Loan Eligibility Prediction Using Machine Learning Techniques

By

Muhammad Abdullah Arshad

Data Analyst

Project Report

Date: 13 August 2025

Phone: +92 303 4292979

Email: malikabdullah5151@gmail.com

LinkedIn: www.linkedin.com/in/muhammad-abdullah-arshad-3966a5245

Contents

1 Introduction **2**

2 Dataset Description **2**

3 Data Analysis and Results **3**
- 3.1 Import Required Libraries . . . 3
- 3.2 Import and Explore the Dataset . . . 3
- 3.3 Data Visualization . . . 5
- 3.4 Handling Missing Values . . . 11
- 3.5 Feature Engineering . . . 12
- 3.6 Splitting Features and Target . . . 14
- 3.7 Train-Test Split . . . 15
- 3.8 Label Encoding . . . 16
- 3.9 Feature Scaling . . . 20
- 3.10 Decision Tree Model . . . 20
- 3.11 Naive Bayes Model . . . 21
- 3.12 Import and Preprocess Test Dataset . . . 21
- 3.13 Test Dataset Encoding and Scaling . . . 25
- 3.14 Final Prediction on Test Data . . . 26

4 Model Evaluation and Discussion **26**
- 4.1 Evaluation Results . . . 27
- 4.2 Discussion . . . 27

1 Introduction

In the financial sector, loan approval is one of the most critical processes for banks and lending institutions. Each loan application requires careful evaluation of the applicant's profile to assess the likelihood of repayment and minimize the risk of default. Traditionally, this process has been performed manually by loan officers, often resulting in delays and potential human bias. With the growing availability of digital banking data and advancements in machine learning techniques, it is now possible to automate and improve the loan approval process with greater accuracy, speed, and fairness.

This project focuses on building a predictive model that determines whether a loan should be approved based on various applicant-related features. The dataset used contains demographic details, financial information, and credit history of applicants. Important variables include *Applicant Income*, *Coapplicant Income*, *Loan Amount*, *Loan Amount Term*, *Credit History*, *Gender*, *Marital Status*, and *Education Level*. The target variable, *Loan_Status*, indicates whether a loan was approved (Y) or not approved (N).

The primary goal of this project is to develop and evaluate machine learning models capable of predicting loan approval decisions accurately. The project involves several key stages:

- Exploratory Data Analysis (EDA) to understand the structure and distribution of the dataset.
- Data Preprocessing, including handling missing values, feature engineering, and encoding categorical variables.
- Model Training and Evaluation using classification algorithms such as Decision Tree and Naive Bayes.
- Performance Comparison to identify the best-performing model.

By applying machine learning techniques to this problem, we aim to streamline the decision-making process for loan approvals, reduce manual workload, and enhance consistency in lending decisions. Such an approach not only benefits financial institutions by improving efficiency and reducing default risks but also ensures a fairer and more transparent evaluation process for applicants.

2 Dataset Description

The dataset used in this project contains information about loan applicants collected by a financial institution. It includes demographic attributes, financial details, and credit history, which are used to predict whether a loan should be approved. The target variable, *Loan_Status*, is a binary indicator representing approval (**Y**) or rejection (**N**) of the loan application.

The dataset consists of the following key features:

- **Gender** – Applicant's gender (*Male* or *Female*).
- **Married** – Marital status of the applicant (*Yes* or *No*).
- **Dependents** – Number of dependents of the applicant.
- **Education** – Education level (*Graduate* or *Not Graduate*).
- **Self_Employed** – Employment type (*Yes* if self-employed, otherwise *No*).

- **ApplicantIncome** – Monthly income of the applicant.
- **CoapplicantIncome** – Monthly income of the co-applicant.
- **LoanAmount** – Loan amount requested (in thousands).
- **Loan_Amount_Term** – Term of the loan in months.
- **Credit_History** – Credit history status (1.0 for good history, 0.0 for poor history).
- **Property_Area** – Type of area (*Urban*, *Semiurban*, *Rural*).
- **Loan_Status** – Target variable: loan approval decision (**Y** or **N**).

The dataset also contains some missing values in certain attributes such as *Gender*, *Married*, *Dependents*, *Self_Employed*, *LoanAmount*, *Loan_Amount_Term*, and *Credit_History*. These missing values are addressed during the data preprocessing stage. Additionally, new derived features such as *LoanAmount_log* and *TotalIncome_log* are created for better model performance.

