Loan Approval Prediction

A Company wants to automate the loan eligibility process based on customer detail provided while filling online application form. These details are Gender, Marital Status, Education, Number of Dependents, Income, Loan Amount, Credit History and others. To automate this process, they have given a problem to identify the customers segments, those are eligible for loan amount so that they can specifically target these customers.

In [163]:
```
# Let us import all the necessary libraries in the Jupyter Notebook
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [164]:
```
# Reading the dataset in a 'df' dataframe using Pandas
df = pd.read_csv('LoanData.csv')
```

In [165]:
```
# Exploring the datset
df.head(10)
```

Out[165]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coa
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	
5	LP001011	Male	Yes	2	Graduate	Yes	5417	
6	LP001013	Male	Yes	0	Not Graduate	No	2333	
7	LP001014	Male	Yes	3+	Graduate	No	3036	
8	LP001018	Male	Yes	2	Graduate	No	4006	
9	LP001020	Male	Yes	1	Graduate	No	12841	

In [166]:
```
# Counting rows and column in the dataset
df.shape
```

Out[166]: (614, 13)

In [167]:
```python
# Understanding datatypes and null values of every column
df.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
```

In [168]:
```python
# Summary of numerical variables of the dataset
df.describe()
```

Out[168]:

	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

Understanding various features of the training dataset

In [169]:
```python
# Relation between credit history and Loan status
pd.crosstab(df['Credit_History'], df['Loan_Status'], margins = True)
```

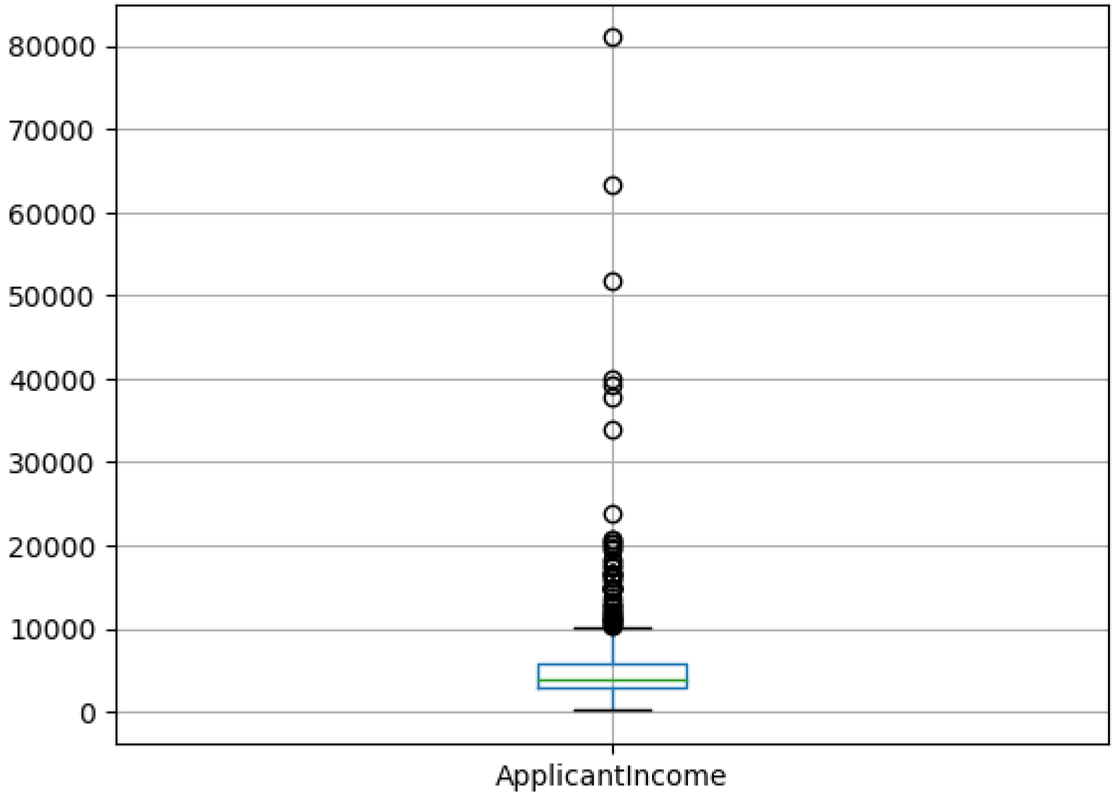
Out[169]:

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

From the above table, we can conclude that applicants who have already have a credit history are more likely to get an approval for the loan.

In [170]:
```
df.boxplot(column= 'ApplicantIncome')
```

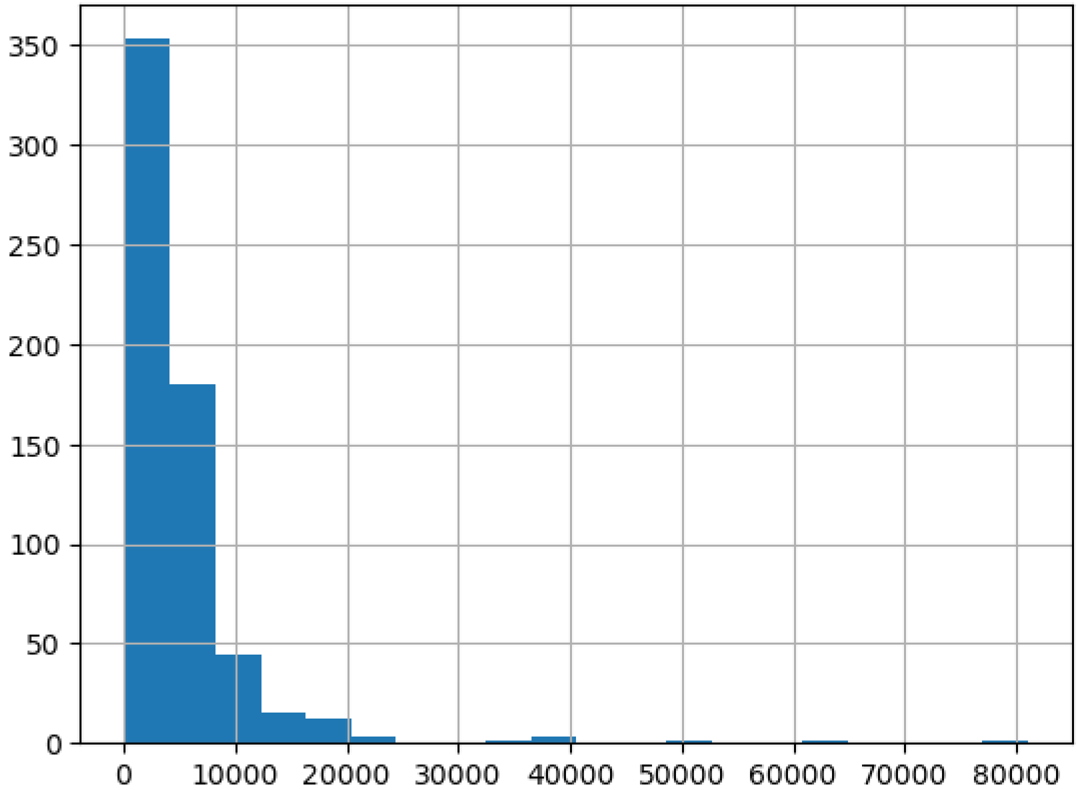
Out[170]: <Axes: >



From the above boxplot, we can say that median Applicant Income is around 5000 and there are considerable number of outliers which shows disparity in income level in the society.

