Loan Eligibility Prediction

Loan eligibility is defined as a set of criteria basis which a financial institution evaluates to decide the eligiblity of a customer for a particular loan.

Criterias Loan amount, Dependents, Marital Status, Applicant Income, Laon amount term, Co-applicant income, Gender, Credit History, Property Area

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

df = pd.read_csv("loan-train.csv")

df.head()
```

```
    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
```

Let's Explore our Data**

```
df.shape
```

```
(614, 13)
```

```
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
```

- Some values are missing from some of the columns.

```
df.describe().T

                   count         mean          std    min     25%
50%  \
ApplicantIncome    614.0  5403.459283  6109.041673  150.0  2877.5
3812.5
CoapplicantIncome  614.0  1621.245798  2926.248369    0.0     0.0
1188.5
LoanAmount         592.0   146.412162    85.587325    9.0   100.0
128.0
Loan_Amount_Term   600.0   342.000000    65.120410   12.0   360.0
360.0
Credit_History     564.0     0.842199     0.364878    0.0     1.0
1.0

                       75%      max
ApplicantIncome    5795.00  81000.0
CoapplicantIncome  2297.25  41667.0
LoanAmount          168.00    700.0
Loan_Amount_Term    360.00    480.0
Credit_History        1.00      1.0
```

How 'Credit history' affects the 'Loan status'?

```
# Using crosstab() to established relationship

# This method is used to compute a simple cross-tabulation of two (or
more) factors.
# By default, computes a frequency table of the factors unless an
array of values and
# an aggregation function are passed.
```

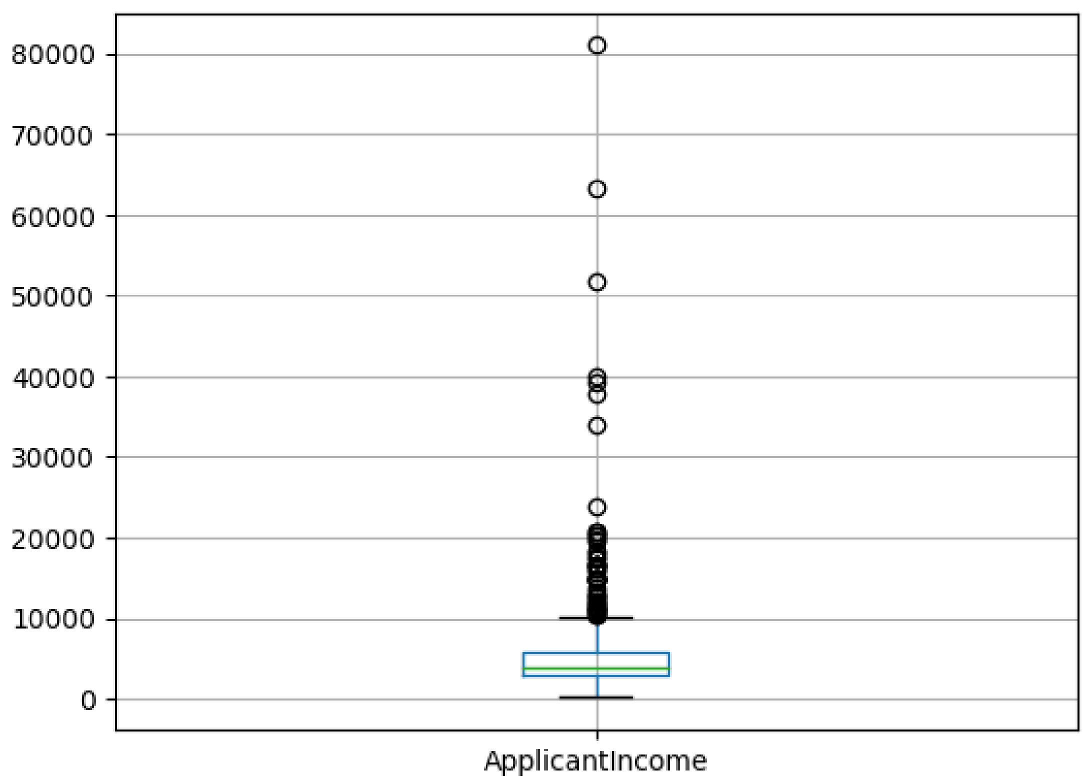
```
pd.crosstab(df['Credit_History'], df['Loan_Status'], margins=True)

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

Applicant with credit history as 1 are more eligible for loan than with credit history = 0 (378 vs 7)

Data Visualization

- Exploring Some of the Variable by visualizing them.

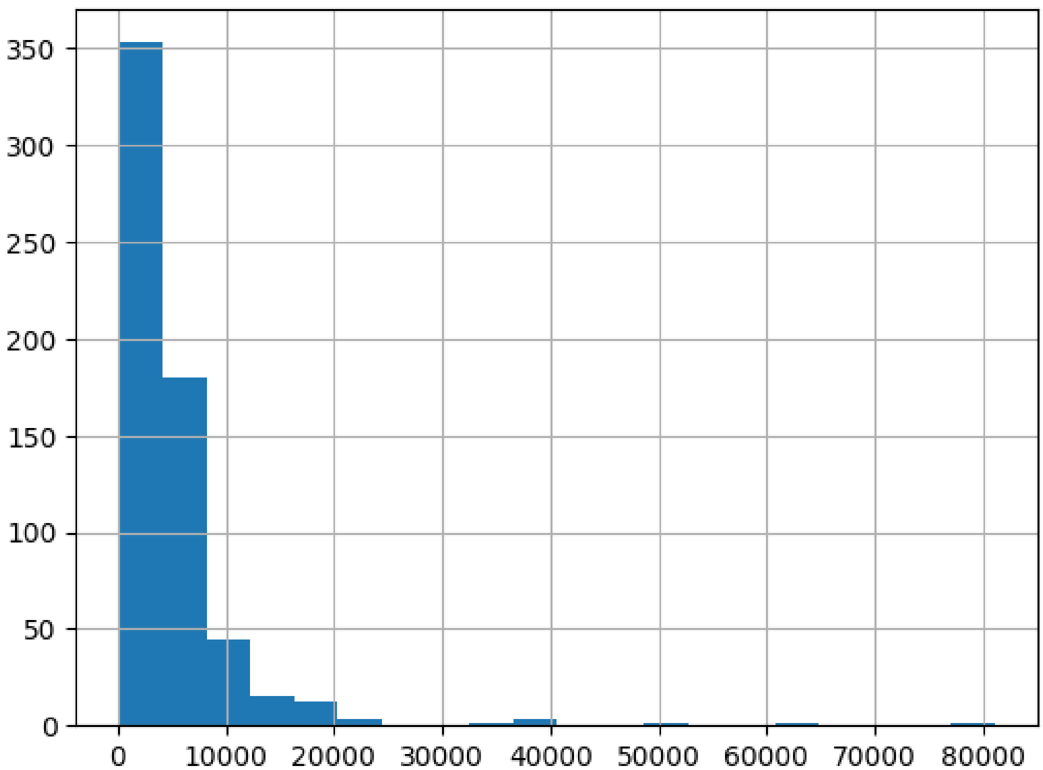
```
# Applicant Income using boxplot, as it helps in identifying outliers
in the dataset.

df.boxplot(column='ApplicantIncome')

<Axes: >
```



