Assignment - 12 - Naive Bayes

1) Prepare a classification model using Naive Bayes for salary data

Data Description:

age -- age of a person

workclass -- A work class is a grouping of work

education -- Education od an individuals

maritalstatus -- Marital status of an individuals

occupation -- Occupation of an individuals

relationship --

race --Race of an Individual

sex -- Gender of an Individuals

capitalgain -- profit received from the sale of an investment

capitalloss -- A decrease in the value of a capital asset

hoursperweek -- number of hours work per week

native -- Native of an individual

Salary -- salary of an individual

Import Libraries

In [93]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import os
import warnings
warnings.filterwarnings('ignore')

from pandas.plotting import scatter_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn import metrics
import statsmodels.api as sm

from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, plot_confusion_matrix
```

Import Dataset

In [94]:
```
salarydata_train = pd.read_csv('SalaryData_Train.csv')
salarydata_train.head()
```

Out[94]:

	age	workclass	education	educationno	maritalstatus	occupation	relationship	race	sex	capitalgain	capitalloss	hoursperweek	native	Salary
0	39	State-gov	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

In [95]:
```
salarydata_test = pd.read_csv('SalaryData_Test.csv')
salarydata_test.head()
```

Out[95]:

	age	workclass	education	educationno	maritalstatus	occupation	relationship	race	sex	capitalgain	capitalloss	hoursperweek	native	Salary
0	25	Private	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United-States	<=50K
1	38	Private	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United-States	<=50K
2	28	Local-gov	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-States	>50K
3	44	Private	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	United-States	>50K
4	34	Private	10th	6	Never-married	Other-service	Not-in-family	White	Male	0	0	30	United-States	<=50K

Exploratory data analysis

In [96]:
```
salarydata_train.shape
```

Out[96]: (30161, 14)

We can see that there are 30161 instances and 14 attributes in the training dataset.

In [97]:
```
salarydata_test.shape
```

Out[97]: (15060, 14)

We can see that there are 15060 instances and 14 attributes in the test dataset

View top 5 rows of dataset

In [98]:
```
# Preview the Training dataset
salarydata_train.head()
```

Out[98]:

	age	workclass	education	educationno	maritalstatus	occupation	relationship	race	sex	capitalgain	capitalloss	hoursperweek	native	Salary
0	39	State-gov	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

In [99]:
```
# Preview the Test dataset
salarydata_test.head()
```

Out[99]:

	age	workclass	education	educationno	maritalstatus	occupation	relationship	race	sex	capitalgain	capitalloss	hoursperweek	native	Salary
0	25	Private	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United-States	<=50K
1	38	Private	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United-States	<=50K
2	28	Local-gov	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-States	>50K
3	44	Private	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	United-States	>50K
4	34	Private	10th	6	Never-married	Other-service	Not-in-family	White	Male	0	0	30	United-States	<=50K

View summary of training dataset

In [100]:
```
salarydata_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30161 entries, 0 to 30160
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   age            30161 non-null  int64 
 1   workclass      30161 non-null  object
 2   education      30161 non-null  object
 3   educationno    30161 non-null  int64 
 4   maritalstatus  30161 non-null  object
 5   occupation     30161 non-null  object
 6   relationship   30161 non-null  object
 7   race           30161 non-null  object
 8   sex            30161 non-null  object
 9   capitalgain    30161 non-null  int64 
 10  capitalloss    30161 non-null  int64 
 11  hoursperweek   30161 non-null  int64 
 12  native         30161 non-null  object
 13  Salary         30161 non-null  object
dtypes: int64(5), object(9)
memory usage: 3.2+ MB
```

In [101]:
```
salarydata_train.describe()
```

Out[101]:

	age	educationno	capitalgain	capitalloss	hoursperweek
count	30161.000000	30161.000000	30161.000000	30161.000000	30161.000000
mean	38.438115	10.121316	1092.044064	88.302311	40.931269
std	13.134830	2.550037	7406.466611	404.121321	11.980182
min	17.000000	1.000000	0.000000	0.000000	1.000000
25%	28.000000	9.000000	0.000000	0.000000	40.000000
50%	37.000000	10.000000	0.000000	0.000000	40.000000
75%	47.000000	13.000000	0.000000	0.000000	45.000000
max	90.000000	16.000000	99999.000000	4356.000000	99.000000

In [102]:
```
salarydata_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15060 entries, 0 to 15059
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   age            15060 non-null  int64 
 1   workclass      15060 non-null  object
 2   education      15060 non-null  object
 3   educationno    15060 non-null  int64 
 4   maritalstatus  15060 non-null  object
 5   occupation     15060 non-null  object
 6   relationship   15060 non-null  object
 7   race           15060 non-null  object
 8   sex            15060 non-null  object
 9   capitalgain    15060 non-null  int64 
 10  capitalloss    15060 non-null  int64 
 11  hoursperweek   15060 non-null  int64 
 12  native         15060 non-null  object
 13  Salary         15060 non-null  object
dtypes: int64(5), object(9)
memory usage: 1.6+ MB
```

In [103]:
```
salarydata_test.describe()
```

Out[103]:

	age	educationno	capitalgain	capitalloss	hoursperweek
count	15060.000000	15060.000000	15060.000000	15060.000000	15060.000000
mean	38.768327	10.112749	1120.301594	89.041899	40.951594
std	13.380676	2.558727	7703.181842	406.283245	12.062831
min	17.000000	1.000000	0.000000	0.000000	1.000000
25%	28.000000	9.000000	0.000000	0.000000	40.000000
50%	37.000000	10.000000	0.000000	0.000000	40.000000
75%	48.000000	13.000000	0.000000	0.000000	45.000000
max	90.000000	16.000000	99999.000000	3770.000000	99.000000

In [104]:
```
# Finding the specia; characters inthe dataframe
salarydata_train.isin(['?']).sum(axis=0)
```

Out[104]:
```
age              0
workclass        0
education        0
educationno      0
maritalstatus    0
occupation       0
relationship     0
race             0
sex              0
capitalgain      0
capitalloss      0
hoursperweek     0
native           0
Salary           0
dtype: int64
```

In [105]:
```
# Finding the special characters in the dataframe
salarydata_test.isin(['?']).sum(axis=0)
```

Out[105]:
```
age              0
workclass        0
education        0
educationno      0
maritalstatus    0
occupation       0
relationship     0
race             0
sex              0
capitalgain      0
capitalloss      0
hoursperweek     0
native           0
Salary           0
dtype: int64
```

In [106]:
```
print(salarydata_train[0:5])
```

