Loan Eligibility Prediction - Machine Learning Project

July 6, 2024

1 Data Cleaning

1.1 Understanding Data

[1]:
```
# Importing necessary packages

import pandas as pd # working with data sets
import numpy as np # perform mathematical operations on arrays
from numpy import r_
import matplotlib.pyplot as plt # visualizing data
%matplotlib inline
```

[2]:
```
# Importing train.CSV data set
dataset = pd.read_csv("loan-train.csv")
```

[3]:
```
# rows and columns count
dataset.shape
```

[3]:
```
(614, 13)
```

[4]:
```
# data view
dataset.head()
```

[4]:
```
    Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001002   Male      No          0      Graduate            No
1  LP001003   Male     Yes          1      Graduate            No
2  LP001005   Male     Yes          0      Graduate           Yes
3  LP001006   Male     Yes          0  Not Graduate            No
4  LP001008   Male      No          0      Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5849                0.0         NaN             360.0
1             4583             1508.0       128.0             360.0
2             3000                0.0        66.0             360.0
3             2583             2358.0       120.0             360.0
4             6000                0.0       141.0             360.0

   Credit_History Property_Area Loan_Status
```

```
0             1.0         Urban           Y
1             1.0         Rural           N
2             1.0         Urban           Y
3             1.0         Urban           Y
4             1.0         Urban           Y
```

[5]:
```
# data general info
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

[6]:
```
# data description
dataset.describe()
```

[6]:
```
       ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
count       614.000000         614.000000  592.000000         600.00000
mean       5403.459283        1621.245798  146.412162         342.00000
std        6109.041673        2926.248369   85.587325          65.12041
min         150.000000           0.000000    9.000000          12.00000
25%        2877.500000           0.000000  100.000000         360.00000
50%        3812.500000        1188.500000  128.000000         360.00000
75%        5795.000000        2297.250000  168.000000         360.00000
max       81000.000000       41667.000000  700.000000         480.00000

       Credit_History
count      564.000000
mean         0.842199
std          0.364878
min          0.000000
```

```
25%            1.000000
50%            1.000000
75%            1.000000
max            1.000000
```

1.2 Exploring & Normalizing Data

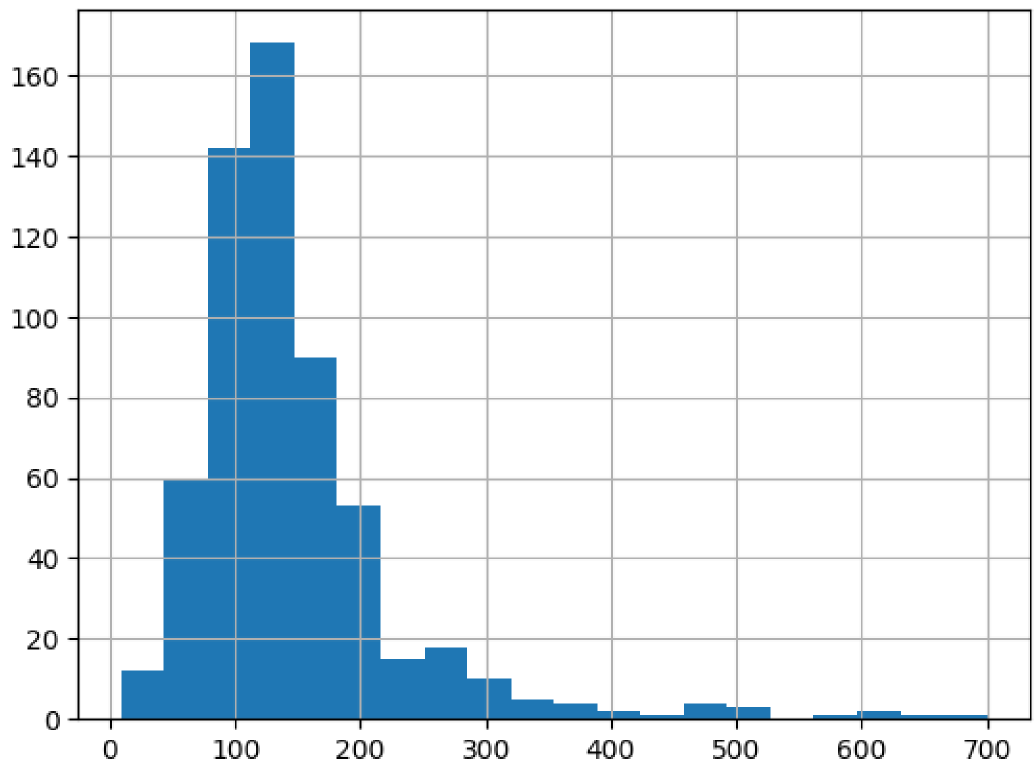
[7]:
```
# Plotting a crosstab to see the effects of credit history on loan status for␣
↪each applicant

pd.crosstab(dataset['Credit_History'], dataset['Loan_Status'], margins=True)
```

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

From the above graph we can see that applicants with credit history of 1.0 are more eligible than 0.0 to take a loan.

[8]:
```
# Plotting a histogram for loan amount variable

dataset['LoanAmount'].hist(bins=20)
```

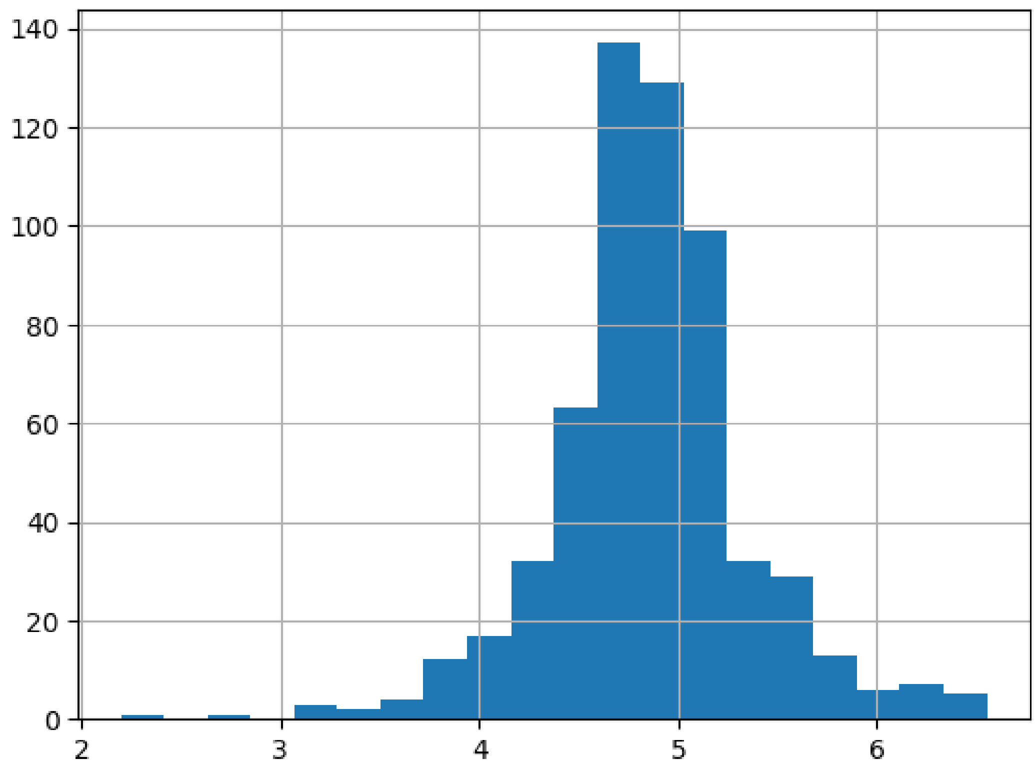
[8]:
```
<Axes: >
```



We can see that the loan amount variable is a little right skewes and to normalize it we will be using the log function.

[9]:
```
# Applying log function to loan amount variable using numpy