Lot's of outliers

```python
# Histogram

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

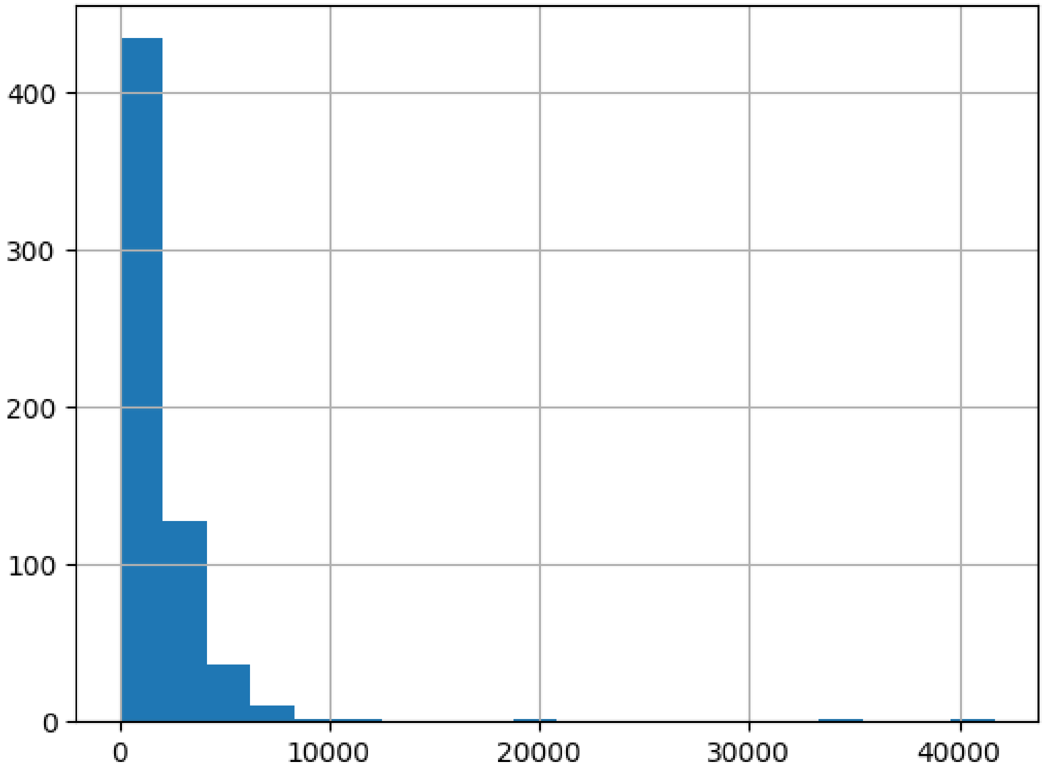
```
<Axes: >
```



Clearly, it is rightly skewed histogram. We have to normalize the values.