```
   age          workclass   education  educationno        maritalstatus  \
0   39          State-gov   Bachelors           13        Never-married   
1   50   Self-emp-not-inc   Bachelors           13   Married-civ-spouse   
2   38            Private     HS-grad            9             Divorced   
3   53            Private        11th            7   Married-civ-spouse   
4   28            Private   Bachelors           13   Married-civ-spouse   

           occupation    relationship    race      sex  capitalgain  \
0        Adm-clerical   Not-in-family   White     Male         2174   
1     Exec-managerial         Husband   White     Male            0   
2   Handlers-cleaners   Not-in-family   White     Male            0   
3   Handlers-cleaners         Husband   Black     Male            0   
4      Prof-specialty            Wife   Black   Female            0   

   capitalloss  hoursperweek          native  Salary  
0            0            40   United-States   <=50K  
1            0            13   United-States   <=50K  
2            0            40   United-States   <=50K  
3            0            40   United-States   <=50K  
4            0            40            Cuba   <=50K  
```

Explore categorical variables

In [107]:
```
# Find categorical variables

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

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

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

```
There are 9 categorical variables

There categorical variables are:

 ['workclass', 'education', 'maritalstatus', 'occupation', 'relationship', 'race', 'sex', 'native', 'Salary']
```

In [108]:
```
# View the categorical variables
salarydata_train[categorical].head()
```