3 Data Analysis and Results

3.1 Import Required Libraries

[7]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

3.2 Import and Explore the Dataset

[8]:
```
dataset = pd.read_csv(r"C:\Users\abdul\Downloads\Loan Data.csv")
```

[9]:
```
dataset.head()
```

[9]:
```
    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
```

[10]:
```
dataset.shape
```

[10]:
```
(614, 13)
```

[11]:
```
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
```

[12]:
```
dataset.describe()
```

[12]:
```
       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
```

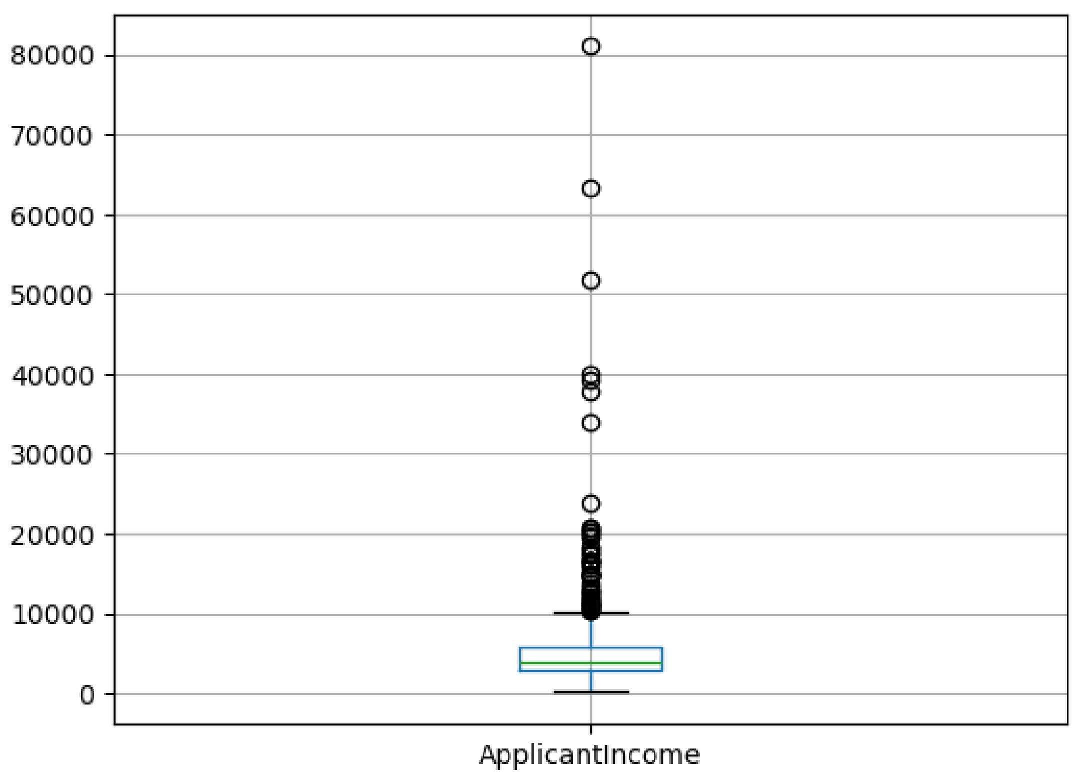
[13]:
```python
pd.crosstab(dataset['Credit_History'], dataset['Loan_Status'])
```

[13]:
```
Loan_Status      N    Y
Credit_History
0.0             82    7
1.0             97  378
```

3.3 Data Visualization

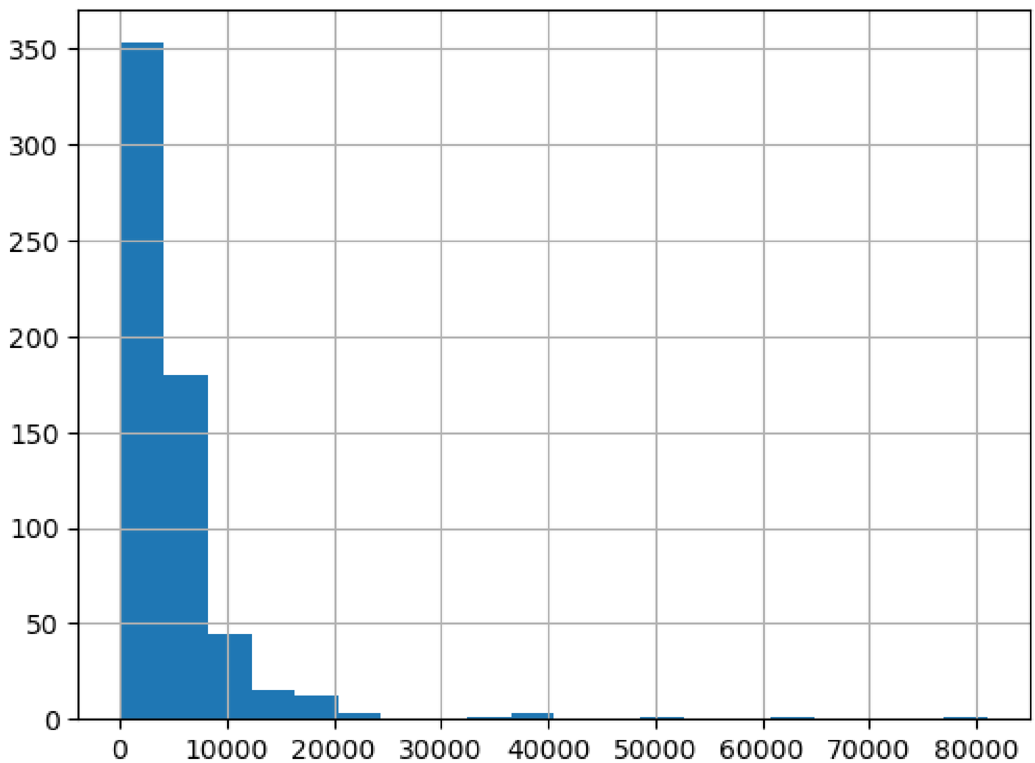
[15]:
```python
dataset.boxplot(column = "ApplicantIncome")
```

