19SE02IT058 SEIT4013

In [1]:

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

In [2]:

```
import warnings

warnings.filterwarnings('ignore')
```

In [3]:

```
data = 'adult.csv'

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

In [4]:

```
# view dimensions of dataset

df.shape
```

Out[4]:

```
(32561, 15)
```

We can see that there are 32561 instances and 15 attributes in the data set.

In [5]:

```
# preview the dataset

df.head()
```

Out[5]:

	0	1	2	3	4	5	6	7	8	9	10	11
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0

In [6]:

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

df.columns = col_names

df.columns
```

Out[6]:

```
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 [7]:

```
# let's again preview the dataset

df.head()
```

Out[7]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife

In [8]:

```
# 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
```

In [9]:

```
# find categorical variables

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

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

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

```
There are 9 categorical variables

The categorical variables are :

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

In [10]:

```
# view the categorical variables

df[categorical].head()
```

Out[10]:

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

In [11]:

```
# check missing values in categorical variables

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

Out[11]:

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

In [12]:

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

for var in categorical:

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

```
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: workclass, dtype: int64
HS-grad         10501
Some-college     7291
Bachelors        5355
Masters          1723
Assoc-voc        1382
11th             1175
Assoc-acdm       1067
10th              933
7th-8th           646
```

In [13]:

```
# view frequency distribution of categorical variables

for var in categorical:

    print(df[var].value_counts()/np.float(len(df)))
```

```
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: workclass, dtype: float64
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
```

Now, we can see that there are several variables like `workclass`, `occupation` and `native_country` which contain missing values. Generally, the missing values are coded as `NaN` and python will detect them

But, in this case the missing values are coded as `?`. Python fail to detect these as missing values because it do not consider `?` as missing values. So, I have to replace `?` with `NaN` so that Python can detect these missing values.

I will explore these variables and replace `?` with `NaN`.

In [14]:

```
# check labels in workclass variable

df.workclass.unique()
```

Out[14]:

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

In [15]:

```
# check frequency distribution of values in workclass variable

df.workclass.value_counts()
```

Out[15]:

```
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: workclass, dtype: int64
```

We can see that there are 1836 values encoded as `?` in workclass variable. I will replace these `?` with `NaN`.

In [16]:

```
# replace '?' values in workclass variable with `NaN`


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

In [17]:

```
# again check the frequency distribution of values in workclass variable

df.workclass.value_counts()
```

Out[17]:

```
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: workclass, dtype: int64
```

Now, we can see that there are no values encoded as `?` in the `workclass` variable.

I will adopt similar approach with `occupation` and `native_country` column.

In [18]:

```
# check labels in occupation variable

df.occupation.unique()
```

Out[18]:

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

In [19]:

```
# check frequency distribution of values in occupation variable

df.occupation.value_counts()
```

Out[19]:

```
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: occupation, dtype: int64
```

We can see that there are 1843 values encoded as `?` in `occupation` variable. I will replace these `?` with `NaN`.

In [20]:

```
# replace '?' values in occupation variable with `NaN`

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

In [21]:

```
# again check the frequency distribution of values in occupation variable

df.occupation.value_counts()
```

Out[21]:

```
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: occupation, dtype: int64
```

In [22]:

```
# check labels in native_country variable

df.native_country.unique()
```

Out[22]:

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

In [23]:

```
# check frequency distribution of values in native_country variable

df.native_country.value_counts()
```

Out[23]:

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

We can see that there are 583 values encoded as `?` in `native_country` variable. I will replace these `?` with `NaN`.

In [24]:

```
# replace '?' values in native_country variable with `NaN`

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

In [25]:

```
# again check the frequency distribution of values in native_country variable

df.native_country.value_counts()
```

Out[25]:

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

In [26]:

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

Out[26]:

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

In [27]:

```
# 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
```

In [28]:

```
# 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_ga
in', 'capital_loss', 'hours_per_week']
```

In [29]:

```
# view the numerical variables

df[numerical].head()
```

Out[29]:

	age	fnlwgt	education_num	capital_gain	capital_loss	hours_per_week
0	39	77516	13	2174	0	40
1	50	83311	13	0	0	13
2	38	215646	9	0	0	40
3	53	234721	7	0	0	40
4	28	338409	13	0	0	40

In [30]:

```
# check missing values in numerical variables

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

Out[30]:

```
age               0
fnlwgt            0
education_num     0
capital_gain      0
capital_loss      0
hours_per_week    0
dtype: int64
```

We can see that all the 6 numerical variables do not contain missing values.