Out[108]:

	workclass	education	maritalstatus	occupation	relationship	race	sex	native	Salary
0	State-gov	Bachelors	Never-married	Adm-clerical	Not-in-family	White	Male	United-States	<=50K
1	Self-emp-not-inc	Bachelors	Married-civ-spouse	Exec-managerial	Husband	White	Male	United-States	<=50K
2	Private	HS-grad	Divorced	Handlers-cleaners	Not-in-family	White	Male	United-States	<=50K
3	Private	11th	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	United-States	<=50K
4	Private	Bachelors	Married-civ-spouse	Prof-specialty	Wife	Black	Female	Cuba	<=50K

Summary of categorical variables

There are of categorical variables

The Categorical variables are given by workclass, education, marital status, occupation, relationship, race, sex, native and Salary.

Salary is the target variables

Explore problems within categorical variables

In [109]:
```
# Check missing values in categorical variables
salarydata_train[categorical].isnull().sum()
```

Out[109]:
```
workclass        0
education        0
maritalstatus    0
occupation       0
relationship     0
race             0
sex              0
native           0
Salary           0
dtype: int64
```

We can see that there are no missing values in the categorical variables. I will confirm this further.

In [110]:
```python
# View frequency counts of values in catgoriables variables
for var in categorical:

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

```
 Private              22285
 Self-emp-not-inc      2499
 Local-gov             2067
 State-gov             1279
 Self-emp-inc          1074
 Federal-gov            943
 Without-pay             14
Name: workclass, dtype: int64
 HS-grad         9840
 Some-college    6677
 Bachelors       5044
 Masters         1627
 Assoc-voc       1307
 11th            1048
 Assoc-acdm      1008
 10th             820
 7th-8th          557
 Prof-school      542
 9th              455
```

In [111]:
```python
# check labels in workclass variable
salarydata_train.workclass.unique()
```

Out[111]:
```
array([' State-gov', ' Self-emp-not-inc', ' Private', ' Federal-gov',
       ' Local-gov', ' Self-emp-inc', ' Without-pay'], dtype=object)
```

In [112]:
```python
# Check frequency distribution of values in workclass variable
salarydata_train.workclass.value_counts()
```

Out[112]:
```
 Private              22285
 Self-emp-not-inc      2499
 Local-gov             2067
 State-gov             1279
 Self-emp-inc          1074
 Federal-gov            943
 Without-pay             14
Name: workclass, dtype: int64
```

Explore occupation variables

In [113]:
```python
# Check labels in occupation variables
salarydata_train.occupation.unique()
```

Out[113]:
```
array([' Adm-clerical', ' Exec-managerial', ' Handlers-cleaners',
       ' Prof-specialty', ' Other-service', ' Sales', ' Transport-moving',
       ' Farming-fishing', ' Machine-op-inspct', ' Tech-support',
       ' Craft-repair', ' Protective-serv', ' Armed-Forces',
       ' Priv-house-serv'], dtype=object)
```

In [114]:
```python
# Check frequency distribution values in occupation variables
salarydata_train.occupation.value_counts()
```

Out[114]:
```
 Prof-specialty       4038
 Craft-repair         4030
 Exec-managerial      3992
 Adm-clerical         3721
 Sales                3584
 Other-service        3212
 Machine-op-inspct    1965
 Transport-moving     1572
 Handlers-cleaners    1350
 Farming-fishing       989
 Tech-support          912
 Protective-serv       644
 Priv-house-serv       143
 Armed-Forces            9
Name: occupation, dtype: int64
```

Explore native_country variable

In [115]:
```
# Check labels in native_conutry variables
salarydata_train.native.unique()
```

Out[115]:
```
array([' United-States', ' Cuba', ' Jamaica', ' India', ' Mexico',
       ' Puerto-Rico', ' Honduras', ' England', ' Canada', ' Germany',
       ' Iran', ' Philippines', ' Poland', ' Columbia', ' Cambodia',
       ' Thailand', ' Ecuador', ' Laos', ' Taiwan', ' Haiti', ' Portugal',
       ' Dominican-Republic', ' El-Salvador', ' France', ' Guatemala',
       ' Italy', ' China', ' South', ' Japan', ' Yugoslavia', ' Peru',
       ' Outlying-US(Guam-USVI-etc)', ' Scotland', ' Trinadad&Tobago',
       ' Greece', ' Nicaragua', ' Vietnam', ' Hong', ' Ireland',
       ' Hungary'], dtype=object)
```

In [116]:
```
# Check frequency distribution of values in native_conutry variable.
salarydata_train.native.value_counts()
```

Out[116]:
```
 United-States                 27504
 Mexico                          610
 Philippines                     188
 Germany                         128
 Puerto-Rico                     109
 Canada                          107
 India                           100
 El-Salvador                     100
 Cuba                             92
 England                          86
 Jamaica                          80
 South                            71
 China                            68
 Italy                            68
 Dominican-Republic               67
 Vietnam                          64
 Guatemala                        63
 Japan                            59
 Poland                           56
 Columbia                         56
 Iran                             42
 Taiwan                           42
 Haiti                            42
 Portugal                         34
 Nicaragua                        33
 Peru                             30
 Greece                           29
 France                           27
 Ecuador                          27
 Ireland                          24
 Hong                             19
 Cambodia                         18
 Trinadad&Tobago                  18
 Laos                             17
 Thailand                         17
 Yugoslavia                       16
 Outlying-US(Guam-USVI-etc)       14
 Hungary                          13
 Honduras                         12
 Scotland                         11
Name: native, dtype: int64
```

Number of labels: Cardinality

In [117]:
```
# Check for cardinality in categorical variables
for var in categorical:

    print(var, ' contains ', len(salarydata_train[var].unique()),'labels')
```

```
workclass  contains  7 labels
education  contains  16 labels
maritalstatus  contains  7 labels
occupation  contains  14 labels
relationship  contains  6 labels
race  contains  5 labels
sex  contains  2 labels
native  contains  40 labels
Salary  contains  2 labels
```

Explore Numerical Variables

In [118]:

```
# Find numerical variables
numerical = [var for var in salarydata_train.columns if salarydata_train[var].dtype!='O']

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

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

```
There are 5 numerical variables

The numerical variables are : ['age', 'educationno', 'capitalgain', 'capitalloss', 'hoursperweek']
```

In [119]:

```
# View the numerical variables
salarydata_train[numerical].head()
```

Out[119]:

	age	educationno	capitalgain	capitalloss	hoursperweek
0	39	13	2174	0	40
1	50	13	0	0	13
2	38	9	0	0	40
3	53	7	0	0	40
4	28	13	0	0	40

Summary of numerical variables

There are 5 mintues variables

These are given by age, education, capitalgain, capitaloss and hoursperweek.All the numerical variables are of discrete data type.

Explore problems with in numerical variables

In [120]:

```
#Check missing values in numerical variables
salarydata_train[numerical].isnull().sum()
```

Out[120]:

```
age             0
educationno     0
capitalgain     0
capitalloss     0
hoursperweek    0
dtype: int64
```

Declare feature vector and target variables

In [121]:

```
X = salarydata_train.drop(['Salary'], axis=1)

y = salarydata_train['Salary']
```

Split data into separate training and test set

In [122]:

```
# 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_state = 0)
```

In [123]:

```
# check the shape of X_train and X_test

X_train.shape, X_test.shape
```

Out[123]:

```
((21112, 13), (9049, 13))
```

Feature Engineering

In [124]:
```
X_train.dtypes
```

Out[124]:
```
age               int64
workclass        object
education        object
educationno       int64
maritalstatus    object
occupation       object
relationship     object
race             object
sex              object
capitalgain       int64
capitalloss       int64
hoursperweek      int64
native           object
dtype: object
```

In [125]:
```
X_test.dtypes
```

Out[125]:
```
age               int64
workclass        object
education        object
educationno       int64
maritalstatus    object
occupation       object
relationship     object
race             object
sex              object
capitalgain       int64
capitalloss       int64
hoursperweek      int64
native           object
dtype: object
```

In [126]:
```
# Display categorical variables

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

categorical
```

Out[126]:
```
['workclass',
 'education',
 'maritalstatus',
 'occupation',
 'relationship',
 'race',
 'sex',
 'native']
```