```python
# Coapplicant Income

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

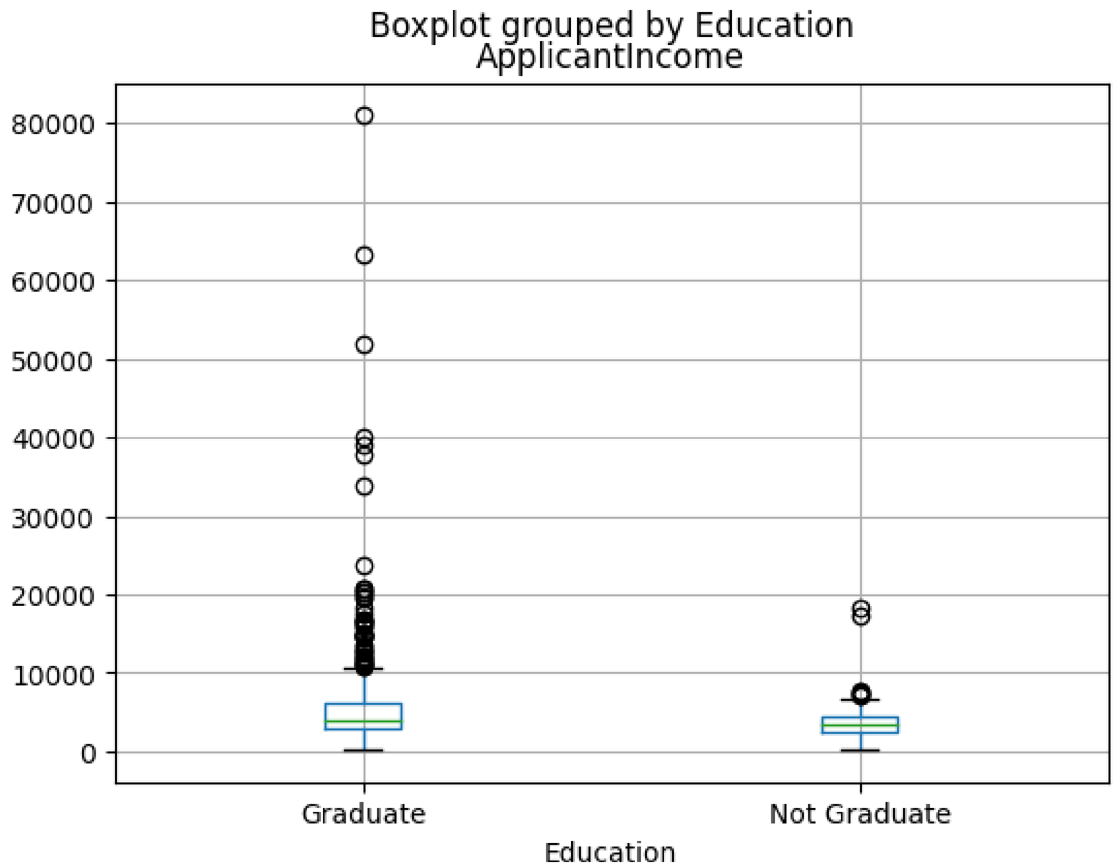
```
<Axes: >
```



Also right skewed.

```
# Now, Let's explore relantionship between applicantIncome and their
education through boxplot

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

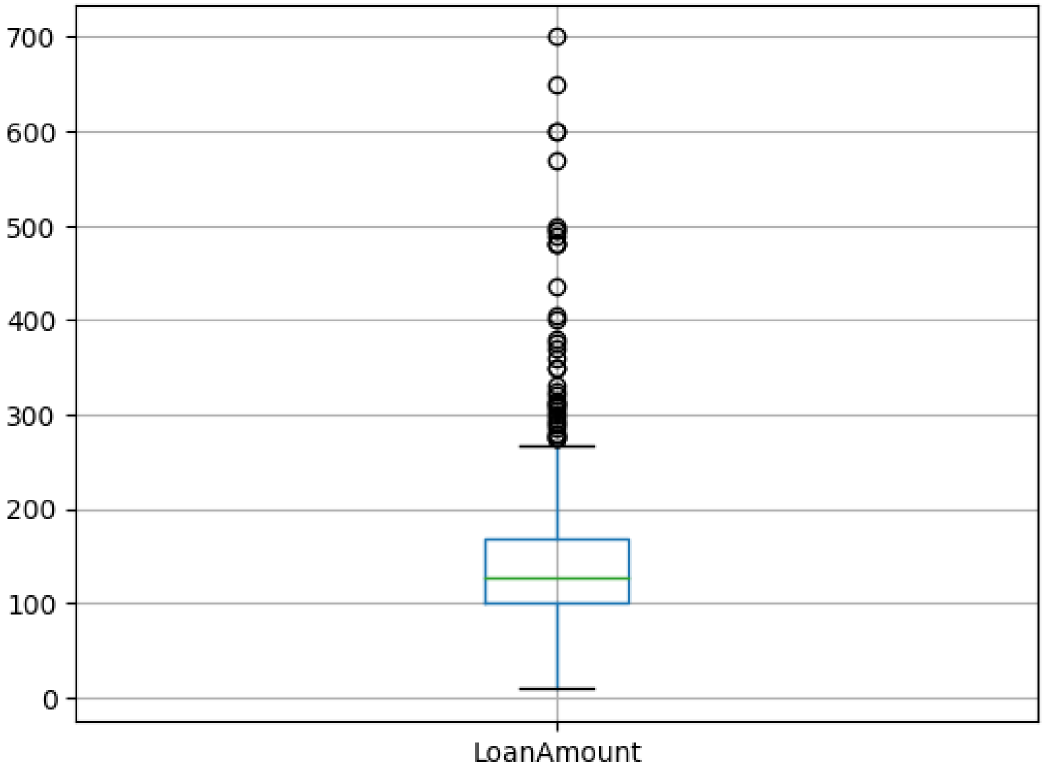
```
<Axes: title={'center': 'ApplicantIncome'}, xlabel='Education'>
```



- Median Salary doesn't vary too much for Graduate vs Not Graduate.
- But Some of the Graduates have very high Salary.This kind of variation is quite common.
- But Normalising and Scaling these value is one important step we've to follow and implement for pre-processing.

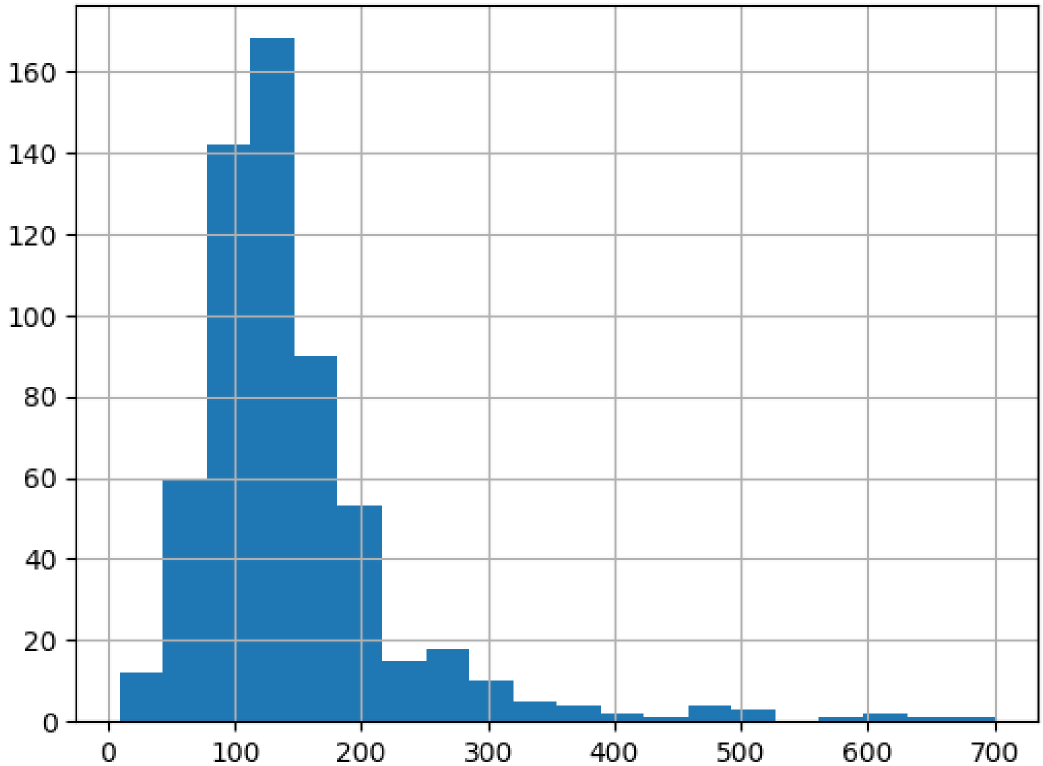
```
# Loan Amount

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

```
<Axes: >
```



Lot's of Outlier.

```
#Let's also draw histogram for loan amount variant

df['LoanAmount'].hist(bins=20)

<Axes: >
```

