average.
6. If we look at all the 10 scores produced by the 10-fold cross-validation, we can also conclude that there is a relatively small variance in the accuracy between folds, ranging from 81.35% accuracy to 79.64% accuracy. So, we can conclude that the model is independent of the particular folds used for training.
7. Our original model accuracy is 0.8083, but the mean cross-validation accuracy is 0.8063. So, the 10-fold cross-validation accuracy does not result in performance improvement for this model.