<

In [44]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [7]:
```
os.getcwd()
```

Out[7]:
```
'C:\\Users\\Oscar\\Desktop\\Coding'
```

In [23]:
```
os.chdir("C:\\Users\\Oscar\\Desktop\\Future Interns Project\\loan.csv")
```

```
---------------------------------------------------------------------------
NotADirectoryError                        Traceback (most recent call last)
Cell In[23], line 1
----> 1 os.chdir("C:\\Users\\Oscar\\Desktop\\Future Interns Project\\loan.csv")

NotADirectoryError: [WinError 267] The directory name is invalid: 'C:\\Users\\Oscar
\\Desktop\\Future Interns Project\\loan.csv'
```

In [26]:
```
data=pd.read_csv("loan.csv")
```

In [27]:
```
data.head()
```

Out[27]:

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

In [28]:
```
data.shape
```

Out[28]:
```
(614, 13)
```

In [29]:
```
data.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 [30]:
```
data.describe()
```

Out[30]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_Histo
count	614.000000	614.000000	592.000000	600.00000	564.0000
mean	5403.459283	1621.245798	146.412162	342.00000	0.8421
std	6109.041673	2926.248369	85.587325	65.12041	0.3648
min	150.000000	0.000000	9.000000	12.00000	0.0000
25%	2877.500000	0.000000	100.000000	360.00000	1.0000
50%	3812.500000	1188.500000	128.000000	360.00000	1.0000
75%	5795.000000	2297.250000	168.000000	360.00000	1.0000
max	81000.000000	41667.000000	700.000000	480.00000	1.0000

