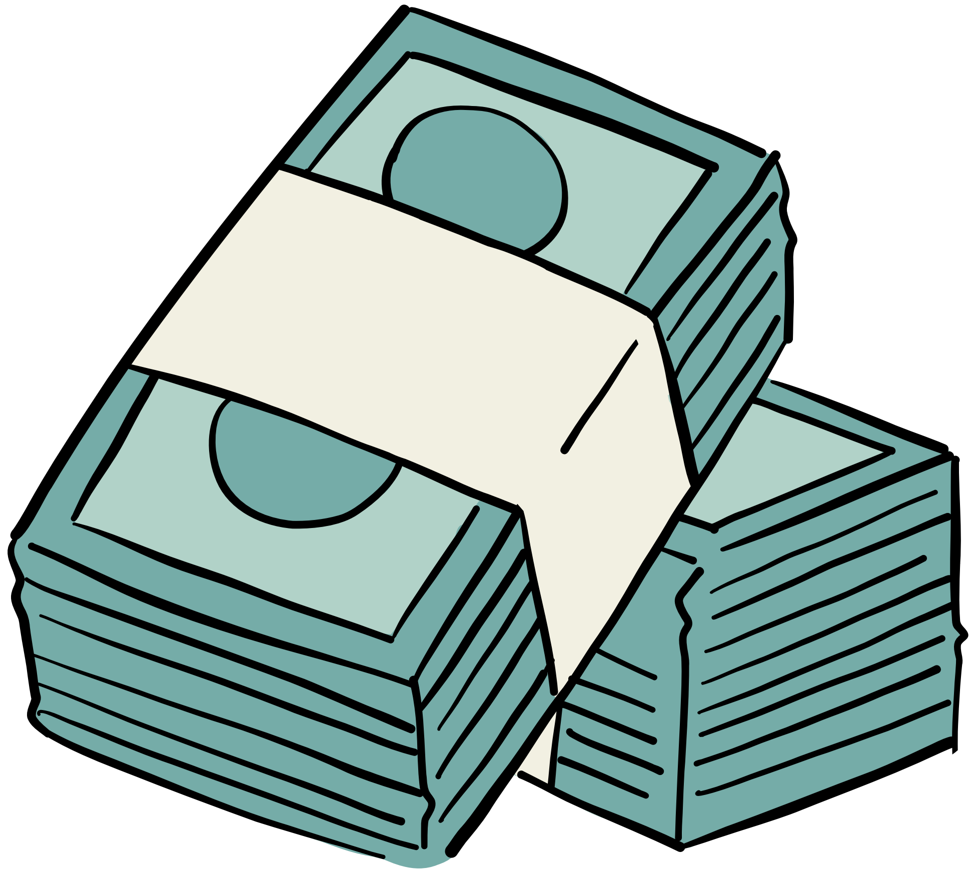
Loan Approval Predicton Using Python



Created by Kamal

HELLO EVERYONE

I AM KAMAL AND IN THIS PROJECT I UTILIZE PYTHON CODE TO PREDICT THE LONE APPROVEL

step 01

Installing all required libraries

Loan_Approval_predicton

In [114]:

```
## Installing all required libraries'

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

step 02 Importing CSV File to Jupter Notebook

```
In [8]: df= pd.read_csv('E:\Downloads\Dataset.csv')
```

step 03 Viewing the Dataset

```
[9]: df
```

[9]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_Hi
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	

step 04

Viewing columns Info

In [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
```

step 05

Viewing Columns name

```
In [10]: df.columns

Out[10]: Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
               'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
               'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
              dtype='object')
```

step 06

Finding Null values in Columns

```
In [17]: df.isnull().sum()

Out[17]: Loan_ID               0
         Gender               13
         Married               3
         Dependents           15
         Education             0
         Self_Employed        32
         ApplicantIncome       0
         CoapplicantIncome     0
         LoanAmount           22
         Loan_Amount_Term     14
```

