```
Naive Bayes is a classification algorithm based on Bayes' theorem with an assumption of independence between features. It is commonly used for text classification tasks and works well with high-dimensional data. Here's a brief explanation of how the Naive Bayes algorithm works:

Bayes' Theorem: Naive Bayes is based on Bayes' theorem, which states that the probability of a hypothesis (class label) given the evidence (features) is proportional to the probability of the evidence given the hypothesis multiplied by the prior probability of the hypothesis. Mathematically, it can be represented as:

P(H|E) = (P(E|H) * P(H)) / P(E)

P(H|E) is the posterior probability of hypothesis H given evidence E.
P(E|H) is the probability of evidence E given hypothesis H.
P(H) is the prior probability of hypothesis H.
P(E) is the prior probability of evidence E.
```

In [86]:

```python
#import the library
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [87]:

```python
import os
```

In [88]:

```python
os.getcwd()
```

Out[88]:

'/Users/myyntiimac'

In [89]:

```python
df=pd.read_csv("/Users/myyntiimac/Desktop/adult.csv")
df.head()
```

Out[89]:

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

In [90]:

```python
df.shape
```

Out[90]:

(32561, 15)

In [91]:

```
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 [92]:

```
df.isnull().any()
```

Out[92]:

```
age               False
workclass         False
fnlwgt            False
education         False
education.num     False
marital.status    False
occupation        False
relationship      False
race              False
sex               False
capital.gain      False
capital.loss      False
hours.per.week    False
native.country    False
income            False
dtype: bool
```

In [93]:

```
df.isnull().sum().sum()
```

Out[93]:

```
0
```

In [10]:

```
df.columns
```

Out[10]:

```
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 [94]:

```
df.size
```

Out[94]:

```
488415
```

In [95]:

```
for column in df.columns:
    # Check if the column contains the question mark value
    if '?' in df[column].values:
        # Find the rows with the question mark value in the column
        rows_with_question_mark = df[df[column] == '?']

        # Print the column name and the rows that contain the question mark value
        print(f"Column: {column}")
        print(rows_with_question_mark)

```

```
...                      ...                  ...            ...
32459  Married-civ-spouse            Sales        Husband
32476       Never-married   Prof-specialty  Not-in-family
32498            Divorced            Sales      Own-child
32515  Married-civ-spouse   Prof-specialty        Husband
32528            Divorced                ?      Unmarried

                     race     sex  capital.gain  capital.loss  hours.per.week  \
9                   White    Male             0          3004              60
18                  Black    Male             0          2824              40
65                  White    Male             0          2415              70
86                  White    Male             0          2415              50
87     Asian-Pac-Islander    Male             0          2415              55
...                   ...     ...           ...           ...             ...
32459               White    Male             0             0              50
32476               White  Female             0             0              99
32498               White    Male             0             0              50
32515               White    Male             0             0              45
32528               White  Female             0             0               1
```

```
insight: 3 column contain ? mark , they are workclass, ocupation, native country
    As python does not detect question mark as nan, we replace the question mark with NAN
```

In [96]:

```
df.workclass.unique()
```

Out[96]:

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

In [97]:

```
df.workclass.value_counts()
```

Out[97]:

```
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
```

```
insight:we saw 1836 values are ? in workclass, now we change it with nun
```

In [98]:

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

In [22]:

```
df.workclass.value_counts()
```

Out[22]:

```
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
```

In [99]:

```
df.workclass.isnull().sum()
```

Out[99]:

```
1836
```

In [100]:

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

In [101]:

```
df.occupation.isnull().sum()
```

Out[101]:

```
1843
```

In [102]:

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

In [103]:

```
df["native.country"].isnull().sum()
```

Out[103]:

```
583
```

In [104]:

```
#if we want to see all the null value
df.isnull().any()
```

Out[104]:

```
age               False
workclass          True
fnlwgt            False
education         False
education.num     False
marital.status    False
occupation         True
relationship      False
race              False
sex               False
capital.gain      False
capital.loss      False
hours.per.week    False
native.country     True
income            False
dtype: bool
```

In [105]:

```
#now impute this catagorical null values with mode
df['workclass'].fillna(df['workclass'].mode()[0], inplace=True)
df['occupation'].fillna(df['occupation'].mode()[0], inplace=True)
df['native.country'].fillna(df['native.country'].mode()[0], inplace=True)
```

In [106]:

```
df.isnull().sum().sum()
```

Out[106]:

0

```
Insight:now we have no ? mark and nan value
```

In [107]:

```
df.head()
```

Out[107]:

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

In [108]:

```
# 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', 'ca
pital.loss', 'hours.per.week']
```

In [109]:

```
# find numerical variables

df[numerical].head()
```

Out[109]:

	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

In [110]:

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

catagorical
```

Out[110]:

```
['workclass',
 'education',
 'marital.status',
 'occupation',
 'relationship',
 'race',
 'sex',
 'native.country',
 'income']
```

In [111]:

```
df[catagorical].head()
```

Out[111]:

	workclass	education	marital.status	occupation	relationship	race	sex	native.country	income
0	Private	HS-grad	Widowed	Prof-specialty	Not-in-family	White	Female	United-States	<=50K
1	Private	HS-grad	Widowed	Exec-managerial	Not-in-family	White	Female	United-States	<=50K
2	Private	Some-college	Widowed	Prof-specialty	Unmarried	Black	Female	United-States	<=50K
3	Private	7th-8th	Divorced	Machine-op-inspct	Unmarried	White	Female	United-States	<=50K
4	Private	Some-college	Separated	Prof-specialty	Own-child	White	Female	United-States	<=50K

In [112]:

```
df[catagorical].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   workclass       32561 non-null  object
 1   education       32561 non-null  object
 2   marital.status  32561 non-null  object
 3   occupation      32561 non-null  object
 4   relationship    32561 non-null  object
 5   race            32561 non-null  object
 6   sex             32561 non-null  object
 7   native.country  32561 non-null  object
 8   income          32561 non-null  object
dtypes: object(9)
memory usage: 2.2+ MB
```

```
We will convert dtype object to catagory
```

In [114]:

```
col = ['workclass', 'education', 'marital.status', 'occupation', 'relationship', 'race', 'sex
df[col] = df[col].astype('category')
```

In [115]:

```
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  category
 2   fnlwgt          32561 non-null  int64   
 3   education       32561 non-null  category
 4   education.num   32561 non-null  int64   
 5   marital.status  32561 non-null  category
 6   occupation      32561 non-null  category
 7   relationship    32561 non-null  category
 8   race            32561 non-null  category
 9   sex             32561 non-null  category
 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  category
 14  income          32561 non-null  category
dtypes: category(9), int64(6)
memory usage: 1.8 MB
```

```
Insight: we see our df contain 9 catagorical variable, and 6 numerical variable

```

In [58]:

```
#now we convert the catagoical values with numerical values by one hot encoder from module ca
!pip install category_encoders
import category_encoders as ce
```

```
Collecting category_encoders
  Downloading category_encoders-2.6.1-py2.py3-none-any.whl (81 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 81.9/81.9 kB 534.3 kB/s eta 0:00:000:01
Requirement already satisfied: statsmodels>=0.9.0 in ./anaconda3/lib/python3.10/site
-packages (from category_encoders) (0.13.5)
Requirement already satisfied: patsy>=0.5.1 in ./anaconda3/lib/python3.10/site-packa
ges (from category_encoders) (0.5.3)
Requirement already satisfied: scipy>=1.0.0 in ./anaconda3/lib/python3.10/site-packa
ges (from category_encoders) (1.10.0)
Requirement already satisfied: pandas>=1.0.5 in ./anaconda3/lib/python3.10/site-pack
ages (from category_encoders) (1.5.3)
Requirement already satisfied: numpy>=1.14.0 in ./anaconda3/lib/python3.10/site-pack
ages (from category_encoders) (1.23.5)
Requirement already satisfied: scikit-learn>=0.20.0 in ./anaconda3/lib/python3.10/si
te-packages (from category_encoders) (1.2.1)
Requirement already satisfied: python-dateutil>=2.8.1 in ./anaconda3/lib/python3.10/
site-packages (from pandas>=1.0.5->category_encoders) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in ./anaconda3/lib/python3.10/site-packa
ges (from pandas>=1.0.5->category_encoders) (2022.7)
Requirement already satisfied: six in ./anaconda3/lib/python3.10/site-packages (from
patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.1.1 in ./anaconda3/lib/python3.10/site-pack
ages (from scikit-learn>=0.20.0->category_encoders) (1.1.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in ./anaconda3/lib/python3.10/si
te-packages (from scikit-learn>=0.20.0->category_encoders) (2.2.0)
Requirement already satisfied: packaging>=21.3 in ./anaconda3/lib/python3.10/site-pa
ckages (from statsmodels>=0.9.0->category_encoders) (22.0)
Installing collected packages: category_encoders
Successfully installed category_encoders-2.6.1
```

```
Insight: we see our df contain 9 catagorical variable, and 6 numerical variable
```