dataset['LoanAmount_log']=np.log(dataset['LoanAmount'])

dataset['LoanAmount_log'].hist(bins=20) # Histogram for loan amount log␣
  ↪variable
```

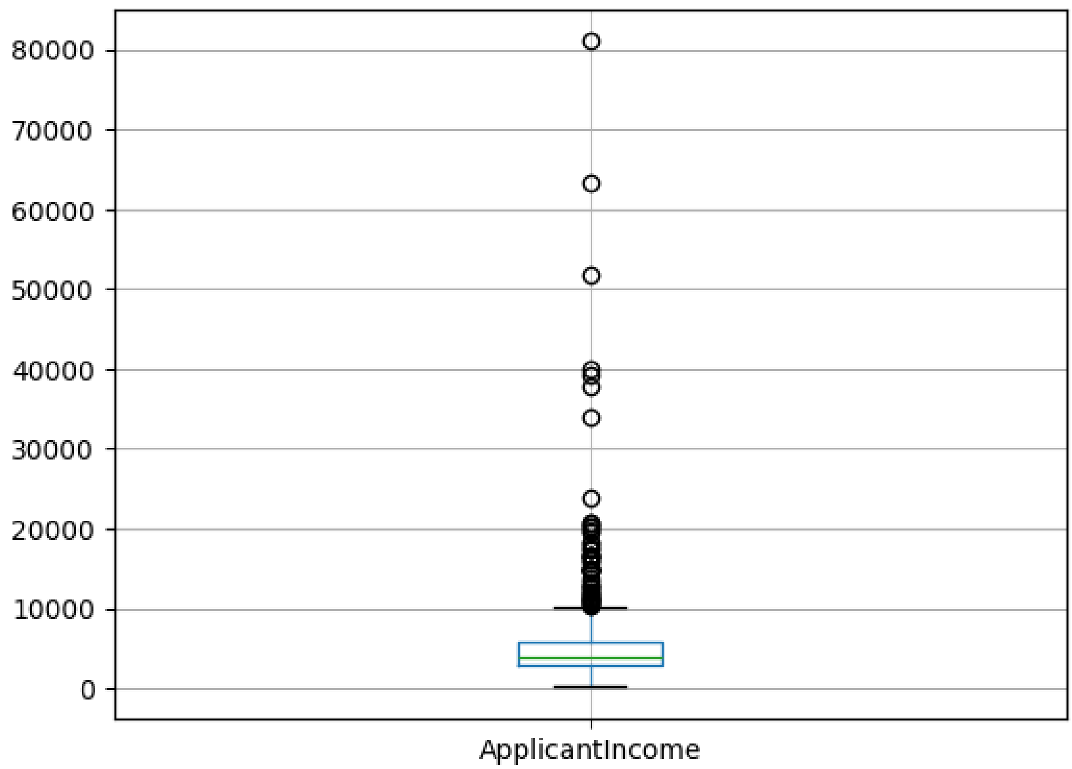
[9]: <Axes: >



We can see that the loan amount log variable histogram is now normalized.

[10]:
```
# Plotting a boxplot for applicant and their income

dataset.boxplot(column='ApplicantIncome')
```

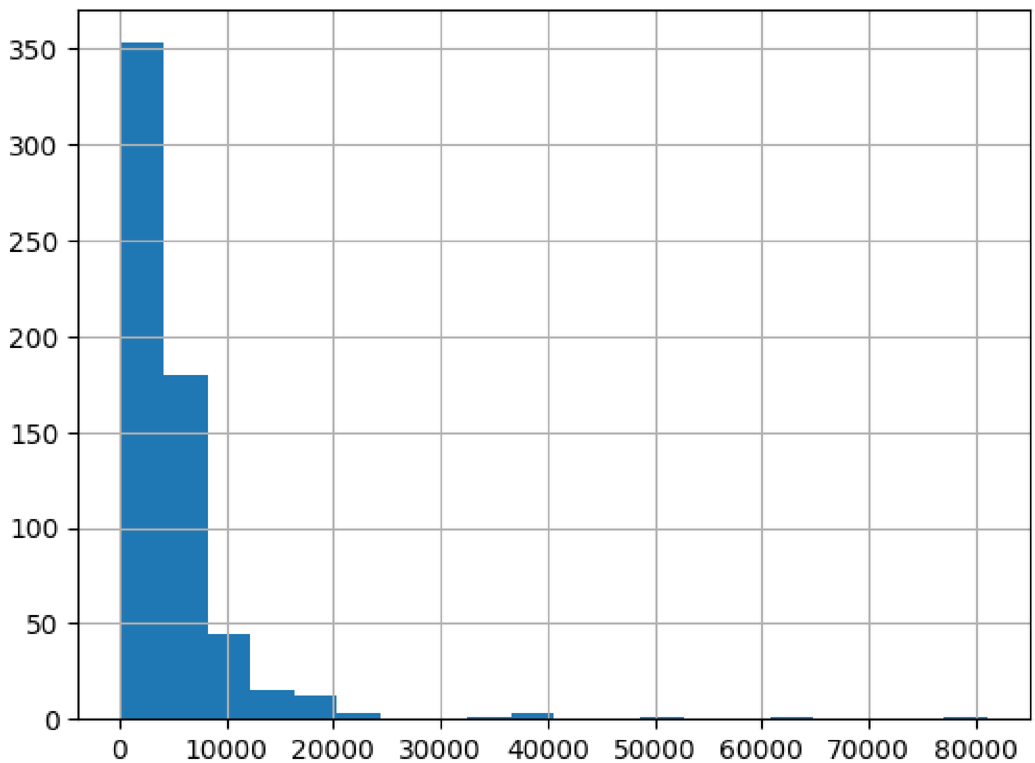
[10]: <Axes: >



We can see that there are lot of outliers for this variable which we have to handle for better data analysis and prediction.

[11]:
```
# Plotting a histogram for applicant income

