Credit Score Classification using Naive Bayes

January 22, 2024

In [62]:
```
import pandas as pd
import numpy as np
df=pd.read_csv("C:/Users/anarg/OneDrive/Documents/Credit Score Classification Dataset
df.head()
```

Out[62]:

	Age	Gender	Income	Education	Marital Status	Number of Children	Home Ownership	Credit Score
0	25	Female	50000.0	Bachelor's Degree	Single	0	Rented	High
1	30	Male	100000.0	Master's Degree	Married	2	Owned	High
2	35	Female	75000.0	Doctorate	Married	1	Owned	High
3	40	Male	125000.0	High School Diploma	Single	0	Owned	High
4	45	Female	100000.0	Bachelor's Degree	Married	3	Owned	High

In [63]:
```
df.tail()
```

Out[63]:

	Age	Gender	Income	Education	Marital Status	Number of Children	Home Ownership	Credit Score
159	29	Female	27500.0	High School Diploma	Single	0	Rented	Low
160	34	Male	47500.0	Associate's Degree	Single	0	Rented	Average
161	39	Female	62500.0	Bachelor's Degree	Married	2	Owned	High
162	44	Male	87500.0	Master's Degree	Single	0	Owned	High
163	49	Female	77500.0	Doctorate	Married	1	Owned	High

In [64]:
```
df.shape
```

Out[64]:
```
(164, 8)
```

In [65]:
```
df.columns
```

Out[65]:
```
Index(['Age', 'Gender', 'Income', 'Education', 'Marital Status',
       'Number of Children', 'Home Ownership', 'Credit Score'],
      dtype='object')
```

Finding missing values

In [66]:
```
df.isna().sum()
```

Out[66]:
```
Age                 0
Gender              6
Income              9
Education           0
Marital Status      0
Number of Children  0
Home Ownership      0
Credit Score        0
dtype: int64
```

Filling missing values

In [67]:
```
gender_mode=df['Gender'].mode()[0]
df['Gender'].fillna(gender_mode,inplace=True)

income_mean=df['Income'].mean()
df['Income'].fillna(income_mean,inplace=True)
```

In [68]:
```
df.isna().sum()
```

Out[68]:
```
Age                  0
Gender               0
Income               0
Education            0
Marital Status       0
Number of Children   0
Home Ownership       0
Credit Score         0
dtype: int64
```

In [69]:
```
df.dtypes
```

Out[69]:
```
Age                   int64
Gender               object
Income              float64
Education            object
Marital Status       object
Number of Children    int64
Home Ownership       object
Credit Score         object
dtype: object
```

Label encoding

In [70]:
```
from sklearn.preprocessing import LabelEncoder
lab=LabelEncoder()
df['Gender']=lab.fit_transform(df['Gender'])
df['Education']=lab.fit_transform(df['Education'])
df['Marital Status']=lab.fit_transform(df['Marital Status'])
df['Home Ownership']=lab.fit_transform(df['Home Ownership'])
```

In [71]:
```
df
```

Out[71]:

	Age	Gender	Income	Education	Marital Status	Number of Children	Home Ownership	Credit Score
0	25	0	50000.0	1	1	0	1	High
1	30	1	100000.0	4	0	2	0	High
2	35	0	75000.0	2	0	1	0	High
3	40	1	125000.0	3	1	0	0	High
4	45	0	100000.0	1	0	3	0	High
...	...	...	...	...	...	...	...	...
159	29	0	27500.0	3	1	0	1	Low
160	34	1	47500.0	0	1	0	1	Average
161	39	0	62500.0	1	0	2	0	High
162	44	1	87500.0	4	1	0	0	High
163	49	0	77500.0	2	0	1	0	High