[15]:
```
<Axes: >
```



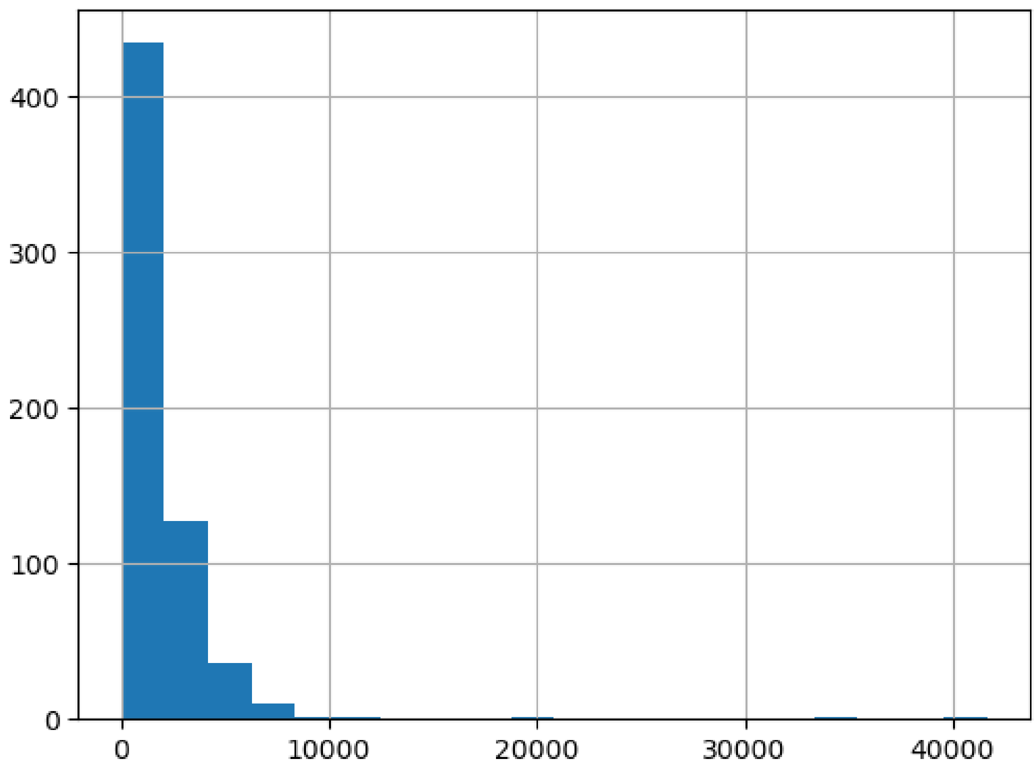
[16]:
```python
dataset['ApplicantIncome'].hist(bins=20)
```

[16]:
```
<Axes: >
```



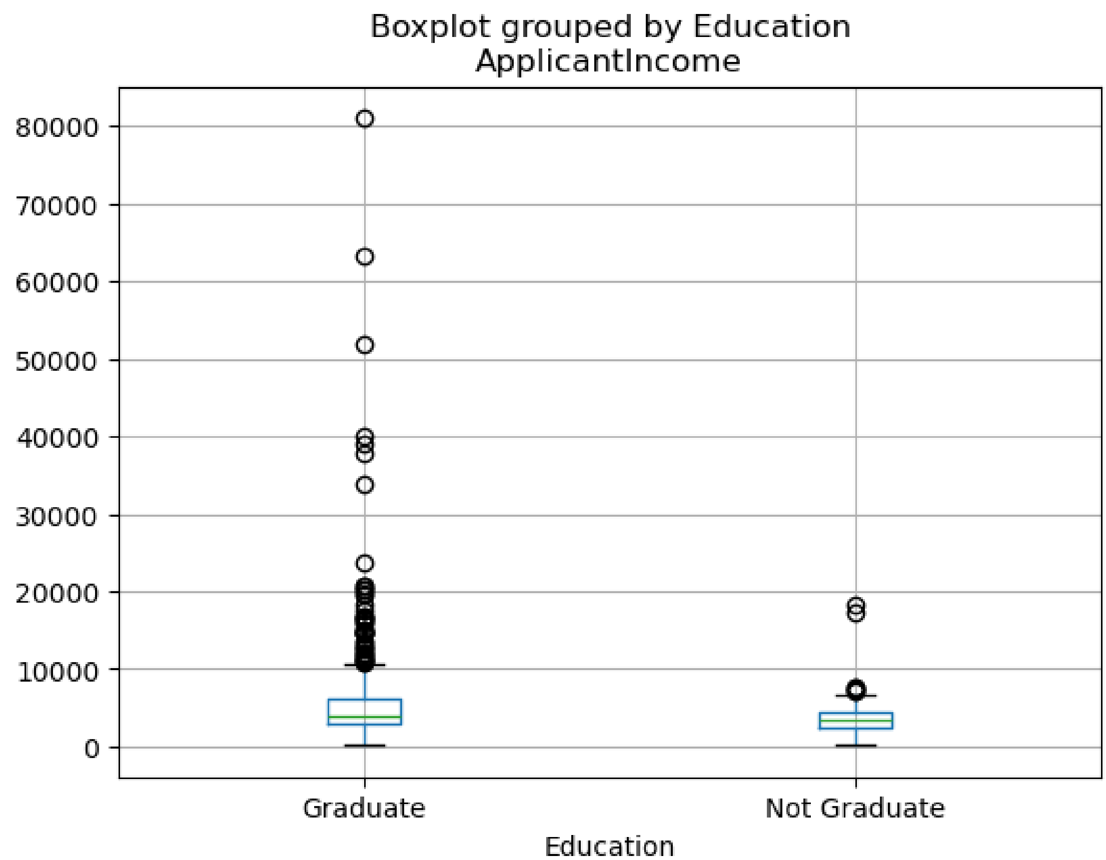
[17]:
```
dataset['CoapplicantIncome'].hist(bins=20)
```