dataset['ApplicantIncome'].hist(bins=20)
```

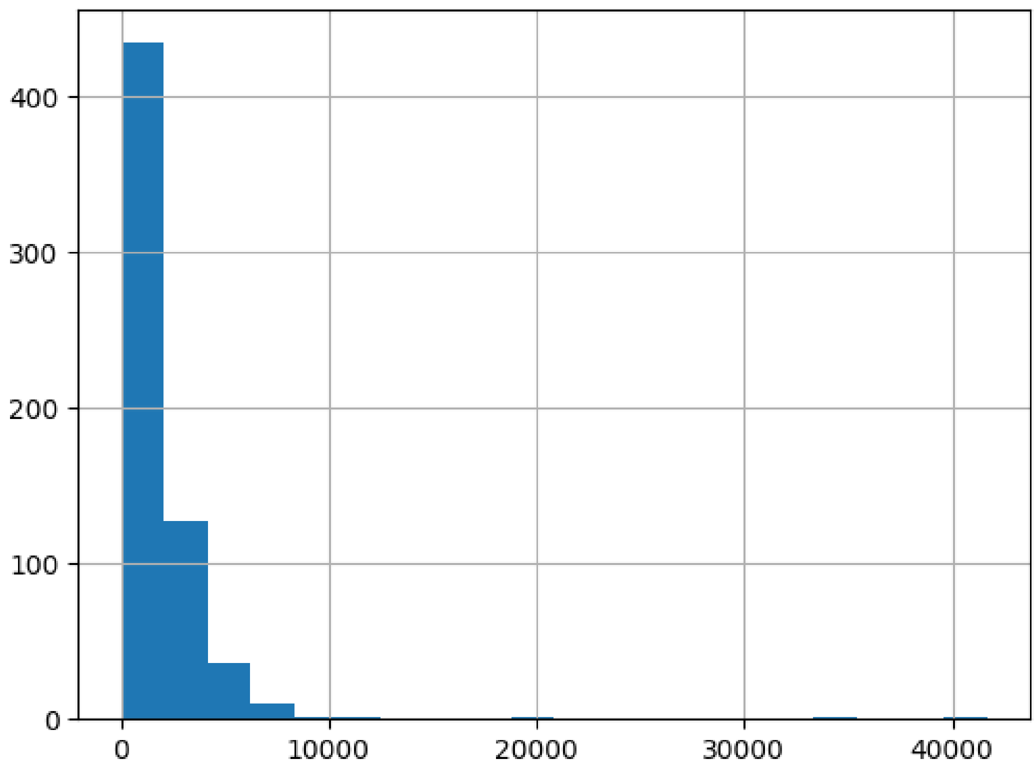
[11]: <Axes: >



We can see that its a right skewes histogram and moving forward we have to normalize the values.

[12]:
```python
# Plotting a histogram for coapplicant income

dataset['CoapplicantIncome'].hist(bins=20)
```

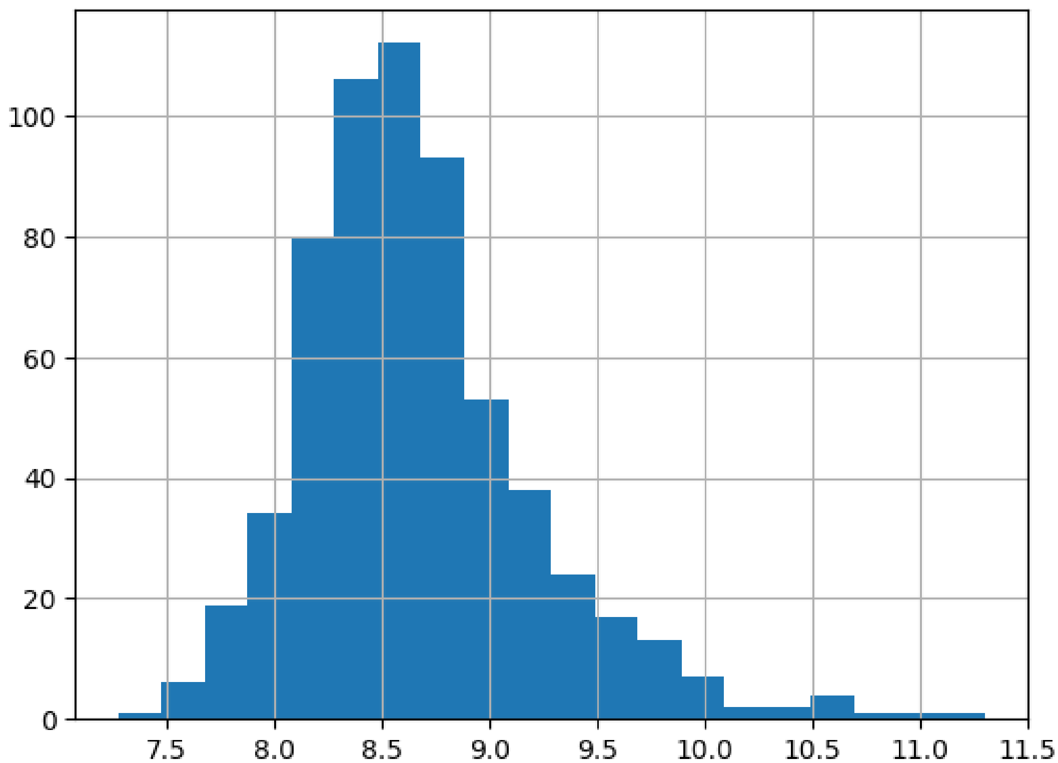
[12]: <Axes: >



Similarly we can see that the coapplicants income graph is also right skewed. So, we have to normalize it as well.

[13]:
```
# Applying log function to the sum of applicant and coapplicant income i.e.␣
↪total income using numpy

dataset['TotalIncome']= dataset['ApplicantIncome'] +␣
↪dataset['CoapplicantIncome']
dataset['TotalIncome_log']= np.log(dataset['TotalIncome'])

