VIDYAVARDHINI'S COLLEGE OF ENGINEERING & TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY
K.T. Marg, Vasai Road (W), Dist-Palghar - 401202, Maharashtra

DS Mini-project

Name : Abhishek Jani
Class : BEIT_16

Title : Loan Status Prediction

Problem Statement:

The problem at hand revolves around the development of a loan status prediction system utilizing Python and employing various classification models to enhance predictive accuracy. The primary challenge lies in harnessing a dataset comprising loan-related information, and undertaking essential data preprocessing tasks such as addressing missing values, eliminating duplicates, managing outliers, and encoding categorical variables. Furthermore, to gain a comprehensive understanding of the data, an in-depth exploratory data analysis will be conducted, leveraging visualization techniques.

The dataset may potentially exhibit class imbalance in terms of approved and denied loan applications, which necessitates careful handling. Subsequently, the data will be partitioned into training and testing sets to facilitate model evaluation. The core of the project involves the implementation of at least four distinct machine learning classification models, each designed to predict loan approval status. The performance of these models will be rigorously assessed through various performance metrics, ultimately allowing us to determine the most effective model for predicting loan outcomes.

Dataset Description:

The dataset offers a rich repository of information pertaining to loan applications, capturing a wide array of applicant attributes and the eventual loan approval status. These attributes encompass critical aspects of the applicants' profiles, including their gender, marital status, the number of dependents they have, their educational background, self-employment status, income details, the amount of the loan they are seeking, the term for which they want the loan, their credit history, the area where the property is located, and, most importantly, the outcome of the loan application – whether it was approved (indicated as 'Y') or denied (indicated as 'N').

This dataset is a valuable resource for the development of predictive models that can discern and forecast the likelihood of loan approval based on the provided applicant information. By examining this dataset, financial institutions and lenders can gain insights into the factors that most significantly influence loan approval decisions, allowing them to make more informed and data-driven choices when assessing loan applications. Moreover, such models can contribute to streamlining the loan application process, thereby enhancing the efficiency and accuracy of lending practices.

The dataset for loan status prediction comprises the following columns:

Sr No .	Variable	Description
1.	Loan_ID	**Unique Loan ID**
2.	Gender	**Male/ Female**
3.	Married	**Applicant married (Y/N)**
4.	Dependents	**Number of dependents**
5.	Education	**Applicant Education (Graduate/ Undergraduate)**
6.	Education	**Self employed (Y/N)**
7.	ApplicantIncome	**Applicant income**
8.	CoapplicantIncome	**Coapplicant income**
9.	LoanAmount	**Loan amount in thousands**
10.	Loan_Amount_Term	**Term of loan in months**
11.	Credit_History	**credit history meets guidelines**
12.	Property_Area	**Urban/ Semi Urban/ Rural**
13.	Loan_Status	**Loan approved (Y/N)**

Loan Status Prediction

Import modules

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

Loading the dataset

In [ ]:
```
df = pd.read_csv("Loan Prediction Dataset.csv")
df.head()
```

Out[ ]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coap
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 [ ]:
```
df.describe()
```

Out[ ]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

In [ ]:
```
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
```

Preprocessing the dataset

In [ ]:
```
# find the null values
df.isnull().sum()
```

Out[ ]:
```
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
dtype: int64
```

In [ ]:
```
# fill the missing values for numerical terms - mean
df['LoanAmount'] = df['LoanAmount'].fillna(df['LoanAmount'].mean())
df['Loan_Amount_Term'] = df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].me
df['Credit_History'] = df['Credit_History'].fillna(df['Credit_History'].mean())
```

In [ ]:
```
# fill the missing values for categorical terms - mode
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])
df['Self_Employed'] = df["Self_Employed"].fillna(df['Self_Employed'].mode()[0])
```