[17]: <Axes: >



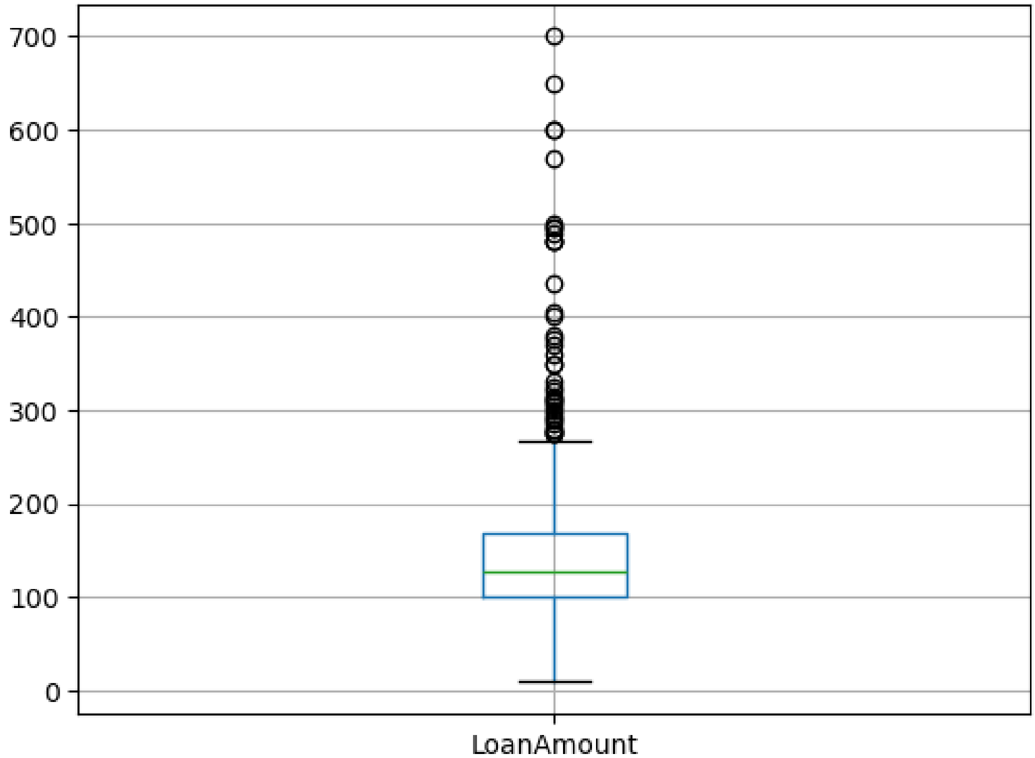

```
[18]: dataset.boxplot(column = 'ApplicantIncome', by = 'Education')
```

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



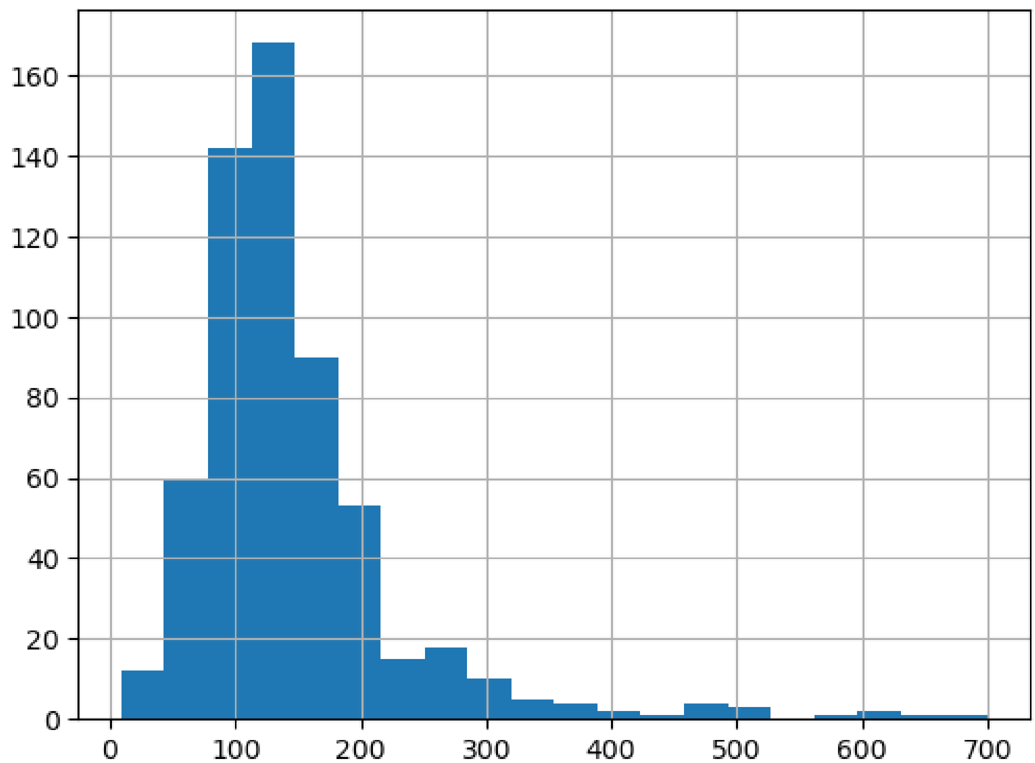
[19]:
```
dataset.boxplot(column = 'LoanAmount')
```