dataset['TotalIncome_log'].hist(bins=20) # Histogram for total income log␣
↪variable
```

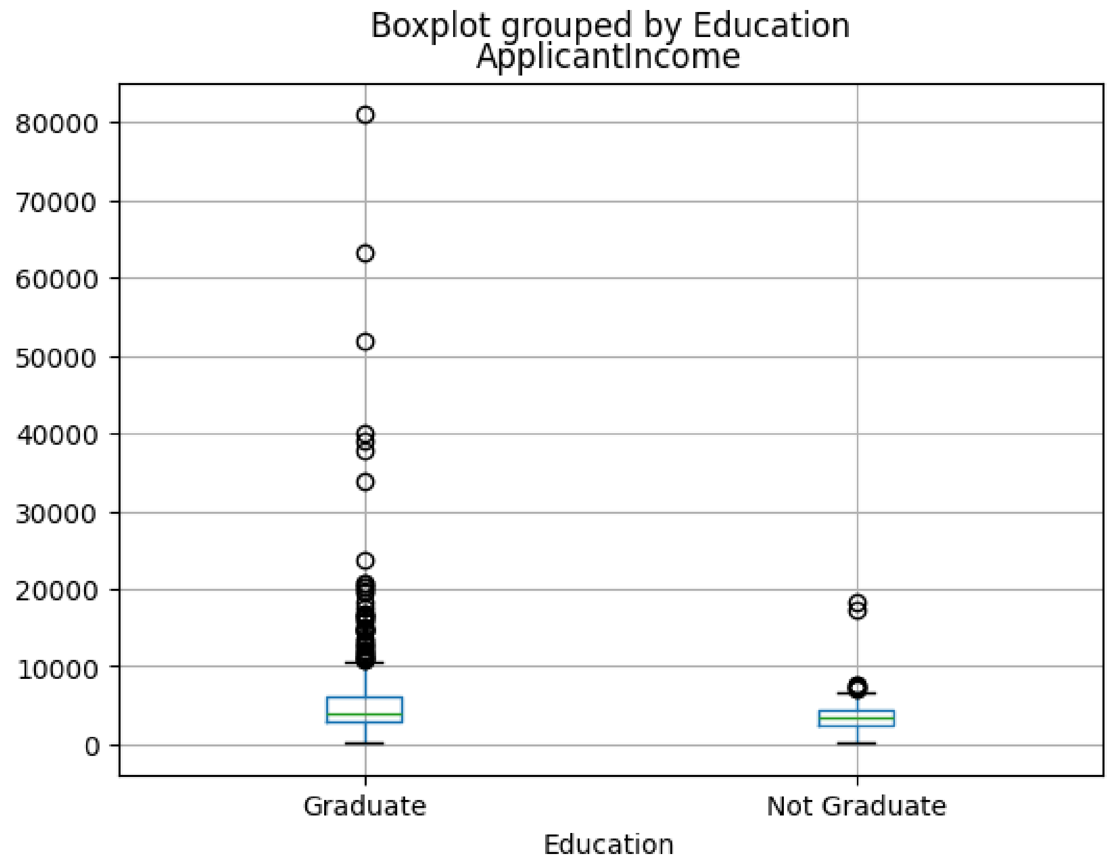
[13]:
```
<Axes: >
```



We can see that the sum of applicant and coapplicant income i.e. total income log variable is now normalized.

[14]:
```python
# Plotting a boxplot to see the relationship between applicants income and␣
↪their education

dataset.boxplot(column='ApplicantIncome', by= 'Education')
```

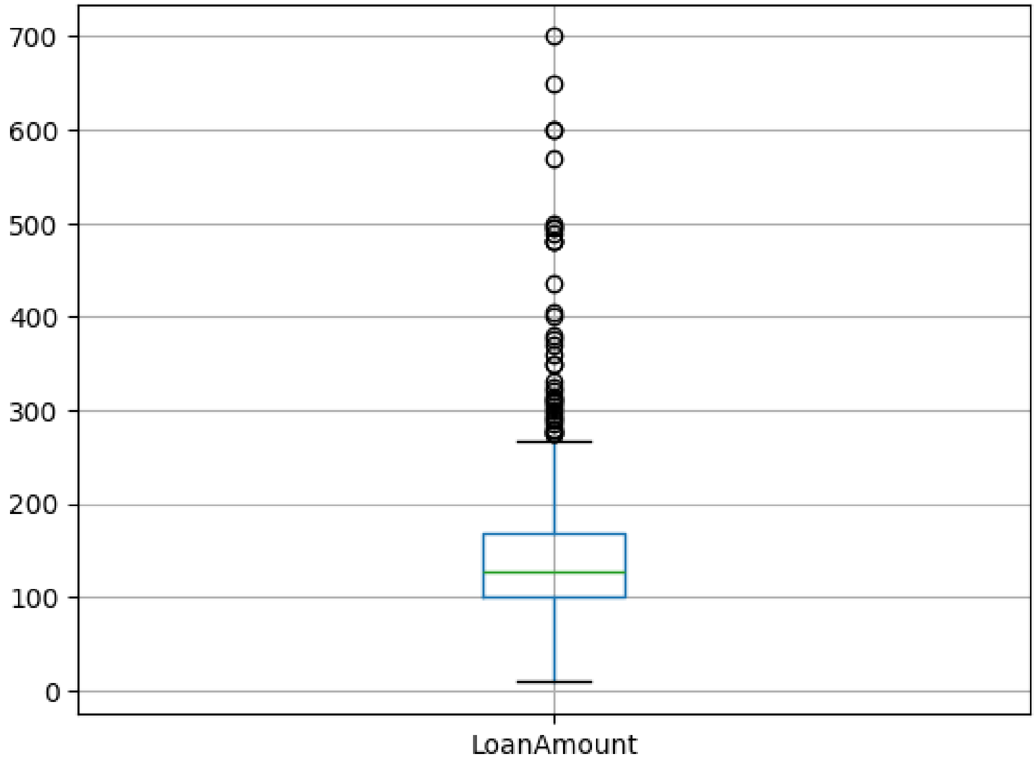
[14]:
```
<Axes: title={'center': 'ApplicantIncome'}, xlabel='Education'>
```



We can see that the median salary do not vary much for both graduates and non-graduates where as some graduates have higher salary as compared to the non graduates. These kinds of differences are quite common in such variables so normalizing and scaling is necessary to avoid bias.

[15]:
```
# plotting a boxplot for loan amount variable

dataset.boxplot(column='LoanAmount')
```

[15]: <Axes: >



We can see that there are outliers as before which means in general most of our variables have outliers which needs to be handled.

1.3 Fixing Null Values

[16]:
```
# check null values

dataset.isnull().sum()
```

[16]:
```
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
```

```
LoanAmount_log       22
TotalIncome           0
TotalIncome_log       0
dtype: int64
```

[17]:
```python
# Filling missing values using mode function in dataset

dataset['Gender'] = dataset['Gender'].fillna(dataset['Gender'].mode()[0])
dataset['Married'] = dataset['Married'].fillna(dataset['Married'].mode()[0])
dataset['Dependents'] = dataset['Dependents'].fillna(dataset['Dependents'].
 ↪mode()[0])
dataset['Self_Employed'] = dataset['Self_Employed'].
 ↪fillna(dataset['Self_Employed'].mode()[0])
dataset['Loan_Amount_Term'] = dataset['Loan_Amount_Term'].
 ↪fillna(dataset['Loan_Amount_Term'].mode()[0])
dataset['Credit_History'] = dataset['Credit_History'].
 ↪fillna(dataset['Credit_History'].mode()[0])
```

[18]:
```python
# Importing simpleimputer from sklearn

from sklearn.impute import SimpleImputer

# filling missing values using mean strategy in dataset

imputer = SimpleImputer(strategy='mean')
dataset['LoanAmount'] = imputer.fit_transform(dataset[['LoanAmount']])
dataset['LoanAmount_log'] = imputer.fit_transform(dataset[['LoanAmount_log']])
```

[19]:
```python
# check null values

dataset.isnull().sum()
```

[19]:
```
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
TotalIncome          0
```