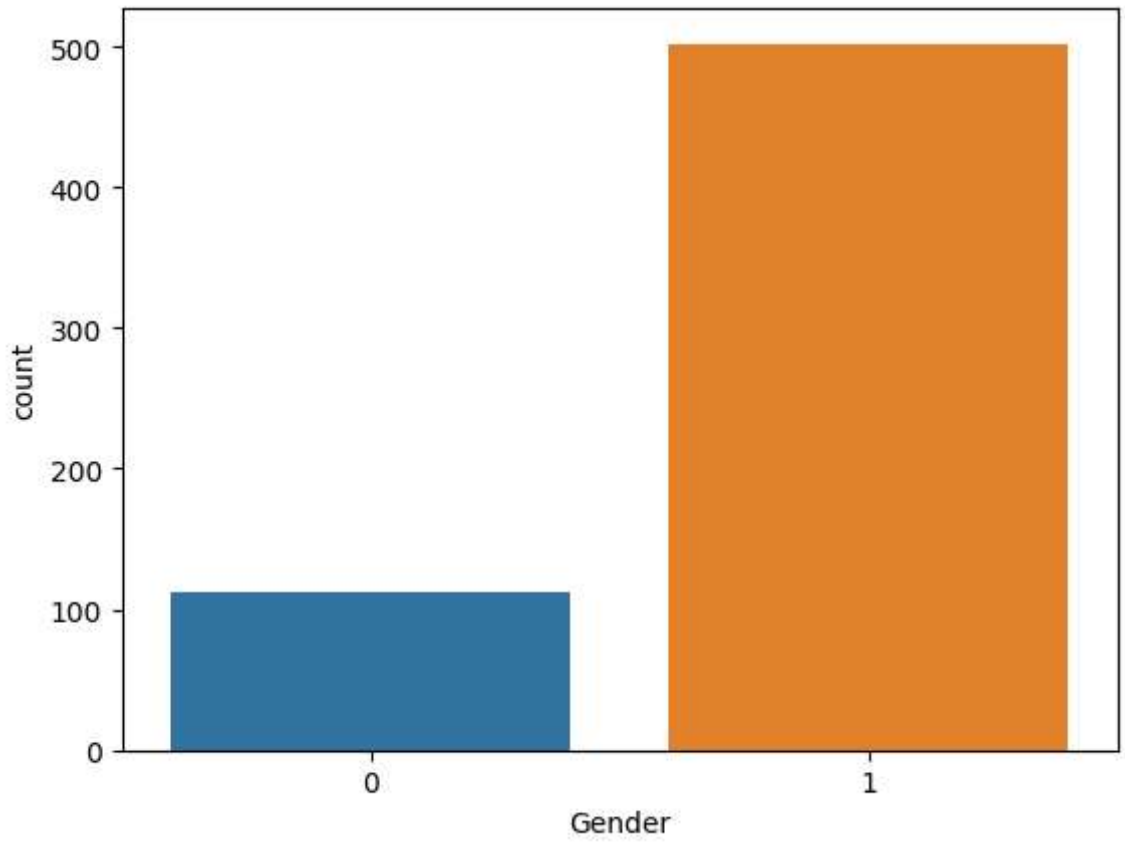
In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
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
```

Exploratory Data Analysis

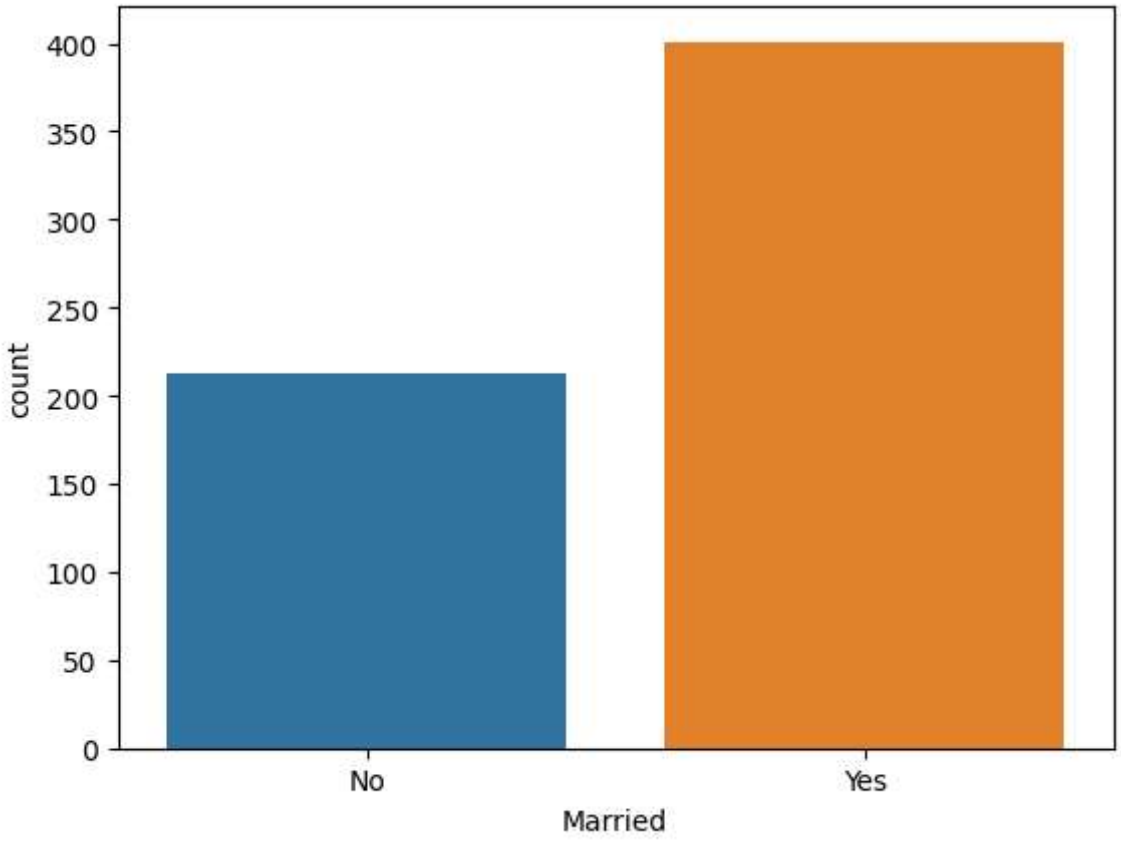
In [ ]:
```
# categorical attributes visualization
sns.countplot(x='Gender', data=df)
```

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