In [127]:
```
# Display numerical variables

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

numerical
```

Out[127]:
```
['age', 'educationno', 'capitalgain', 'capitalloss', 'hoursperweek']
```

In [128]:
```
# Print percentage of missing values in the categorical variables in training set

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

Out[128]:
```
workclass        0.0
education        0.0
maritalstatus    0.0
occupation       0.0
relationship     0.0
race             0.0
sex              0.0
native           0.0
dtype: float64
```

In [129]:
```
# Print categorical variables with missing data
for col in categorical:
    if X_train[col].isnull().mean()>0:
        print(col, (X_train[col].isnull().mean()))
```

In [130]:
```
# 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'].fillna(X_train['native'].mode()[0], inplace=True)
```

In [131]:
```
# Check missing value in categorical variables in X_train

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

```

Out[131]:
```
workclass        0
education        0
maritalstatus    0
occupation       0
relationship     0
race             0
sex              0
native           0
dtype: int64
```

In [132]:
```
# Check missing value in categorical variables in X_test

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

```

Out[132]:
```
workclass        0
education        0
maritalstatus    0
occupation       0
relationship     0
race             0
sex              0
native           0
dtype: int64
```

In [133]:
```
# Check missing values in X_train

X_train.isnull().sum()

```

Out[133]:
```
age              0
workclass        0
education        0
educationno      0
maritalstatus    0
occupation       0
relationship     0
race             0
sex              0
capitalgain      0
capitalloss      0
hoursperweek     0
native           0
dtype: int64
```

In [134]:
```
# Check missing values in X_test

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

Out[134]:
```
age              0
workclass        0
education        0
educationno      0
maritalstatus    0
occupation       0
relationship     0
race             0
sex              0
capitalgain      0
capitalloss      0
hoursperweek     0
native           0
dtype: int64
```

Encoder categorical variables

In [135]:
```
# Print categorical variables

categorical
```

Out[135]:
```
['workclass',
 'education',
 'maritalstatus',
 'occupation',
 'relationship',
 'race',
 'sex',
 'native']
```

In [136]:
```
X_train[categorical].head()
```

Out[136]:

	workclass	education	maritalstatus	occupation	relationship	race	sex	native
8166	Local-gov	Some-college	Married-civ-spouse	Protective-serv	Husband	White	Male	United-States
7138	Private	Some-college	Never-married	Other-service	Own-child	White	Male	United-States
437	Private	HS-grad	Never-married	Transport-moving	Not-in-family	White	Male	United-States
5436	Private	HS-grad	Married-civ-spouse	Craft-repair	Husband	White	Male	United-States
6541	Self-emp-not-inc	HS-grad	Married-civ-spouse	Tech-support	Husband	White	Male	United-States

In [137]:
```
!pip install category_encoders
```

```
Requirement already satisfied: category_encoders in c:\users\admin\anaconda3\lib\site-packages (2.5.1.post0)
Requirement already satisfied: scikit-learn>=0.20.0 in c:\users\admin\anaconda3\lib\site-packages (from category_encoders) (1.
1.3)
Requirement already satisfied: statsmodels>=0.9.0 in c:\users\admin\anaconda3\lib\site-packages (from category_encoders) (0.12.
2)
Requirement already satisfied: scipy>=1.0.0 in c:\users\admin\anaconda3\lib\site-packages (from category_encoders) (1.7.1)
Requirement already satisfied: pandas>=1.0.5 in c:\users\admin\anaconda3\lib\site-packages (from category_encoders) (1.3.4)
Requirement already satisfied: numpy>=1.14.0 in c:\users\admin\anaconda3\lib\site-packages (from category_encoders) (1.20.3)
Requirement already satisfied: patsy>=0.5.1 in c:\users\admin\anaconda3\lib\site-packages (from category_encoders) (0.5.2)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\admin\anaconda3\lib\site-packages (from pandas>=1.0.5->catego
ry_encoders) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in c:\users\admin\anaconda3\lib\site-packages (from pandas>=1.0.5->category_encoder
s) (2021.3)
Requirement already satisfied: six in c:\users\admin\anaconda3\lib\site-packages (from patsy>=0.5.1->category_encoders) (1.16.
0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\admin\anaconda3\lib\site-packages (from scikit-learn>=0.20.0->c
ategory_encoders) (2.2.0)
Requirement already satisfied: joblib>=1.0.0 in c:\users\admin\anaconda3\lib\site-packages (from scikit-learn>=0.20.0->category
_encoders) (1.1.0)
```

