Loan Eligibility Prediction Using Machine Learing Algorithms

Loan Eligibility in broad terms means your ability / capacity to avail the loan from lender banks/ NBFCs. It's the measurement tool of the banks to decide to lend what quantum of funds on the basis of your financial credentials (Financial credentials means your income, other obligations, additional source of income, etc.

Problem / Objective Statement.

1. The informations / critera needed to get an approval for loan takes alot of time for verification. The reason was because people had to verify the documents provided by customers one after the order. So building a machine learning model was the perfect solution to the problem.

1. Banks also needed a model that would give a very good accuracy score to avoid running after their customers due to refusal to pay their loans on time.

1. Customers also needed fast and urgent replies on there loan approval.

Solution to the Above Statement.

The best option was to build a machine learning model with a high performane to help both the banks and customers for a greater satisfaction.

In [3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [4]:
```
data = pd.read_csv('loan_dataset_train.csv')
```

In [5]:
```
data.head()
```

Out[5]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplicant
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	
4	LP001008	Male	No	0	Graduate	No	6000	

In [6]:
```
data.describe()
```

Out[6]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

In [7]:
```
data.isnull().sum()
```

Out[7]:
```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

In [8]:
```
data.shape
```

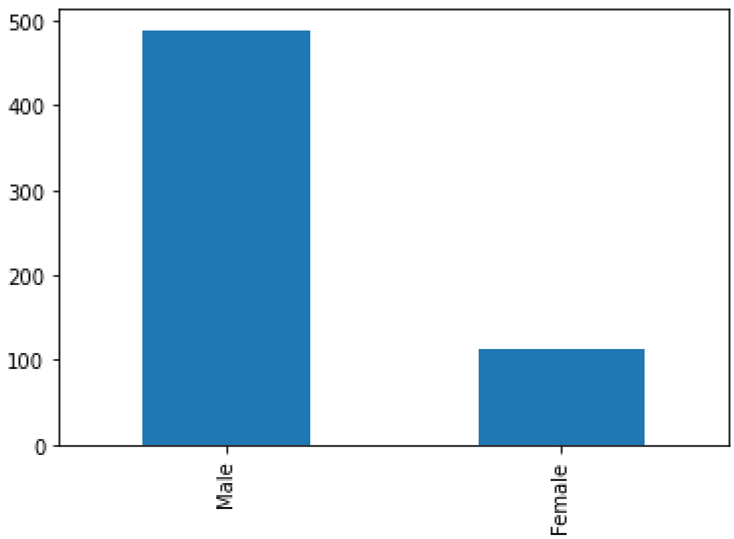
Out[8]:
```
(614, 13)
```

In [9]:
```
data.dtypes
```

Out[9]:
```
Loan_ID              object
Gender               object
Married              object
Dependents           object
Education            object
Self_Employed        object
ApplicantIncome       int64
CoapplicantIncome   float64
LoanAmount          float64
Loan_Amount_Term    float64
Credit_History      float64
Property_Area        object
Loan_Status          object
dtype: object
```

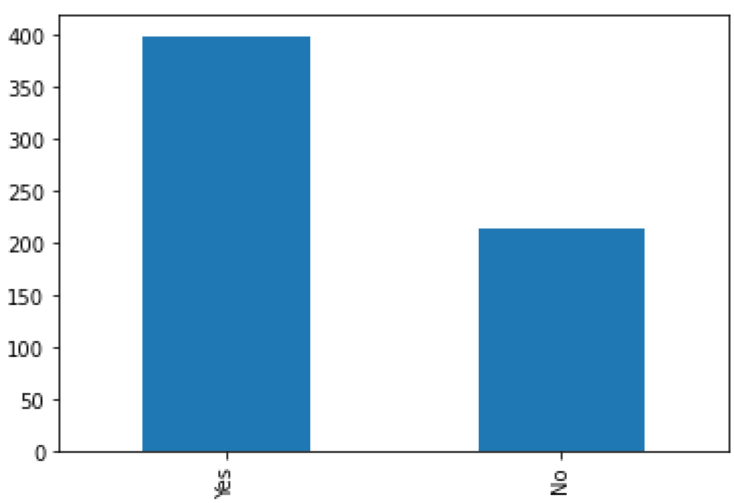
In [10]:
```
data['Gender'].value_counts().plot.bar()
```

Out[10]: <AxesSubplot:>



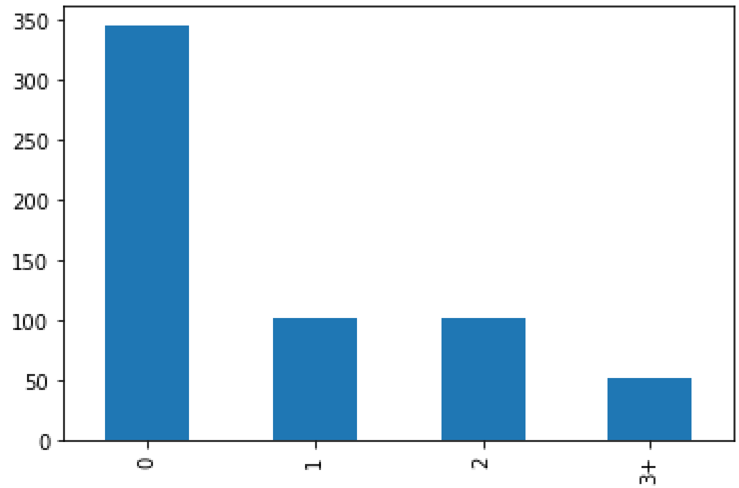
In [9]:
```
data['Married'].value_counts().plot.bar()
```

Out[9]: <AxesSubplot:>



In [11]:
```
data['Dependents'].value_counts().plot.bar()
```