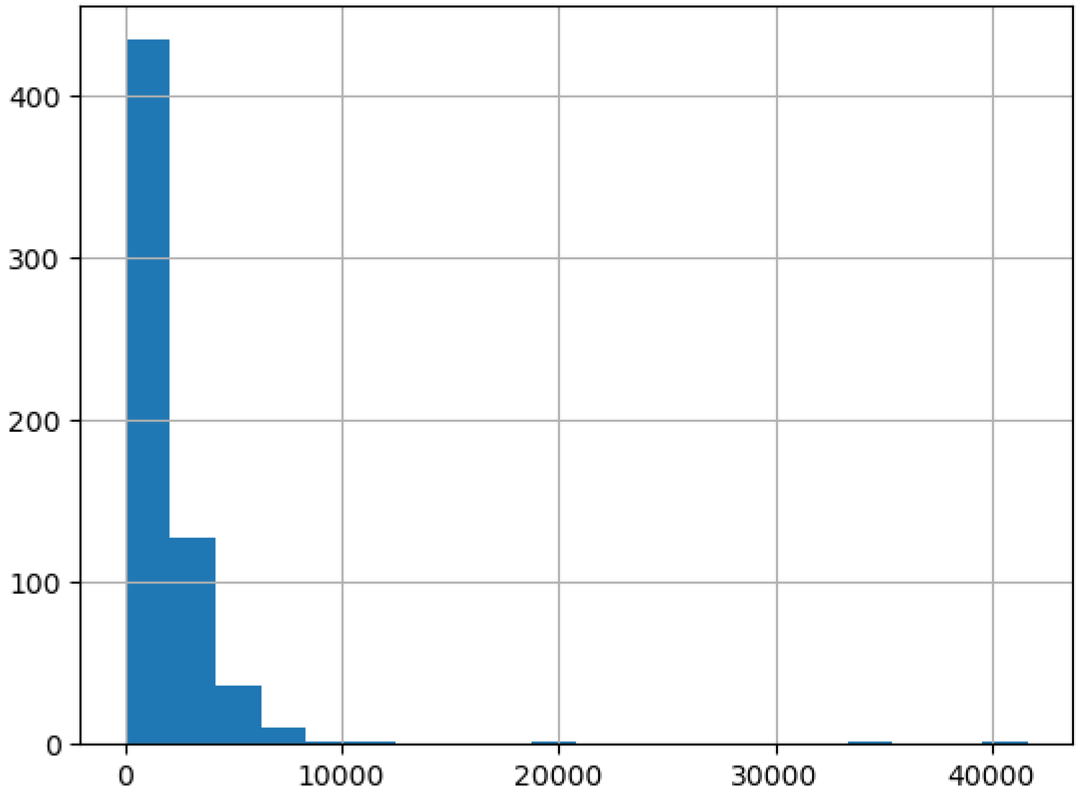
In [171]:
```
# Histogram of variable Applicant income
df['ApplicantIncome'].hist(bins=20)
```

Out[171]: <Axes: >



In [172]:
```
# Histogram of variable Coapplicant income
df['CoapplicantIncome'].hist(bins=20)
```

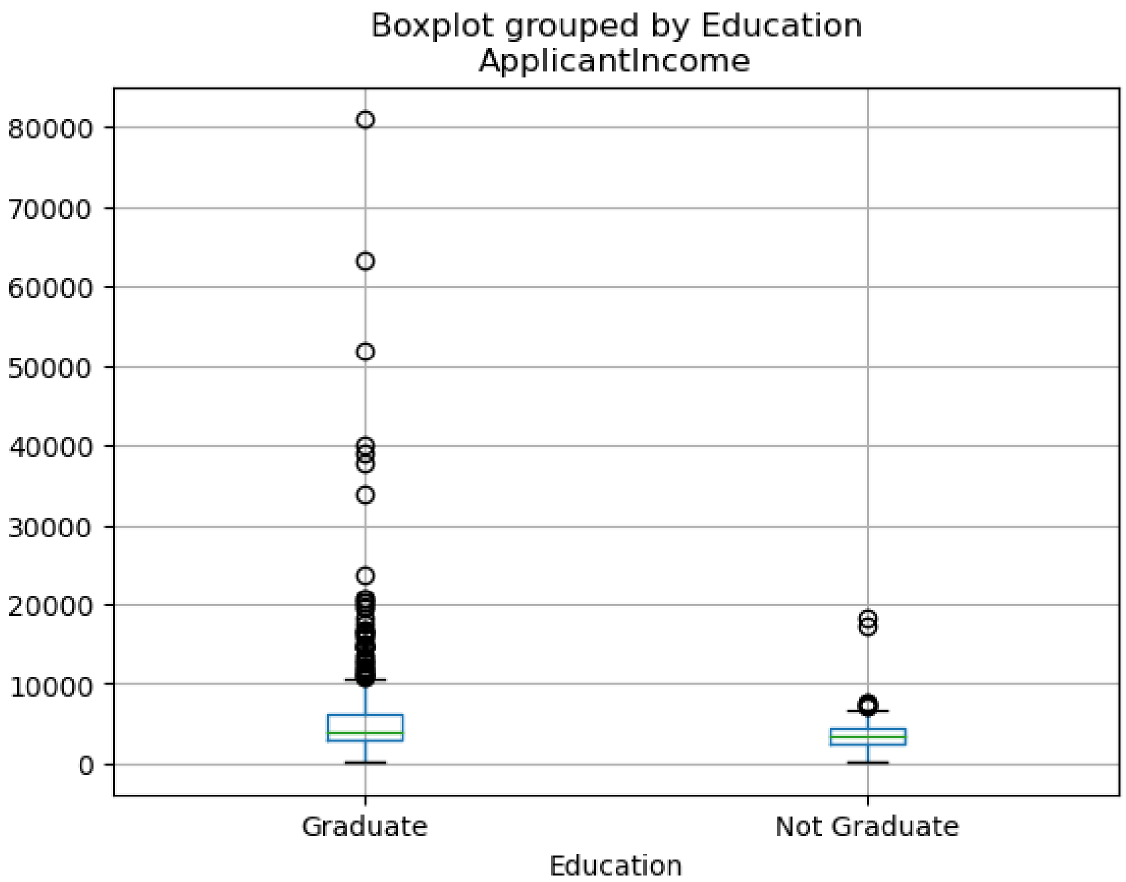
Out[172]: <Axes: >



Both Applicant income and Coapplicant income distribution is right skewd.

In [173]:
```
# Box Plot for variable ApplicantIncome by variable Education
df.boxplot(column= 'ApplicantIncome', by= 'Education')
```

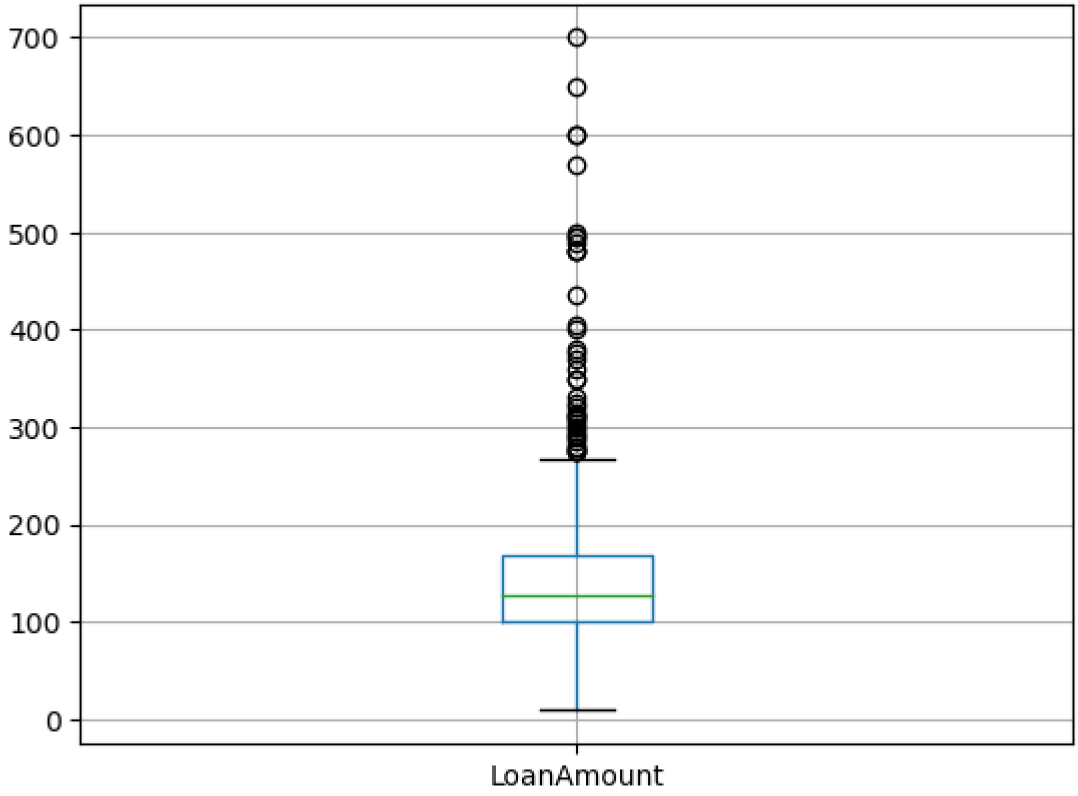
Out[173]: `<Axes: title={'center': 'ApplicantIncome'}, xlabel='Education'>`



Median income of Graduate as well as Non Graduate applicants is similar but graduate applicants have highest income's.

In [174]:
```
# Boxplot and histogram of variable LoanAmount
df.boxplot('LoanAmount')
```

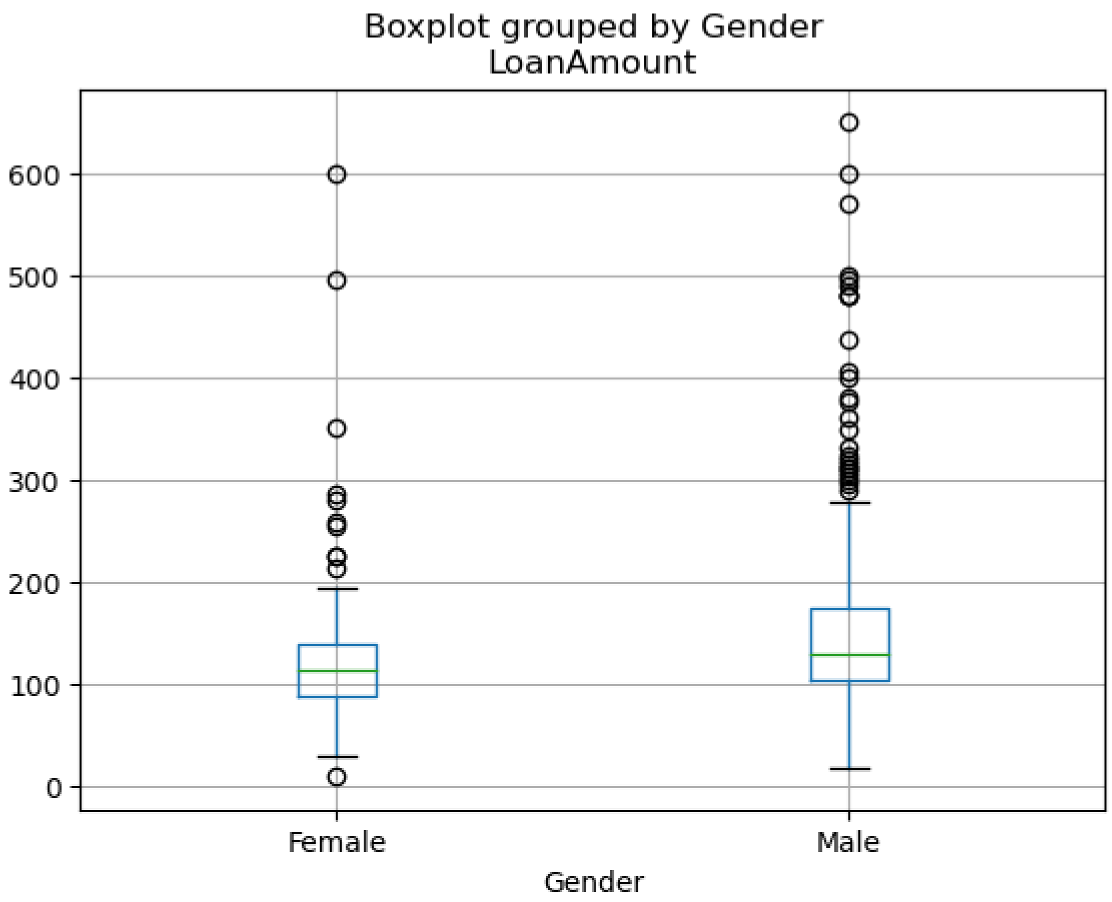
Out[174]: <Axes: >



In [175]:
```python
# Box Plot for variable LoanAmount by variable Gender of training data