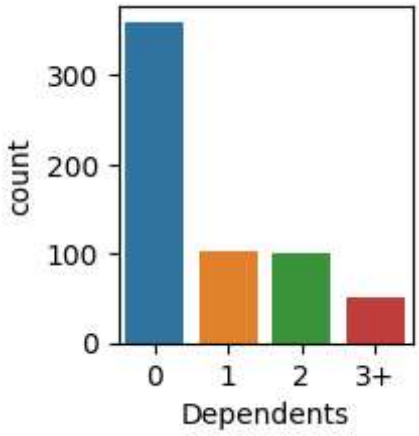
In [ ]:
```
sns.countplot(x='Married', data=df)
```

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



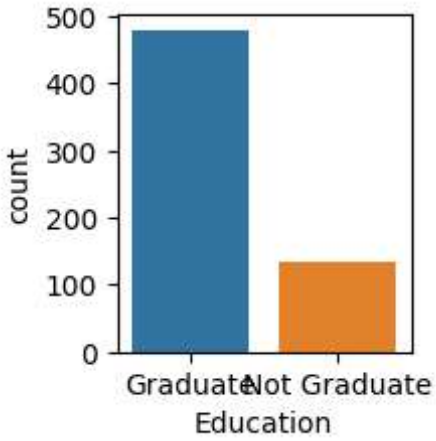
In [ ]:
```
# plt.subplot(2, 3, 2)
sns.countplot(x='Dependents', data=df)
```

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



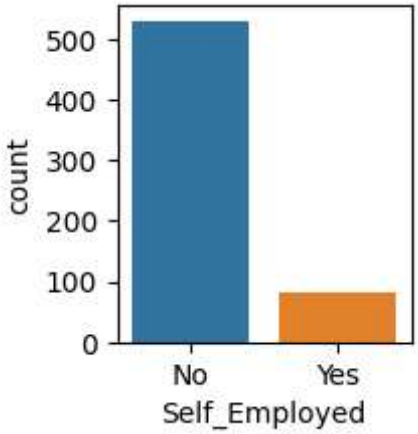
In [ ]:
```
plt.subplot(2, 3, 3)