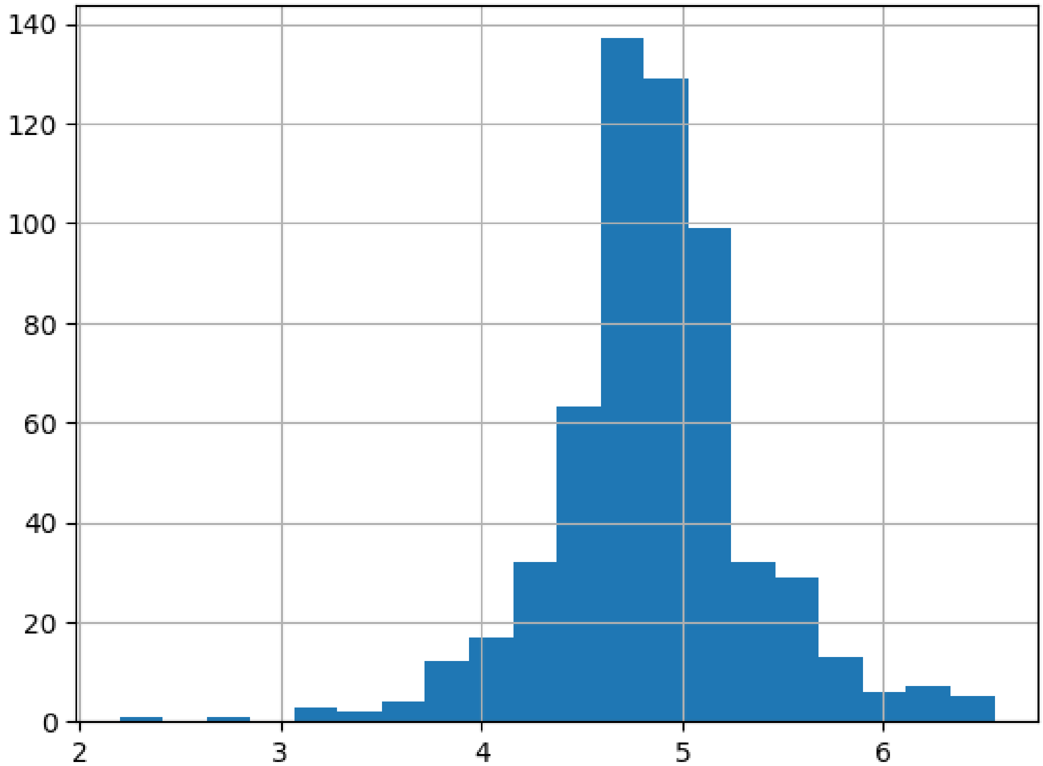


Little right skewed.

Normalising right skewed data.

- We'll be using **Log function**.

```
# Normalizing Loan Amount

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

# Visualizing LoanAmount_Log
df['LoanAmount_log'].hist(bins=20)

<Axes: >
```



Looks a lot more normalized then before.

Look for missing values

```
df.isnull().sum()
```

```
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
```

Handling missing values

Gender

```
# Since Gender is a categorical variable, we will be using mode
function

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

Married

```
df['Married'].fillna(df['Married'].mode()[0], inplace=True)
```

Dependent

```
df['Dependents'].fillna(df['Dependents'].mode()[0], inplace=True)
```

Self_Employed

```
df['Self_Employed'].fillna(df['Self_Employed'].mode()[0],
inplace=True)
```

LoanAmount & LoanAmount_log

```
# It is not a categorical value, but a quantitative value. Hence we
will be using mean() to replace the missing value

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

LoanAmount_Term

```
df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mode()[0],
inplace=True)
```

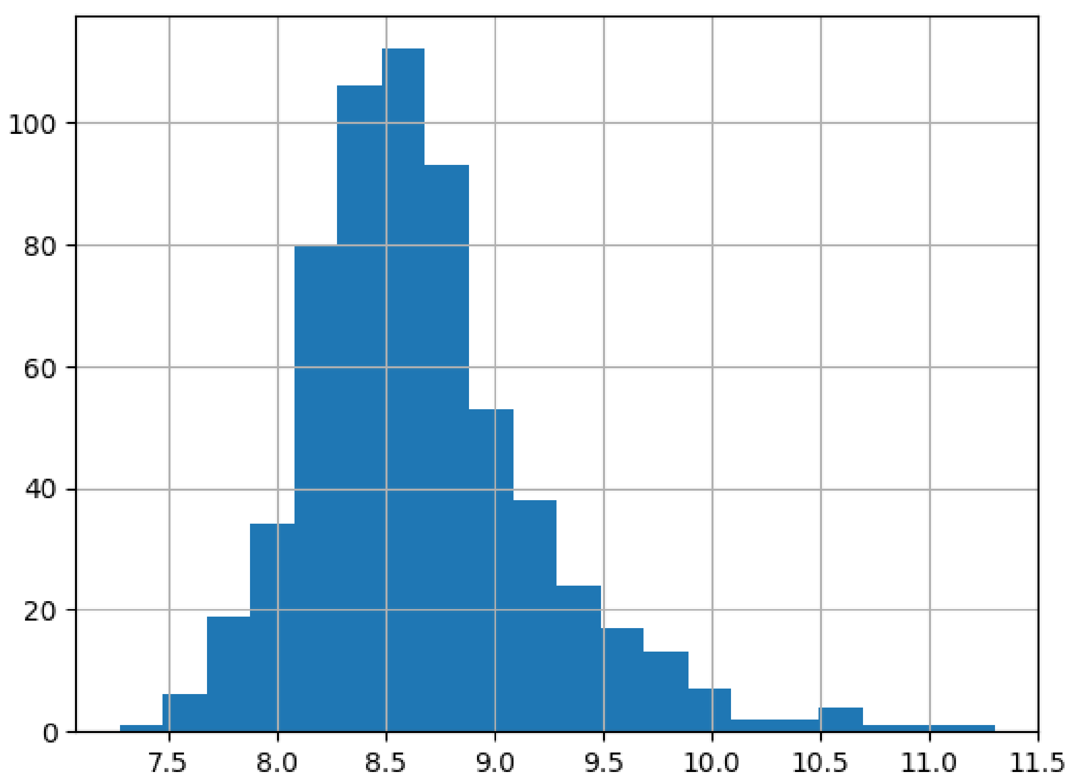
Credit_History

```
# It is in 0 and 1.

df['Credit_History'].fillna(df['Credit_History'].mode()[0],
inplace=True)

# Let's check if the missing values are handled or not

df.isnull().sum()

Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
```

```
ApplicantIncome        0
CoapplicantIncome      0
LoanAmount             0
Loan_Amount_Term       0
Credit_History         0
Property_Area          0
Loan_Status            0
LoanAmount_log         0
dtype: int64
```