[19]: <Axes: >



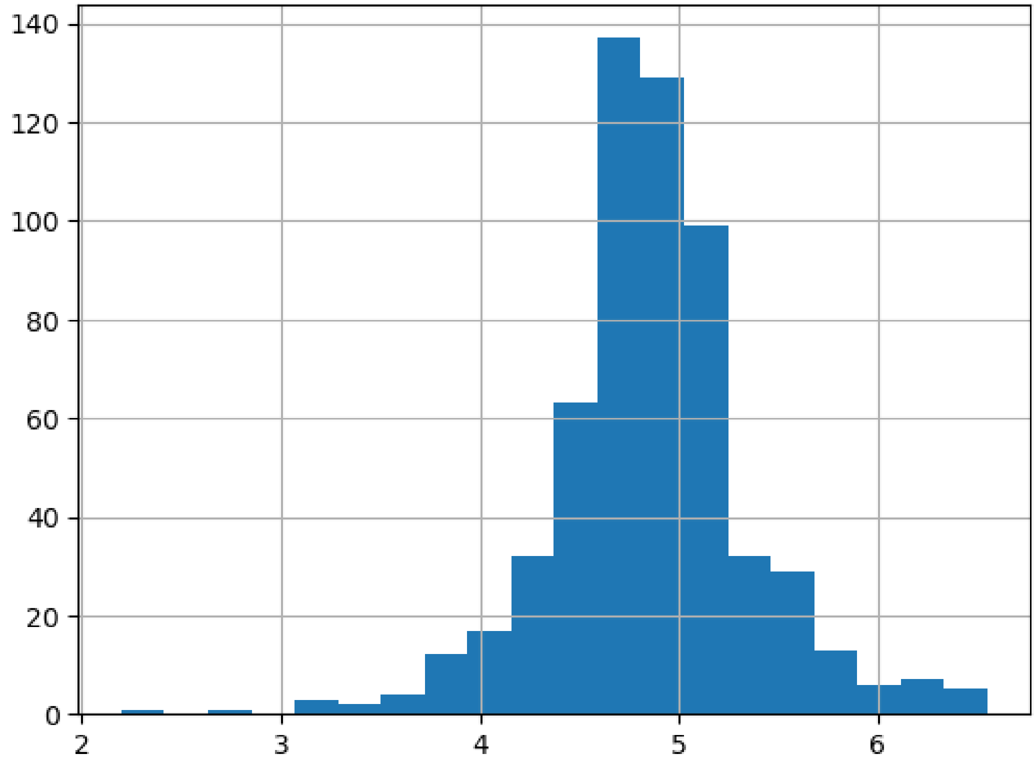
[20]:
```
dataset['LoanAmount'].hist(bins=20)
```

[20]: <Axes: >



[21]:
```
dataset['LoanAmount_log'] = np.log(dataset['LoanAmount'])
dataset['LoanAmount_log'].hist(bins=20)
```

[21]: <Axes: >



3.4 Handling Missing Values

[23]:
```
dataset.isnull().sum()
```

[23]:
```
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
dtype: int64
```

[24]:
```
dataset['Gender'].fillna(dataset['Gender'].mode()[0], inplace=True)
```

[25]:
```
dataset['Married'].fillna(dataset['Married'].mode()[0], inplace=True)
```

[26]:
```
dataset['Dependents'].fillna(dataset['Dependents'].mode()[0], inplace=True)
```

[27]:
```
dataset['Self_Employed'].fillna(dataset['Self_Employed'].mode()[0], inplace=True)
```

[28]:
```
dataset.LoanAmount = dataset.LoanAmount.fillna(dataset.LoanAmount.mean())
dataset.LoanAmount_log = dataset.LoanAmount_log.fillna(dataset.LoanAmount_log.
 ↪mean())
```

[29]:
```
dataset['Loan_Amount_Term'].fillna(dataset['Loan_Amount_Term'].mode()[0],␣
 ↪inplace=True)
```

[34]:
```
dataset['Credit_History'].fillna(dataset['Credit_History'].mode()[0],␣
 ↪inplace=True)
```

[35]:
```
dataset.isnull().sum()
```

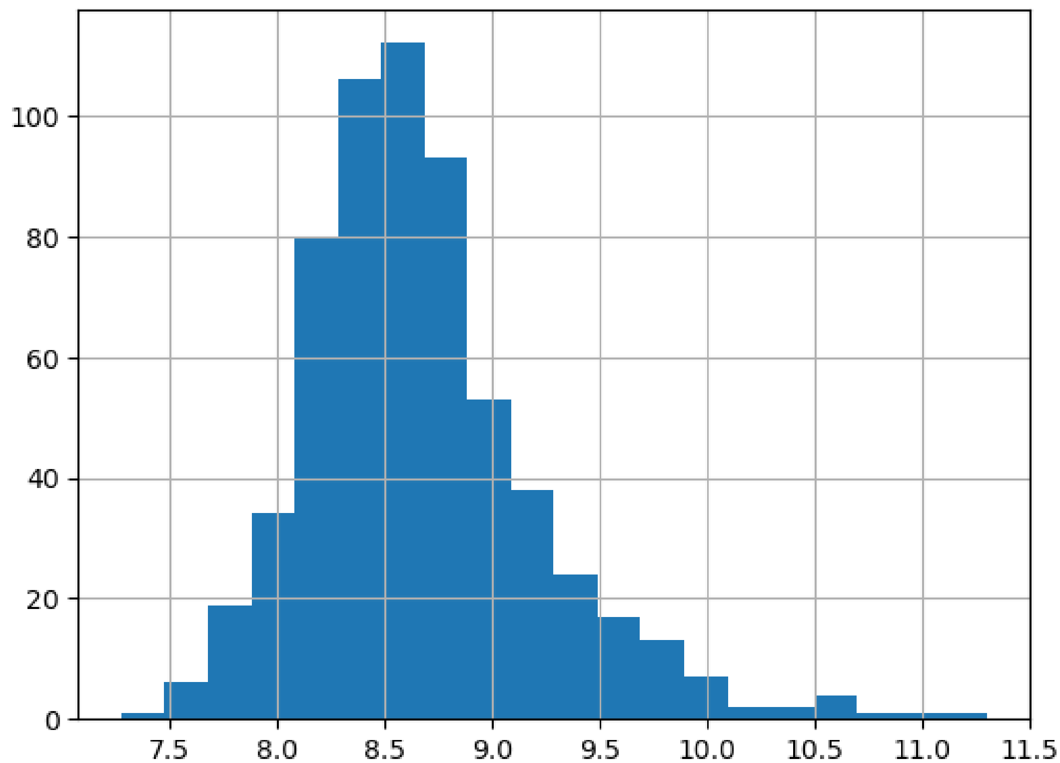
[35]:
```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
LoanAmount_log       0
TotalIncome          0
TotalIncome_log      0
dtype: int64
```