164 rows × 8 columns

Spliting into training and testing data

In [72]:
```
x=df.iloc[:,:-1].values
y=df.iloc[:,-1].values
```

In [87]:
```
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=42)
```

In [88]:
```
x_train
```

Out[88]:
```
array([[4.50000000e+01, 0.00000000e+00, 1.15000000e+05, 1.00000000e+00,
        0.00000000e+00, 3.00000000e+00, 0.00000000e+00],
       [5.10000000e+01, 1.00000000e+00, 1.35000000e+05, 1.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [4.50000000e+01, 0.00000000e+00, 1.10000000e+05, 1.00000000e+00,
        0.00000000e+00, 3.00000000e+00, 0.00000000e+00],
       [3.40000000e+01, 1.00000000e+00, 4.75000000e+04, 0.00000000e+00,
        1.00000000e+00, 0.00000000e+00, 1.00000000e+00],
       [2.60000000e+01, 0.00000000e+00, 4.00000000e+04, 0.00000000e+00,
        1.00000000e+00, 0.00000000e+00, 1.00000000e+00],
       [4.40000000e+01, 1.00000000e+00, 7.50000000e+04, 4.00000000e+00,
        1.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [4.20000000e+01, 1.00000000e+00, 1.05000000e+05, 4.00000000e+00,
        1.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [3.10000000e+01, 1.00000000e+00, 6.50000000e+04, 1.00000000e+00,
        1.00000000e+00, 0.00000000e+00, 1.00000000e+00],
       [4.10000000e+01, 1.00000000e+00, 1.10000000e+05, 2.00000000e+00,
        1.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [2.70000000e+01, 0.00000000e+00, 3.75000000e+04, 3.00000000e+00,
```

In [85]:
```
y_train
```

Out[85]:
```
array(['High', 'High', 'High', 'Average', 'Average', 'High', 'High',
       'Average', 'High', 'Low', 'High', 'High', 'High', 'High', 'High',
       'Average', 'High', 'High', 'Low', 'High', 'Average', 'High',
       'High', 'High', 'High', 'High', 'High', 'High', 'Average', 'High',
       'High', 'High', 'Low', 'High', 'Average', 'High', 'Low', 'High',
       'Low', 'High', 'Average', 'Low', 'High', 'Average', 'High',
       'Average', 'High', 'High', 'Average', 'High', 'Low', 'High',
       'Average', 'High', 'High', 'High', 'Average', 'Average', 'High',
       'High', 'High', 'Average', 'High', 'High', 'Average', 'High',
       'High', 'High', 'Low', 'Average', 'High', 'High', 'High', 'High',
       'Average', 'High', 'High', 'High', 'High', 'High', 'High', 'High',
       'High', 'High', 'Average', 'High', 'High', 'High', 'Average',
       'Average', 'High', 'High', 'High', 'High', 'High', 'High',
       'Average', 'High', 'High', 'High', 'High', 'High', 'Average',
       'High', 'High', 'Average', 'High', 'High', 'High', 'High',
       'Average', 'High', 'High', 'High'], dtype=object)
```

Normalization using MinMaxScaler

In [74]:
```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaler.fit(x_train)
x_train=scaler.transform(x_train)
x_test=scaler.transform(x_test)
```

In [75]:
```
x_train
```

Out[75]:
```
array([[0.71428571, 0.        , 0.68627451, 0.25      , 0.        ,
        1.        , 0.        ],
       [0.92857143, 1.        , 0.84313725, 0.25      , 0.        ,
        0.        , 0.        ],
       [0.71428571, 0.        , 0.64705882, 0.25      , 0.        ,
        1.        , 0.        ],
       [0.32142857, 1.        , 0.15686275, 0.        , 1.        ,
        0.        , 1.        ],
       [0.03571429, 0.        , 0.09803922, 0.        , 1.        ,
        0.        , 1.        ],
       [0.67857143, 1.        , 0.37254902, 1.        , 1.        ,
        0.        , 0.        ],
       [0.60714286, 1.        , 0.60784314, 1.        , 1.        ,
        0.        , 0.        ],
       [0.21428571, 1.        , 0.29411765, 0.25      , 1.        ,
        0.        , 1.        ],
       [0.57142857, 1.        , 0.64705882, 0.5       , 1.        ,
        0.        , 0.        ],
       [0.07142857, 0.        , 0.07843137, 0.75      , 1.        ,
```

Model creation using Naive bayes algorithm

In [77]:
```
from sklearn.naive_bayes import GaussianNB
nb=GaussianNB()
nb.fit(x_train,y_train)
y_pred=nb.predict(x_test)
y_pred
```