In [138]:
```
# Import category encoders

import category_encoders as ce

```

In [139]:
```
# encoding remaining variables with one-hot encoding

encoder = ce.OneHotEncoder(cols=['workclass','education','maritalstatus','occupation','relationship','race',
                                 'sex','native'])

X_train = encoder.fit_transform(X_train)

X_test = encoder.transform(X_test)
```

In [140]:
```
X_train.head()
```

Out[140]:

	age	workclass_1	workclass_2	workclass_3	workclass_4	workclass_5	workclass_6	workclass_7	education_1	education_2	...	native_31	native_32	na
8166	54	1	0	0	0	0	0	0	1	0	...	0	0	
7138	21	0	1	0	0	0	0	0	1	0	...	0	0	
437	30	0	1	0	0	0	0	0	0	1	...	0	0	
5436	42	0	1	0	0	0	0	0	0	1	...	0	0	
6541	37	0	0	1	0	0	0	0	0	1	...	0	0	

5 rows × 102 columns

In [141]:
```
X_train.shape
```

Out[141]:
```
(21112, 102)
```

In [142]:
```
X_test.head()
```

Out[142]:

	age	workclass_1	workclass_2	workclass_3	workclass_4	workclass_5	workclass_6	workclass_7	education_1	education_2	...	native_31	native_32
25338	21	0	1	0	0	0	0	0	0	1	...	0	0
18840	21	0	1	0	0	0	0	0	0	0	...	0	0
8391	56	0	1	0	0	0	0	0	0	0	...	0	0
18258	43	1	0	0	0	0	0	0	1	0	...	0	0
16669	53	0	0	0	1	0	0	0	0	0	...	0	1

5 rows × 102 columns

In [143]:
```
X_test.shape
```

Out[143]: (9049, 102)

We now have training and testing set ready for model building.Before that, we should map all the feature variables onto the same scale.It is called Feature Scaling

Feature Scaling

In [144]:
```
cols = X_train.columns
```

In [145]:
```
from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

In [146]:
```
X_train = pd.DataFrame(X_train, columns=[cols])
```

In [147]:
```
X_test = pd.DataFrame(X_test, columns=[cols])
```

In [148]:
```
X_train.head()
```

Out[148]:

	age	workclass_1	workclass_2	workclass_3	workclass_4	workclass_5	workclass_6	workclass_7	education_1	education_2	...	native_31	native_32
0	0.894737	1.0	-1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	0.0	0.0
1	-0.842105	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	0.0	0.0
2	-0.368421	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	0.0
3	0.263158	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	0.0
4	0.000000	0.0	-1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	0.0

5 rows × 102 columns

We now have X_train dataset ready to be fed into the Gaussian Naives Bayes Classifier.

Model training

In [149]:
```
# 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[149]: GaussianNB()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Predict the results

In [150]:
```
y_pred = gnb.predict(X_test)

y_pred
```

Out[150]:
```
array([' <=50K', ' <=50K', ' <=50K', ..., ' <=50K', ' <=50K', ' >50K'],
      dtype='<U6')
```

In [151]:
```
y_pred = gnb.predict(X_test)

y_pred
```

Out[151]:
```
array([' <=50K', ' <=50K', ' <=50K', ..., ' <=50K', ' <=50K', ' >50K'],
      dtype='<U6')
```

Check accuracy score

In [152]:
```
from sklearn.metrics import accuracy_score

print('Model accuracy score:{0:04f}'.format(accuracy_score(y_test,y_pred)))
```

```
Model accuracy score:0.799536
```

Here,y_test are the true class labels and y_pred are the predicted class labels in the test_set.