- All missing values are handled.

```
# ApplicantIncome and CoapplicantIncome, both are right skewed. Instead of normalizing them separately,
# we will be combining them and then log over the total value

df['TotalIncome'] = df['ApplicantIncome'] + df['CoapplicantIncome']

df['TotalIncome_log'] = np.log(df['TotalIncome'])

df['TotalIncome_log'].hist(bins=20)

<Axes: >
```



- Looks normalize compare to before.

```
df.head()
```

```
    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
```

Dividing dataset into Dependent (y) and Independent variables (X)

```
# X represent all the independent variables
X = df.iloc[:,np.r_[1:5,9:11,13:15]].values

# y is the dependent variable that we need to predict, which is the
'loan status' column at 12th index
y = df.iloc[:,12].values

X
```

```
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)
```

```
y
```

```
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)
```

Split dataset into train and test dataset

```
# Run the below comman to install scikit module, if not installed

# pip install scikit-learn

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,
random_state = 0)

# test_size=0.2 # as we want 80% data for training the model
# random_state = 0 # as we want the same result to change in every
cycle. If not given zero, it will keep changing

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]]
```

There are categorical values in here, we need to change that we will be using labelEncoder to convert these categorical values into numeric format.

Encoding

```
from sklearn.preprocessing import LabelEncoder

labelencoder_X = LabelEncoder()

# Let use fit_tranform() func. of LabelEncoder() class to convert the
indexes we want from String to numeric format

for i in range(0, 5):
    X_train[:,i] = labelencoder_X.fit_transform(X_train[:,i])

# similarly, do that for 7th index

X_train[:,7] = labelencoder_X.fit_transform(X_train[:,7])

X_train
```

```
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)
```

- Textual format is converted to numeric format now.