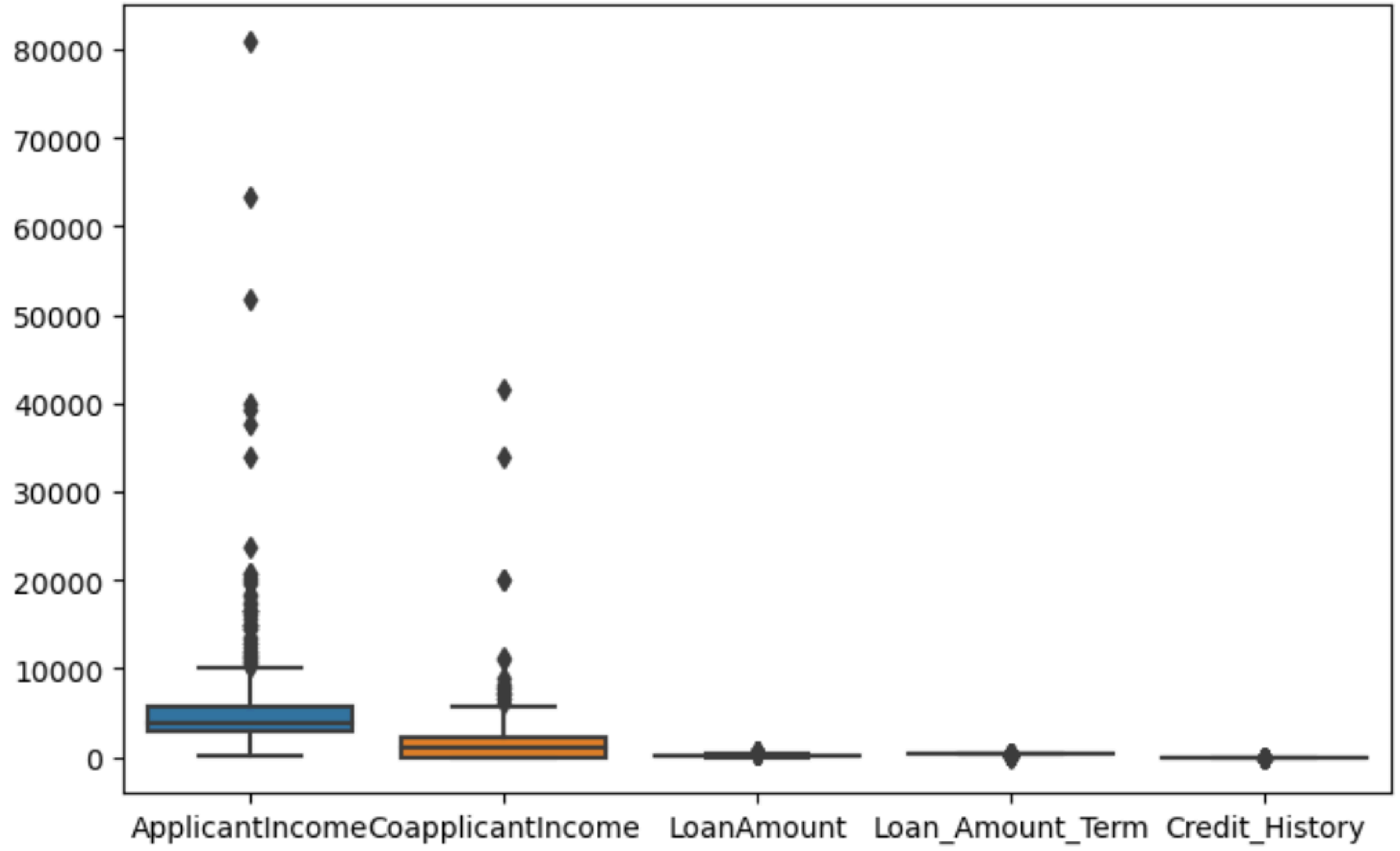
step 07

Checking for Outliners

In [7]:
```
## Checking for Outliners

plt.figure(figsize=(8,5))
sns.boxplot(data=df)
```

Out[7]: <Axes: >



step 07

Null Values of Qualitative Columns Fill with Median

In [23]:
```
df['LoanAmount']=df['LoanAmount'].fillna(df['LoanAmount'].median())
df['Loan_Amount_Term']=df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mean())
df['Credit_History']=df['Credit_History'].fillna(df['Credit_History'].median())
```

step 08

Null Values of Categorical Columns Fill with Mode

In [28]:
```
df['Gender']=df['Gender'].fillna(df['Gender'].mode()[0])
df['Married']=df['Married'].fillna(df['Married'].mode()[0])
df['Dependents']=df['Dependents'].fillna(df['Dependents'].mode()[0])
```

In [30]:
```
df['Self_Employed']=df['Self_Employed'].fillna(df['Self_Employed'].mode()[0])
```

step 09

Checking Null Values Again

```
In [31]: df.isnull().sum()

Out[31]: 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
         dtype: int64
```