3.5 Feature Engineering

[36]:
```
dataset['TotalIncome'] = dataset['ApplicantIncome'] +␣
 ↪dataset['CoapplicantIncome']
dataset['TotalIncome_log'] = np.log(dataset['TotalIncome'])
```

[37]:
```
dataset['TotalIncome_log'].hist(bins=20)
```

[37]:
```
<Axes: >
```



[38]:
```
dataset.head()
```

[38]:
```
    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
```

3.6 Splitting Features and Target

[39]:
```
x = dataset.iloc[:,np.r_[1:5,9:11,13:15]].values
y = dataset.iloc[:,12].values
```

[40]:
```
x
```

[40]:
```
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)
```

[41]:
```
y
```

[41]:
```
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)
```

3.7 Train-Test Split

[42]:
```
pip install scikit-learn
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: scikit-learn in c:\program files\orange\lib\site-
packages (1.5.2)
Requirement already satisfied: numpy>=1.19.5 in c:\program
files\orange\lib\site-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in c:\program files\orange\lib\site-
packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in c:\program
files\orange\lib\site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\program
files\orange\lib\site-packages (from scikit-learn) (3.5.0)
Note: you may need to restart the kernel to use updated packages.
```

[43]:
```
from sklearn.model_selection import train_test_split
```

[44]:
```
x_train , x_test, y_train, y_test = train_test_split(x,y,test_size = 0.2,␣
↪random_state=0)
```

[45]:
```
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]]
```

3.8 Label Encoding

[46]:
```
from sklearn.preprocessing import LabelEncoder
labelencoder_x = LabelEncoder()
```

[47]:
```
for i in range(0, 5):
    x_train[:, i] = labelencoder_x.fit_transform(x_train[:, i])
```

[48]:
```
x_train[:, 7] = labelencoder_x.fit_transform(x_train[:, 7])
```

[49]:
```
x_train
```

[49]:
```
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)
```

[50]:
```
labelencoder_y  = LabelEncoder()
y_train = labelencoder_y.fit_transform(y_train)
```

[51]:
```
y_train
```

[51]:
```
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])
```

[52]:
```python
for i in range(0, 5):
    x_test[:, i] = labelencoder_x.fit_transform(x_test[:, i])
```

[53]:
```python
x_test[:, 7] = labelencoder_x.fit_transform(x_test[:, 7])
```

[54]:
```python
labelencoder_y  = LabelEncoder()
y_test = labelencoder_y.fit_transform(y_test)
```

[55]:
```python
x_test
```

[55]:
```
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)
```

[56]:
```
y_test
```

[56]:
```
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])
```

3.9 Feature Scaling

[58]:
```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
x_train = ss.fit_transform(x_train)
x_test = ss.fit_transform(x_test)
```

3.10 Decision Tree Model

[60]:
```
from sklearn.tree import DecisionTreeClassifier
DTClassifier = DecisionTreeClassifier(criterion = 'entropy',random_state=0)
DTClassifier.fit(x_train, y_train)
```

[60]:
```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

[61]:
```
y_pred = DTClassifier.predict(x_test)
y_pred
```

[61]:
```
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])
```

[ ]:
```
from sklearn import metrics
```

[63]:
```
print("The Accuracy of Decision Tree Model is:",metrics.
 ↪accuracy_score(y_pred,y_test))
```

```
The Accuracy of Decision Tree Model is: 0.7073170731707317
```

3.11 Naive Bayes Model

[64]:
```
from sklearn.naive_bayes import GaussianNB
NBClassifier = GaussianNB()
```

[65]:
```
NBClassifier.fit(x_train,y_train)
```

[65]:
```
GaussianNB()
```

[66]:
```
y_pred = NBClassifier.predict(x_test)
y_pred
```

[66]:
```
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])
```

[67]:
```
print("The Accuracy of Decision Tree Model is:",metrics.
 ↪accuracy_score(y_pred,y_test))
```

```
The Accuracy of Decision Tree Model is: 0.8292682926829268
```

3.12 Import and Preprocess Test Dataset

[69]:
```
testdata = pd.read_csv("TestData.csv")
```

[70]:
```
testdata.head()
```

[70]:
```
    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
0             1.0         Urban
1             1.0         Rural
2             1.0         Urban
```