Let's encode y_train now

```
labelencoder_y = LabelEncoder()

y_train = labelencoder_y.fit_transform(y_train)

y_train

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])
```

```
# Encoding test data now

X_test
```

```
array([['Male', 'No', '0', 'Graduate', 360.0, 1.0, 4.430816798843313,
        7085.0],
       ['Female', 'No', '0', 'Graduate', 360.0, 1.0,
4.718498871295094,
        4230.0],
       ['Male', 'Yes', '0', 'Graduate', 360.0, 1.0, 5.780743515792329,
        10039.0],
       ['Male', 'Yes', '0', 'Graduate', 360.0, 1.0, 4.700480365792417,
        6784.0],
       ['Male', 'Yes', '2', 'Graduate', 360.0, 1.0, 4.574710978503383,
        3875.0],
       ['Male', 'Yes', '0', 'Not Graduate', 180.0, 0.0,
5.10594547390058,
        6058.0],
       ['Male', 'Yes', '3+', 'Graduate', 180.0, 1.0,
5.056245805348308,
        6417.0],
       ['Male', 'No', '0', 'Graduate', 360.0, 1.0, 6.003887067106539,
        12876.0],
       ['Male', 'No', '0', 'Graduate', 360.0, 0.0, 4.820281565605037,
        5124.0],
       ['Male', 'Yes', '0', 'Graduate', 360.0, 1.0, 4.852030263919617,
        5233.0],
       ['Female', 'No', '0', 'Graduate', 360.0, 1.0,
4.430816798843313,
        2917.0],
       ['Male', 'Yes', '1', 'Graduate', 360.0, 1.0, 4.553876891600541,
        2895.0],
       ['Female', 'No', '0', 'Graduate', 360.0, 1.0,
5.634789603169249,
        8333.0],
       ['Male', 'Yes', '2', 'Graduate', 360.0, 1.0,
```

```
5.4638318050256105,
        8667.0],
       ['Male', 'Yes', '0', 'Graduate', 360.0, 1.0, 4.564348191467836,
        14880.0],
       ['Male', 'Yes', '1', 'Graduate', 360.0, 1.0, 4.204692619390966,
        3875.0],
       ['Male', 'No', '1', 'Not Graduate', 360.0, 1.0,
5.247024072160486,
        4311.0],
       ['Male', 'No', '0', 'Not Graduate', 360.0, 1.0,
4.882801922586371,
        3946.0],
       ['Female', 'No', '0', 'Graduate', 360.0, 1.0,
4.532599493153256,
        2500.0],
       ['Male', 'Yes', '0', 'Not Graduate', 360.0, 0.0,
        5.198497031265826, 4787.0],
       ['Female', 'Yes', '0', 'Graduate', 360.0, 0.0,
4.787491742782046,
        6085.0],
       ['Male', 'Yes', '0', 'Graduate', 360.0, 1.0, 4.962844630259907,
        4765.0],
       ['Male', 'Yes', '2', 'Graduate', 360.0, 1.0, 4.68213122712422,
        7550.0],
       ['Male', 'Yes', '2', 'Graduate', 360.0, 1.0, 5.10594547390058,
        11500.0],
       ['Male', 'Yes', '0', 'Graduate', 360.0, 1.0, 4.060443010546419,
        4521.0],
       ['Male', 'Yes', '1', 'Graduate', 360.0, 1.0, 5.521460917862246,
        8069.0],
       ['Male', 'No', '0', 'Graduate', 360.0, 1.0, 5.231108616854587,
        8724.0],
       ['Male', 'Yes', '0', 'Graduate', 360.0, 1.0, 5.231108616854587,
        11333.0],
       ['Male', 'Yes', '3+', 'Graduate', 360.0, 0.0,
4.852030263919617,
        4680.0],
       ['Female', 'No', '0', 'Graduate', 360.0, 0.0,
4.634728988229636,
        5000.0],
       ['Male', 'Yes', '0', 'Graduate', 360.0, 1.0, 5.429345628954441,
        9083.0],
       ['Male', 'No', '0', 'Not Graduate', 360.0, 1.0,
3.871201010907891,
        4885.0],
       ['Male', 'Yes', '1', 'Not Graduate', 360.0, 1.0,
        4.499809670330265, 5100.0],
       ['Male', 'Yes', '0', 'Graduate', 360.0, 1.0, 5.19295685089021,
        9734.0],
```

```
       ['Male', 'Yes', '0', 'Graduate', 360.0, 1.0, 4.857444178729352,
        8235.0],
       ['Female', 'Yes', '0', 'Not Graduate', 360.0, 0.0,
        5.181783550292085, 5386.0],
       ['Male', 'Yes', '0', 'Graduate', 360.0, 1.0, 5.147494476813453,
        5717.0],
       ['Male', 'No', '0', 'Not Graduate', 360.0, 1.0,
4.836281906951478,
        4592.0],
       ['Male', 'Yes', '0', 'Graduate', 360.0, 1.0, 4.852030263919617,
        6250.0],
       ['Male', 'Yes', '2', 'Not Graduate', 360.0, 1.0,
4.68213122712422,
        3917.0],
       ['Female', 'No', '0', 'Graduate', 360.0, 1.0,
4.382026634673881,
        3244.0],
       ['Male', 'Yes', '3+', 'Graduate', 360.0, 0.0,
4.812184355372417,
        5900.0],
       ['Male', 'Yes', '2', 'Graduate', 120.0, 1.0, 2.833213344056216,
        2385.0],
       ['Male', 'Yes', '1', 'Not Graduate', 360.0, 1.0,
        5.062595033026967, 5783.0],
       ['Male', 'No', '0', 'Graduate', 360.0, 1.0, 4.330733340286331,
        3858.0],
       ['Male', 'No', '0', 'Graduate', 360.0, 1.0, 5.231108616854587,
        12083.0],
       ['Male', 'Yes', '1', 'Graduate', 360.0, 1.0,
4.7535901911063645,
        3750.0],
       ['Female', 'No', '0', 'Graduate', 360.0, 1.0, 4.74493212836325,
        4547.0],
       ['Male', 'Yes', '1', 'Graduate', 360.0, 1.0, 4.852030263919617,
        6091.0],
       ['Male', 'No', '0', 'Graduate', 360.0, 1.0, 4.941642422609304,
        6500.0],
       ['Male', 'Yes', '3+', 'Not Graduate', 360.0, 1.0,
        4.30406509320417, 3173.0],
       ['Male', 'Yes', '0', 'Graduate', 360.0, 1.0, 4.867534450455582,
        7083.0],
       ['Male', 'Yes', '0', 'Not Graduate', 360.0, 1.0,
        4.672828834461906, 4300.0],
       ['Male', 'No', '0', 'Graduate', 360.0, 1.0, 4.857444178729352,
        5505.0],
       ['Male', 'Yes', '0', 'Graduate', 360.0, 1.0, 4.718498871295094,
        3798.0],
       ['Male', 'Yes', '0', 'Graduate', 360.0, 1.0, 5.556828061699537,
        10916.0],
```

```
       ['Male', 'Yes', '0', 'Graduate', 360.0, 1.0, 4.553876891600541,
        4492.0],
       ['Male', 'No', '0', 'Not Graduate', 360.0, 1.0,
4.890349128221754,
        6216.0],
       ['Male', 'Yes', '2', 'Graduate', 360.0, 1.0, 5.123963979403259,
        5532.0],
       ['Male', 'No', '0', 'Graduate', 360.0, 1.0, 4.787491742782046,
        4191.0],
       ['Female', 'No', '0', 'Graduate', 360.0, 0.0,
4.919980925828125,
        11117.0],
       ['Female', 'No', '0', 'Graduate', 360.0, 1.0,
5.365976015021851,
        10000.0],
       ['Male', 'Yes', '0', 'Not Graduate', 360.0, 1.0,
4.74493212836325,
        4567.0],
       ['Female', 'No', '0', 'Graduate', 360.0, 0.0,
4.330733340286331,
        3510.0],
       ['Male', 'Yes', '2', 'Graduate', 360.0, 1.0, 4.890349128221754,
        5935.0],
       ['Male', 'Yes', '1', 'Graduate', 360.0, 1.0, 5.752572638825633,
        11580.0],
       ['Male', 'Yes', '0', 'Graduate', 360.0, 1.0, 5.075173815233827,
        6166.0],
       ['Male', 'No', '0', 'Graduate', 360.0, 1.0, 4.912654885736052,
        4897.0],
       ['Male', 'Yes', '0', 'Graduate', 360.0, 1.0, 5.204006687076795,
        6873.0],
       ['Male', 'No', '0', 'Not Graduate', 360.0, 1.0,
4.564348191467836,
        5484.0],
       ['Male', 'No', '0', 'Graduate', 360.0, 1.0, 4.204692619390966,
        6979.0],
       ['Female', 'Yes', '0', 'Graduate', 360.0, 1.0,
4.867534450455582,
        2928.0],
       ['Male', 'Yes', '2', 'Not Graduate', 360.0, 1.0,
        5.056245805348308, 5398.0],
       ['Male', 'Yes', '1', 'Not Graduate', 180.0, 1.0,
        4.919980925828125, 6608.0],
       ['Female', 'Yes', '0', 'Graduate', 360.0, 1.0,
4.969813299576001,
        5126.0],
       ['Male', 'Yes', '0', 'Not Graduate', 300.0, 1.0,
        4.820281565605037, 5297.0],
       ['Male', 'No', '0', 'Graduate', 360.0, 1.0, 4.499809670330265,
```

```
        35673.0],
       ['Male', 'No', '3+', 'Graduate', 360.0, 1.0, 5.768320995793772,
        15500.0],
       ['Male', 'Yes', '2', 'Graduate', 360.0, 1.0, 4.718498871295094,
        9703.0],
       ['Female', 'No', '0', 'Graduate', 360.0, 0.0,
4.7535901911063645,
        4166.0],
       ['Female', 'No', '0', 'Graduate', 480.0, 1.0,
4.727387818712341,
        4328.0],
       ['Male', 'Yes', '1', 'Graduate', 360.0, 1.0, 6.214608098422191,
        18333.0],
       ['Female', 'No', '0', 'Graduate', 360.0, 1.0,
5.267858159063328,
        7441.0],
       ['Male', 'Yes', '2', 'Graduate', 360.0, 1.0, 5.231108616854587,
        7583.0],
       ['Male', 'No', '0', 'Graduate', 480.0, 1.0, 4.2626798770413155,
        3069.0],
       ['Male', 'Yes', '0', 'Graduate', 12.0, 1.0, 4.709530201312334,
        7482.0],
       ['Male', 'Yes', '0', 'Graduate', 360.0, 1.0, 4.700480365792417,
        4801.0],
       ['Male', 'Yes', '2', 'Graduate', 360.0, 1.0, 5.298317366548036,
        11179.0],
       ['Male', 'No', '1', 'Graduate', 180.0, 1.0, 4.727387818712341,
        3667.0],
       ['Male', 'Yes', '1', 'Graduate', 360.0, 1.0,
4.6443908991413725,
        4545.0],
       ['Female', 'Yes', '0', 'Not Graduate', 360.0, 1.0,
        4.605170185988092, 3572.0],
       ['Male', 'No', '0', 'Graduate', 360.0, 1.0, 4.30406509320417,
        3167.0],
       ['Male', 'Yes', '1', 'Graduate', 84.0, 1.0, 5.147494476813453,
        7283.0],
       ['Male', 'Yes', '3+', 'Graduate', 300.0, 0.0, 5.19295685089021,
        7167.0],
       ['Female', 'No', '0', 'Graduate', 360.0, 1.0,
4.2626798770413155,
        2900.0],
       ['Male', 'No', '0', 'Not Graduate', 180.0, 0.0,
4.836281906951478,
        5454.0],
       ['Male', 'No', '0', 'Graduate', 180.0, 1.0, 5.1647859739235145,
        6950.0],
       ['Male', 'No', '0', 'Graduate', 360.0, 1.0, 4.969813299576001,
        5762.0],
```

```
       ['Male', 'Yes', '2', 'Not Graduate', 360.0, 1.0,
        4.394449154672439, 5093.0],
       ['Male', 'Yes', '1', 'Graduate', 360.0, 1.0, 5.231108616854587,
        9538.0],
       ['Male', 'Yes', '0', 'Graduate', 360.0, 1.0, 5.351858133476067,
        7977.0],
       ['Male', 'Yes', '0', 'Graduate', 360.0, 1.0, 4.605170185988092,
        3539.0],
       ['Male', 'Yes', '2', 'Graduate', 360.0, 1.0, 4.787491742782046,
        10819.0],
       ['Male', 'No', '0', 'Graduate', 180.0, 1.0, 4.787491742782046,
        10383.0],
       ['Male', 'Yes', '3+', 'Graduate', 360.0, 1.0,
4.852030263919617,
        5703.0],
       ['Male', 'No', '0', 'Graduate', 360.0, 1.0, 4.8283137373023015,
        4950.0],
       ['Male', 'No', '0', 'Not Graduate', 360.0, 1.0,
        4.6443908991413725, 4750.0],
       ['Female', 'No', '0', 'Graduate', 360.0, 1.0,
4.477336814478207,
        3410.0],
       ['Male', 'Yes', '0', 'Not Graduate', 360.0, 1.0,
        4.553876891600541, 3849.0],
       ['Male', 'Yes', '3+', 'Not Graduate', 180.0, 1.0,
        4.394449154672439, 3522.0],
       ['Male', 'No', '0', 'Graduate', 360.0, 1.0, 5.298317366548036,
        6400.0],
       ['Female', 'No', '0', 'Graduate', 360.0, 1.0, 4.90527477843843,
        3418.0],
       ['Male', 'No', '0', 'Graduate', 480.0, 1.0, 4.727387818712341,
        3750.0],
       ['Male', 'Yes', '2', 'Graduate', 360.0, 1.0, 4.248495242049359,
        3900.0],
       ['Male', 'Yes', '0', 'Not Graduate', 360.0, 0.0,
        5.303304908059076, 5558.0],
       ['Male', 'Yes', '0', 'Graduate', 180.0, 0.0, 4.499809670330265,
        4949.0],
       ['Female', 'No', '0', 'Graduate', 360.0, 1.0,
4.430816798843313,
        4292.0],
       ['Male', 'No', '0', 'Graduate', 360.0, 1.0, 4.897839799950911,
        4269.0],
       ['Male', 'Yes', '2', 'Graduate', 360.0, 1.0, 5.170483995038151,
        7100.0],
       ['Male', 'Yes', '3+', 'Graduate', 360.0, 1.0,
4.867534450455582,
        13746.0],
       ['Male', 'Yes', '0', 'Graduate', 360.0, 1.0, 6.077642243349034,
        14583.0],
```

```
       ['Male', 'Yes', '3+', 'Not Graduate', 180.0, 0.0,
        4.248495242049359, 4611.0],
       ['Male', 'Yes', '1', 'Graduate', 360.0, 1.0, 4.564348191467836,
        3428.0]], dtype=object)
```