Out[77]:
```
array(['Low', 'High', 'High', 'High', 'High', 'Low', 'High', 'High',
       'High', 'High', 'Average', 'Low', 'High', 'High', 'Low', 'High',
       'Average', 'Low', 'High', 'High', 'High', 'Low', 'Low', 'High',
       'High', 'High', 'High', 'High', 'High', 'Average', 'High', 'High',
       'High', 'High', 'High', 'High', 'High', 'Average', 'High', 'Low',
       'High', 'Low', 'Low', 'Low', 'High', 'Average', 'High', 'High',
       'High', 'Low'], dtype='<U7')
```

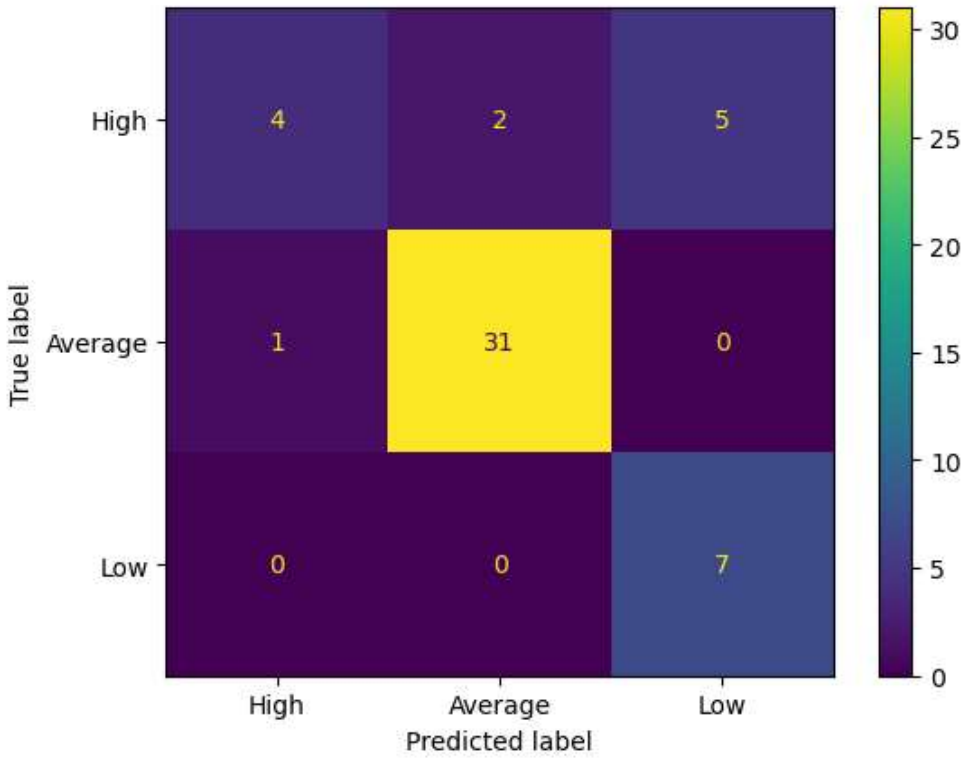
Performance evaluation

In [78]:
```
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay,accuracy_score,cla
cm=confusion_matrix(y_test,y_pred)
cm
```

Out[78]:
```
array([[ 4,  2,  5],
       [ 1, 31,  0],
       [ 0,  0,  7]], dtype=int64)
```

In [79]:
```
label=['High',  'Average', 'Low']
cmd=ConfusionMatrixDisplay(cm,display_labels=label)
cmd.plot()
```

Out[79]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2759c007970>
```



In [80]:
```
score=accuracy_score(y_test,y_pred)
score
```

Out[80]:
```
0.84
```

In [82]:
```
report=classification_report(y_test,y_pred)
print(report)
```

```
              precision    recall  f1-score   support

     Average       0.80      0.36      0.50        11
        High       0.94      0.97      0.95        32
         Low       0.58      1.00      0.74         7

    accuracy                           0.84        50
   macro avg       0.77      0.78      0.73        50
weighted avg       0.86      0.84      0.82        50
```