Check for overfitting and underfitting

In [153]:
```
# 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: 0.8023
Test set score: 0.7995
```

The training-set accuracy score is 0.8023 while the test-set accuracy to be 0.7995. These two values are quite comparable. So, there is no sign of overfitting

Compare model accuracy with null accuracy

In [154]:
```
# Check class distribution in test set

y_test.value_counts()
```

Out[154]:
```
 <=50K    6798
 >50K     2251
Name: Salary, dtype: int64
```

In [155]:
```
# Check null accuracy score

null_accuracy = (7407/(7407+2362))

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

```
Null accuracy score:0.7582
```

We can see that our model accuracy score is 0.8023 but null accuracy score is 0.7582. So, we can conclude that our Gaussian Naive Bayes Classification model is doing a very good job in predicting the class labels.

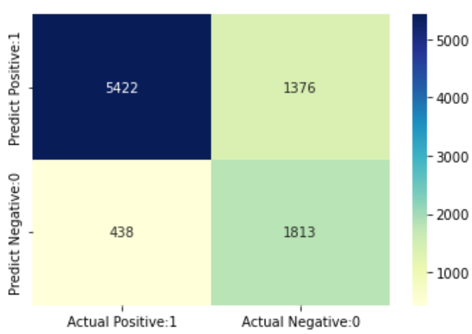
Confusion Matrix

In [156]:
```
# 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 Positive(TP)=' , cm[0,0])

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

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

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

```
confusion matrix

 [[5422 1376]
 [ 438 1813]]

True Positive(TP)= 5422

True Negative(TN)= 1813

False Positive(FP)= 1376

False Negative(FN)= 438
```

In [157]:
```
# 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[157]: <AxesSubplot:>



Classification mertices

In [158]:
```
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      6798
        >50K       0.57      0.81      0.67      2251

    accuracy                           0.80      9049
   macro avg       0.75      0.80      0.76      9049
weighted avg       0.84      0.80      0.81      9049
```

Classification accuracy

In [159]:
```
TP = cm[0,0]
TN = cm[1,1]
FP = cm[0,1]
FN = cm[1,0]
```

In [160]:
```
# Print classification accuracy

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

print('classification accuracy : {0:0.4f}'.format(classification_accuracy))
```

```
classification accuracy : 0.7995
```

Classification error

In [161]:
```
# print classification error

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

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

```
Classification error : 0.2005
```

Precision

In [162]:
```
# print precision score

precision = TP / float(TP + FP)

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

```
Precision : 0.7976
```

Recall

In [163]:
```
recall = TP / float(TP + FN)

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

```
Recall or Sensitivity : 0.9253
```

True Positive Rate.

True Positive Rate is synonymous with Recall.

In [168]:
```
true_postivie_rate = TP / float(TN + FN)

print('True Positive Rate : {0:0.4f}'.format(true_postivie_rate))
```

```
True Positive Rate : 2.4087
```

False Positive Rate

In [167]:
```
false_positive_rate = FP / float(FP + TN)

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

```
False Positive Rate : 0.4315
```

Specificity

In [166]:
```python
specificity = TN / (TN + FP)

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

```
Specificity : 0.5685
```

Calculate class probabilities

In [170]:
```python
# 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[170]:
```
array([[9.99955511e-01, 4.44887598e-05],
       [9.95935549e-01, 4.06445120e-03],
       [8.63901480e-01, 1.36098520e-01],
       [9.99999906e-01, 9.37239455e-08],
       [8.80888343e-02, 9.11911166e-01],
       [9.99562896e-01, 4.37103927e-04],
       [5.34482750e-06, 9.99994655e-01],
       [6.28497161e-01, 3.71502839e-01],
       [5.46536963e-04, 9.99453463e-01],
       [9.99999570e-01, 4.30495598e-07]])
```

Observations:

In each row, the numbers sum to 1.

There are 2 columns which correspond to 2 classes - <= 50k and > 50k.