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

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

y_test = labelencoder_y.fit_transform(y_test)

X_test
```

```
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)
```

```
y_test
```

```
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])
```

Scaling (since different columns have different range)

```
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()

# scaling input variables/features
X_train = ss.fit_transform(X_train)
X_test = ss.fit_transform(X_test)
```

Creating Model (by applying Algorithm on the dataset)

1. Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier

DTClassifier = DecisionTreeClassifier(criterion ='entropy',
random_state=0)
DTClassifier.fit(X_train,y_train)

DecisionTreeClassifier(criterion='entropy', random_state=0)
```

Now Let's use this algorithm to predict the values of test dataset

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

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])

# Calculating accuracy of the prediction

from sklearn import metrics

print('The accuracy of the decision tree is: ',
metrics.accuracy_score(y_pred,y_test))

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

Not so great accuracy. Let's try to use another algorithm

2. Naive_Bayes Algorithm

```
from sklearn.naive_bayes import GaussianNB

NBClassifier = GaussianNB()
NBClassifier.fit(X_train, y_train)

GaussianNB()

y_pred = NBClassifier.predict(X_test)

y_pred

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])

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

The accuracy of Naive Bayes is:  0.8292682926829268
```

Good Accuracy compared to Decision Tree.

Now, import test dataset, which don't have 'loan status' column.

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

    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
```

Going to use Naive_Bayes Algo to predict. But first, let's explore the loan-test.csv

```
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
```

- Okay, this also has some missing values.