df.boxplot(column='LoanAmount', by = 'Gender')
```

Out[175]: <Axes: title={'center': 'LoanAmount'}, xlabel='Gender'>

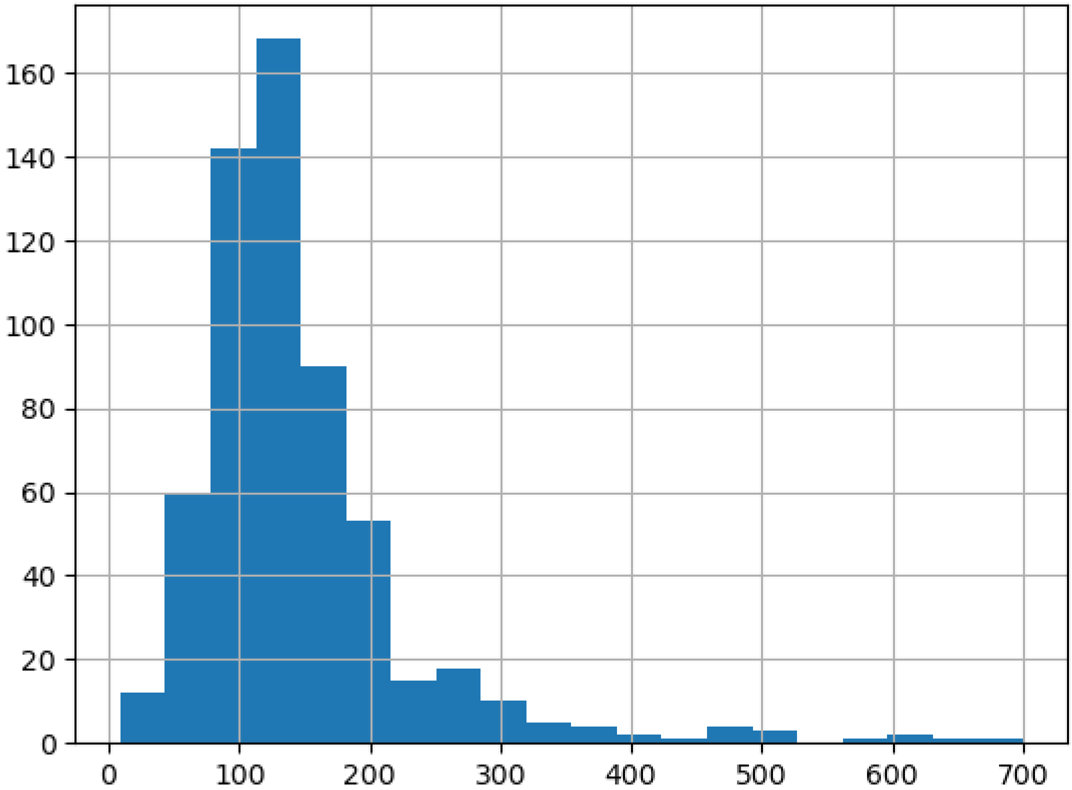


Treating outliers

Loan amount and applicant income has lot of outliers as anyone can apply for higher loan as per their need. Hence we need to normalize this data and we will do that with the help of log function to nullify effects of outliers.

In [176]:
```
# Distribution of Loan amount is right skewd
df['LoanAmount'].hist(bins=20)
```

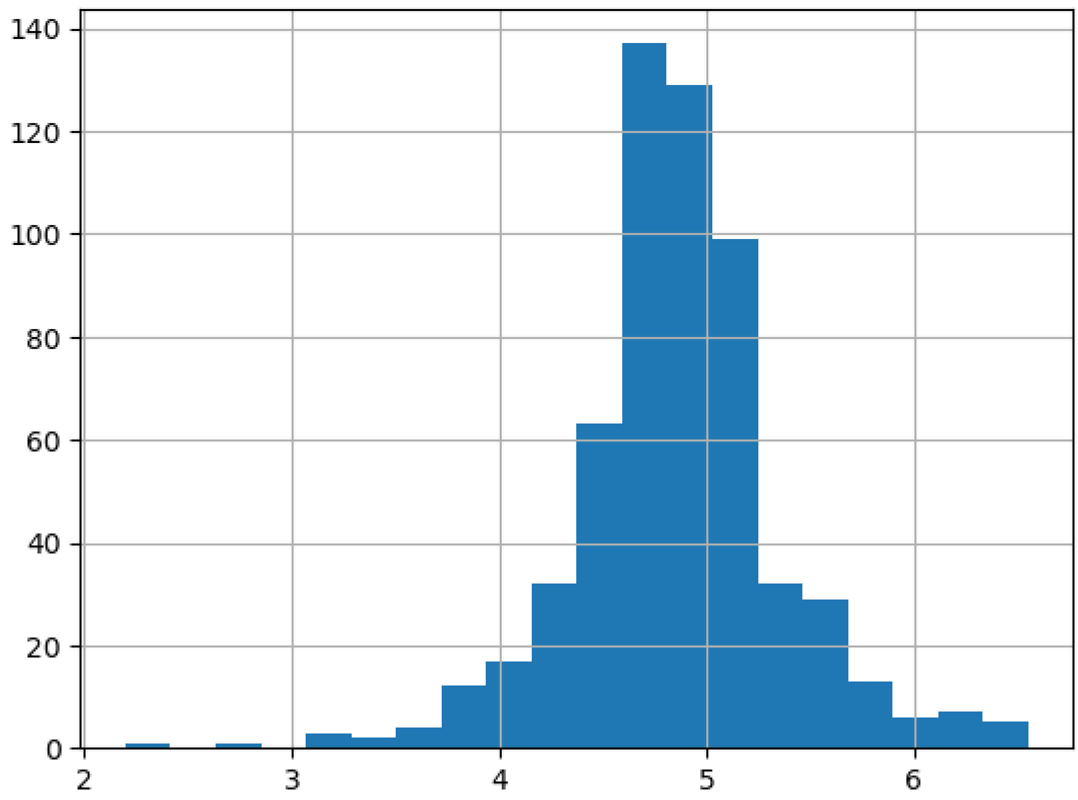
Out[176]: <Axes: >



In [177]:
```
# Applying log function to normalize data
df['LoanAmount_log'] = np.log(df['LoanAmount'])