Define the Independent and dependent variable

In [116]:

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

y = df['income']
```

split the data

In [135]:

```
# 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 = 0.20, random_state = 0)
```

In [146]:

```
# check the shape of X_train and X_test

X_train.shape, X_test.shape
```

Out[146]:

```
((26048, 105), (6513, 105))
```

In [148]:

```
from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

```

In [149]:

```
X_train_scaled = pd.DataFrame((X_train), columns=cols)
X_test_scaled = pd.DataFrame((X_test), columns=cols)
```

In [150]:

```
X_train_scaled
```

Out[150]:

ative.country_33	native.country_34	native.country_35	native.country_36	native.country_37	native.country_38	native.cou
0.0	0.0	0.0	0.0	0.0	0.0	
0.0	0.0	0.0	0.0	0.0	0.0	
0.0	0.0	0.0	0.0	0.0	0.0	
0.0	0.0	0.0	0.0	0.0	0.0	
0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	
0.0	0.0	0.0	0.0	0.0	0.0	
0.0	0.0	0.0	0.0	0.0	0.0	
0.0	0.0	0.0	0.0	0.0	0.0	
0.0	0.0	0.0	0.0	0.0	0.0	
0.0	0.0	0.0	0.0	0.0	0.0	

In [151]:

```
#now the x_train data is data is ready for model training, lets do this with Naive-bayes clas
from sklearn.naive_bayes import GaussianNB
 = GaussianNB()
gnb.fit(X_train, y_train)
```

Out[151]:

```
▾ GaussianNB
GaussianNB()
```

In [153]:

```
#now our model is build for predict, so predict  with  x_test data
y_pred =gnb.predict(X_test)
y_pred
```

Out[153]:

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

In [157]:

```
from sklearn.metrics import accuracy_score
accuracy_score1=accuracy_score(y_test, y_pred)
accuracy_score1
```

Out[157]:

```
0.8036235221863964
```

In [158]:

```
#get the prediction with X_train
y_pred_train =gnb.predict(X_train)
y_pred_train
```

Out[158]:

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

In [159]:

```
#find accuracy
accuracy_score2=accuracy_score(y_train, y_pred_train )
accuracy_score2
```

Out[159]:

```
0.8003685503685504
```

```
Insight:The training-set accuracy score is 0.80362 while the test-set accuracy to be
0.80036. These two values are quite comparable. So, there is no sign of overfitting.
```

#Compare model accuracy with null accuracy comparing the model accuracy with the null accuracy provides context and helps in evaluating the model's performance, ensuring that it is better than a simple baseline and providing insights for further analysis and decision-making. y_test.value_counts()

In [160]:

```
y_test.value_counts()
```

Out[160]:

```
<=50K    4966
>50K     1547
Name: income, dtype: int64
```

In [161]:

```
# find null score
null_accuracy = (4966/(1547+4966))
null_accuracy
```

Out[161]:

```
0.7624750499001997
```

```
insight: Since the model accuracy (0.80) is higher than the null accuracy (0.76), it
indicates our model is outperforming the simple naive approach. This suggests that your
model is learning meaningful patterns from the data and providing better predictions than
simply predicting the majority class.