Out[11]: <AxesSubplot:>



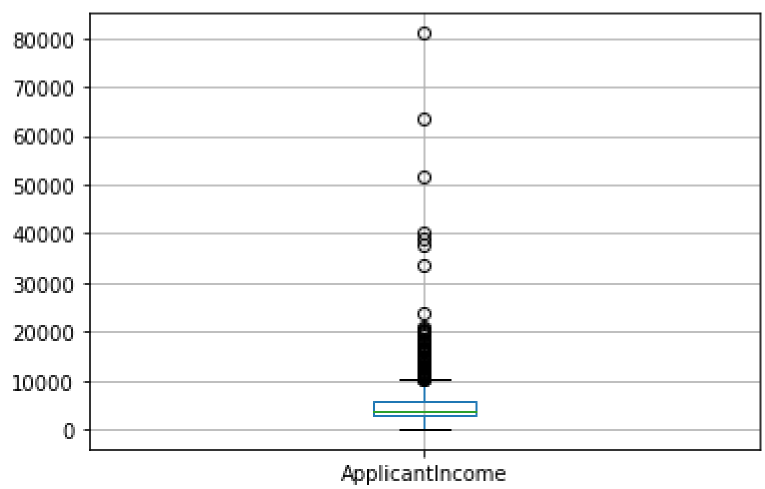
In [12]:
```
pd.crosstab(data['Credit_History'], data['Loan_Status'], margins=True)
```

Out[12]:

Loan_Status	N	Y	All
Credit_History			
0.0	82	7	89
1.0	97	378	475
All	179	385	564

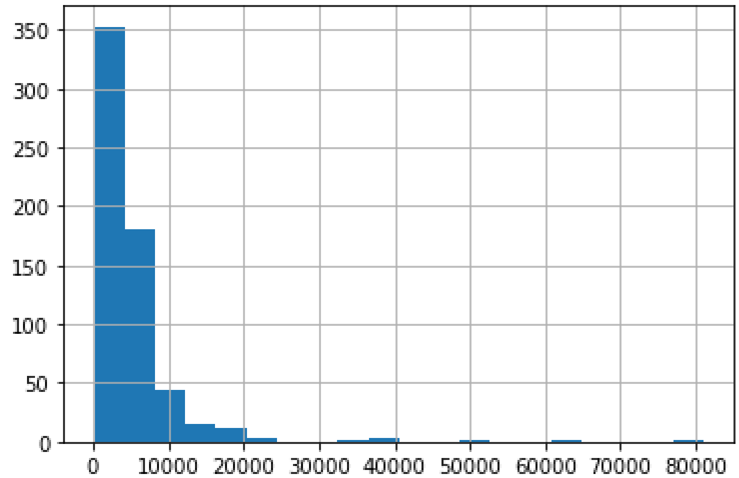
In [13]:
```
data.boxplot('ApplicantIncome')
```

Out[13]: <AxesSubplot:>



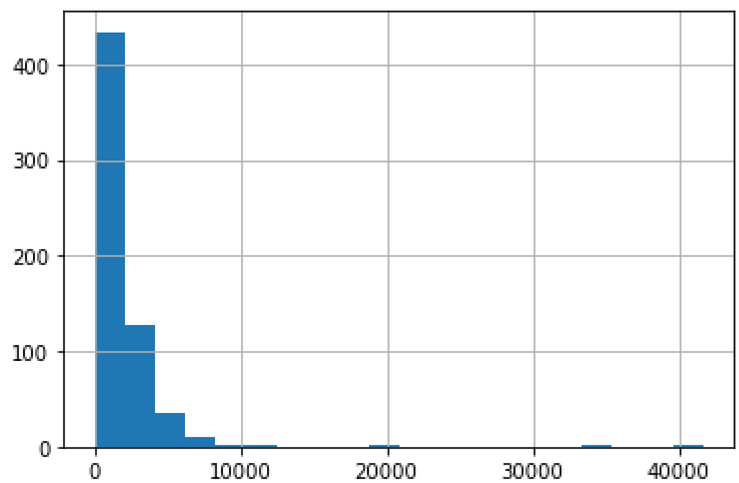
In [14]:
```
data['ApplicantIncome'].hist(bins=20)
```

Out[14]: <AxesSubplot:>



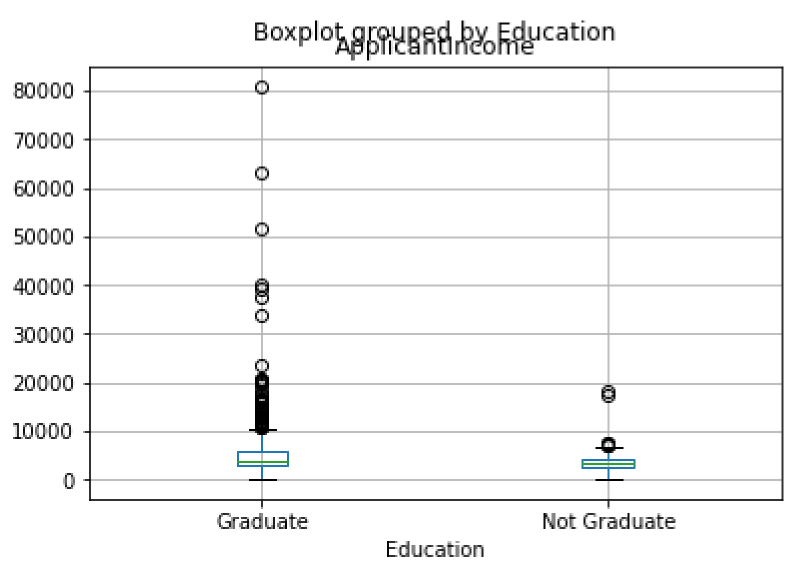
In [15]:
```
data['CoapplicantIncome'].hist(bins=20)
```

Out[15]: <AxesSubplot:>



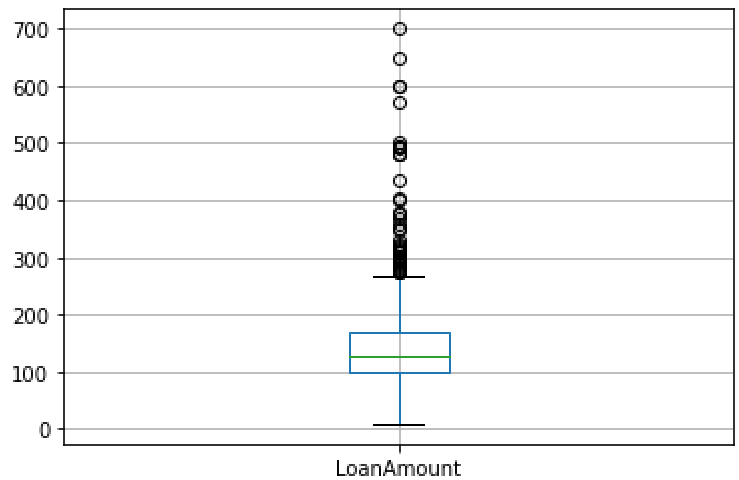
In [16]:
```
data.boxplot('ApplicantIncome', 'Education')
```

Out[16]: <AxesSubplot:title={'center':'ApplicantIncome'}, xlabel='Education'>



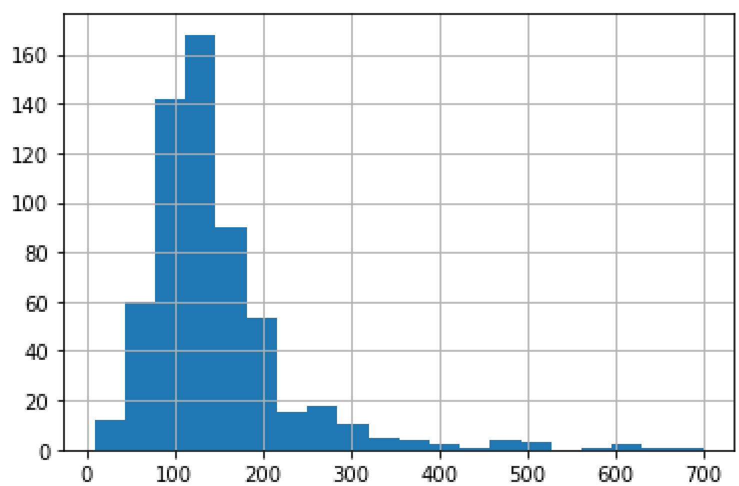
In [17]:
```
data.boxplot('LoanAmount')
```

Out[17]: <AxesSubplot:>



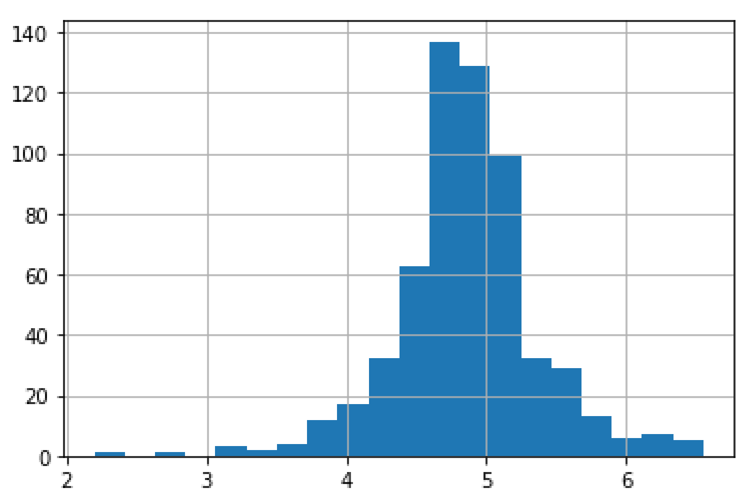
In [18]:
```
data['LoanAmount'].hist(bins=20)
```

Out[18]: <AxesSubplot:>



In [19]:
```
data['LoanAmount_log'] = np.log(data['LoanAmount'])
data['LoanAmount_log'].hist(bins=20)
```

Out[19]: <AxesSubplot:>



In [20]:
```
# fill the missing values for numerical terms - mean
```