```

In [178]:
```
# Distribution of data after log function
df['LoanAmount_log'].hist(bins=20)
```

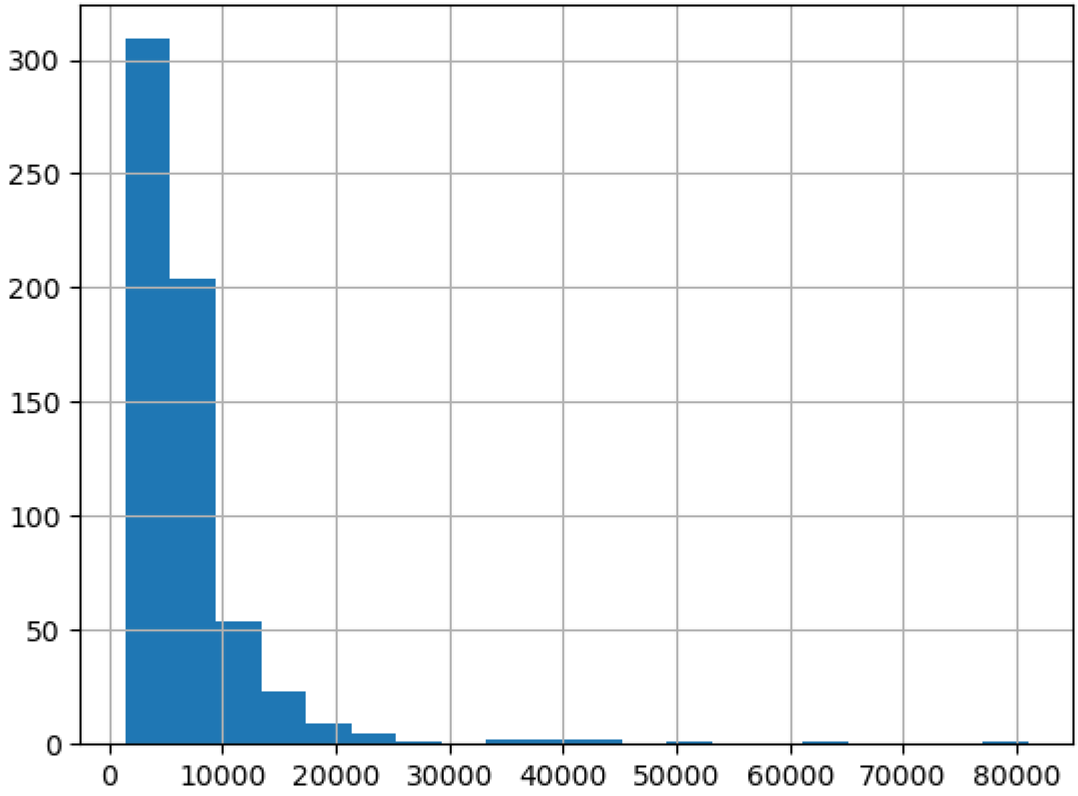
Out[178]: <Axes: >



In [179]:
```
# Adding both ApplicantIncome and CoapplicantIncome to TotalIncome
df['TotalIncome'] = df['ApplicantIncome'] + df['CoapplicantIncome']
```

In [180]:
```
# Distribution of total income is right skewd
df['TotalIncome'].hist(bins=20)
```

Out[180]: <Axes: >

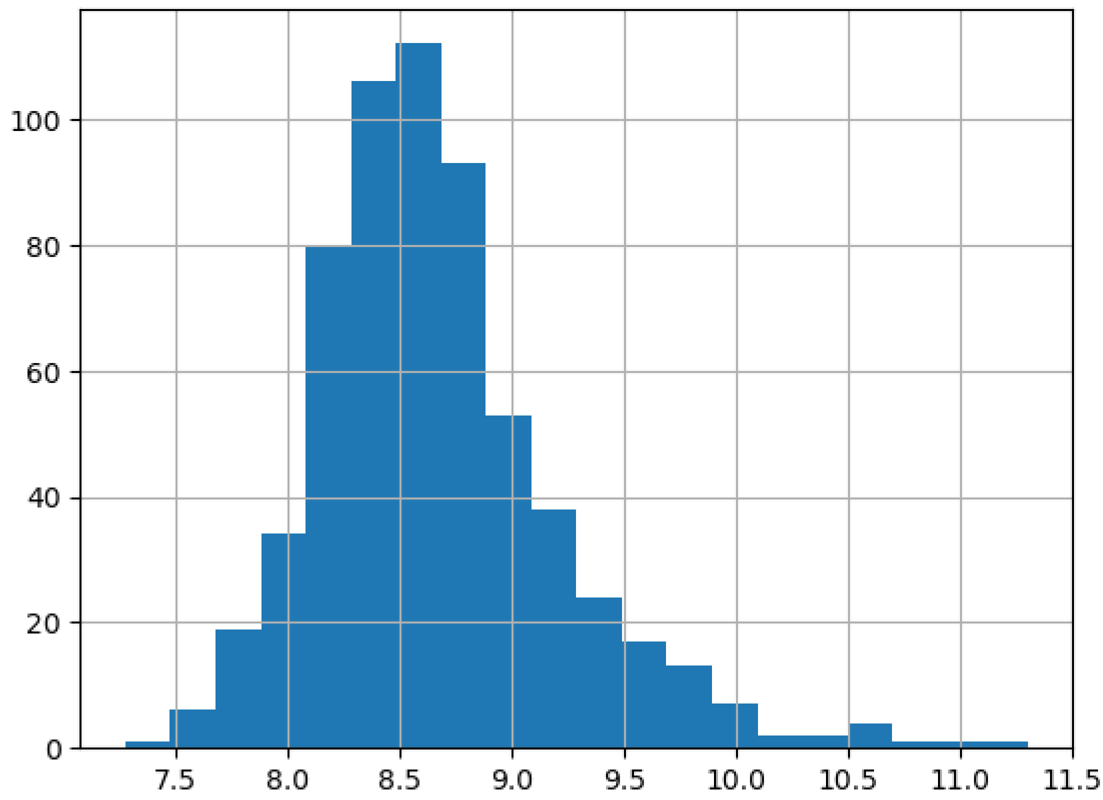


In [181]:
```
# Applying log function to normalize data
df['TotalIncome_log'] = np.log(df['TotalIncome'])
```

In [182]:

```
# Distribution of data after log function
df['TotalIncome_log'].hist(bins=20)
```

Out[182]: <Axes: >



Imputing missing values

In [183]:

```
# Checking null values in the column
df.isnull().sum()
```

Out[183]:

```
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
```

In [184]:
```
# for categorical variable we use mode

df['Gender'].fillna(df['Gender'].mode()[0], inplace= True)
df['Married'].fillna(df['Married'].mode()[0], inplace= True)
df['Dependents'].fillna(df['Dependents'].mode()[0], inplace= True)
df['Self_Employed'].fillna(df['Self_Employed'].mode()[0], inplace= True
df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mode()[0], inplace
df['Credit_History'].fillna(df['Credit_History'].mode()[0], inplace= Tr
```

In [185]:
```
# for numerical variable we use mean

df.LoanAmount = df.LoanAmount.fillna(df.LoanAmount.mean())
df.LoanAmount_log = df.LoanAmount_log.fillna(df.LoanAmount_log.mean())
```

In [186]:
```
df.isnull().sum()
```

Out[186]:
```
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
```

All the null values have been replaced.

In [187]:
```
# Exploring data after imputing missing values
df.head(10)
```

Out[187]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coa
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	
5	LP001011	Male	Yes	2	Graduate	Yes	5417	
6	LP001013	Male	Yes	0	Not Graduate	No	2333	
7	LP001014	Male	Yes	3+	Graduate	No	3036	
8	LP001018	Male	Yes	2	Graduate	No	4006	
9	LP001020	Male	Yes	1	Graduate	No	12841	

Working on training datset

In [188]:
```
# Assigning dependent and independent variables
# 'x' independent (Categorical variables)
# 'y' dependent (Loan_Status)

x = df.iloc[:,np.r_[1:5,9:11,13:15]].values

x
```

Out[188]:
```
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)
```

In [189]:
```
y = df.iloc[:,12].values

y
```

Out[189]:
```
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 [190]:
```
#Importing models from scikit learn module
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0
```

Splitting training data into 'train' and 'test' dataset.

In [191]:
```
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]]
```

In [192]:
```
# Importing models from scikit learn module
# Applying label encoder to convert categorical variables into numerica
from sklearn.preprocessing import LabelEncoder
labelencoder_x = LabelEncoder()
```

In [193]:
```
# Applying label encoder using loops through every column of x_train
for i in range(0,5):
    x_train[:,i] = labelencoder_x.fit_transform(x_train[:,i])
```

In [194]:
```
x_train[:,7] = labelencoder_x.fit_transform(x_train[:,7])
```

In [195]:
```
# Converted data of x_train
x_train
```

Out[195]:
```
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)
```

In [196]:
```
labelencoder_y = LabelEncoder()
```

In [197]:
```
y_train = labelencoder_y.fit_transform(y_train)
```

In [198]:

```python
# Converted data of y_train
y_train
```

Out[198]:

```
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 [199]:

```python
# Applying label encoder using loops through every column of x_test
for i in range(0,5):
    x_test[:,i] = labelencoder_x.fit_transform(x_test[:,i])
```

In [200]:

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

In [235]:

```python
# Converted data of x_test
x_test
```

Out[235]:

```
array([[ 4.66713812e-01, -1.25000000e+00, -6.40593614e-01,
        -5.17726991e-01,  2.99352777e-01,  3.86694596e-01,
        -9.44182815e-01,  7.32623333e-01],
       [-2.14264068e+00, -1.25000000e+00, -6.40593614e-01,
        -5.17726991e-01,  2.99352777e-01,  3.86694596e-01,
        -3.06802355e-01, -8.95402716e-01],
       [ 4.66713812e-01,  8.00000000e-01, -6.40593614e-01,
        -5.17726991e-01,  2.99352777e-01,  3.86694596e-01,
         2.04667756e+00,  1.27529868e+00],
       [ 4.66713812e-01,  8.00000000e-01, -6.40593614e-01,
        -5.17726991e-01,  2.99352777e-01,  3.86694596e-01,
        -3.46723659e-01,  5.89814030e-01],
       [ 4.66713812e-01,  8.00000000e-01,  1.37974009e+00,
        -5.17726991e-01,  2.99352777e-01,  3.86694596e-01,
        -6.25374842e-01, -1.06677388e+00],
       [ 4.66713812e-01,  8.00000000e-01, -6.40593614e-01,
         1.93151993e+00, -2.07615636e+00, -2.58602011e+00,
         5.51613645e-01,  3.04195425e-01],
       [ 4.66713812e-01,  8.00000000e-01,  2.38990694e+00,
```

In [202]:
```
y_test = labelencoder_y.fit_transform(y_test)
```

In [236]:
```
# Converted data of y_test
y_test
```

Out[236]:
```
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 [204]:
```
# Importing StandardScaler model to standardize the data
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
x_train = ss.fit_transform(x_train)
x_test = ss.fit_transform(x_test)
```

Classifiers

Applying different classification techniques to the data to understand the most accurate classifier.