sns.countplot(x='Education', data=df)
```

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



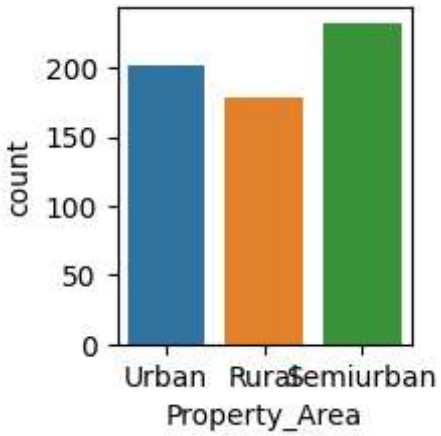
In [ ]:
```
plt.subplot(2, 3, 4)
sns.countplot(x='Self_Employed', data=df)
```

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



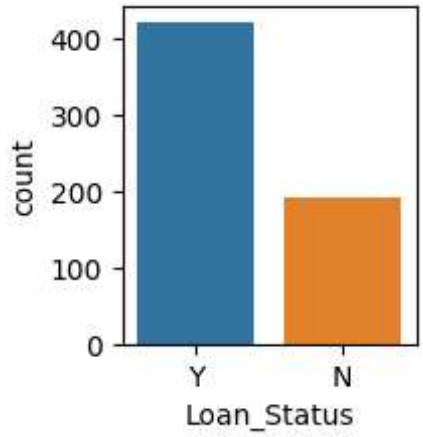
In [ ]:
```
plt.subplot(2, 3, 5)
sns.countplot(x='Property_Area', data=df)