```
data['LoanAmount'] = data['LoanAmount'].fillna(data['LoanAmount'].mean())
data['LoanAmount_log'] = data['LoanAmount_log'].fillna(data['LoanAmount_log'].mean())
data['Loan_Amount_Term'] = data['Loan_Amount_Term'].fillna(data['Loan_Amount_Term'].me
data['Credit_History'] = data['Credit_History'].fillna(data['Credit_History'].mean())
```

In [21]:

```
# fill the missing values for categorical terms - mode
data['Gender'] = data["Gender"].fillna(data['Gender'].mode()[0])
data['Married'] = data["Married"].fillna(data['Married'].mode()[0])
data['Dependents'] = data["Dependents"].fillna(data['Dependents'].mode()[0])
data['Self_Employed'] = data["Self_Employed"].fillna(data['Self_Employed'].mode()[0])
```

In [22]:

```
data.isnull().sum()
```

Out[22]:

```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
LoanAmount_log       0
dtype: int64
```

In [23]:

```
data.head()
```

Out[23]:

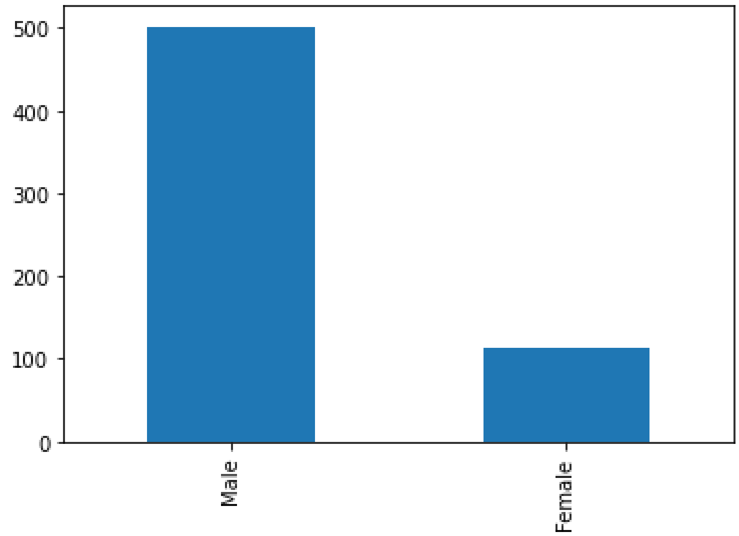
	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplicant
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	
4	LP001008	Male	No	0	Graduate	No	6000	

In [24]:

```
data['Gender'].value_counts().plot.bar()
```

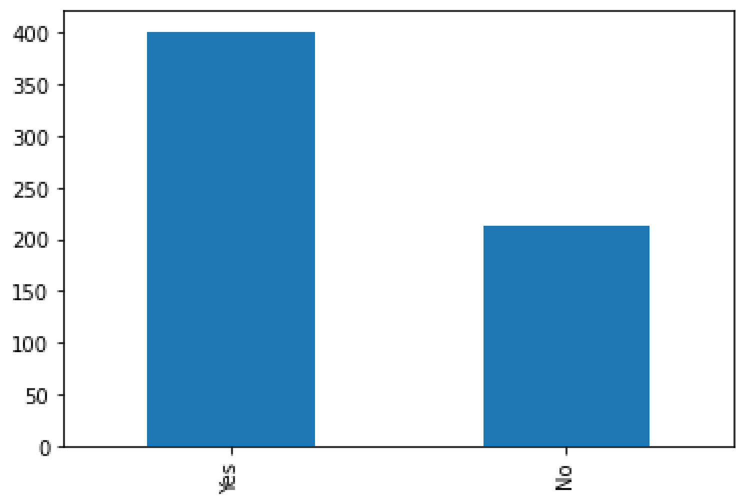
Out[24]:

```
<AxesSubplot:>
```



In [25]:
```
data['Married'].value_counts().plot.bar()
```

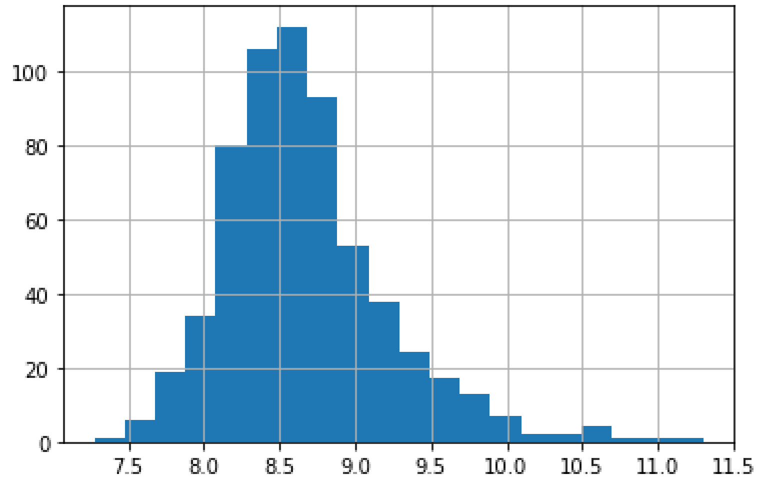
Out[25]: <AxesSubplot:>



In [26]:
```
# total income
data['Total_Income'] = data['ApplicantIncome'] + data['CoapplicantIncome']
data['TotalIncome_log'] = np.log(data['Total_Income'])
```

In [27]:
```
data['TotalIncome_log'].hist(bins=20)
```

Out[27]: <AxesSubplot:>



In [28]:
```
data.head()
```

Out[28]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplicant
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	
4	LP001008	Male	No	0	Graduate	No	6000	

In [29]:
```
X = data.iloc[:,np.r_[1:5,9:11,13:15]].values
y = data.iloc[:,12].values
```

In [30]:
```
X
```

Out[30]:
```
array([['Male', 'No', '0', ..., 1.0, 4.857444178729353, 5849.0],
       ['Male', 'Yes', '1', ..., 1.0, 4.852030263919617, 6091.0],
       ['Male', 'Yes', '0', ..., 1.0, 4.189654742026425, 3000.0],
       ...,
       ['Male', 'Yes', '1', ..., 1.0, 5.53338948872752, 8312.0],
       ['Male', 'Yes', '2', ..., 1.0, 5.231108616854587, 7583.0],
       ['Female', 'No', '0', ..., 0.0, 4.890349128221754, 4583.0]],
      dtype=object)
```

In [31]:
```
y
```

Out[31]:
```
array(['Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y',
       'N', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'N', 'Y', 'N', 'N', 'N', 'Y',
       'Y', 'Y', 'N', 'Y', 'N', 'N', 'N', 'Y', 'N', 'Y', 'N', 'Y', 'Y',
       'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y',
       'N', 'N', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'N',
       'N', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'N', 'N',
       'N', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
       'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N',
       'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'N', 'N', 'N', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'N', 'N', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'N',
       'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'N', 'N', 'N',
       'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y',
       'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'N', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'N',
       'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y',
       'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'N', 'Y', 'N', 'N', 'N', 'N',
       'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y',
       'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N',
       'N', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'N', 'Y', 'Y', 'Y',
       'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y',
       'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'N', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
       'N', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'N', 'Y', 'Y', 'Y',
       'Y', 'N', 'N', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y',
       'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'N', 'N', 'Y',
       'Y', 'N', 'Y', 'Y', 'Y', 'N', 'N', 'N', 'Y', 'N', 'Y', 'N', 'Y',
       'N', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y',
       'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'N', 'N', 'Y', 'N', 'Y', 'Y',
       'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y',
       'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
       'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y',
       'N', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'N', 'N', 'N',
       'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N',
       'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
       'N', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'N', 'N', 'Y', 'N',
       'Y', 'N', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'N',
       'N', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'N',
       'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'N'], dtype=object)
```

In [32]:
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

In [33]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
```