it does not tell anything about the type of errors our classifer is making.
Lets find the answer with  Confusion matrix
```

In [162]:

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
cm
```

Out[162]:

```
array([[4005,  961],
       [ 318, 1229]])
```

In [ ]:

```
Insight:total erros=961+318=979
```

In [163]:

```
#see cm  in heat map
# first convert array to df
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'],
                                 index=['Predict Positive:1', 'Predict Negative:0'])
cm_matrix
```

Out[163]:

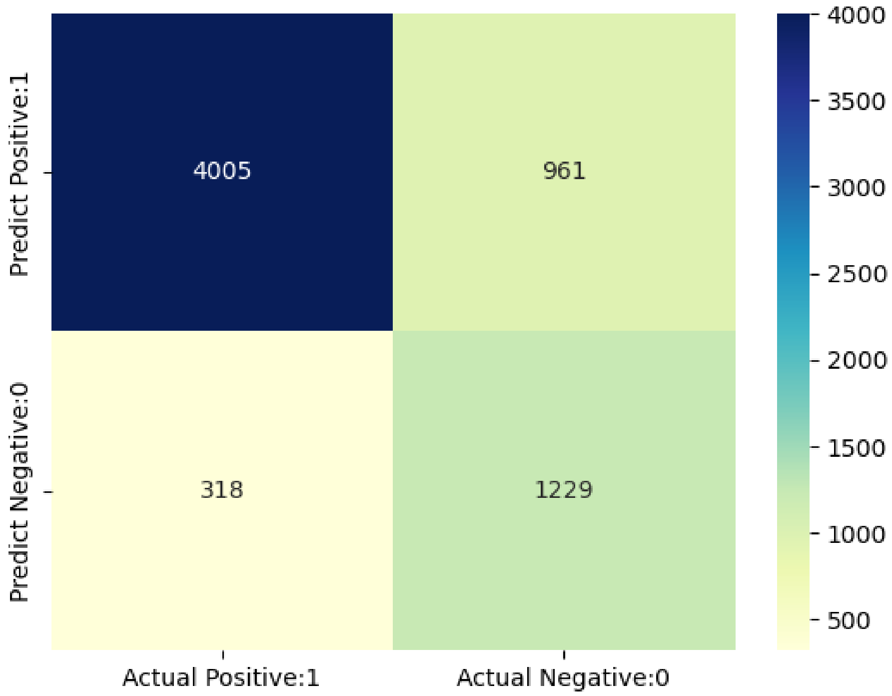
	Actual Positive:1	Actual Negative:0
Predict Positive:1	4005	961
Predict Negative:0	318	1229

In [164]:

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

Out[164]:

<Axes: >



In [166]:

```
from sklearn.metrics import classification_report
classification_report(y_test, y_pred)
```

Out[166]:

```
'              precision    recall  f1-score   support\n\n       <=50K       0.93
0.81      0.86      4966\n        >50K      0.56      0.79      0.66      1547\n\n
accuracy                           0.80      6513\n   macro avg      0.74      0.80
0.76      6513\nweighted avg      0.84      0.80      0.81      6513\n'
```

In [167]:

```
#we can define our cm9 finding separately
TP = cm[0,0]
TN = cm[1,1]
FP = cm[0,1]
FN = cm[1,0]
```

In [168]:

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

Out[168]:

```
0.8036235221863964
```

In [169]:

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

Out[169]:

```
0.19637647781360357
```

In [170]:

```
# print precision score

precision = TP / float(TP + FP)
precision
```

Out[170]:

```
0.806484091824406
```

In [171]:

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

Out[171]:

```
0.43881278538812785
```

In [172]:

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

Out[172]:

```
0.5611872146118722
```

In [173]:

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

Out[173]:

```
0.926439972241499
```

In [ ]:

```
#F1 score
#The F1 score is the harmonic mean of precision and recall. It is calculated using the follow

#F1 score = 2 * (precision * recall) / (precision + recall)

#The F1 score ranges from 0 to 1, with 1 being the best score indicating perfect precision and
```

In [174]:

```
F1_score = 2 * (precision * recall) / (precision + recall)
F1_score
```