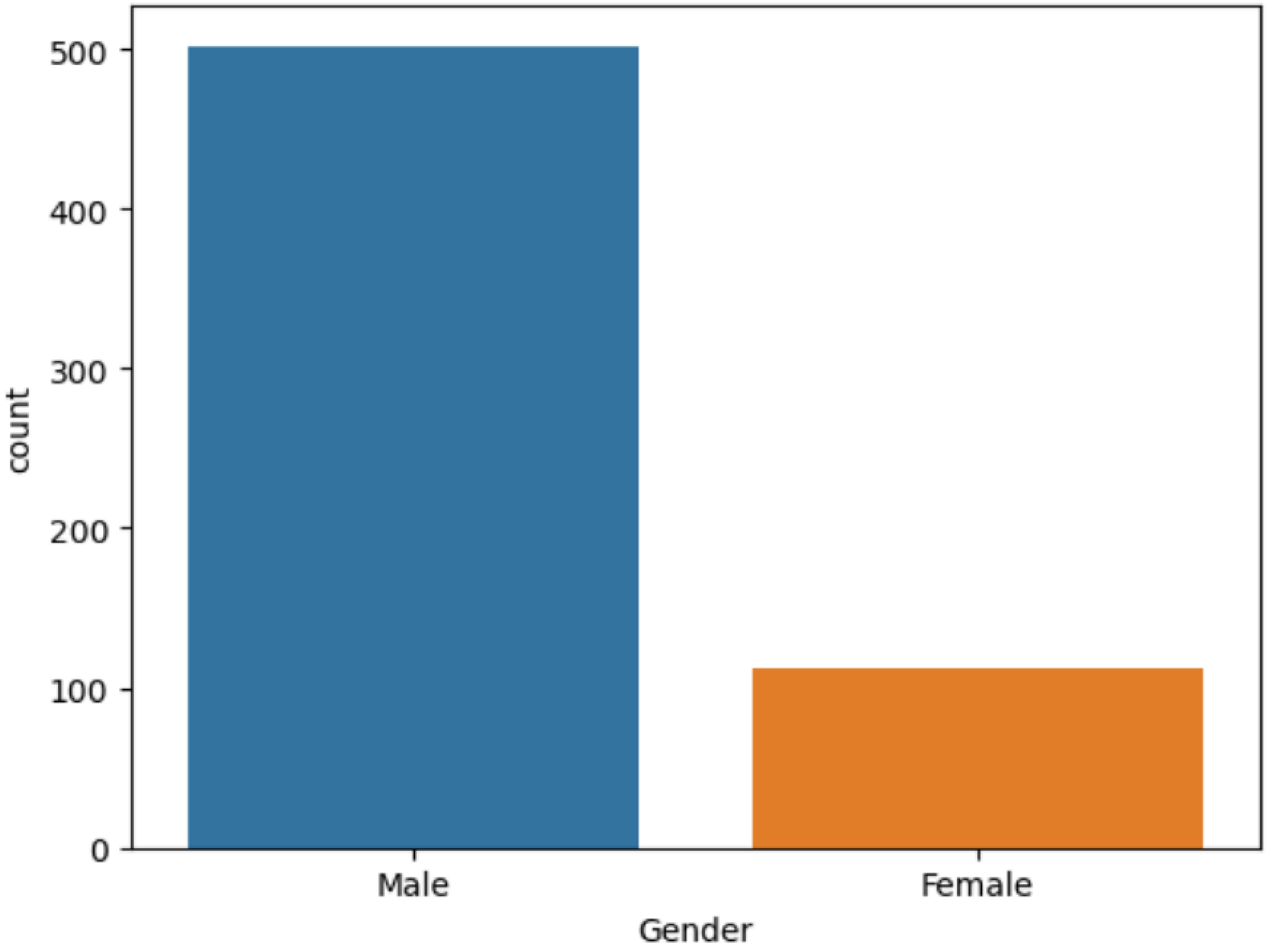
step 10

Finding People who taken the loan by Gender

```
In [38]: print('People who taken the loan by Gender')
         print(df['Gender'].value_counts())
         sns.countplot(x='Gender',data=df)
```

```
People who taken the loan by Gender
Gender
Male      502
Female    112
Name: count, dtype: int64
```

```
Out[38]: <Axes: xlabel='Gender', ylabel='count'>
```