In [205]:
```
# Importing DecisionTreeClassifier from Scikit learn module
from sklearn.tree import DecisionTreeClassifier
DTC = DecisionTreeClassifier(criterion= 'entropy', random_state= 0)
DTC.fit(x_train,y_train)
```

Out[205]:
```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [206]:
```
# Predicting outcome for x_test data using DecisionTreeClassifier
y_pred = DTC.predict(x_test)
y_pred
```

Out[206]:
```
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])
```

In [207]:
```
# Checking accuracy of predicted x_test outcome with y_test
from sklearn import metrics
print ("The accuracy of decision tree is", metrics.accuracy_score(y_pre
```

```
The accuracy of decision tree is 0.7073170731707317
```

In [208]:
```
# Importing Naive bayes from Scikit learn module
from sklearn.naive_bayes import GaussianNB
NB = GaussianNB()
NB.fit(x_train, y_train)
```

Out[208]:
```
GaussianNB()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [237]:
```
# Predicting outcome for x_test data using Naive bayes
y_pred = NB.predict(x_test)
y_pred
```

Out[237]:
```
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 [210]:
```
# Checking accuracy of predicted x_test outcome with y_test
print ("The accuracy of Naive Bayes is", metrics.accuracy_score(y_pred,
```

```
The accuracy of Naive Bayes is 0.8292682926829268
```

In [211]:
```
# Importing RandomForestClassifier from Scikit learn module
from sklearn.ensemble import RandomForestClassifier
RFC = RandomForestClassifier()
RFC.fit(x_train,y_train)
```

Out[211]:
```
RandomForestClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [212]:
```
# Predicting outcome for x_test data using RandomForestClassifier
y_pred = RFC.predict(x_test)
y_pred
```

Out[212]:
```
array([1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 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, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1])
```

In [213]:
```
# Checking accuracy of predicted x_test outcome with y_test
print ("The accuracy of Random Forest is", metrics.accuracy_score(y_pre
```

```
The accuracy of Random Forest is 0.7723577235772358
```

In [214]:
```
# Importing KNeighborsClassifier from Scikit learn module
from sklearn.neighbors import KNeighborsClassifier
KNC = KNeighborsClassifier()
KNC.fit(x_train,y_train)
```

Out[214]: KNeighborsClassifier()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [215]:
```
# Predicting outcome for x_test data using KNeighborsClassifier
y_pred = KNC.predict(x_test)
y_pred
```

Out[215]:
```
array([1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 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, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1])
```

In [216]:
```
# Checking accuracy of predicted x_test outcome with y_test
print ("The accuracy of KNeighbors is", metrics.accuracy_score(y_pred,
```

```
The accuracy of KNeighbors is 0.7967479674796748
```

After applying different classification techniques on the data, we can conclude that 'Naive Bayes' is the most accuarate classifier with accuracy of 83%. So, we will use Naive Bayes classifier to predict our actual data.

Working on the actual data

In [245]:
```
# Reading CSV file of actual data
test = pd.read_csv('TestData.csv')
```

In [246]:

```
# Exploring the data
test.head(10)
```

Out[246]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coa
0	LP001015	Male	Yes	0	Graduate	No	5720	
1	LP001022	Male	Yes	1	Graduate	No	3076	
2	LP001031	Male	Yes	2	Graduate	No	5000	
3	LP001035	Male	Yes	2	Graduate	No	2340	
4	LP001051	Male	No	0	Not Graduate	No	3276	
5	LP001054	Male	Yes	0	Not Graduate	Yes	2165	
6	LP001055	Female	No	1	Not Graduate	No	2226	
7	LP001056	Male	Yes	2	Not Graduate	No	3881	
8	LP001059	Male	Yes	2	Graduate	NaN	13633	
9	LP001067	Male	No	0	Not Graduate	No	2400	

In [247]:

```
test.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
```

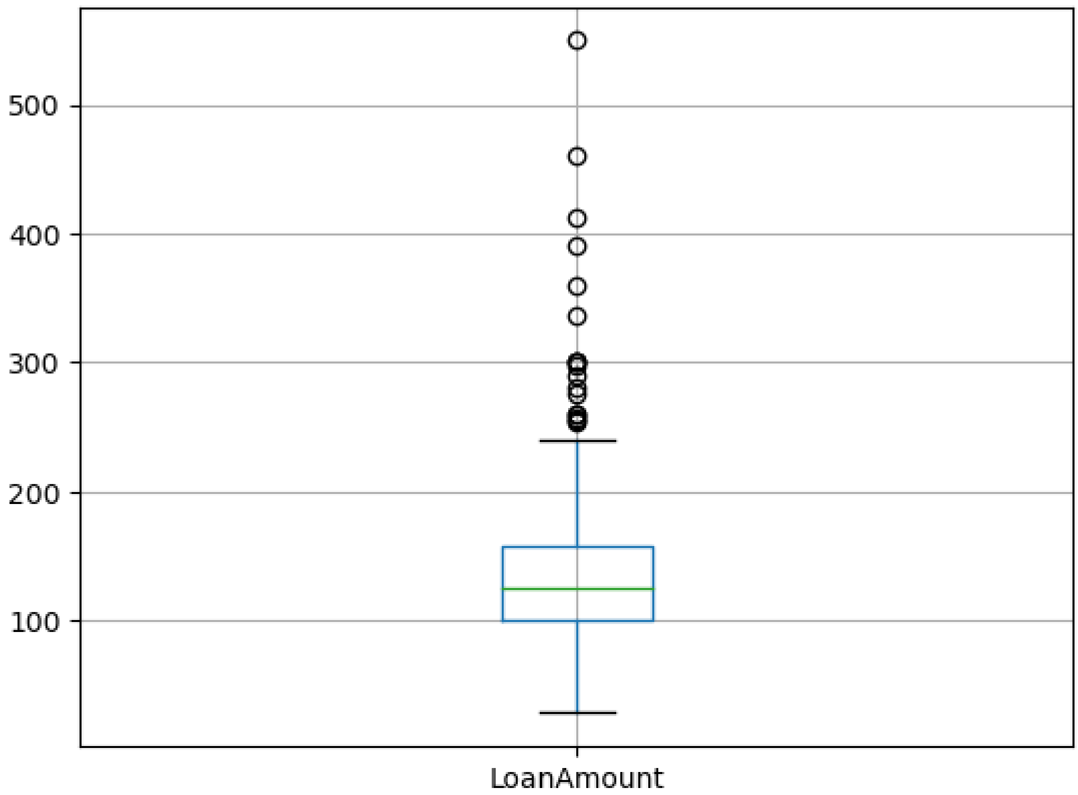
In [248]:
```
# Checking null values
test.isnull().sum()
```