Out[174]:

```
0.8623102594466574
```

In [176]:

```
y_pred_prob =gnb.predict_proba(X_test)[0:20]

y_pred_prob
```

Out[176]:

```
array([[9.99999604e-01, 3.96418796e-07],
       [1.00000000e+00, 1.44294273e-10],
       [9.99999997e-01, 3.12016675e-09],
       [1.02419444e-03, 9.98975806e-01],
       [9.18190838e-04, 9.99081809e-01],
       [9.99502162e-01, 4.97838406e-04],
       [9.99999505e-01, 4.94690101e-07],
       [9.67007719e-01, 3.29922815e-02],
       [9.99999921e-01, 7.90638033e-08],
       [1.58882375e-03, 9.98411176e-01],
       [8.10889466e-01, 1.89110534e-01],
       [1.34721162e-08, 9.99999987e-01],
       [9.99998254e-01, 1.74627829e-06],
       [9.99427437e-01, 5.72562953e-04],
       [7.76999315e-04, 9.99223001e-01],
       [1.00000000e+00, 1.57528603e-12],
       [9.88958128e-01, 1.10418718e-02],
       [1.36170920e-03, 9.98638291e-01],
       [3.02460808e-01, 6.97539192e-01],
       [9.97882166e-01, 2.11783395e-03]])
```

```
Insight:
    There are 2 values(=<50k,>50k) which correspond to classes - 0 and 1.

Class 0 - predicted probability that there is income=<50k

Class 1 - predicted probability that there is income >50k

Importance of predicted probabilities

We can rank the observations by probability of income range

for Predicting the probabilities

Choose the class with the highest probability

```

```
Classification threshold level

There is a classification threshold level of 0.50.

Class 0 - probability if probability> 0.5.

Class 1 - probability if probability < 0.5.
```

In [178]:

```python
y_pred_prob_df = pd.DataFrame(data=y_pred_prob, columns=['Prob of - <=50K', 'Prob of - >50K']

y_pred_prob_df
```

Out[178]:

	Prob of - <=50K	Prob of - >50K
0	9.999996e-01	3.964188e-07
1	1.000000e+00	1.442943e-10
2	1.000000e+00	3.120167e-09
3	1.024194e-03	9.989758e-01
4	9.181908e-04	9.990818e-01
5	9.995022e-01	4.978384e-04
6	9.999995e-01	4.946901e-07
7	9.670077e-01	3.299228e-02
8	9.999999e-01	7.906380e-08
9	1.588824e-03	9.984112e-01
10	8.108895e-01	1.891105e-01
11	1.347212e-08	1.000000e+00
12	9.999983e-01	1.746278e-06
13	9.994274e-01	5.725630e-04
14	7.769993e-04	9.992230e-01
15	1.000000e+00	1.575286e-12
16	9.889581e-01	1.104187e-02
17	1.361709e-03	9.986383e-01
18	3.024608e-01	6.975392e-01
19	9.978822e-01	2.117834e-03

In [180]:

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

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

Out[180]:

```
array([3.96418796e-07, 1.44294273e-10, 3.12016675e-09, 9.98975806e-01,
       9.99081809e-01, 4.97838406e-04, 4.94690101e-07, 3.29922815e-02,
       7.90638033e-08, 9.98411176e-01])
```

In [181]:

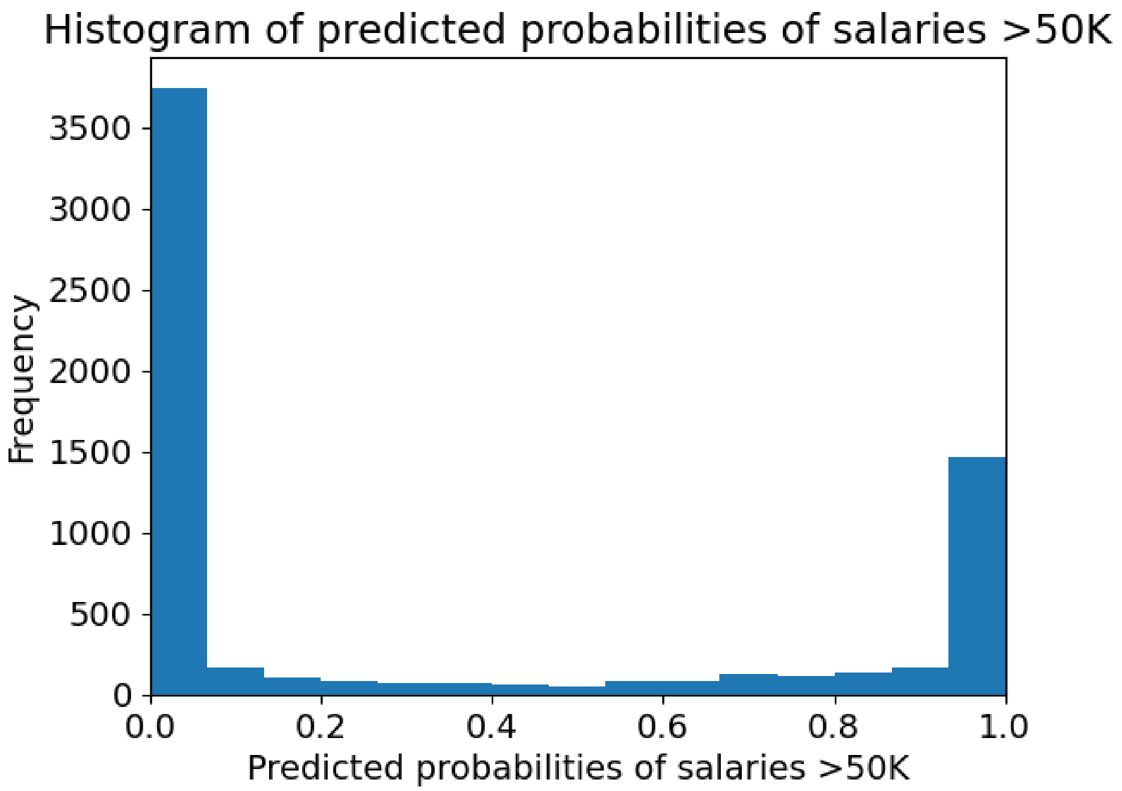
```python
# store the predicted probabilities for class 1 - Probability of >50K

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

In [183]:

```python
# plot histogram of predicted probabilities


# adjust the font size
plt.rcParams['font.size'] = 14


# plot histogram with 10 bins
plt.hist(y_pred1, bins = 15)


# 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[183]:

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




```
Insight:
    We can see that the above histogram is highly positive skewed.

The first column tell us that there are approximately 5700 observations with probability
between 0.0 and 0.1 whose salary is <=50K.

There are relatively small number of observations with probability > 0.5.

So, these small number of observations predict that the salaries will be >50K.

```

```
Majority of observations predcit that the salaries will be <=50K.
```

Cross validation:

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))

In [184]:

```python
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:[0.8        0.80652591 0.78925144 0.8        0.80307102 0.79
923225
 0.79769674 0.79731286 0.79953917 0.80645161]
```

In [185]:

```python
# take the average score
Avg_scoreof_CV=scores.mean()
Avg_scoreof_CV
```

Out[185]:

```
0.7999080994542577
```

```
insight:In this case, the average accuracy across all 10 iterations was0.7999080994542577 . This means that, on average, the model correctly classified approximately 79% of the instances in the test sets.

An accuracy of 0.7999080994542577 indicates that the model is performing well and has a high level of predictive accuracy. It suggests that the model is capable of making accurate predictions on unseen data.
```

Conclusion

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.

The training-set accuracy score is 0.8003 while the test-set accuracy to be 0.8083. These two values are quite comparable. So, there is no sign of overfitting.

I have compared the model accuracy score which is 0.8083 with null accuracy score which is 0.7624. So, we can conclude that our Gaussian Naïve Bayes classifier model is doing a very good job in predicting the class labels.

Using the mean cross-validation, we can conclude that we expect the model to be around 79.99% accurate on average.

Our original model accuracy is 0.8083, but the mean cross-validation accuracy is 0.7999. So, the 10-fold cross-validation accuracy does not result in performance improvement for this model.