In [31]:

```
X = df.drop(['income'], axis=1)

y = df['income']
```

In [32]:

```
# 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 [33]:

```
# check the shape of X_train and X_test

X_train.shape, X_test.shape
```

Out[33]:

```
((22792, 14), (9769, 14))
```

In [34]:

```
# check data types in X_train

X_train.dtypes
```

Out[34]:

```
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 [35]:

```
# display categorical variables

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

categorical
```

Out[35]:

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

In [36]:

```
# display numerical variables

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

numerical
```

Out[36]:

```
['age',
 'fnlwgt',
 'education_num',
 'capital_gain',
 'capital_loss',
 'hours_per_week']
```

In [37]:

```
# print percentage of missing values in the categorical variables in training set

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

Out[37]:

```
workclass         0.055985
education         0.000000
marital_status    0.000000
occupation        0.056072
relationship      0.000000
race              0.000000
sex               0.000000
native_country    0.018164
dtype: float64
```

In [38]:

```
# 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 0.055984555984555984
occupation 0.05607230607230607
native_country 0.018164268164268166
```

In [39]:

```
# 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 [40]:

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

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

Out[40]:

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

In [41]:

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

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

Out[41]:

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

As a final check, I will check for missing values in X_train and X_test.

In [42]:

```
# check missing values in X_train

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

Out[42]:

```
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 [43]:

```
# check missing values in X_test

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

Out[43]:

```
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 [44]:

```
# print categorical variables

categorical
```

Out[44]:

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

In [45]:

```
X_train[categorical].head()
```

Out[45]:

	workclass	education	marital_status	occupation	relationship	race	sex	native_cou
32098	Private	HS-grad	Married-civ-spouse	Craft-repair	Husband	White	Male	United-S
25206	State-gov	HS-grad	Divorced	Adm-clerical	Unmarried	White	Female	United-S
23491	Private	Some-college	Married-civ-spouse	Sales	Husband	White	Male	United-S
12367	Private	HS-grad	Never-married	Craft-repair	Not-in-family	White	Male	Guate
7054	Private	7th-8th	Never-married	Craft-repair	Not-in-family	White	Male	Ger

In [46]:

```
# import category encoders

import category_encoders as ce
```

In [47]:

```
# encode remaining variables with one-hot encoding

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

X_train = encoder.fit_transform(X_train)

X_test = encoder.transform(X_test)
```

In [48]:

```
X_train.head()
```

Out[48]:

	age	workclass_1	workclass_2	workclass_3	workclass_4	workclass_5	workclass_6
32098	45	1	0	0	0	0	0
25206	47	0	1	0	0	0	0
23491	48	1	0	0	0	0	0
12367	29	1	0	0	0	0	0
7054	23	1	0	0	0	0	0

5 rows × 105 columns

In [49]:

```
X_train.shape
```

Out[49]:

```
(22792, 105)
```

In [50]:

```
X_test.head()
```

Out[50]:

	age	workclass_1	workclass_2	workclass_3	workclass_4	workclass_5	workclass_6	w
22278	27	1	0	0	0	0	0	
8950	27	1	0	0	0	0	0	
7838	25	1	0	0	0	0	0	
16505	46	1	0	0	0	0	0	
19140	45	1	0	0	0	0	0	

5 rows × 105 columns

In [51]:

```
X_test.shape
```

Out[51]:

```
(9769, 105)
```

In [52]:

```
cols = X_train.columns
```

In [53]:

```
from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

In [54]:

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

In [55]:

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

In [56]:

```
X_train.head()
```

Out[56]:

	age	workclass_1	workclass_2	workclass_3	workclass_4	workclass_5	workclass_6	work
0	0.40	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.50	-1.0	1.0	0.0	0.0	0.0	0.0	
2	0.55	0.0	0.0	0.0	0.0	0.0	0.0	
3	-0.40	0.0	0.0	0.0	0.0	0.0	0.0	
4	-0.70	0.0	0.0	0.0	0.0	0.0	0.0	

5 rows × 105 columns

In [57]:

```
# 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[57]:

```
GaussianNB()
```

In [58]:

```
y_pred = gnb.predict(X_test)

y_pred
```

Out[58]:

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

In [59]:

```
from sklearn.metrics import accuracy_score

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

```
Model accuracy score: 0.8083
```

In [60]:

```
y_pred_train = gnb.predict(X_train)

y_pred_train
```

Out[60]:

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

In [61]:

```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train)
```

```
Training-set accuracy score: 0.8067
```

In [62]:

```
# check class distribution in test set

y_test.value_counts()
```

Out[62]:

```
<=50K    7407
>50K     2362
Name: income, dtype: int64
```

In [63]:

```
# check null accuracy score

null_accuracy = (7407/(7407+2362))

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

```
Null accuracy score: 0.7582
```

In [64]:

```
# 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])
```

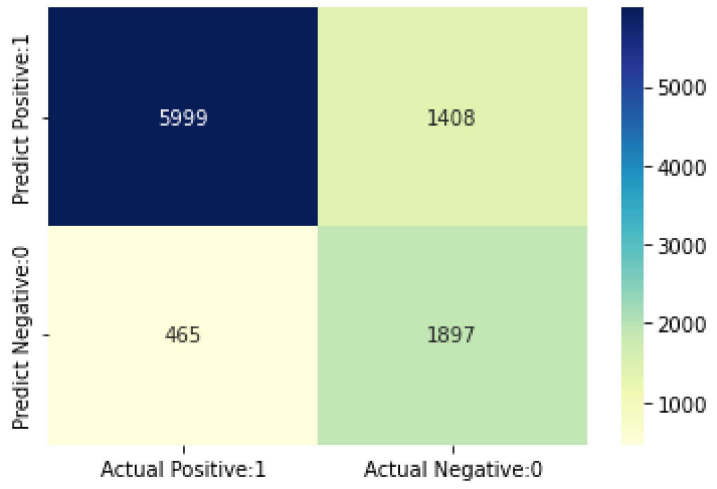
```
Confusion matrix

 [[5999 1408]
 [ 465 1897]]

True Positives(TP) =  5999

True Negatives(TN) =  1897

False Positives(FP) =  1408

False Negatives(FN) =  465
```

In [65]:

```
# 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[65]:

```
<AxesSubplot:>
```



In [66]:

```
from sklearn.metrics import classification_report

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

```
              precision    recall  f1-score   support

       <=50K       0.93      0.81      0.86      7407
        >50K       0.57      0.80      0.67      2362

    accuracy                           0.81      9769
   macro avg       0.75      0.81      0.77      9769
weighted avg       0.84      0.81      0.82      9769
```

In [67]:

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

In [68]:

```
# print classification accuracy

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

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

```
Classification accuracy : 0.8083
```

In [69]:

```
# print classification error

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

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

```
Classification error : 0.1917
```

In [70]:

```
# print precision score

precision = TP / float(TP + FP)


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

```
Precision : 0.8099
```

In [71]:

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

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

```
Recall or Sensitivity : 0.9281
```

In [72]:

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


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

```
True Positive Rate : 0.9281
```

In [73]:

```
false_positive_rate = FP / float(FP + TN)


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

```
False Positive Rate : 0.4260
```

In [74]:

```
specificity = TN / (TN + FP)

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

```
Specificity : 0.5740
```

In [75]:

```
# 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[75]:

```
array([[9.99999426e-01, 5.74152436e-07],
       [9.99687907e-01, 3.12093456e-04],
       [1.54405602e-01, 8.45594398e-01],
       [1.73624321e-04, 9.99826376e-01],
       [8.20121011e-09, 9.99999992e-01],
       [8.76844580e-01, 1.23155420e-01],
       [9.99999927e-01, 7.32876705e-08],
       [9.99993460e-01, 6.53998797e-06],
       [9.87738143e-01, 1.22618575e-02],
       [9.99999996e-01, 4.01886317e-09]])
```

In [76]:

```
# 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[76]:

	Prob of - <=50K	Prob of - >50K
0	9.999994e-01	5.741524e-07
1	9.996879e-01	3.120935e-04
2	1.544056e-01	8.455944e-01
3	1.736243e-04	9.998264e-01
4	8.201210e-09	1.000000e+00
5	8.768446e-01	1.231554e-01
6	9.999999e-01	7.328767e-08
7	9.999935e-01	6.539988e-06
8	9.877381e-01	1.226186e-02
9	1.000000e+00	4.018863e-09

In [77]:

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

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

Out[77]:

```
array([5.74152436e-07, 3.12093456e-04, 8.45594398e-01, 9.99826376e-01,
       9.99999992e-01, 1.23155420e-01, 7.32876705e-08, 6.53998797e-06,
       1.22618575e-02, 4.01886317e-09])
```

In [78]:

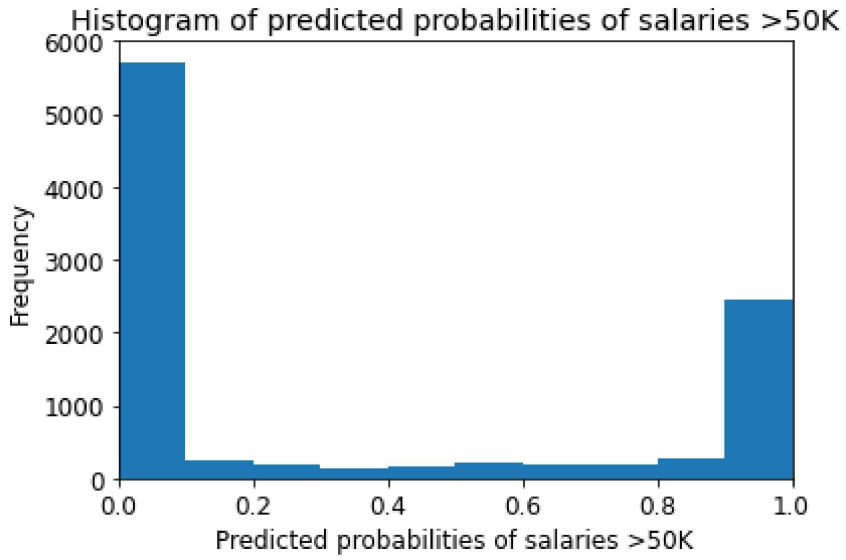
```
# store the predicted probabilities for class 1 - Probability of >50K

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

In [79]:

```
# 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')
```

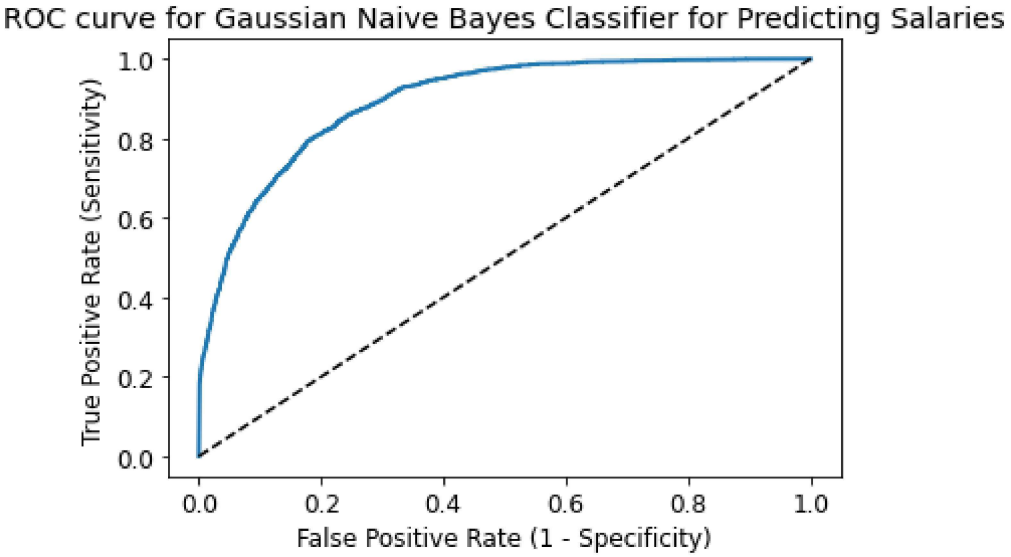
Out[79]:

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



In [80]:

```
# 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 [81]:

```
# 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 : 0.8941
```

In [82]:

```
# 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').m

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

```
Cross validated ROC AUC : 0.8938
```

In [83]:

```
# 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:[0.81359649 0.80438596 0.81175954 0.8056165  0.79596
314 0.79684072
 0.81044318 0.81175954 0.80210619 0.81044318]
```

In [84]:

```
# compute Average cross-validation score

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

```
Average cross-validation score: 0.8063
```

In [ ]:

```

```