Do a crosstab to determine how the credit history affects loan status of applicants

In [31]:
```
pd.crosstab(data['Credit_History'],data['Loan_Status'],margins=True)
```

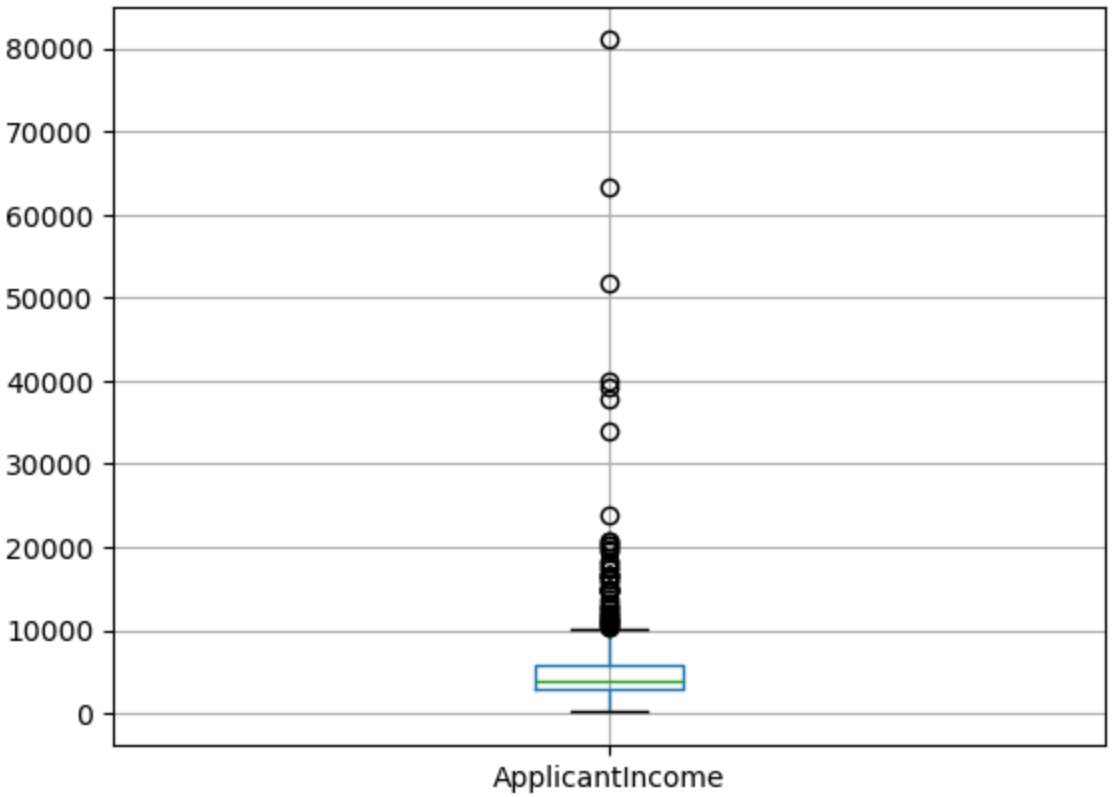
Out[31]:

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

mean,mode,median of ApplicantIncome using boxplot

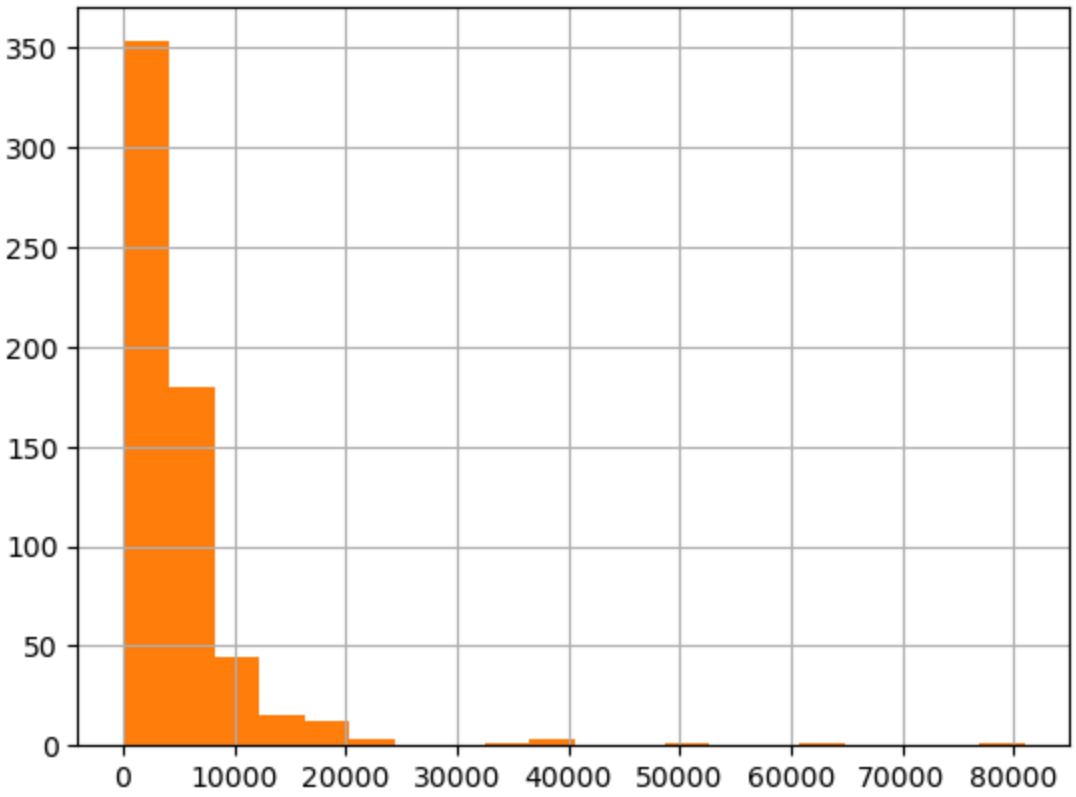
In [32]:
```
data.boxplot(column='ApplicantIncome')
```

Out[32]: <Axes: >



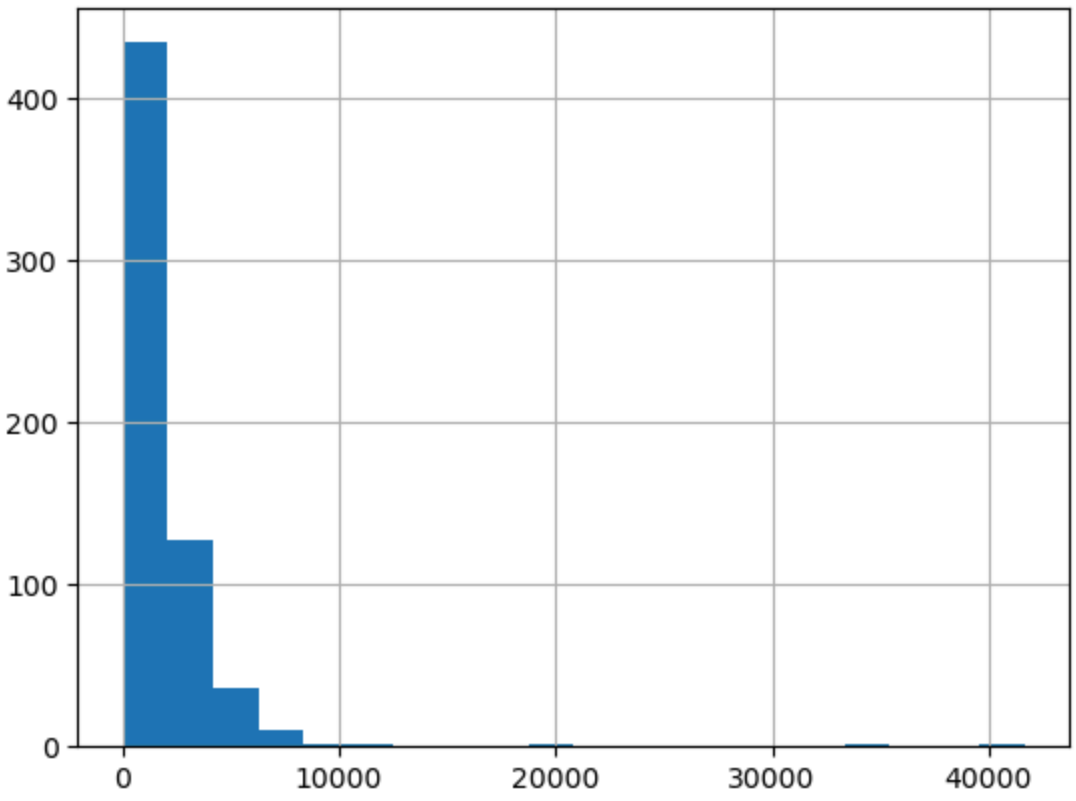
perform a histogram

In [57]:
```
data['ApplicantIncome'].hist(bins=20)
plt.show()
```



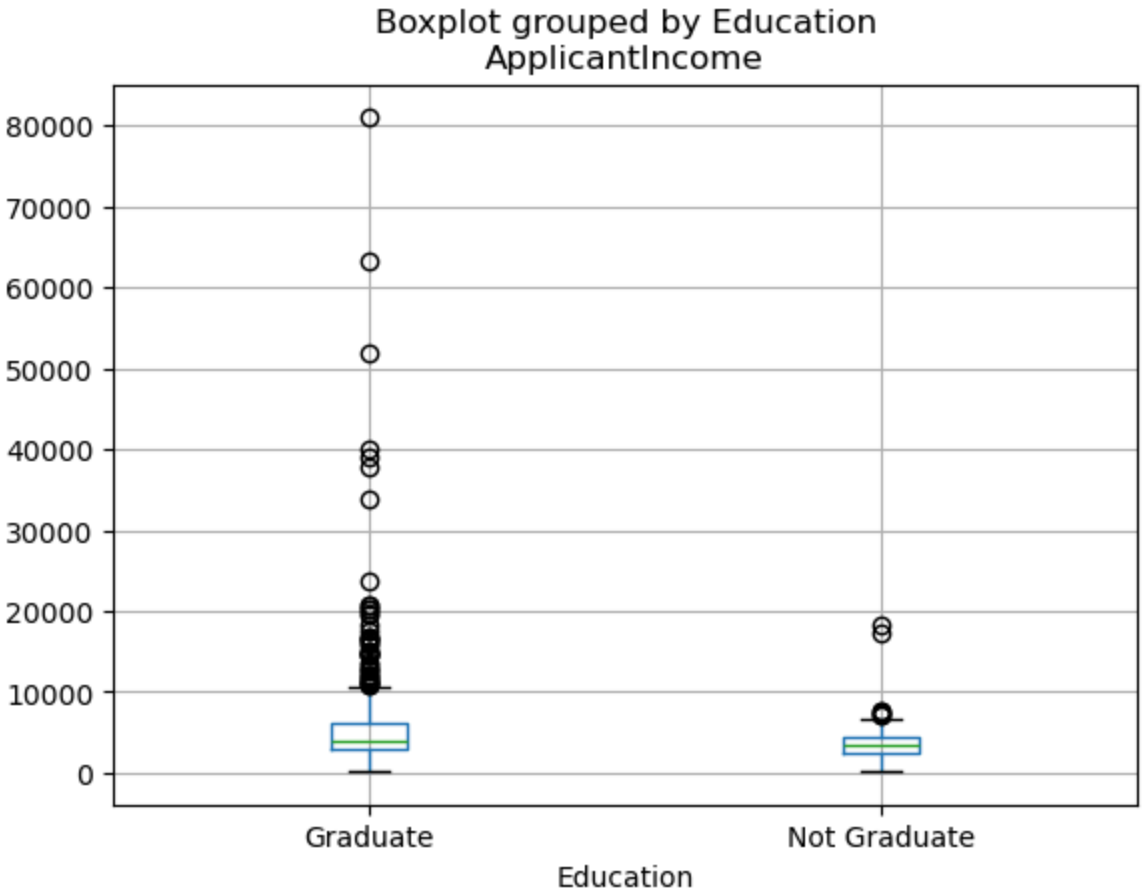
perform a histogram

In [58]:
```
data['CoapplicantIncome'].hist(bins=20)
plt.show()
```



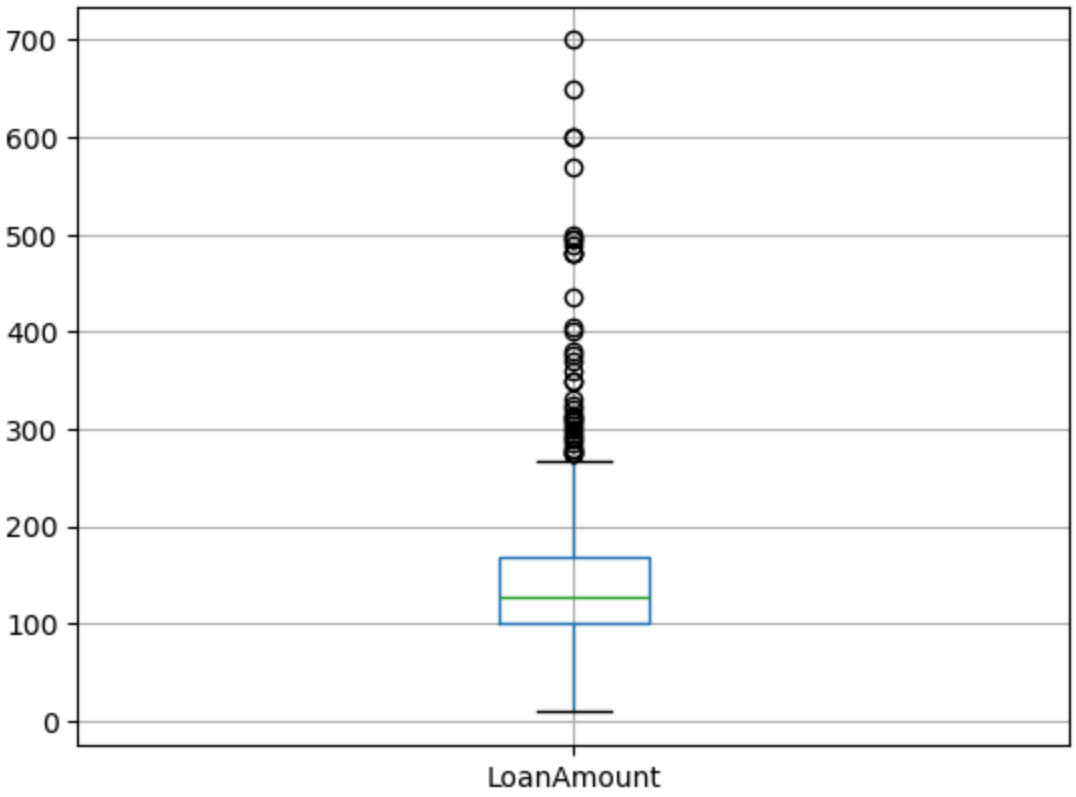