```
TotalIncome_log      0
dtype: int64
```

[20]:
```
# data view

dataset.head()
```

[20]:
```
    Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001002   Male      No          0      Graduate            No
1  LP001003   Male     Yes          1      Graduate            No
2  LP001005   Male     Yes          0      Graduate           Yes
3  LP001006   Male     Yes          0  Not Graduate            No
4  LP001008   Male      No          0      Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5849                0.0  146.412162             360.0
1             4583             1508.0  128.000000             360.0
2             3000                0.0   66.000000             360.0
3             2583             2358.0  120.000000             360.0
4             6000                0.0  141.000000             360.0

   Credit_History Property_Area Loan_Status  LoanAmount_log  TotalIncome  \
0             1.0         Urban           Y        4.857444       5849.0
1             1.0         Rural           N        4.852030       6091.0
2             1.0         Urban           Y        4.189655       3000.0
3             1.0         Urban           Y        4.787492       4941.0
4             1.0         Urban           Y        4.948760       6000.0

   TotalIncome_log
0         8.674026
1         8.714568
2         8.006368
3         8.505323
4         8.699515
```

2 Data Modeling

2.1 Classifying into Dependent & Independent Variables

[21]:
```
# Classifying the variables into dependent(y) and independent(x) variables

x= dataset.iloc[:,np.r_[1:5,9:11, 13:15]].values
y= dataset.iloc[:,12].values
```

[22]:
```
x
```

```
[22]: array([['Male', 'No', '0', …, 1.0, 4.857444178729352, 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)
```

```
[23]: y
```

```
[23]: 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)
```

2.2 Spliting Variables into Test & Train Datasets

[24]:
```python
# Importing train test split function from sklearn

from sklearn.model_selection import train_test_split

# Spliting the data set into train and test data sets

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
 ↪random_state=0)
```

[25]:
```python
# checking x_train data set

print(x_train)
```

```
[['Male' 'Yes' '0' … 1.0 4.875197323201151 5858.0]
 ['Male' 'No' '1' … 1.0 5.278114659230517 11250.0]
 ['Male' 'Yes' '0' … 0.0 5.003946305945459 5681.0]
 …
 ['Male' 'Yes' '3+' … 1.0 5.298317366548036 8334.0]
 ['Male' 'Yes' '0' … 1.0 5.075173815233827 6033.0]
 ['Female' 'Yes' '0' … 1.0 5.204006687076795 6486.0]]
```

We can see some categorical values in the data set but we need to convert them into "0's" and "1's" for the machine to understand.

2.3 Converting Categorical Values into Numerical Format

[26]:
```python
# importing data labelencoder from sklearn

from sklearn.preprocessing import LabelEncoder

# converting categorical values into numerical format using labelencoder
```

```
labelencoder_x = LabelEncoder()
```

[27]:
```
# Assigning labelencoder instance to dataset index

for i in range(0,5):
    x_train[:,i]= labelencoder_x.fit_transform(x_train[:,i])
```

[28]:
```
x_train[:,7]= labelencoder_x.fit_transform(x_train[:,7])
```

[29]:
```
# View x_train data set

x_train
```

[29]:
```
array([[1, 1, 0, …, 1.0, 4.875197323201151, 267],
       [1, 0, 1, …, 1.0, 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)
```

[63]:
```
# converting categorical values into numerical format using labelencoder

labelencoder_y = LabelEncoder()

# Assigning labelencoder instance to dataset index

y_train= labelencoder_y.fit_transform(y_train)
```

[31]:
```
# View y_train data set

y_train
```

[31]:
```
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])
```

[32]:
```
# Assigning labelencoder instance to dataset index

for i in range(0,5):
    x_test[:,i]= labelencoder_x.fit_transform(x_test[:,i])
```

[33]:
```
x_test[:,7]= labelencoder_x.fit_transform(x_test[:,7])
```

[34]:
```
# Assigning labelencoder instance to dataset index

labelencoder_y = LabelEncoder()
y_test= labelencoder_y.fit_transform(y_test)
```

[35]:
```
# View x_test data set

x_test
```

[35]:
```
array([[1, 0, 0, 0, 5, 1.0, 4.430816798843313, 85],
       [0, 0, 0, 0, 5, 1.0, 4.718498871295094, 28],
       [1, 1, 0, 0, 5, 1.0, 5.780743515792329, 104],
       [1, 1, 0, 0, 5, 1.0, 4.700480365792417, 80],
       [1, 1, 2, 0, 5, 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, 5, 1.0, 6.003887067106539, 114],
       [1, 0, 0, 0, 5, 0.0, 4.820281565605037, 53],
       [1, 1, 0, 0, 5, 1.0, 4.852030263919617, 55],
       [0, 0, 0, 0, 5, 1.0, 4.430816798843313, 4],
       [1, 1, 1, 0, 5, 1.0, 4.553876891600541, 2],
       [0, 0, 0, 0, 5, 1.0, 5.634789603169249, 96],
       [1, 1, 2, 0, 5, 1.0, 5.4638318050256105, 97],
       [1, 1, 0, 0, 5, 1.0, 4.564348191467836, 117],
       [1, 1, 1, 0, 5, 1.0, 4.204692619390966, 22],
       [1, 0, 1, 1, 5, 1.0, 5.247024072160486, 32],
       [1, 0, 0, 1, 5, 1.0, 4.882801922586371, 25],
       [0, 0, 0, 0, 5, 1.0, 4.532599493153256, 1],
       [1, 1, 0, 1, 5, 0.0, 5.198497031265826, 44],
       [0, 1, 0, 0, 5, 0.0, 4.787491742782046, 71],
```