In [34]:
```python
labelencoder_X = LabelEncoder()
for i in range(0, 5):
    X_train[:,i] = labelencoder_X.fit_transform(X_train[:,i])
```

In [35]:
```python
X_train[:,7] = labelencoder_X.fit_transform(X_train[:,7])
```

In [36]:
```
X_train
```

Out[36]:
```
array([[1, 1, 0, ..., 1.0, 4.875197323201151, 267],
       [1, 0, 1, ..., 0.8421985815602837, 5.278114659230517, 407],
       [1, 1, 0, ..., 0.0, 5.003946305945459, 249],
       ...,
       [1, 1, 3, ..., 1.0, 5.298317366548036, 363],
       [1, 1, 0, ..., 1.0, 5.075173815233827, 273],
       [0, 1, 0, ..., 1.0, 5.204006687076795, 301]], dtype=object)
```

In [37]:
```
labelencoder_y = LabelEncoder()
y_train = labelencoder_y.fit_transform(y_train)
```

In [38]:
```
y_train
```

Out[38]:
```
array([1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1,
       1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0,
       0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1,
       0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1,
       1, 1, 1, 0, 1, 0, 1])
```

In [39]:
```
for i in range(0, 5):
    X_test[:,i] = labelencoder_X.fit_transform(X_test[:,i])
```

In [40]:
```
X_test[:,7] = labelencoder_X.fit_transform(X_test[:,7])
```

In [41]:
```
labelencoder_y = LabelEncoder()
y_test = labelencoder_y.fit_transform(y_test)
```