boxplot of application when booked with education

In [59]:
```
data.boxplot(column='ApplicantIncome',by = 'Education')
plt.show()
```



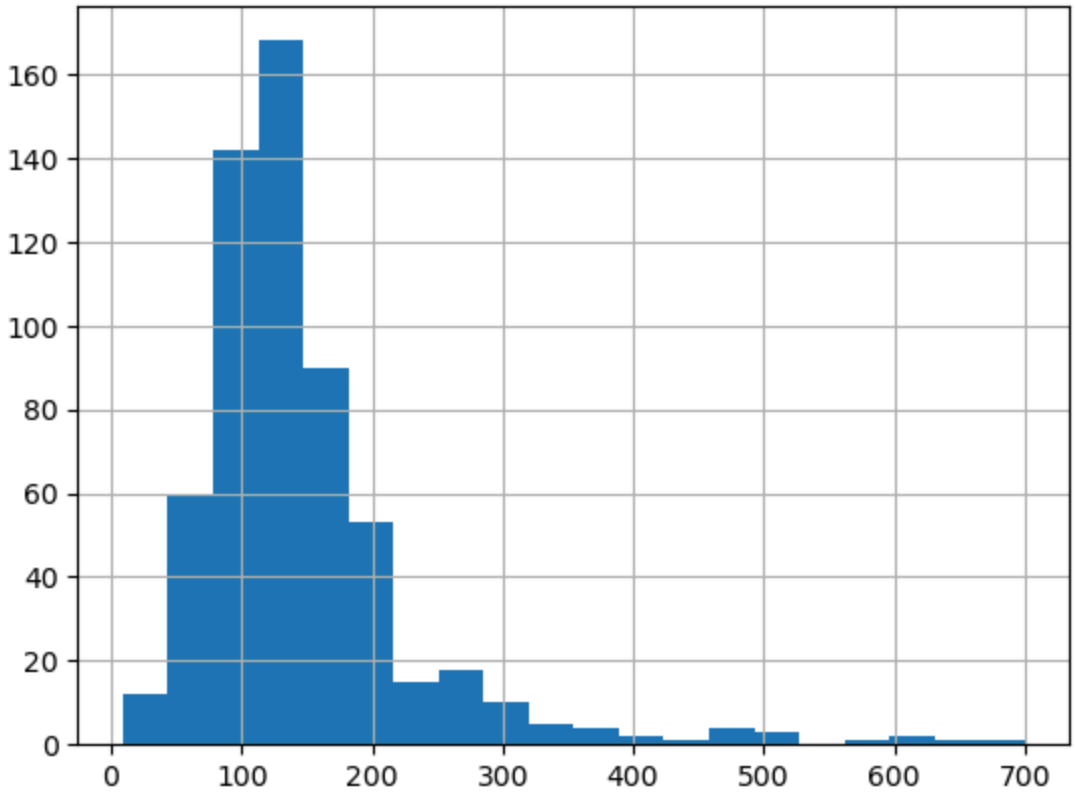
boxplot of loanamount

In [60]:
```
data.boxplot(column='LoanAmount')
plt.show()
```



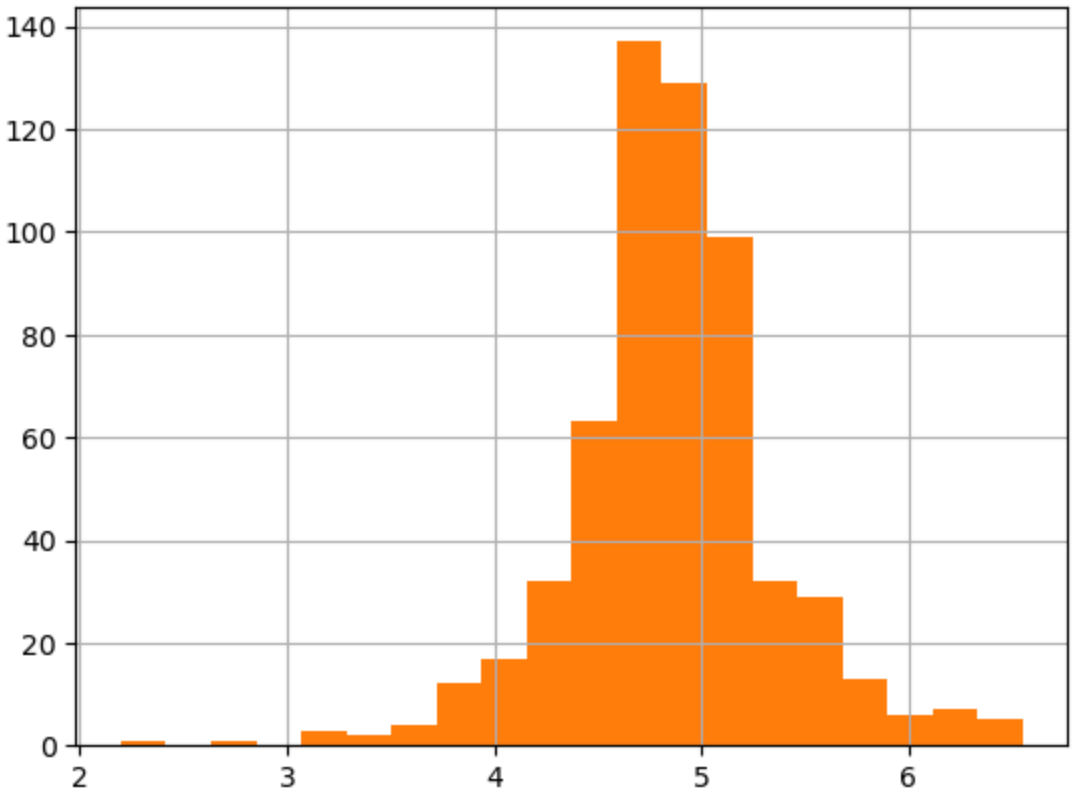
histogram for loanamount

In [61]:
```
data['LoanAmount'].hist(bins=20)
plt.show()
```



normalising loanamount using logfunction

In [64]:
```
data['LoanAmount_log']=np.log(data['LoanAmount'])
data['LoanAmount_log'].hist(bins=20)
plt.show()
```



checking missing values in the dataset

In [65]:
```
data.isnull().sum()
```

Out[65]:
```
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
```

filling missing values

In [162…