```
[1, 1, 0, 0, 5, 1.0, 4.962844630259907, 43],
[1, 1, 2, 0, 5, 1.0, 4.68213122712422, 91],
[1, 1, 2, 0, 5, 1.0, 5.10594547390058, 111],
[1, 1, 0, 0, 5, 1.0, 4.060443010546419, 35],
[1, 1, 1, 0, 5, 1.0, 5.521460917862246, 94],
[1, 0, 0, 0, 5, 1.0, 5.231108616854587, 98],
[1, 1, 0, 0, 5, 1.0, 5.231108616854587, 110],
[1, 1, 3, 0, 5, 0.0, 4.852030263919617, 41],
[0, 0, 0, 0, 5, 0.0, 4.634728988229636, 50],
[1, 1, 0, 0, 5, 1.0, 5.429345628954441, 99],
[1, 0, 0, 1, 5, 1.0, 3.871201010907891, 46],
[1, 1, 1, 1, 5, 1.0, 4.499809670330265, 52],
[1, 1, 0, 0, 5, 1.0, 5.19295685089021, 102],
[1, 1, 0, 0, 5, 1.0, 4.857444178729352, 95],
[0, 1, 0, 1, 5, 0.0, 5.181783550292085, 57],
[1, 1, 0, 0, 5, 1.0, 5.147494476813453, 65],
[1, 0, 0, 1, 5, 1.0, 4.836281906951478, 39],
[1, 1, 0, 0, 5, 1.0, 4.852030263919617, 75],
[1, 1, 2, 1, 5, 1.0, 4.68213122712422, 24],
[0, 0, 0, 0, 5, 1.0, 4.382026634673881, 9],
[1, 1, 3, 0, 5, 0.0, 4.812184355372417, 68],
[1, 1, 2, 0, 2, 1.0, 2.833213344056216, 0],
[1, 1, 1, 1, 5, 1.0, 5.062595033026967, 67],
[1, 0, 0, 0, 5, 1.0, 4.330733340286331, 21],
[1, 0, 0, 0, 5, 1.0, 5.231108616854587, 113],
[1, 1, 1, 0, 5, 1.0, 4.7535901911063645, 18],
[0, 0, 0, 0, 5, 1.0, 4.74493212836325, 37],
[1, 1, 1, 0, 5, 1.0, 4.852030263919617, 72],
[1, 0, 0, 0, 5, 1.0, 4.941642422609304, 78],
[1, 1, 3, 1, 5, 1.0, 4.30406509320417, 8],
[1, 1, 0, 0, 5, 1.0, 4.867534450455582, 84],
[1, 1, 0, 1, 5, 1.0, 4.672828834461906, 31],
[1, 0, 0, 0, 5, 1.0, 4.857444178729352, 61],
[1, 1, 0, 0, 5, 1.0, 4.718498871295094, 19],
[1, 1, 0, 0, 5, 1.0, 5.556828061699537, 107],
[1, 1, 0, 0, 5, 1.0, 4.553876891600541, 34],
[1, 0, 0, 1, 5, 1.0, 4.890349128221754, 74],
[1, 1, 2, 0, 5, 1.0, 5.123963979403259, 62],
[1, 0, 0, 0, 5, 1.0, 4.787491742782046, 27],
[0, 0, 0, 0, 5, 0.0, 4.919980925828125, 108],
[0, 0, 0, 0, 5, 1.0, 5.365976015021851, 103],
[1, 1, 0, 1, 5, 1.0, 4.74493212836325, 38],
[0, 0, 0, 0, 5, 0.0, 4.330733340286331, 13],
[1, 1, 2, 0, 5, 1.0, 4.890349128221754, 69],
[1, 1, 1, 0, 5, 1.0, 5.752572638825633, 112],
[1, 1, 0, 0, 5, 1.0, 5.075173815233827, 73],
[1, 0, 0, 0, 5, 1.0, 4.912654885736052, 47],
```

```
[1, 1, 0, 0, 5, 1.0, 5.204006687076795, 81],
[1, 0, 0, 1, 5, 1.0, 4.564348191467836, 60],
[1, 0, 0, 0, 5, 1.0, 4.204692619390966, 83],
[0, 1, 0, 0, 5, 1.0, 4.867534450455582, 5],
[1, 1, 2, 1, 5, 1.0, 5.056245805348308, 58],
[1, 1, 1, 1, 3, 1.0, 4.919980925828125, 79],
[0, 1, 0, 0, 5, 1.0, 4.969813299576001, 54],
[1, 1, 0, 1, 4, 1.0, 4.820281565605037, 56],
[1, 0, 0, 0, 5, 1.0, 4.499809670330265, 120],
[1, 0, 3, 0, 5, 1.0, 5.768320995793772, 118],
[1, 1, 2, 0, 5, 1.0, 4.718498871295094, 101],
[0, 0, 0, 0, 5, 0.0, 4.7535901911063645, 26],
[0, 0, 0, 0, 6, 1.0, 4.727387818712341, 33],
[1, 1, 1, 0, 5, 1.0, 6.214608098422191, 119],
[0, 0, 0, 0, 5, 1.0, 5.267858159063328, 89],
[1, 1, 2, 0, 5, 1.0, 5.231108616854587, 92],
[1, 0, 0, 0, 6, 1.0, 4.2626798770413155, 6],
[1, 1, 0, 0, 0, 1.0, 4.709530201312334, 90],
[1, 1, 0, 0, 5, 1.0, 4.700480365792417, 45],
[1, 1, 2, 0, 5, 1.0, 5.298317366548036, 109],
[1, 0, 1, 0, 3, 1.0, 4.727387818712341, 17],
[1, 1, 1, 0, 5, 1.0, 4.6443908991413725, 36],
[0, 1, 0, 1, 5, 1.0, 4.605170185988092, 16],
[1, 0, 0, 0, 5, 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, 5, 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, 5, 1.0, 4.969813299576001, 66],
[1, 1, 2, 1, 5, 1.0, 4.394449154672439, 51],
[1, 1, 1, 0, 5, 1.0, 5.231108616854587, 100],
[1, 1, 0, 0, 5, 1.0, 5.351858133476067, 93],
[1, 1, 0, 0, 5, 1.0, 4.605170185988092, 15],
[1, 1, 2, 0, 5, 1.0, 4.787491742782046, 106],
[1, 0, 0, 0, 3, 1.0, 4.787491742782046, 105],
[1, 1, 3, 0, 5, 1.0, 4.852030263919617, 64],
[1, 0, 0, 0, 5, 1.0, 4.8283137373023015, 49],
[1, 0, 0, 1, 5, 1.0, 4.6443908991413725, 42],
[0, 0, 0, 0, 5, 1.0, 4.477336814478207, 10],
[1, 1, 0, 1, 5, 1.0, 4.553876891600541, 20],
[1, 1, 3, 1, 3, 1.0, 4.394449154672439, 14],
[1, 0, 0, 0, 5, 1.0, 5.298317366548036, 76],
[0, 0, 0, 0, 5, 1.0, 4.90527477843843, 11],
[1, 0, 0, 0, 6, 1.0, 4.727387818712341, 18],
[1, 1, 2, 0, 5, 1.0, 4.248495242049359, 23],
[1, 1, 0, 1, 5, 0.0, 5.303304908059076, 63],
```