```
# Handling those misssing values

testdata.isnull().sum()

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
```

```
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)
```

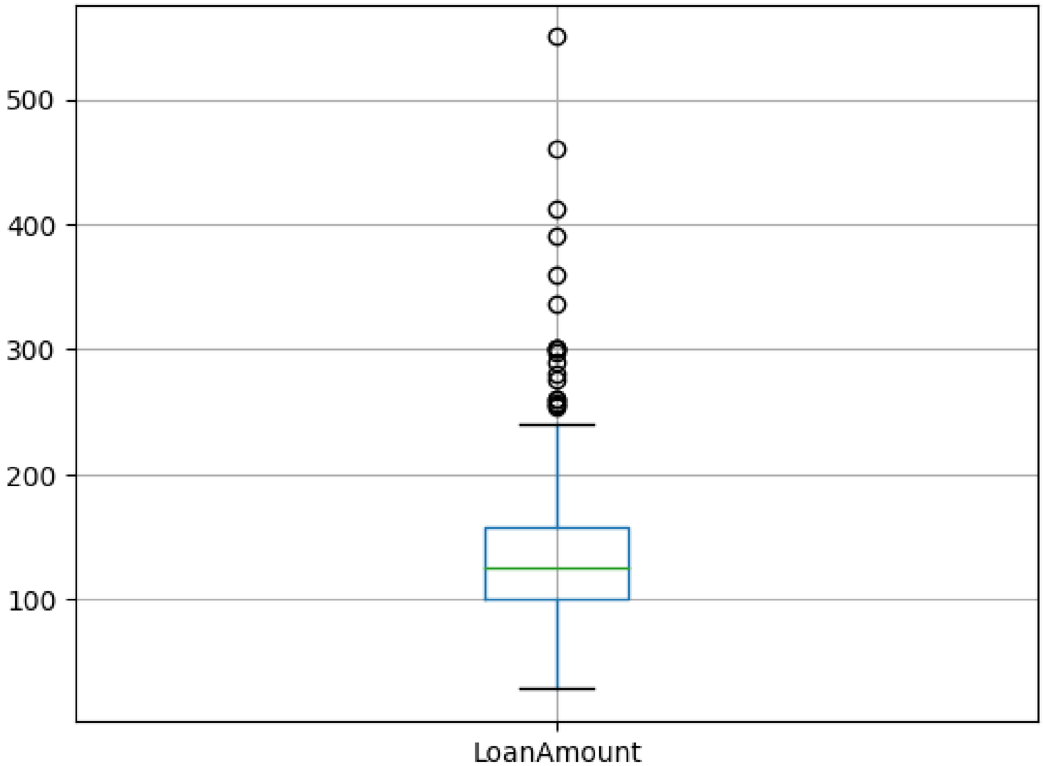
```
testdata.isnull().sum()
```

```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           5
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
dtype: int64
```

- Now, only in Loan Amount

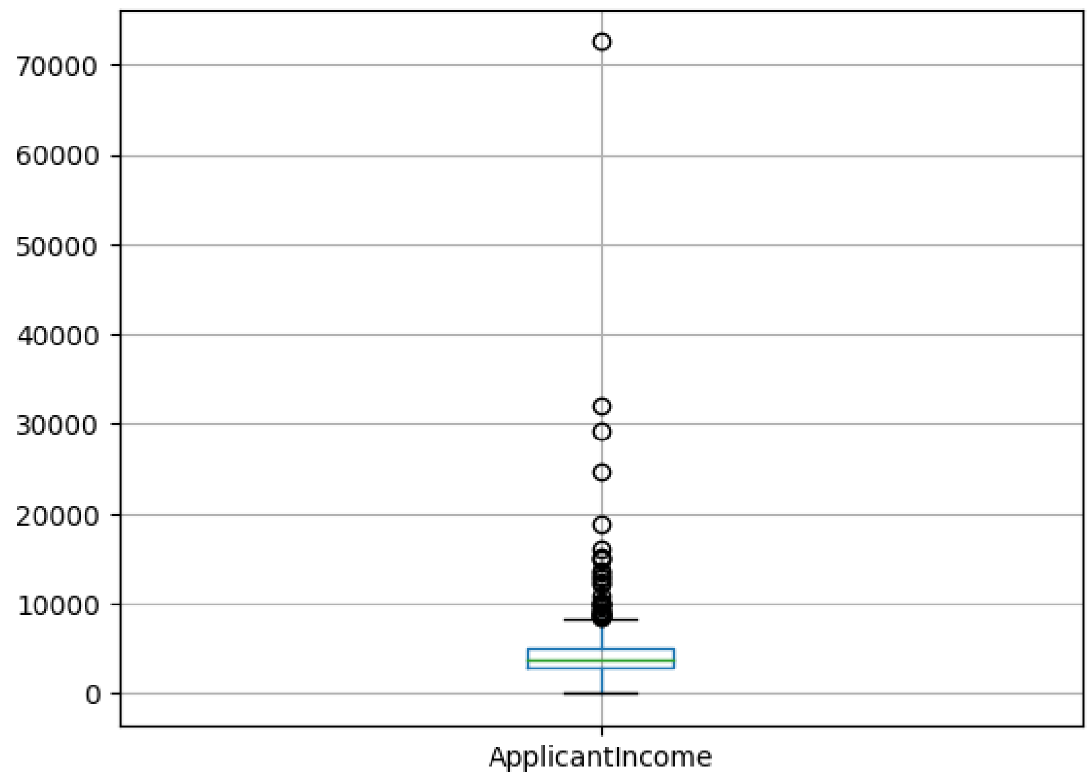
```
#Let's Visulaize Loan Amount Box-plot

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

```
<Axes: >
```




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

```
<Axes: >
```




```
# filling null values in LoanAmount column with mean value

testdata.LoanAmount =
testdata.LoanAmount.fillna(testdata.LoanAmount.mean())

testdata.isnull().sum()

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
dtype: int64
```

- All missing values are handled in test dataset.

```
# Normalise the LoanAmount

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

```python
# Let's calculate TotalIncome similar to what we had done, while
trainning our model

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

testdata.head()
```

```
    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
```

```python
# Spiltting dataset into X and y

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

# Encoding

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

# 7 index as well
test[:,7] = labelencoder_X.fit_transform(test[:,7])

test
```

```
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)
```

- All textual value is converted into numeric now.

```
# Scaling test data

test = ss.fit_transform(test)  # instances have already been created
while training the model
```

Using Naive Bayes Algo to predict for test dataset

```
pred = NBClassifier.predict(test)

pred   # prediction

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])
```

- 1 - eligible
- 0 - not eligible

```
```