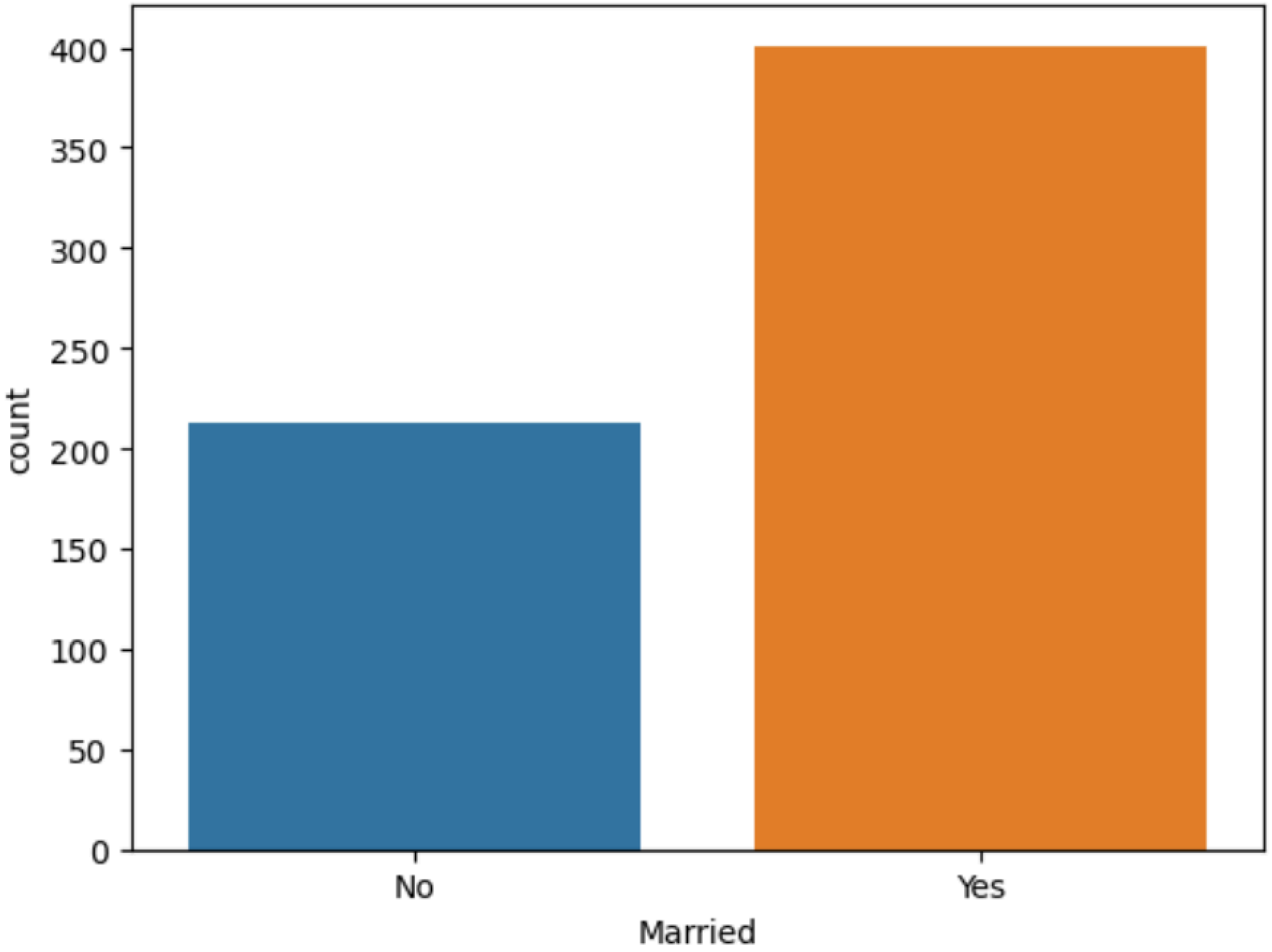
step 11

Finding People who taken the loan by Married Statues

In [39]:
```
print('People who taken the loan by Married')
print(df['Married'].value_counts())
sns.countplot(x='Married',data=df)
```

```
People who taken the loan by Married
Married
Yes     401
No      213
Name: count, dtype: int64
```

Out[39]: <Axes: xlabel='Married', ylabel='count'>

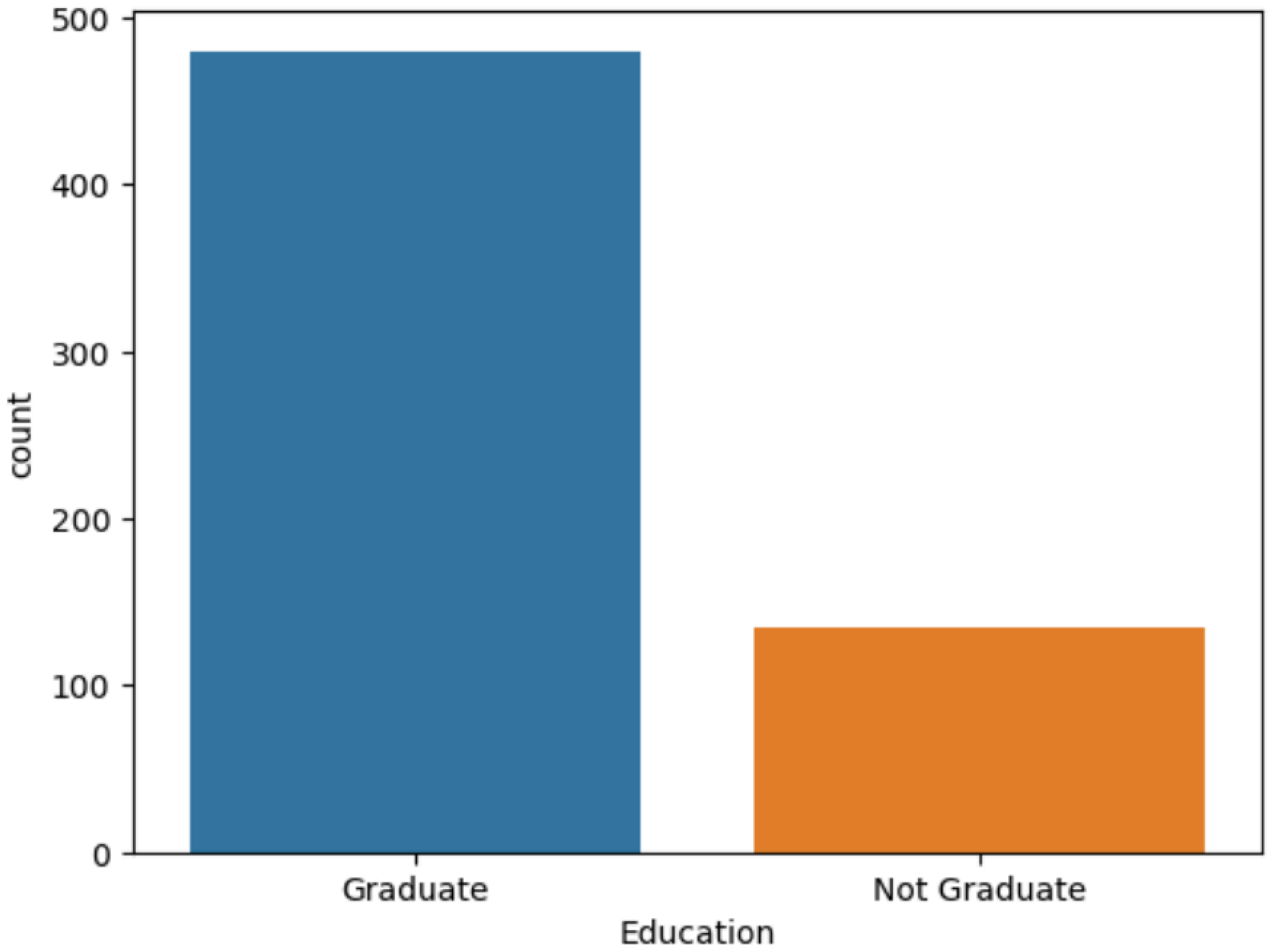


step 12

Finding People who taken the loan For Education

In [40]:
```
print('People who taken the loan by Education')
print(df['Education'].value_counts())
sns.countplot(x='Education',data=df)
```