```
1. Class 0 => <= 50k - class that a person makes less than equal to 50k.
```

```
2.   Class 1 => >50k - class that a person makes more than 50k.
```

Importance of predicted probabilities

```
   1. We can rank the observations by probability of whether a person makes less than or equal to 50K or more than
 50K.
   predict_proba process

   2. Predicts the probabilities

   3.Choose the class with the highest probability
```

Classification threshold level

```
1. There is a classification threshold level of 0.5.

2. Class 0 => <=50K - probability of salary less than or equal to 50K is predicted if probability < 0.5.

3. Class 1 => >50K - probability of salary more than 50K is predicted if probability > 0.5.
```

In [173]:
```
# store the probabilities in dataframe

y_pred_prob_df = pd.DataFrame(data=y_pred_prob, columns=['Prob of - <=50k', 'Prob of - >50k'])

y_pred_prob_df
```

Out[173]:

	Prob of - <=50k	Prob of - >50k
0	0.999956	4.448876e-05
1	0.995936	4.064451e-03
2	0.863901	1.360985e-01
3	1.000000	9.372395e-08
4	0.088089	9.119112e-01
5	0.999563	4.371039e-04
6	0.000005	9.999947e-01
7	0.628497	3.715028e-01
8	0.000547	9.994535e-01
9	1.000000	4.304956e-07

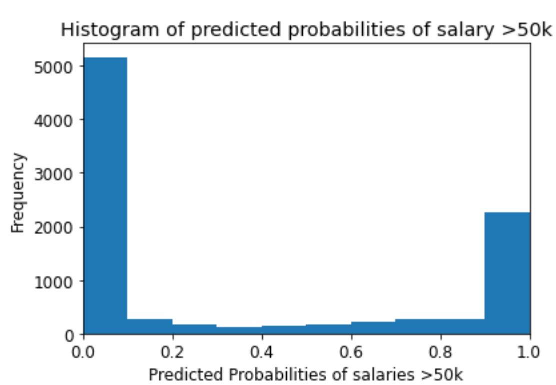
In [174]:
```
# print the first 10 predicted probabilities for class 1 - probability of >50k

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

Out[174]:
```
array([4.44887598e-05, 4.06445120e-03, 1.36098520e-01, 9.37239455e-08,
       9.11911166e-01, 4.37103927e-04, 9.99994655e-01, 3.71502839e-01,
       9.99453463e-01, 4.30495598e-07])
```

In [175]:
```
# Store the predicted probabilities for class 1 - probability of >50k

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

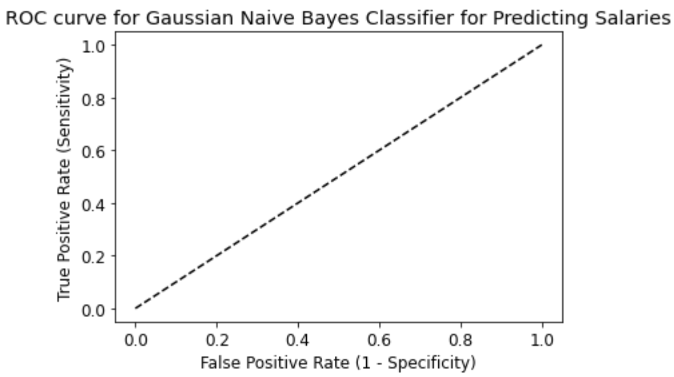
In [176]:
```
#plot histogram of predicted probabilities

#adjust the front 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 salary >50k')

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

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

Out[176]:
```
Text(0, 0.5, 'Frequency')
```



In [177]:

```
# 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([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 [178]:

```
# 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 : 0.8902
```

Interpretation

In [182]:

```
#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_auc').mean()

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

```
Cross Validated ROC AUC : 0.8923
```

k - Fold Cross Validation

In [183]:

```
#Appling 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:[0.81676136 0.79829545 0.79014685 0.81288489 0.80388441 0.79062056
 0.80767409 0.7925154  0.79630507 0.80909522]
```

In [184]:
```
#compute Average cross-validation score

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

```
Average Cross-Validation score:0.8018
```

In [ ]:
```

```