```
        [1, 1, 0, 0, 3, 0.0, 4.499809670330265, 48],
        [0, 0, 0, 0, 5, 1.0, 4.430816798843313, 30],
        [1, 0, 0, 0, 5, 1.0, 4.897839799950911, 29],
        [1, 1, 2, 0, 5, 1.0, 5.170483995038151, 86],
        [1, 1, 3, 0, 5, 1.0, 4.867534450455582, 115],
        [1, 1, 0, 0, 5, 1.0, 6.077642243349034, 116],
        [1, 1, 3, 1, 3, 0.0, 4.248495242049359, 40],
        [1, 1, 1, 0, 5, 1.0, 4.564348191467836, 12]], dtype=object)
```

[36]:
```
# View y_test data set

y_test
```

[36]:
```
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])
```

2.4 Scaling the Dataset

[37]:
```
# Importing standard scaler from sklearn to scale the dataset

from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
x_train=ss.fit_transform(x_train)
x_test=ss.fit_transform(x_test)
```

3 Data Training

3.1 Decision Tree Classifier Algorithm

[38]:
```
# Importing decision tree classifier algorithm from sklearn to train the model

from sklearn.tree import DecisionTreeClassifier
DTClassifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
DTClassifier.fit(x_train,y_train)
```

[38]:
```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

[39]:
```
# Using algorithm to predict values of test data set

y_pred= DTClassifier.predict(x_test)
y_pred
```

[39]: array([0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1,
       1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1])

[40]:
```
# Importing metrics from sklearn to check the accuracy of prediction

from sklearn import metrics
print('The accuracy of decision tree is: ', metrics.
 ↪accuracy_score(y_pred,y_test))
```

The accuracy of decision tree is: 0.7073170731707317

We have got an accuracy of 70% which is generally not a good score. So, we are using Naive Bayes Classifier.

3.2 Naive Bayes Classifier Algorithm

[41]:
```
# Importing naive bayes classifier algorithm from sklearn to train the model

from sklearn.naive_bayes import GaussianNB
NBClassifier = GaussianNB()
NBClassifier.fit(x_train,y_train)
```

[41]: GaussianNB()

[42]:
```
# Using algorithm to predict values of test data set

y_pred= NBClassifier.predict(x_test)
y_pred
```

[42]: 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])

[43]:
```
# To check the accuracy of prediction

print('The accuracy of Naive Bayes is: ',metrics.accuracy_score(y_pred,y_test))
```

The accuracy of Naive Bayes is: 0.8292682926829268

We have got an accuracy of 82% which is much better score as compared to decision tree algorithm.

4 Testing the Model using Test Dataset

4.1 Understanding Test Data

[44]:
```
# Importing test.CSV dataset

testdata = pd.read_csv("loan-test.csv")
```

[45]:
```
# Viewing test dataset

testdata.head()
```

[45]:
```
    Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001015   Male     Yes          0      Graduate            No
1  LP001022   Male     Yes          1      Graduate            No
2  LP001031   Male     Yes          2      Graduate            No
3  LP001035   Male     Yes          2      Graduate            No
4  LP001051   Male      No          0  Not Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5720                  0       110.0             360.0
1             3076               1500       126.0             360.0
2             5000               1800       208.0             360.0
3             2340               2546       100.0             360.0
4             3276                  0        78.0             360.0

   Credit_History Property_Area
0             1.0         Urban
1             1.0         Urban
2             1.0         Urban
3             NaN         Urban
4             1.0         Urban
```

[46]:
```
# Viewing test dataset info

testdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 367 entries, 0 to 366
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            367 non-null    object
 1   Gender             356 non-null    object
 2   Married            367 non-null    object
 3   Dependents         357 non-null    object
 4   Education          367 non-null    object
 5   Self_Employed      344 non-null    object
```

```
 6   ApplicantIncome    367 non-null    int64
 7   CoapplicantIncome  367 non-null    int64
 8   LoanAmount         362 non-null    float64
 9   Loan_Amount_Term   361 non-null    float64
 10  Credit_History     338 non-null    float64
 11  Property_Area      367 non-null    object
dtypes: float64(3), int64(2), object(7)
memory usage: 34.5+ KB
```

[47]:
```
# Checking for missing values

testdata.isnull().sum()
```

[47]:
```
Loan_ID               0
Gender               11
Married               0
Dependents           10
Education             0
Self_Employed        23
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            5
Loan_Amount_Term      6
Credit_History       29
Property_Area         0
dtype: int64
```

4.2 Filling Null Values in Test Data

[64]:
```
# Filling missing values using mode function in test dataset

testdata['Gender'] = testdata['Gender'].fillna(testdata['Gender'].mode()[0])
testdata['Dependents'] = testdata['Dependents'].fillna(testdata['Dependents'].
  ↪mode()[0])
testdata['Self_Employed'] = testdata['Self_Employed'].
  ↪fillna(testdata['Self_Employed'].mode()[0])
testdata['Loan_Amount_Term'] = testdata['Loan_Amount_Term'].
  ↪fillna(testdata['Loan_Amount_Term'].mode()[0])
testdata['Credit_History'] = testdata['Credit_History'].
  ↪fillna(testdata['Credit_History'].mode()[0])
```

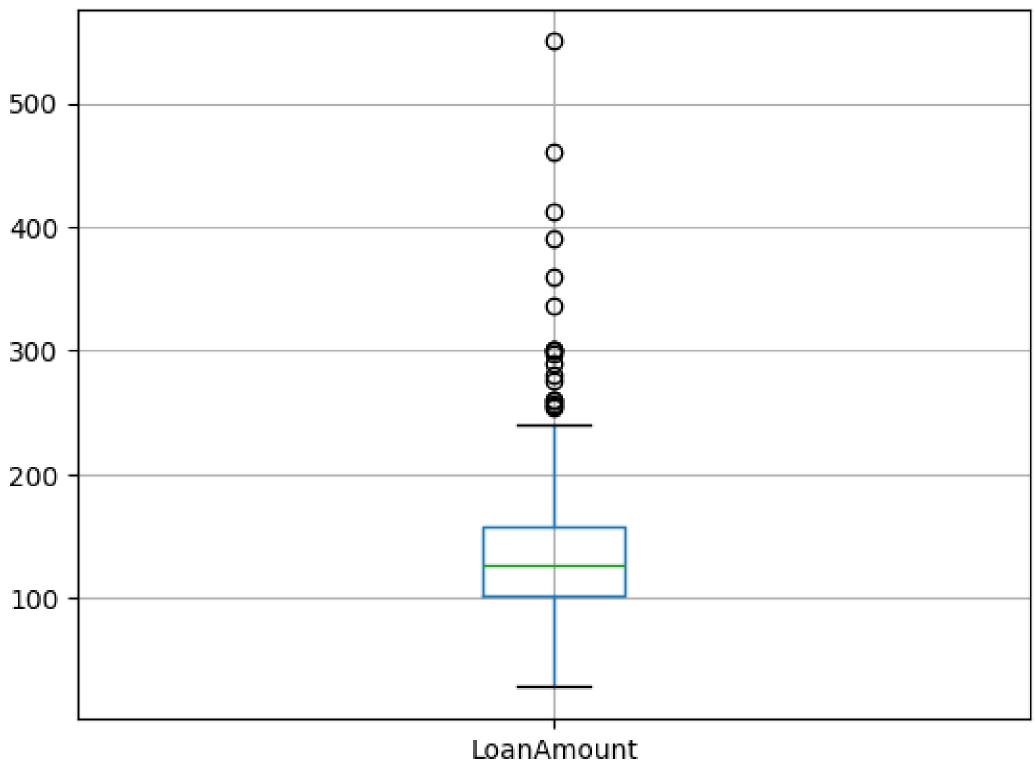
[65]:
```
# Filling missing values using mean strategy in test dataset

testdata.LoanAmount= testdata.LoanAmount.fillna(testdata.LoanAmount.mean())
testdata['LoanAmount_log']= np.log(testdata['LoanAmount'])
```