Out[40]: <Axes: xlabel='Education', ylabel='count'>




```
People who taken the loan by Education
Education
Graduate        480
Not Graduate    134
Name: count, dtype: int64
```

step 13

Calculating Total Income

In [56]:
```
df['Total_income']=df['ApplicantIncome']+df['CoapplicantIncome']
df.head()
```

Out[56]:

oan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_
)01002	Male	No	0	Graduate	No	5849	0.0	128.0	360.0	1.0	
)01003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	
)01005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	
)01006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	
)01008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	

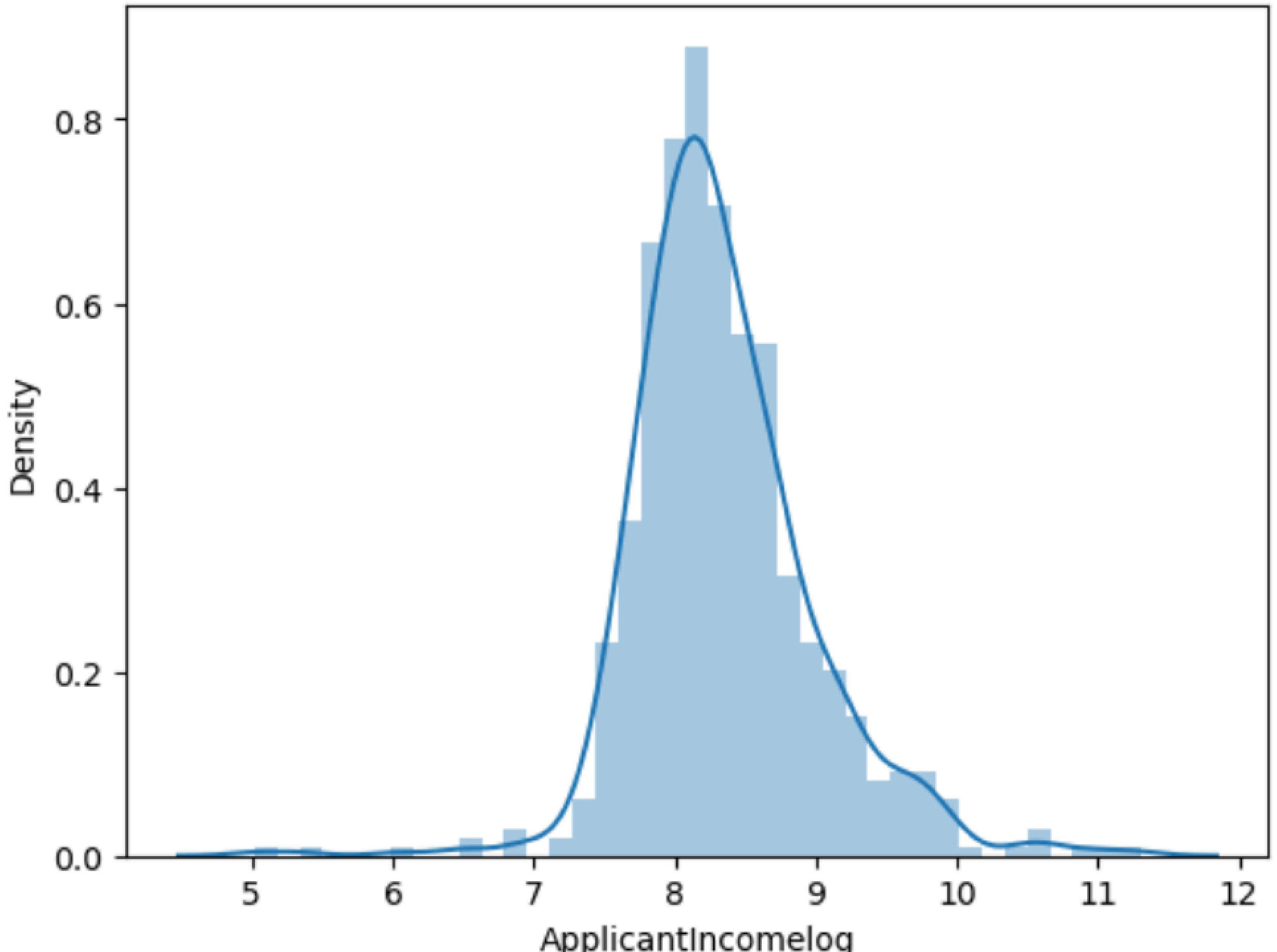
step 14

Applying Log Transformation on Applicant-Income

In [51]:
```
## Applying Log Transformation

df['ApplicantIncomelog'] = np.log(df['ApplicantIncome']+1)
sns.distplot(df['ApplicantIncomelog'])
```