```
data['Gender'].fillna(data['Gender'].mode()[0],inplace=True)
data['Married'].fillna(data['Married'].mode()[0],inplace=True)
data['Self_Employed'].fillna(data['Self_Employed'].mode()[0],inplace=True)
data.LoanAmount = data.LoanAmount.fillna(data.LoanAmount.mean())
data.LoanAmount_log= data.LoanAmount_log.fillna(data.LoanAmount_log.mean())
data['Loan_Amount_Term'].fillna(data['Loan_Amount_Term'].mode()[0],inplace=True)
data['Credit_History'].fillna(data['Credit_History'].mode()[0],inplace=True)
```

```
C:\Users\Oscar\AppData\Local\Temp\ipykernel_8072\801618550.py:1: FutureWarning: A va
lue is trying to be set on a copy of a DataFrame or Series through chained assignmen
t using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  data['Gender'].fillna(data['Gender'].mode()[0],inplace=True)
C:\Users\Oscar\AppData\Local\Temp\ipykernel_8072\801618550.py:2: FutureWarning: A va
lue is trying to be set on a copy of a DataFrame or Series through chained assignmen
t using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  data['Married'].fillna(data['Married'].mode()[0],inplace=True)
C:\Users\Oscar\AppData\Local\Temp\ipykernel_8072\801618550.py:3: FutureWarning: A va
lue is trying to be set on a copy of a DataFrame or Series through chained assignmen
t using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  data['Self_Employed'].fillna(data['Self_Employed'].mode()[0],inplace=True)
C:\Users\Oscar\AppData\Local\Temp\ipykernel_8072\801618550.py:6: FutureWarning: A va
lue is trying to be set on a copy of a DataFrame or Series through chained assignmen
t using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  data['Loan_Amount_Term'].fillna(data['Loan_Amount_Term'].mode()[0],inplace=True)
C:\Users\Oscar\AppData\Local\Temp\ipykernel_8072\801618550.py:7: FutureWarning: A va
lue is trying to be set on a copy of a DataFrame or Series through chained assignmen
t using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
```

```
the operation inplace on the original object.