Out[248]:
```
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
```

In [249]:
```
# Imputing null values for categorical variables
test['Gender'].fillna(test['Gender'].mode()[0], inplace= True)
test['Dependents'].fillna(test['Dependents'].mode()[0], inplace= True)
test['Self_Employed'].fillna(test['Self_Employed'].mode()[0], inplace=
test['Loan_Amount_Term'].fillna(test['Loan_Amount_Term'].mode()[0], inp
test['Credit_History'].fillna(test['Credit_History'].mode()[0], inplace
```

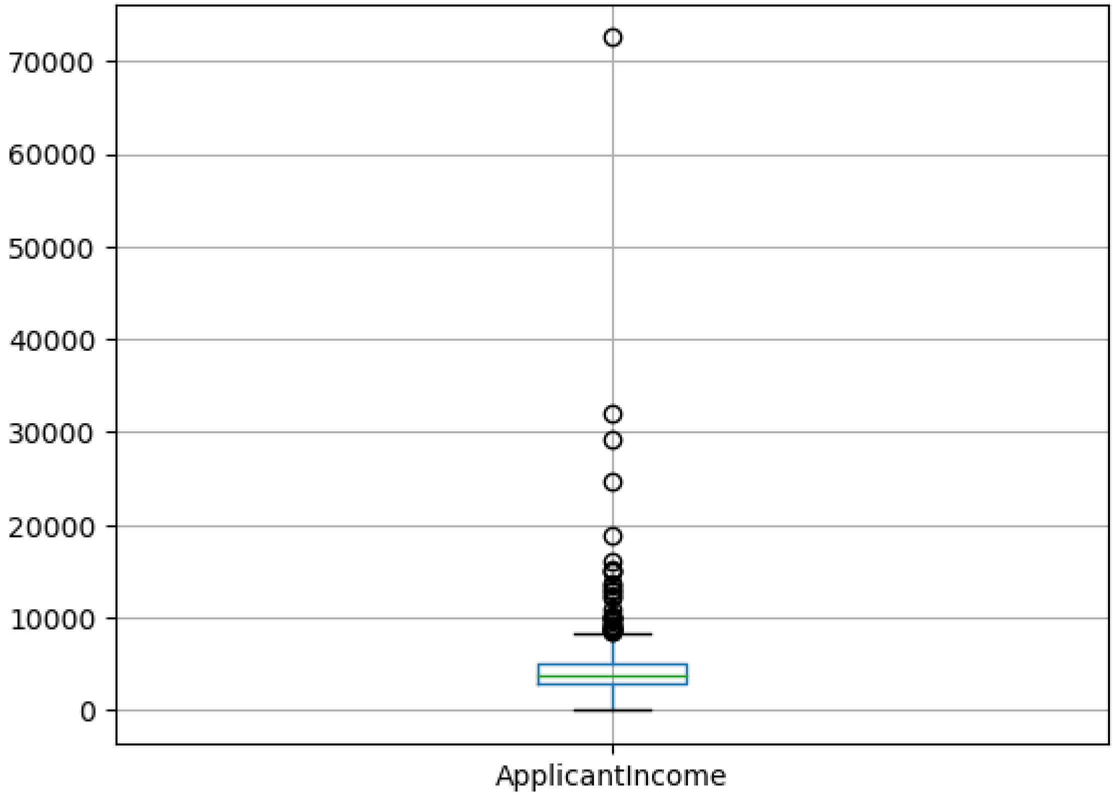
In [250]:
```
# Boxplot of variable LoanAmount
test.boxplot(column= 'LoanAmount')
```

Out[250]: <Axes: >



In [251]:
```
# Boxplot of variable ApplicantIncome
test.boxplot(column= 'ApplicantIncome')
```

Out[251]: <Axes: >



In [252]:
```
# Imputing null values for numerical variable
test.LoanAmount = test.LoanAmount.fillna(test.LoanAmount.mean())
```

In [253]:
```
test['LoanAmount_log'] = np.log(test['LoanAmount'])
```

In [254]:
```
test['TotalIncome'] = test['ApplicantIncome'] + test['CoapplicantIncome
```

In [255]:
```
test['TotalIncome_log'] = np.log(test['TotalIncome'])
```

In [256]:
```
# Replacing all the null values
test.isnull().sum()
```

Out[256]:
```
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
```

In [257]:
```
test.head(10)
```

Out[257]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coa
0	LP001015	Male	Yes	0	Graduate	No	5720	
1	LP001022	Male	Yes	1	Graduate	No	3076	
2	LP001031	Male	Yes	2	Graduate	No	5000	
3	LP001035	Male	Yes	2	Graduate	No	2340	
4	LP001051	Male	No	0	Not Graduate	No	3276	
5	LP001054	Male	Yes	0	Not Graduate	Yes	2165	
6	LP001055	Female	No	1	Not Graduate	No	2226	
7	LP001056	Male	Yes	2	Not Graduate	No	3881	
8	LP001059	Male	Yes	2	Graduate	No	13633	
9	LP001067	Male	No	0	Not Graduate	No	2400	

In [230]:
```
testDS = test.iloc[:, np.r_[1:5,9:11,13:15]].values
```

In [231]:
```
# Converting categorical variable into numerical variable
for i in range(0,5):
    testDS[:,i] = labelencoder_x.fit_transform(testDS[:,i])
    testDS[:,7] = labelencoder_x.fit_transform(testDS[:,7])
```

In [232]:
```
testDS
```

Out[232]:
```
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)
```

In [233]:
```
# Using StandardScaler model to standardize data
testDS = ss.fit_transform(testDS)
```

In [259]:

```
# Predicting Loan Status of all the applicant
pred = NB.predict(testDS)
pred
```

Out[259]:

```
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])
```

0 - Loan not approved, 1 - Loan approved

Conclusion

In conclusion, this Python project successfully achieved its objectives. We began by cleaning the training dataset, ensuring data quality. Through exploratory data analysis, we gained insights into the dataset's characteristics. Categorical variables were transformed into numerical form for modeling. The dataset was split into 'train' and 'test' subsets to evaluate model performance. Employing various classification techniques, we determined that Naive Bayes exhibited the highest accuracy, achieving an impressive 83% accuracy rate. This accuracy allowed us to predict the 'Loan status' of applicants, making this project a valuable tool for decision-making in the lending industry.