Out[51]: <Axes: xlabel='ApplicantIncomelog', ylabel='Density'>

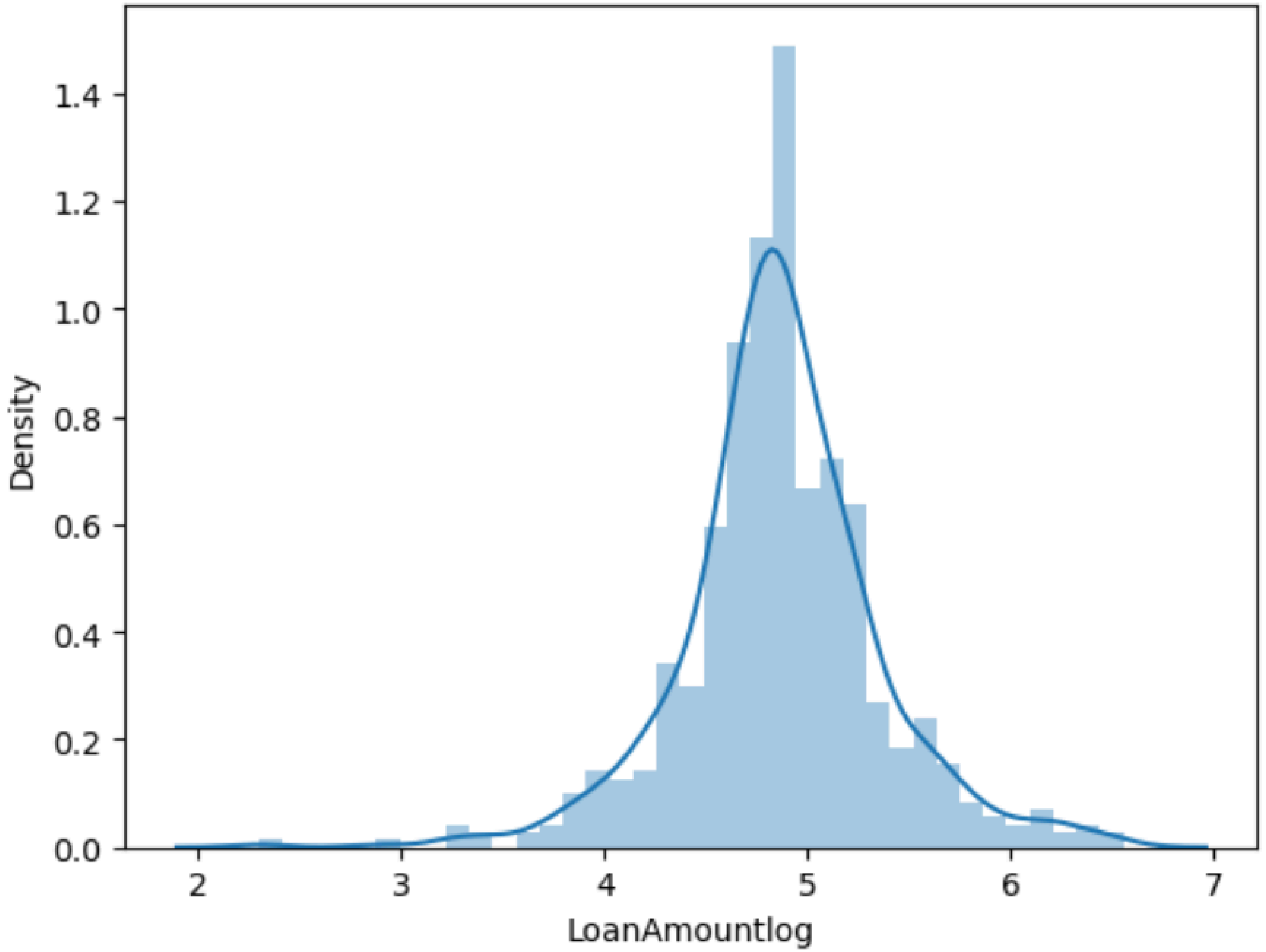


step 15

Applying Log Transformation on Loan-Amount

In [53]:
```
df['LoanAmountlog'] = np.log(df['LoanAmount']+1)
sns.distplot(df['LoanAmountlog'])
```