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

ensuring that there are no missing values

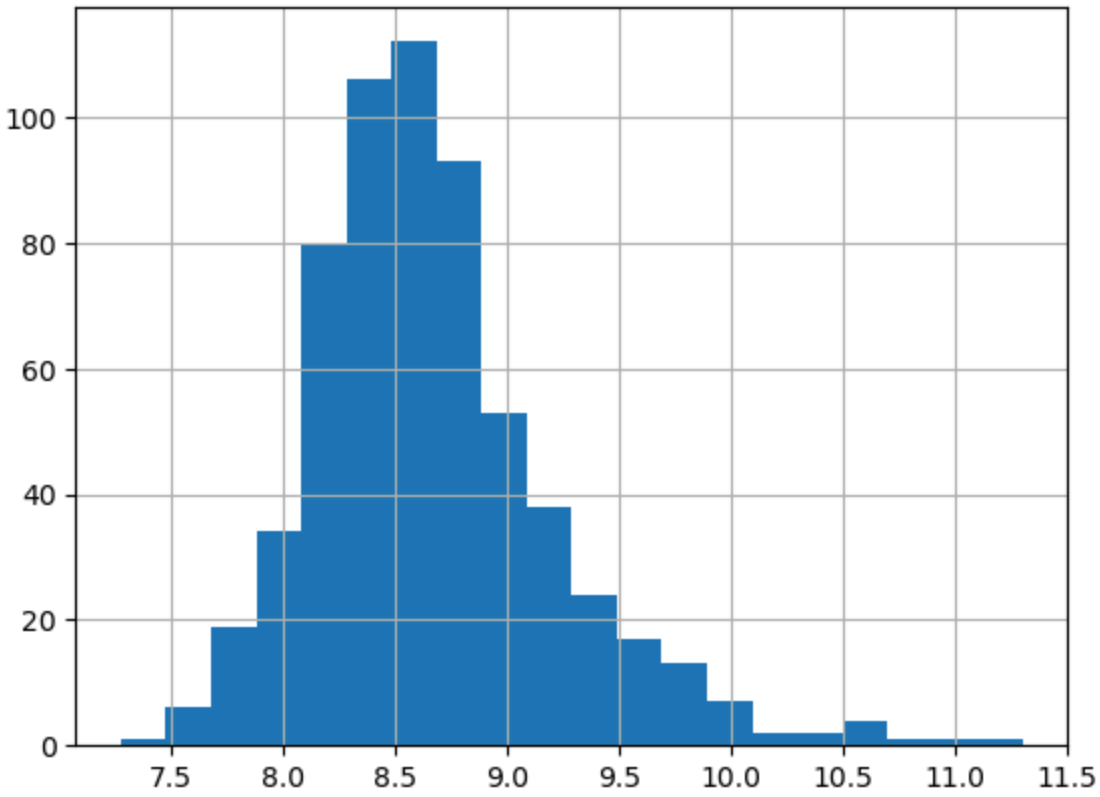
In [163…
```
data.isnull().sum()
```

Out[163…
```
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
```

finding sum of applicantincome and coapplicantincome

In [101…
```
data['TotalIncome']= data['ApplicantIncome'] + data['CoapplicantIncome']
data['TotalIncome_log']= np.log(data['TotalIncome'])
```

In [102…
```
data['TotalIncome_log'].hist(bins=20)
plt.show()
```



In [103…
```
data.head()
```

Out[103…

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	C
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	

identifying independent and dependent variables

In [108…
```
X= data.iloc[:,np.r_[1:5,9:11,13:15]].values
y= data.iloc[:,12].values
```

In [109…
```
X
```

Out[109…

```
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 [111…

```
y
```

Out[111…

```
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', 'Y', 'N', 'Y', '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 [116…

```
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_stat
```

print(X_train)

In [118…

```
from sklearn.preprocessing import LabelEncoder
labelencoder_X =LabelEncoder()
```

In [123…
```
for i in range (0, 5):
    X_train[:,i]= labelencoder_X.fit_transform(X_train[:,i])
```

In [125…
```
X_train[: ,7]= labelencoder_X.fit_transform(X_train[:,7])
```

In [126…
```
X_train
```

Out[126…
```
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 [127…
```
labelencoder_y=LabelEncoder()
y_train= labelencoder_y.fit_transform(y_train)
```

In [128…
```
y_train
```

Out[128…
```
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, 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, 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, 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, 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 [129…
```
for i in range (0, 5):
    X_test[:,i]= labelencoder_X.fit_transform(X_test[:,i])
```

In [130…
```
X_test[: ,7]= labelencoder_X.fit_transform(X_test[:,7])
```

In [131…
```
labelencoder_y=LabelEncoder()
y_test= labelencoder_y.fit_transform(y_test)
```

In [132…
```
X_test
```

Out[132…

```
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)
```

In [133…
```
y_test
```

Out[133…
```
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 [147…
```
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
X_train=ss.fit_transform(X_train)
X_test=ss.fit_transform(X_test)
```

In [148…
```
from sklearn.tree import DecisionTreeClassifier
DTClassifier = DecisionTreeClassifier(criterion='entropy',random_state=0)
DTClassifier.fit(X_train,y_train)
```

Out[148…

DecisionTreeClassifier

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

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

In [149…
```
y_pred = DTClassifier.predict(X_test)
y_pred
```

Out[149…
```
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 [150…
```
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
```

In [151…
```
from sklearn.naive_bayes import GaussianNB
NBClassifier = GaussianNB()
NBClassifier.fit(X_train,y_train)
```

Out[151…

GaussianNB

GaussianNB()