```
3              1.0          Urban
4              1.0          Urban
```

[71]:
```
testdata.isnull().sum()
```

[71]:
```
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
dtype: int64
```

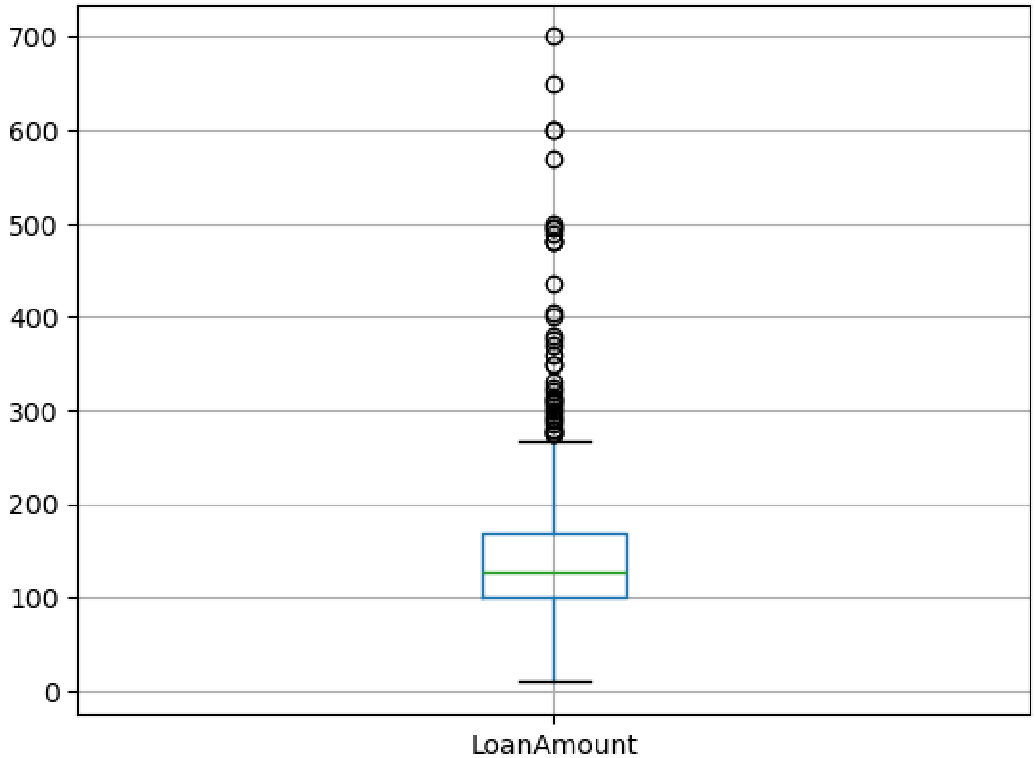
[72]:
```
testdata['Gender'].fillna(testdata['Gender'].mode()[0], inplace=True)
testdata['Dependents'].fillna(testdata['Dependents'].mode()[0], inplace=True)
testdata['Self_Employed'].fillna(testdata['Self_Employed'].mode()[0],␣
 ↪inplace=True)
testdata['Loan_Amount_Term'].fillna(testdata['Loan_Amount_Term'].mode()[0],␣
 ↪inplace=True)
testdata['Credit_History'].fillna(testdata['Credit_History'].mode()[0],␣
 ↪inplace=True)
testdata['Married'].fillna(testdata['Married'].mode()[0], inplace=True)
```

[73]:
```
testdata.isnull().sum()
```

[73]:
```
Loan_ID               0
Gender                0
Married               0
Dependents            0
Education             0
Self_Employed         0
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term      0
Credit_History        0
Property_Area         0
dtype: int64
```

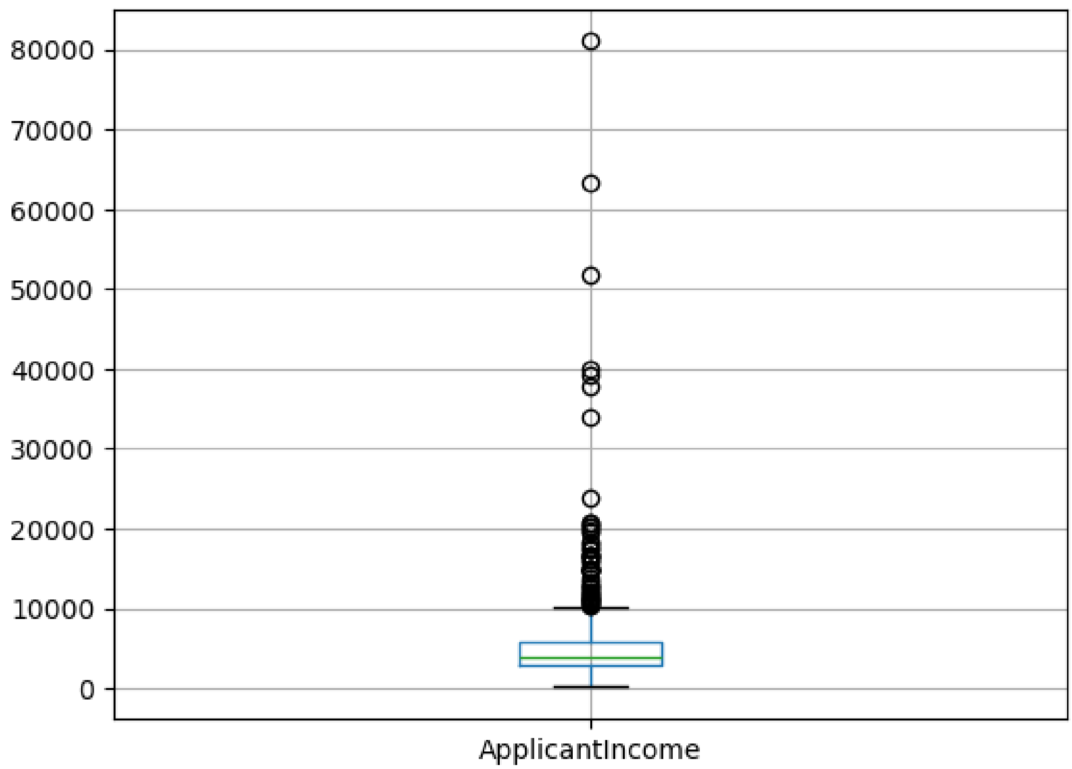
[74]:
```
testdata.boxplot(column = 'LoanAmount')
```

[74]: <Axes: >



[75]:
```
testdata.boxplot(column = 'ApplicantIncome')
```

[75]: <Axes: >



[76]:
```
testdata.LoanAmount = testdata.LoanAmount.fillna(testdata.LoanAmount.mean())
```

[77]:
```
testdata['LoanAmount_log'] = np.log(testdata['LoanAmount'])
```

[78]:
```
testdata.isnull().sum()
```

[78]:
```
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
dtype: int64
```

[79]:
```
testdata['TotalIncome'] = testdata['ApplicantIncome'] +␣
↪testdata['CoapplicantIncome']
testdata['TotalIncome_log'] = np.log(testdata['TotalIncome'])
```

[80]:
```
testdata.head()
```

[80]:
```
     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  LoanAmount_log  TotalIncome  TotalIncome_log
0             1.0         Urban        4.986426       5849.0         8.674026
1             1.0         Rural        4.852030       6091.0         8.714568
2             1.0         Urban        4.189655       3000.0         8.006368
3             1.0         Urban        4.787492       4941.0         8.505323
4             1.0         Urban        4.948760       6000.0         8.699515
```

3.13 Test Dataset Encoding and Scaling

[82]:
```
test = testdata.iloc[:,np.r_[1:5,9:11,13:15]].values
```

[83]:
```
for i in range(0, 5):
    test[:, i] = labelencoder_x.fit_transform(test[:, i])
```

[84]:
```
test[:, 7] = labelencoder_x.fit_transform(test[:, 7])
```

[85]:
```
test
```

[85]:
```
array([[1, 0, 0, ..., 1.0, 5849.0, 320],
       [1, 1, 1, ..., 1.0, 6091.0, 333],
       [1, 1, 0, ..., 1.0, 3000.0, 42],
       ...,
       [1, 1, 1, ..., 1.0, 8312.0, 436],
       [1, 1, 2, ..., 1.0, 7583.0, 416],
       [0, 0, 0, ..., 0.0, 4583.0, 185]], dtype=object)
```

[86]:
```
test = ss.fit_transform(test)
```

3.14 Final Prediction on Test Data

[88]:
```python
predict = NBClassifier.predict(test)
```

[89]:
```python
predict
```

[89]:
```
array([1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 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, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 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, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 0, 0, 1, 1, 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, 1, 1, 0, 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, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 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, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 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, 0, 1, 1, 1, 1, 0, 1, 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, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 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, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 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, 0,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 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, 1, 1, 0])