Out[53]: <Axes: xlabel='LoanAmountlog', ylabel='Density'>

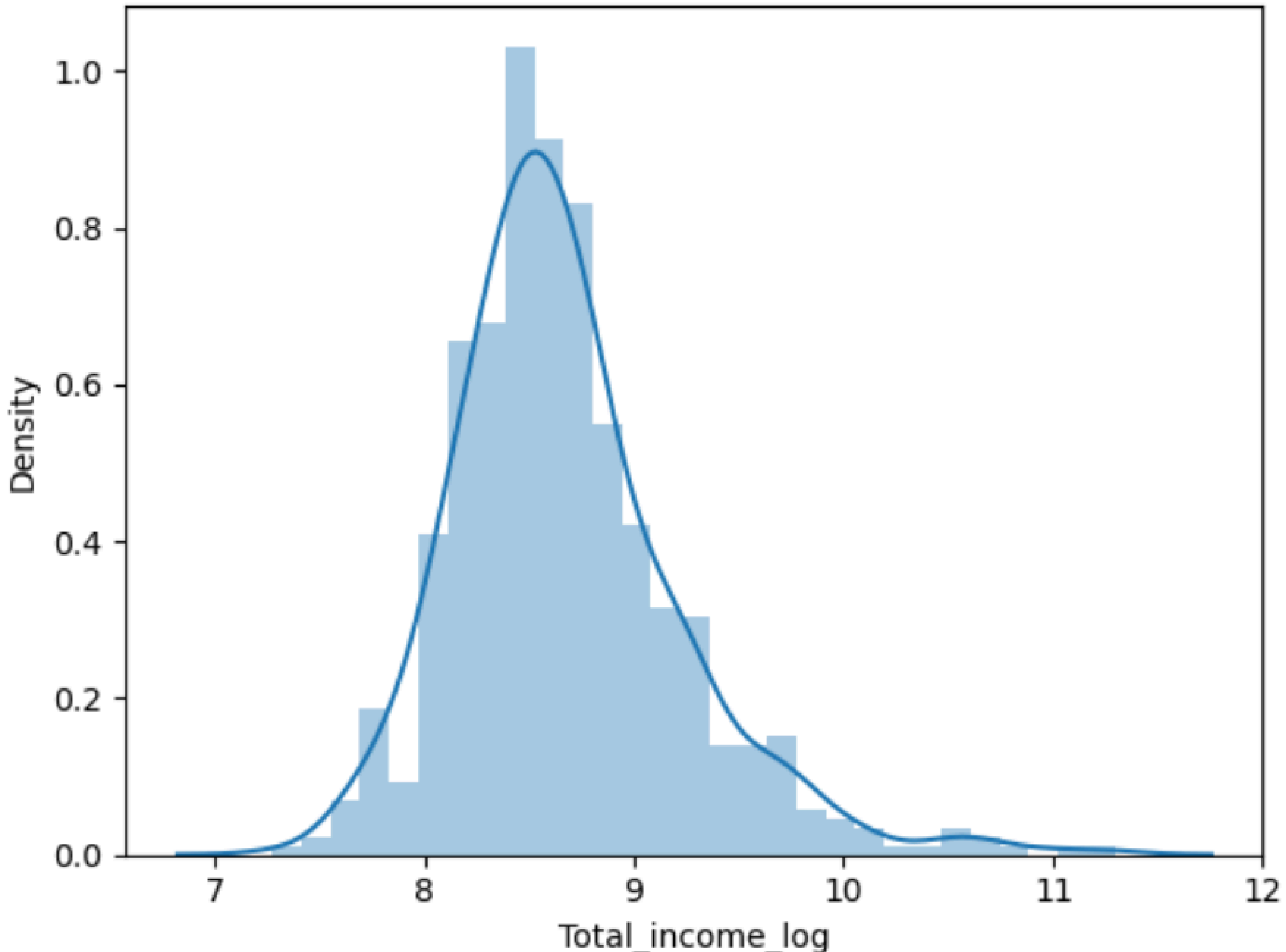


step 16

Applying Log Transformation on Total-Income

In [57]:
```
df['Total_income_log'] = np.log(df['Total_income']+1)
sns.distplot(df['Total_income_log'])
```

Out[57]: <Axes: xlabel='Total_income_log', ylabel='Density'>



step 17 Droping unnecessary columns

In [59]:

```
## Droping unnecessary columns

cls = ['ApplicantIncome','CoapplicantIncome','LoanAmount','Loan_Amount_Term','Total_income','Loan_ID']
df = df.drop(columns = cls , axis =1)
df.head()
```

Out[59]:

	Gender	Married	Dependents	Education	Self_Employed	Credit_History	Property_Area	Loan_Status	ApplicantIncomelog	LoanAmountlog	Loan_Amount_Te
0	Male	No	0	Graduate	No	1.0	Urban	Y	8.674197	4.859812	5.88
1	Male	Yes	1	Graduate	No	1.0	Rural	N	8.430327	4.859812	5.88
2	Male	Yes	0	Graduate	Yes	1.0	Urban	Y	8.006701	4.204693	5.88
3	Male	Yes	0	Not Graduate	No	1.0	Urban	Y	7.857094	4.795791	5.88
4	Male	No	0	Graduate	No	1.0	Urban	Y	8.699681	4.955827	5.88

step 18

Encoding Technique

Encoding Technique = Label Encoding or One Hot Encoding (We use Label encoding when we have only two types of outcomes like y/n so there we can easly place 0/1 and on other hand when we have multiple or different out comes we use One Hot Encoding)

In [65]:
```
## Encoding Technique = Label Encoding or One Hot Encoding (We use Label encoding when we have only t

from sklearn.preprocessing import LabelEncoder
cols = ['Gender','Married','Education','Self_Employed','Dependents','Property_Area','Loan_Status']
le = LabelEncoder()
for col in cols:
    df[col] = le.fit_transform(df[col])
```

step 19

After Converting All Elements into Numerical Form

```
In [67]: df.head(10)
```

Out[67]:

	Gender	Married	Dependents	Education	Self_Employed	Credit_History	Property_Area	Loan_Status	ApplicantIncomelog	LoanAmountlog	Loan_Amount_Te
0	1	0	0	0	0	1.0	2	1	8.674197	4.859812	5.88
1	1	1	1	0	0	1.0	0	0	8.430327	4.859812	5.88
2	1	1	0	0	1	1.0	2	1	8.006701	4.204693	5.88
3	1	1	0	1	0	1.0	2	1	7.857094	4.795791	5.88
4	1	0	0	0	0	1.0	2	1	8.699681	4.955827	5.88
5	1	1	2	0	1	1.0	2	1	8.597482	5.590987	5.88
6	1	1	0	1	0	1.0	2	1	7.755339	4.564348	5.88
7	1	1	3	0	0	0.0	1	0	8.018625	5.068904	5.88
8	1	1	2	0	0	1.0	2	1	8.295798	5.129899	5.88
9	1	1	1	0	0	1.0	1	0	9.460476	5.857933	5.88

step 20

Splitting the Data-Set into Dependent or independent.