In [152…
```
y_pred=NBClassifier.predict(X_test)
```

In [153…
```
y_pred
```

Out[153…
```
array([1, 1, 1, 1, 1, 0, 1, 1, 0, 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 [154…
```
print('The accuracy of Naive Bayes is:',metrics.accuracy_score(y_pred,y_test))
```

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

In [29]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [30]:
```
testdata=pd.read_csv("loan.csv")
```

In [32]:
```
testdata.drop(['Loan_Status'],axis=1 ,inplace=True)
```

In [33]:
```
testdata.head()
```

Out[33]:

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

In [34]:
```
testdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 12 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
dtypes: float64(4), int64(1), object(7)
memory usage: 57.7+ KB
```

In [35]:
```
testdata.isnull().sum()
```

Out[35]:
```
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
```

handling missing data

In [46]:
```
testdata['Gender'].fillna(testdata['Gender'].mode()[0],inplace=True)
testdata['Married'].fillna(testdata['Married'].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=
testdata['Credit_History'].fillna(testdata['Credit_History'].mode()[0],inplace=True
```

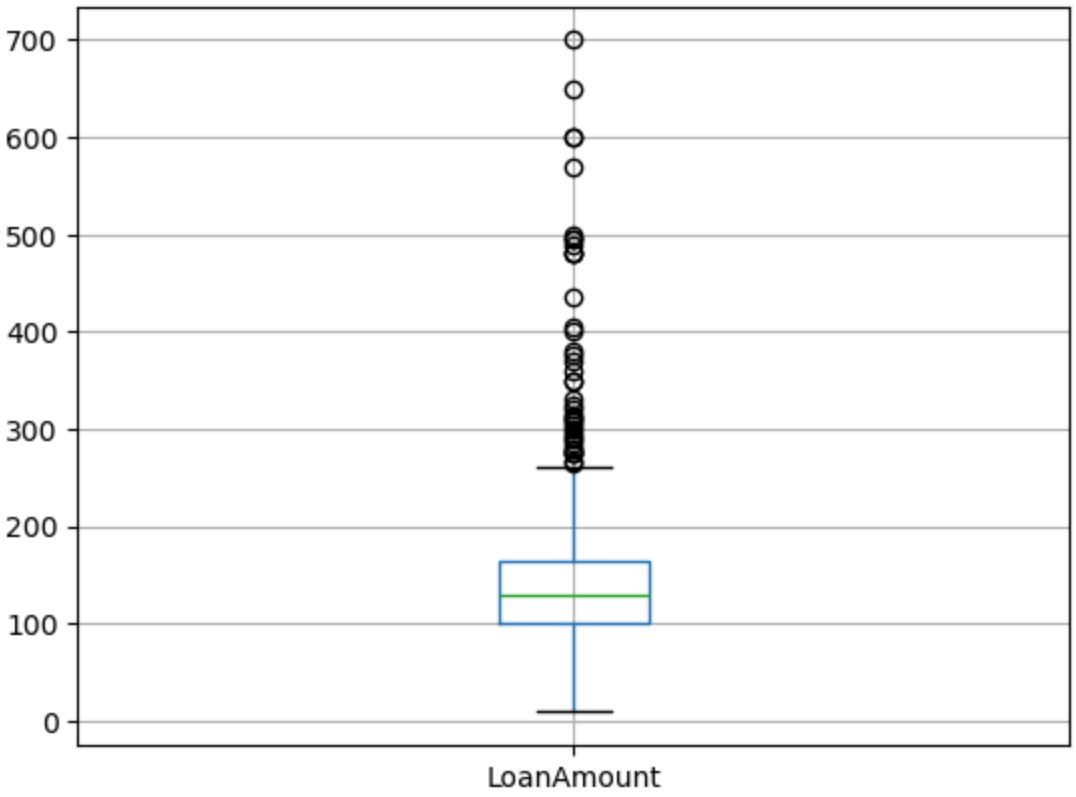
```
C:\Users\Oscar\AppData\Local\Temp\ipykernel_8368\733620056.py:1: FutureWarning: A va
lue is trying to be set on a copy of a DataFrame or Series through chained assignmen
t using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  testdata['Gender'].fillna(testdata['Gender'].mode()[0],inplace=True)
C:\Users\Oscar\AppData\Local\Temp\ipykernel_8368\733620056.py:2: FutureWarning: A va
lue is trying to be set on a copy of a DataFrame or Series through chained assignmen
t using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  testdata['Married'].fillna(testdata['Married'].mode()[0],inplace=True)
C:\Users\Oscar\AppData\Local\Temp\ipykernel_8368\733620056.py:3: FutureWarning: A va
lue is trying to be set on a copy of a DataFrame or Series through chained assignmen
t using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  testdata['Dependents'].fillna(testdata['Dependents'].mode()[0],inplace=True)
C:\Users\Oscar\AppData\Local\Temp\ipykernel_8368\733620056.py:4: FutureWarning: A va
lue is trying to be set on a copy of a DataFrame or Series through chained assignmen
t using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  testdata['Self_Employed'].fillna(testdata['Self_Employed'].mode()[0],inplace=True)
C:\Users\Oscar\AppData\Local\Temp\ipykernel_8368\733620056.py:5: FutureWarning: A va
lue is trying to be set on a copy of a DataFrame or Series through chained assignmen
t using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
```

```
the operation inplace on the original object.