```

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



In [ ]:
```
plt.subplot(2, 3, 6)
sns.countplot(x='Loan_Status', data=df)
```

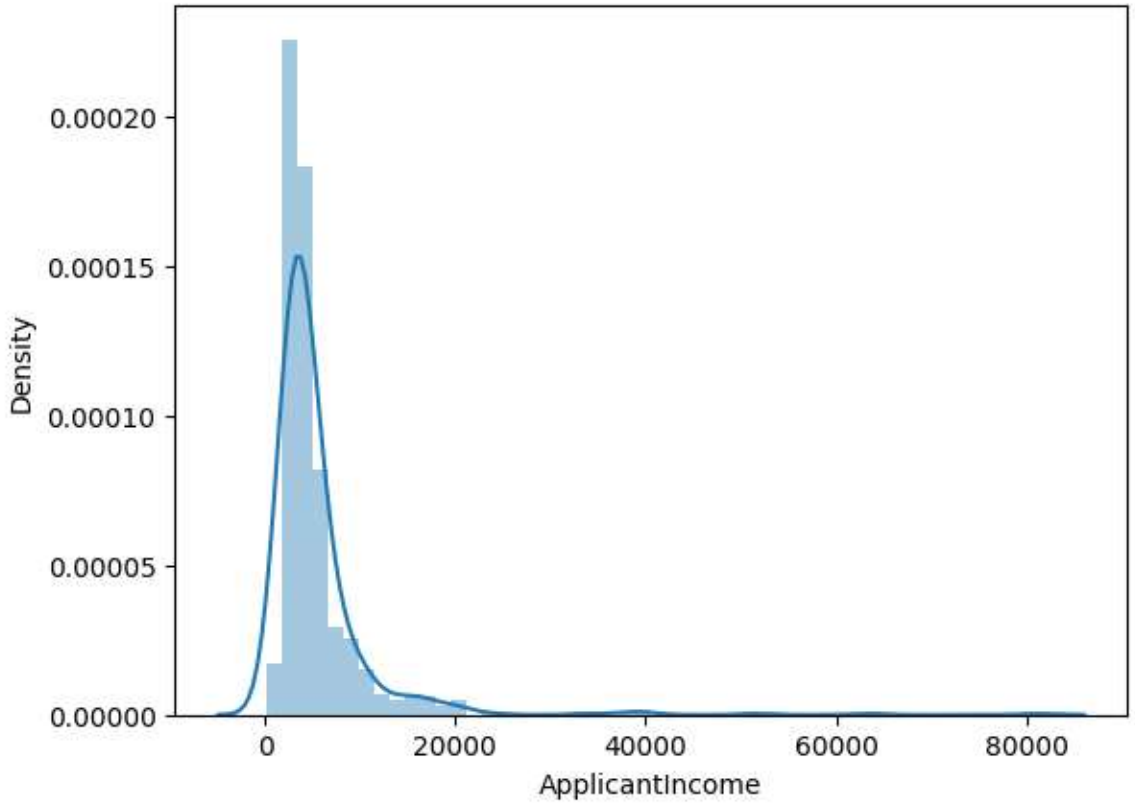
Out[ ]:
```
<Axes: xlabel='Loan_Status', ylabel='count'>
```



In [ ]:

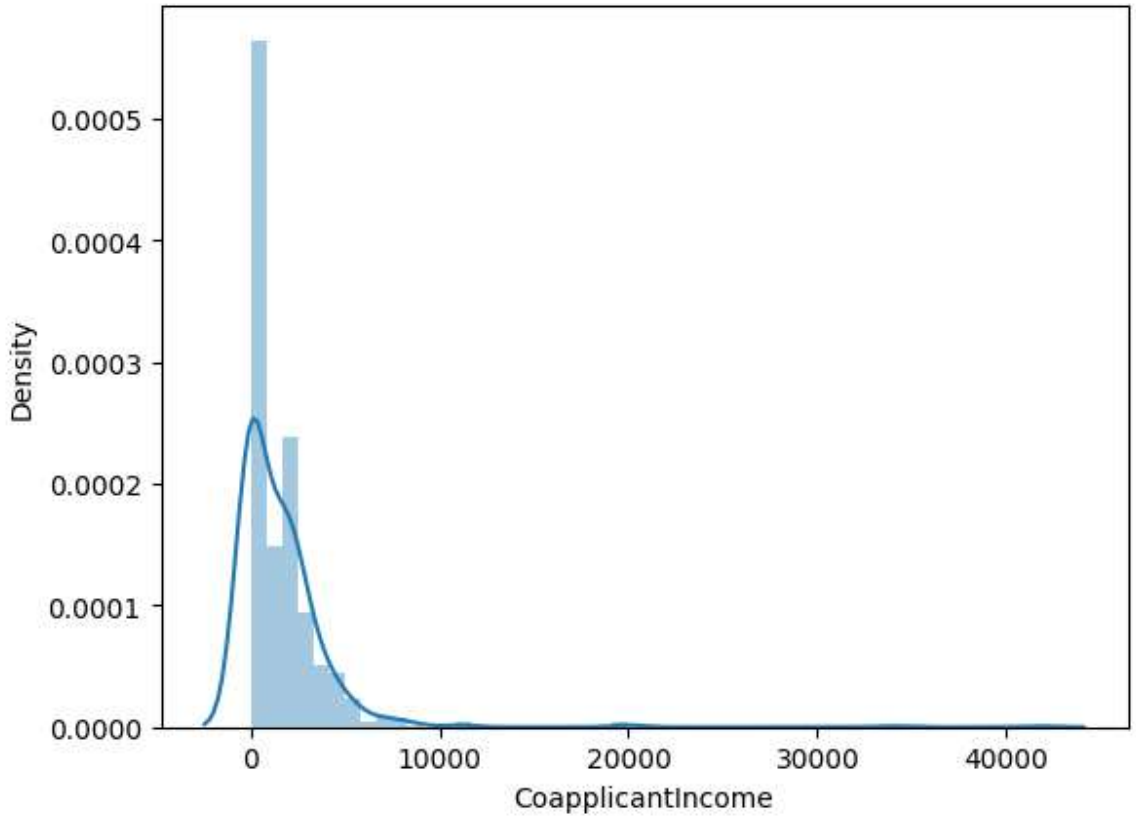
In [ ]:
```
# numerical attributes visualization
# plt.subplot(1, 3, 1)
sns.distplot(df["ApplicantIncome"])