```

Final Note: In our predictions, a value of 1 represents an eligible loan applicant, and a value of 0 represents an ineligible applicant.

4 Model Evaluation and Discussion

The primary objective of this project was to evaluate the performance of different machine learning algorithms for the task of loan approval prediction. After preprocessing the dataset, handling missing values, encoding categorical variables, and standardizing numerical features, two models were implemented and tested: the Decision Tree Classifier and the Gaussian Naive Bayes Classifier. Both models were trained on the same training data and evaluated using the same test data to ensure a fair comparison.

4.1 Evaluation Results

The Decision Tree Classifier was constructed using the `entropy` criterion, which measures the information gain for each split. This model is capable of capturing non-linear patterns in the data and producing an interpretable tree structure. On the test set, the Decision Tree achieved an accuracy of 70%. While this indicates that the model was able to correctly classify the majority of test samples, the performance leaves room for improvement.

The Gaussian Naive Bayes Classifier, on the other hand, is a probabilistic model based on Bayes' theorem with the assumption that the features follow a Gaussian distribution and are conditionally independent given the class label. Despite its simplicity, this model achieved a slightly higher accuracy of 82% on the test set.

Model	**Accuracy**
Decision Tree Classifier	70%
Gaussian Naive Bayes Classifier	82%

4.2 Discussion

The results indicate that the Gaussian Naive Bayes model marginally outperformed the Decision Tree in this specific application. There are several possible explanations for this outcome:

1. **Model Complexity:** Decision Trees, while powerful, can overfit the training data if not properly regularized. This can result in slightly weaker generalization performance, as seen in this case.

2. **Feature Independence Assumption:** The Naive Bayes model assumes independence between features, which is often unrealistic. However, in this dataset, the correlation between features may not have been strong enough to significantly violate this assumption, allowing Naive Bayes to perform well.

3. **Data Distribution:** Naive Bayes works best when features follow a normal distribution. The logarithmic transformations applied to income and loan amount features may have made the data more Gaussian-like, favoring Naive Bayes.

It is also important to note that accuracy is not the only metric for evaluating models. In real-world loan approval scenarios, false positives (approving risky loans) and false negatives (rejecting creditworthy applicants) can have very different costs. A more thorough evaluation could include metrics such as precision, recall, F1-score, and ROC-AUC to better understand each model's strengths and weaknesses.

From a computational standpoint, the Gaussian Naive Bayes Classifier is significantly faster to train and requires fewer resources than a Decision Tree, which makes it attractive for large-scale or real-time applications. However, Decision Trees offer better interpretability, which is often valued in the financial sector where model transparency is important for compliance and trust.