In [42]:
```
X_test
```

Out[42]:
```
array([[1, 0, 0, 0, 6, 1.0, 4.430816798843313, 85],
       [0, 0, 0, 0, 6, 1.0, 4.718498871295094, 28],
       [1, 1, 0, 0, 6, 1.0, 5.780743515792329, 104],
       [1, 1, 0, 0, 6, 1.0, 4.700480365792417, 80],
       [1, 1, 2, 0, 6, 1.0, 4.574710978503383, 22],
       [1, 1, 0, 1, 3, 0.0, 5.10594547390058, 70],
       [1, 1, 3, 0, 3, 1.0, 5.056245805348308, 77],
       [1, 0, 0, 0, 6, 1.0, 6.003887067106539, 114],
       [1, 0, 0, 0, 5, 0.0, 4.820281565605037, 53],
       [1, 1, 0, 0, 6, 1.0, 4.852030263919617, 55],
       [0, 0, 0, 0, 6, 1.0, 4.430816798843313, 4],
       [1, 1, 1, 0, 6, 1.0, 4.553876891600541, 2],
       [0, 0, 0, 0, 6, 1.0, 5.634789603169249, 96],
       [1, 1, 2, 0, 6, 1.0, 5.4638318050256105, 97],
       [1, 1, 0, 0, 6, 1.0, 4.564348191467836, 117],
       [1, 1, 1, 0, 6, 1.0, 4.204692619390966, 22],
       [1, 0, 1, 1, 6, 0.8421985815602837, 5.247024072160486, 32],
       [1, 0, 0, 1, 6, 1.0, 4.882801922586371, 25],
       [0, 0, 0, 0, 6, 0.8421985815602837, 4.532599493153256, 1],
       [1, 1, 0, 1, 6, 0.0, 5.198497031265826, 44],
       [0, 1, 0, 0, 6, 0.0, 4.787491742782046, 71],
       [1, 1, 0, 0, 6, 1.0, 4.962844630259907, 43],
       [1, 1, 2, 0, 6, 1.0, 4.68213122712422, 91],
       [1, 1, 2, 0, 6, 1.0, 5.10594547390058, 111],
       [1, 1, 0, 0, 6, 0.8421985815602837, 4.060443010546419, 35],
       [1, 1, 1, 0, 6, 1.0, 5.521460917862246, 94],
       [1, 0, 0, 0, 6, 1.0, 5.231108616854587, 98],
       [1, 1, 0, 0, 6, 1.0, 5.231108616854587, 110],
       [1, 1, 3, 0, 6, 0.0, 4.852030263919617, 41],
       [0, 0, 0, 0, 6, 0.0, 4.634728988229636, 50],
       [1, 1, 0, 0, 6, 1.0, 5.429345628954441, 99],
       [1, 0, 0, 1, 6, 1.0, 3.871201010907891, 46],
       [1, 1, 1, 1, 6, 1.0, 4.499809670330265, 52],
       [1, 1, 0, 0, 6, 1.0, 5.19295685089021, 102],
       [1, 1, 0, 0, 6, 1.0, 4.857444178729353, 95],
       [0, 1, 0, 1, 6, 0.0, 5.181783550292085, 57],
       [1, 1, 0, 0, 6, 1.0, 5.147494476813453, 65],
       [1, 0, 0, 1, 6, 1.0, 4.836281906951478, 39],
       [1, 1, 0, 0, 6, 1.0, 4.852030263919617, 75],
       [1, 1, 2, 1, 6, 1.0, 4.68213122712422, 24],
       [0, 0, 0, 0, 6, 1.0, 4.382026634673881, 9],
       [1, 1, 3, 0, 6, 0.0, 4.812184355372417, 68],
       [1, 1, 2, 0, 2, 1.0, 2.833213344056216, 0],
       [1, 1, 1, 1, 6, 1.0, 5.062595033026967, 67],
       [1, 0, 0, 0, 6, 1.0, 4.330733340286331, 21],
       [1, 0, 0, 0, 6, 1.0, 5.231108616854587, 113],
       [1, 1, 1, 0, 6, 1.0, 4.7535901911063645, 18],
       [0, 0, 0, 0, 6, 1.0, 4.74493212836325, 37],
       [1, 1, 1, 0, 6, 1.0, 4.852030263919617, 72],
       [1, 0, 0, 0, 6, 1.0, 4.941642422609304, 78],
       [1, 1, 3, 1, 6, 1.0, 4.30406509320417, 8],
       [1, 1, 0, 0, 6, 1.0, 4.867534450455582, 84],
       [1, 1, 0, 1, 6, 1.0, 4.672828834461906, 31],
       [1, 0, 0, 0, 6, 1.0, 4.857444178729353, 61],
       [1, 1, 0, 0, 6, 1.0, 4.718498871295094, 19],
       [1, 1, 0, 0, 6, 0.8421985815602837, 5.556828061699537, 107],
       [1, 1, 0, 0, 6, 1.0, 4.553876891600541, 34],
       [1, 0, 0, 1, 6, 1.0, 4.890349128221754, 74],
       [1, 1, 2, 0, 6, 1.0, 5.123963979403259, 62],
       [1, 0, 0, 0, 6, 1.0, 4.787491742782046, 27],
```

```
[0, 0, 0, 0, 6, 0.0, 4.919980925828125, 108],
[0, 0, 0, 0, 6, 1.0, 5.365976015021851, 103],
[1, 1, 0, 1, 6, 1.0, 4.74493212836325, 38],
[0, 0, 0, 0, 6, 0.0, 4.330733340286331, 13],
[1, 1, 2, 0, 6, 1.0, 4.890349128221754, 69],
[1, 1, 1, 0, 6, 1.0, 5.752572638825633, 112],
[1, 1, 0, 0, 6, 0.8421985815602837, 5.075173815233827, 73],
[1, 0, 0, 0, 6, 1.0, 4.912654885736052, 47],
[1, 1, 0, 0, 5, 1.0, 5.204006687076795, 81],
[1, 0, 0, 1, 6, 0.8421985815602837, 4.564348191467836, 60],
[1, 0, 0, 0, 6, 0.8421985815602837, 4.204692619390966, 83],