In [68]:
```
## Splitting the Data-Set into Dependent or independent.

X = df.drop(columns='Loan_Status',axis=1)
Y = df['Loan_Status']
```

In [69]: X

Out[69]:

	Gender	Married	Dependents	Education	Self_Employed	Credit_History	Property_Area	ApplicantIncomelog	LoanAmountlog	Loan_Amount_Termlog	Total
0	1	0	0	0	0	1.0	2	8.674197	4.859812	5.888878	
1	1	1	1	0	0	1.0	0	8.430327	4.859812	5.888878	
2	1	1	0	0	1	1.0	2	8.006701	4.204693	5.888878	
3	1	1	0	1	0	1.0	2	7.857094	4.795791	5.888878	
4	1	0	0	0	0	1.0	2	8.699681	4.955827	5.888878	
...	...	...	...	...	...	...	...	...	...	...	
609	0	0	0	0	0	1.0	0	7.972811	4.276666	5.888878	
610	1	1	3	0	0	1.0	0	8.320448	3.713572	5.198497	
611	1	1	1	0	0	1.0	2	8.996280	5.537334	5.888878	
612	1	1	2	0	0	1.0	2	8.933796	5.236442	5.888878	
613	0	0	0	0	1	0.0	1	8.430327	4.897840	5.888878	

614 rows × 11 columns

In [70]: Y

Out[70]:

```
0      1
1      0
2      1
3      1
4      1
      ..
609    1
610    1
611    1
612    1
613    0
Name: Loan_Status, Length: 614, dtype: int32
```

step 21

Importing all Machine_learing Models Which is reqired for ML-Algorithm

In [72]:

```
## Importing all Machine_learing Models Which is reqired for ML-Algorithm

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
```

step 22

Now we are going to Split our data-set in form of Traning and Teasting

```
In [73]: ## Now we are going to Split our data-set in form of Traning and Teasting .

X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.25,random_state=42)
```

step 23

Preforming Logistic Regression

In [89]:
```
## Logistic Regression

model1 = LogisticRegression()
model1.fit(X_train,Y_train)
y_pred_model1 = model1.predict(X_test)
accuracy = accuracy_score(Y_test,y_pred_model1)
```

In [90]:
```
accuracy

## Converting into Percentage.

accuracy*100
```

Out[90]:
```
77.27272727272727
```

In [78]:
```
## Accuarcy = The ratio of the correctly predicted values  to the total values.
```

In [79]:
```
score = cross_val_score(model1,X,Y,cv=6)
score
```

Out[79]:
```
array([0.82524272, 0.77669903, 0.7745098 , 0.81372549, 0.83333333,
       0.83333333])
```

In [80]:
```
np.mean(score)*100
```

Out[80]:
```
80.94739513928548
```

step 24

Preforming Decision Tree Classifier

In [91]:
```
## Decision_Tree_Classifier

model2 = DecisionTreeClassifier()
model2.fit(X_train,Y_train)
y_pred_model2 = model2.predict(X_test)
accuracy = accuracy_score(Y_test,y_pred_model2)
print("Accuracy of Decision tree Model : ",accuracy*100)
```

```
Accuracy of Decision tree Model :  71.42857142857143
```

In [92]:
```
score = cross_val_score(model2,X,Y,cv=6)
score
np.mean(score)*100
```

Out[92]:
```
70.84047211117456
```

step 25

K_Neighbors_Classifier

In [94]:

```python
## K_Neighbors_Classifier

model4 = KNeighborsClassifier(n_neighbors =3)
model4.fit(X_train,Y_train)
y_pred_model4 = model4.predict(X_test)
accuracy = accuracy_score(Y_test,y_pred_model4)
print("Accuracy of K Neighbors Model : ",accuracy*100)
```

```
Accuracy of K Neighbors Model :  71.42857142857143
```

step 26

genreate_classification_report

In [97]:
```
from sklearn.metrics import classification_report

def genreate_classification_report(model_name,Y_test,y_pred):
    report = classification_report(Y_test,y_pred)
    print(f"classification report for {model_name}:\n{report}\n")

genreate_classification_report(model1,Y_test,y_pred_model1)
genreate_classification_report(model2,Y_test,y_pred_model2)
genreate_classification_report(model3,Y_test,y_pred_model3)
genreate_classification_report(model4,Y_test,y_pred_model4)
```

classification report for LogisticRegression():

	precision	recall	f1-score	support
0	0.91	0.39	0.55	54
1	0.75	0.98	0.85	100
accuracy			0.77	154
macro avg	0.83	0.68	0.70	154
weighted avg	0.81	0.77	0.74	154

classification report for DecisionTreeClassifier():

	precision	recall	f1-score	support
0	0.61	0.50	0.55	54
1	0.75	0.83	0.79	100
accuracy			0.71	154
macro avg	0.68	0.67	0.67	154
weighted avg	0.71	0.71	0.71	154

classification report for RandomForestClassifier():

	precision	recall	f1-score	support
0	0.89	0.46	0.61	54
1	0.77	0.97	0.86	100
accuracy			0.79	154
macro avg	0.83	0.72	0.73	154
weighted avg	0.81	0.79	0.77	154

classification report for KNeighborsClassifier(n_neighbors=3):

	precision	recall	f1-score	support
0	0.63	0.44	0.52	54
1	0.74	0.86	0.80	100
accuracy			0.71	154
macro avg	0.69	0.65	0.66	154
weighted avg	0.70	0.71	0.70	154

Thank You