  testdata['Loan_Amount_Term'].fillna(testdata['Loan_Amount_Term'].mode()[0],inplace
=True)
C:\Users\Oscar\AppData\Local\Temp\ipykernel_8368\733620056.py:6: FutureWarning: A va
lue is trying to be set on a copy of a DataFrame or Series through chained assignmen
t using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  testdata['Credit_History'].fillna(testdata['Credit_History'].mode()[0],inplace=Tru
e)
```

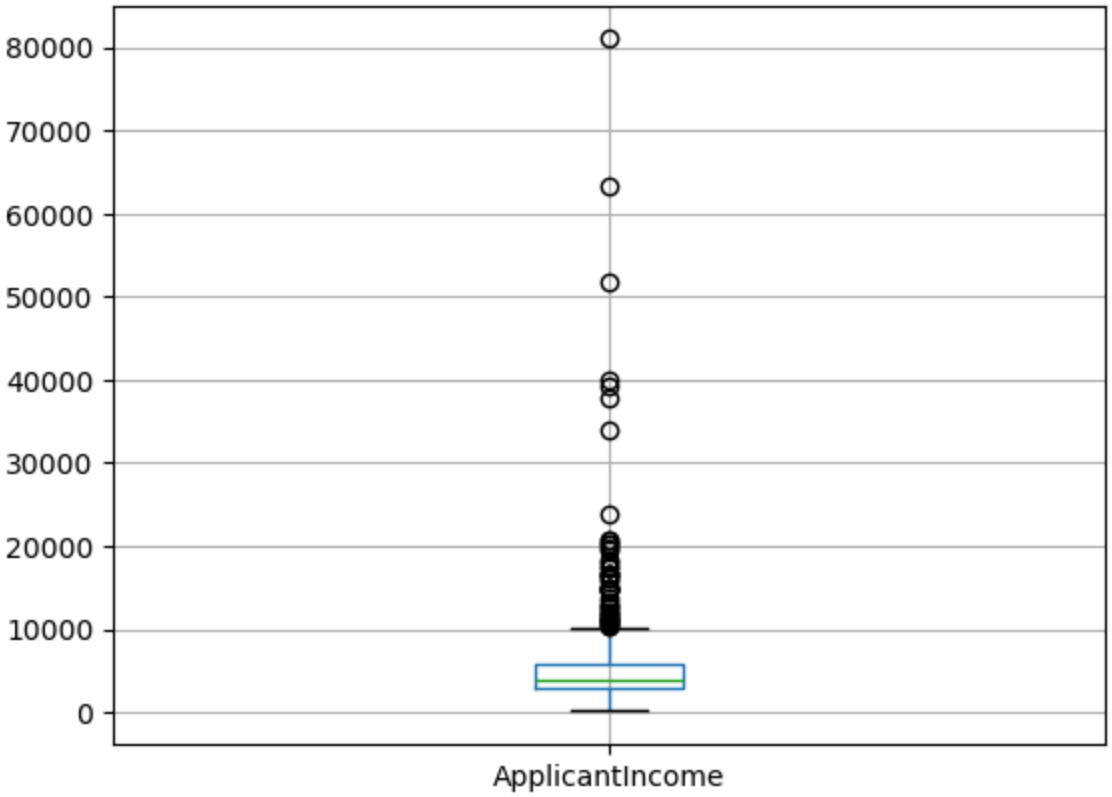
In [47]:
```
testdata.isnull().sum()
```

Out[47]:
```
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
```

In [48]:
```
testdata.boxplot(column='LoanAmount')
plt.show()
```



In [49]:
```
testdata.boxplot(column='ApplicantIncome')
plt.show()
```



In [50]:
```
testdata.LoanAmount=testdata.LoanAmount.fillna(testdata.LoanAmount.mean())
testdata.LoanAmount=testdata.LoanAmount.fillna(testdata.LoanAmount.mean())
```

In [51]:
```
testdata['LoanAmount_log']=np.log(testdata['LoanAmount'])
```

In [52]:
```
testdata.isnull().sum()
```

Out[52]:
```
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
```

In [53]:
```
testdata['TotalIncome']= testdata['ApplicantIncome'] + testdata['CoapplicantIncome'
testdata['TotalIncome_log']= np.log(testdata['TotalIncome'])
```

In [54]:
```
testdata.head()
```

Out[54]:

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

In [55]:
```
test= testdata.iloc[:,np.r_[1:5,9:11,13:15]].values
```

In [81]:
```
from sklearn.preprocessing import LabelEncoder
labelencoder_X =LabelEncoder()
```

In [85]:
```
for i in range(0,5):
     test[:,i]=labelencoder_X.fit_transform(test[:,i])
```

In [86]:
```
test[:,7]=labelencoder_X.fit_transform(test[:,7])
```

In [87]:
```
test
```

Out[87]:
```
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)
```

In [88]:
```
from sklearn .preprocessing import StandardScaler
from sklearn.cluster import KMeans
```

In [89]:
```
KMeans(n_clusters=2, random_state=0)
```

Out[89]:
```
KMeans
KMeans(n_clusters=2, random_state=0)
```

In [90]:
```
kmeans = KMeans(n_clusters=2,random_state=0)
kmeans.fit(test)
```

Out[90]:
```
KMeans
KMeans(n_clusters=2, random_state=0)
```

In [92]:
```
predict = kmeans.predict(test)
```

In [94]:
```
predict
```

```
Out[94]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

1 represents that a customer is eligible for the loan while 0 shows that a customer is not eligible for the loan