[0, 1, 0, 0, 6, 1.0, 4.867534450455582, 5],
[1, 1, 2, 1, 6, 1.0, 5.056245805348308, 58],
[1, 1, 1, 1, 3, 1.0, 4.919980925828125, 79],
[0, 1, 0, 0, 6, 1.0, 4.969813299576001, 54],
[1, 1, 0, 1, 4, 1.0, 4.820281565605037, 56],
[1, 0, 0, 0, 6, 1.0, 4.499809670330265, 120],
[1, 0, 3, 0, 6, 1.0, 5.768320995793772, 118],
[1, 1, 2, 0, 6, 1.0, 4.718498871295094, 101],
[0, 0, 0, 0, 6, 0.0, 4.7535901911063645, 26],
[0, 0, 0, 0, 7, 1.0, 4.727387818712341, 33],
[1, 1, 1, 0, 6, 1.0, 6.214608098422191, 119],
[0, 0, 0, 0, 6, 1.0, 5.267858159063328, 89],
[1, 1, 2, 0, 6, 1.0, 5.231108616854587, 92],
[1, 0, 0, 0, 7, 1.0, 4.2626798770413155, 6],
[1, 1, 0, 0, 0, 0.8421985815602837, 4.709530201312334, 90],
[1, 1, 0, 0, 6, 1.0, 4.700480365792417, 45],
[1, 1, 2, 0, 6, 1.0, 5.298317366548036, 109],
[1, 0, 1, 0, 3, 1.0, 4.727387818712341, 17],
[1, 1, 1, 0, 6, 1.0, 4.6443908991413725, 36],
[0, 1, 0, 1, 6, 1.0, 4.605170185988092, 16],
[1, 0, 0, 0, 6, 1.0, 4.30406509320417, 7],
[1, 1, 1, 0, 1, 1.0, 5.147494476813453, 88],
[1, 1, 3, 0, 4, 0.0, 5.19295685089021, 87],
[0, 0, 0, 0, 6, 1.0, 4.2626798770413155, 3],
[1, 0, 0, 1, 3, 0.0, 4.836281906951478, 59],
[1, 0, 0, 0, 3, 1.0, 5.1647859739235145, 82],
[1, 0, 0, 0, 6, 1.0, 4.969813299576001, 66],
[1, 1, 2, 1, 6, 1.0, 4.394449154672439, 51],
[1, 1, 1, 0, 6, 1.0, 5.231108616854587, 100],
[1, 1, 0, 0, 6, 1.0, 5.351858133476067, 93],
[1, 1, 0, 0, 6, 1.0, 4.605170185988092, 15],
[1, 1, 2, 0, 6, 1.0, 4.787491742782046, 106],
[1, 0, 0, 0, 3, 1.0, 4.787491742782046, 105],
[1, 1, 3, 0, 6, 1.0, 4.852030263919617, 64],
[1, 0, 0, 0, 6, 1.0, 4.8283137373023015, 49],
[1, 0, 0, 1, 6, 1.0, 4.6443908991413725, 42],
[0, 0, 0, 0, 5, 1.0, 4.477336814478207, 10],
[1, 1, 0, 1, 6, 1.0, 4.553876891600541, 20],
[1, 1, 3, 1, 3, 1.0, 4.394449154672439, 14],
[1, 0, 0, 0, 6, 1.0, 5.298317366548036, 76],
[0, 0, 0, 0, 6, 1.0, 4.90527477843843, 11],
[1, 0, 0, 0, 7, 1.0, 4.727387818712341, 18],
[1, 1, 2, 0, 6, 1.0, 4.248495242049359, 23],
[1, 1, 0, 1, 6, 0.0, 5.303304908059076, 63],
[1, 1, 0, 0, 3, 0.0, 4.499809670330265, 48],
[0, 0, 0, 0, 6, 0.8421985815602837, 4.430816798843313, 30],
[1, 0, 0, 0, 6, 1.0, 4.897839799950911, 29],
[1, 1, 2, 0, 6, 1.0, 5.170483995038151, 86],
[1, 1, 3, 0, 6, 1.0, 4.867534450455582, 115],
```

```
       [1, 1, 0, 0, 6, 1.0, 6.077642243349034, 116],
       [1, 1, 3, 1, 3, 0.0, 4.248495242049359, 40],
       [1, 1, 1, 0, 6, 1.0, 4.564348191467836, 12]], dtype=object)
```

In [43]:
```
y_test
```

Out[43]:
```
array([1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1,
       1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1])
```

In [44]:
```
from sklearn.preprocessing import StandardScaler
stand = StandardScaler()
X_Train_stand = stand.fit_transform(X_train)
X_test_stand = stand.fit_transform(X_test)
```

In [45]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
DTClassifier = DecisionTreeClassifier(criterion='entropy', random_state=0)
DTClassifier.fit(X_train, y_train)
```

Out[45]:
```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

In [46]:
```
y_pred = DTClassifier.predict(X_test)
y_pred
```

Out[46]:
```
array([1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0,
       1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0,
       0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0])
```

In [47]:
```
metrics.accuracy_score(y_pred, y_test)*100
```

Out[47]:
```
57.72357723577236
```

In [48]:
```
from sklearn.naive_bayes import GaussianNB
NBClassifier = GaussianNB()
NBClassifier.fit(X_train, y_train)
```

Out[48]:
```
GaussianNB()
```

In [49]:
```
y_pred = NBClassifier.predict(X_test)
y_pred
```

Out[49]:
```
array([1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1])
```

In [50]:
```
metrics.accuracy_score(y_test, y_pred)*100
```

Out[50]:
```
82.92682926829268
```