[66]:
```python
# Checking for missing values

testdata.isnull().sum()
```

[66]:
```
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
LoanAmount_log       0
TotalIncome          0
TotalIncome_log      0
dtype: int64
```

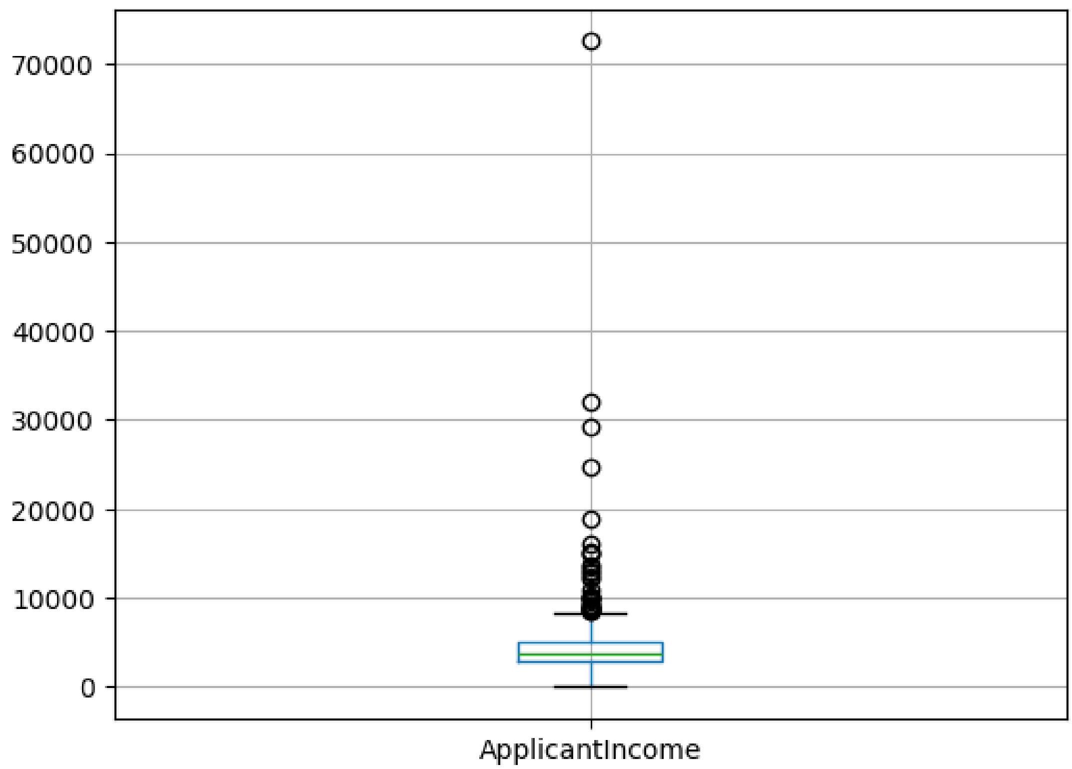
[67]:
```python
# Plotting boxplot for loan amount in test dataset

testdata.boxplot(column='LoanAmount')
```

[67]:
```
<Axes: >
```



[68]:
```
# Plotting boxplot for applicant income in test dataset

testdata.boxplot(column='ApplicantIncome')
```

[68]: <Axes: >



[54]:
```
# Applying log function to the sum of applicant and coapplicant income i.e.␣
↪total income in test dataset using numpy

testdata['TotalIncome']= testdata['ApplicantIncome'] +␣
↪testdata['CoapplicantIncome']
testdata['TotalIncome_log']= np.log(testdata['TotalIncome'])
```

[55]:
```
# Viewing test dataset

testdata.head()
```

[55]:
```
    Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001015   Male     Yes          0      Graduate            No
1  LP001022   Male     Yes          1      Graduate            No
2  LP001031   Male     Yes          2      Graduate            No
3  LP001035   Male     Yes          2      Graduate            No
4  LP001051   Male      No          0  Not Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5720                  0       110.0             360.0
1             3076               1500       126.0             360.0
2             5000               1800       208.0             360.0
```

```
3            2340              2546       100.0              360.0
4            3276                 0        78.0              360.0

   Credit_History Property_Area  LoanAmount_log  TotalIncome  TotalIncome_log
0             1.0         Urban        4.700480         5720         8.651724
1             1.0         Urban        4.836282         4576         8.428581
2             1.0         Urban        5.337538         6800         8.824678
3             1.0         Urban        4.605170         4886         8.494129
4             1.0         Urban        4.356709         3276         8.094378
```

4.3 Modeling & Converting Test Dataset

[56]:
```python
# Classifying the independent variables columns as test variable

test= testdata.iloc[:,np.r_[1:5,9:11,13:15]].values
```

[57]:
```python
# Converting the categorical values as numerical format

for i in range(0,5):
    test[:,i]=labelencoder_x.fit_transform(test[:,i])
```

[58]:
```python
test[:,7]= labelencoder_x.fit_transform(test[:,7])
```

[59]:
```python
# Viewing test dataset

test
```

[59]:
```
array([[1, 1, 0, …, 1.0, 5720, 207],
       [1, 1, 1, …, 1.0, 4576, 124],
       [1, 1, 2, …, 1.0, 6800, 251],
       …,
       [1, 0, 0, …, 1.0, 5243, 174],
       [1, 1, 0, …, 1.0, 7393, 268],
       [1, 0, 0, …, 1.0, 9200, 311]], dtype=object)
```

4.4 Scaling & Predicting Loan Eligibility of Test Dataset

[60]:
```python
# Scaling the test dataset

test= ss.fit_transform(test)
```

[61]:
```python
# Creating a variable to predict loan eligibility using the naive bayes␣
↪classifier algorithm

predict_loan_eligibility= NBClassifier.predict(test)
```

[62]:
```python
# Viewing - pridict loan eligibility variable for test dataset

predict_loan_eligibility
```

[62]:
```
array([1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 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, 0, 1, 1,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
       0, 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, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0,
       1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

We finally have the predictions indicating which applicants are eligible for the loan. In this prediction using Naive Bayes Classifier Algorithm, a "1" represents applicants who are eligible for the loan, while a "0" represents those who are not eligible.