```

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



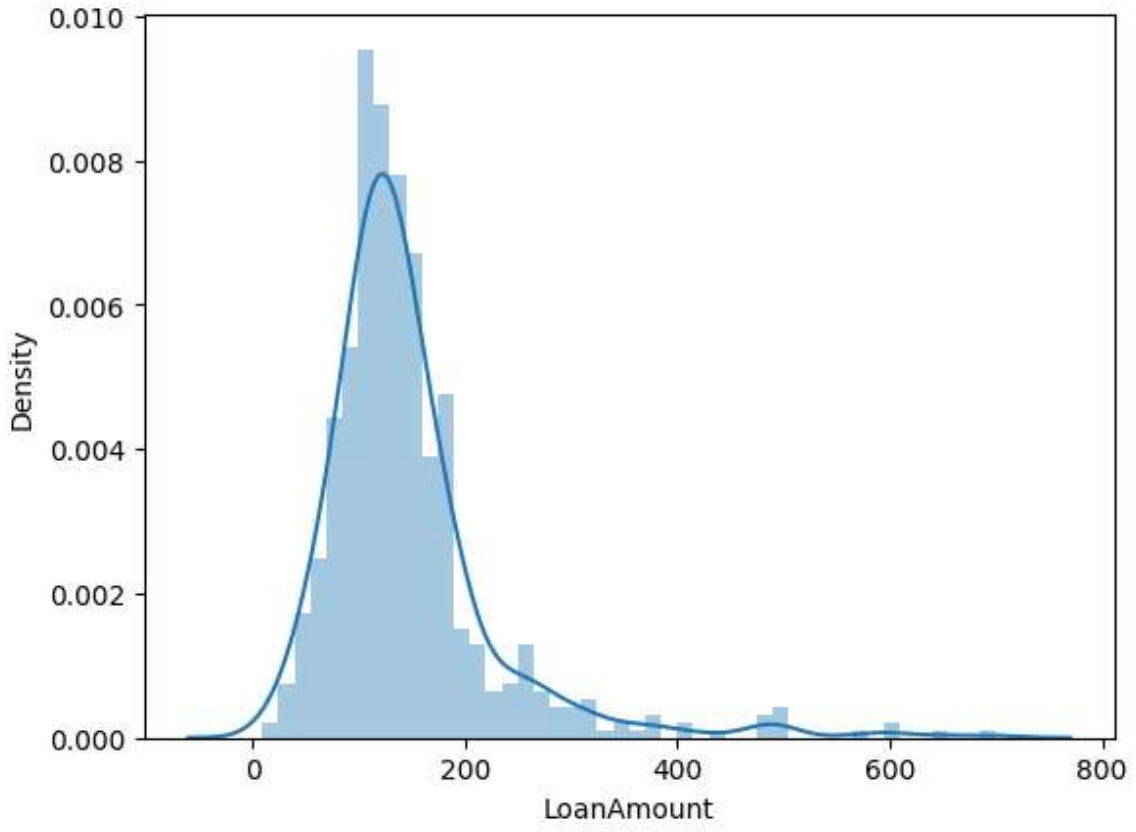
In [ ]:
```
sns.distplot(df["CoapplicantIncome"])
```

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



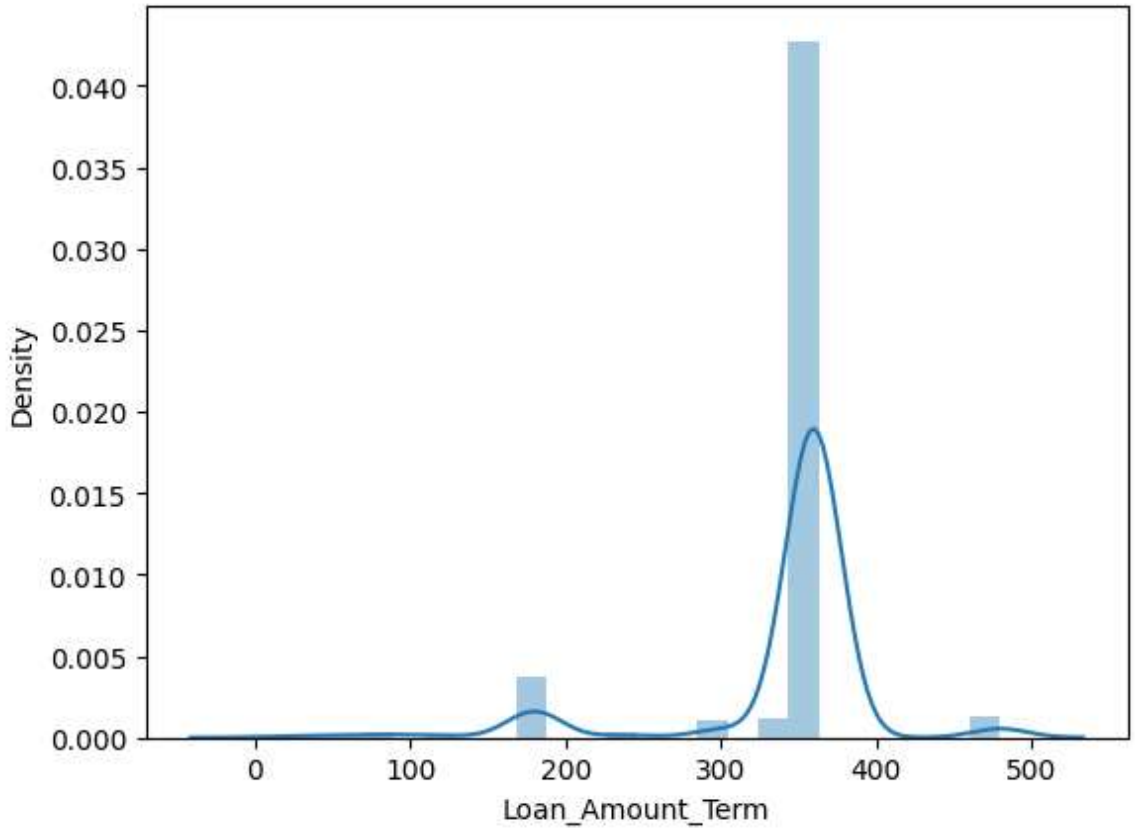
In [ ]:
```
sns.distplot(df["LoanAmount"])
```

Out[ ]:
```
<Axes: xlabel='LoanAmount', ylabel='Density'>
```



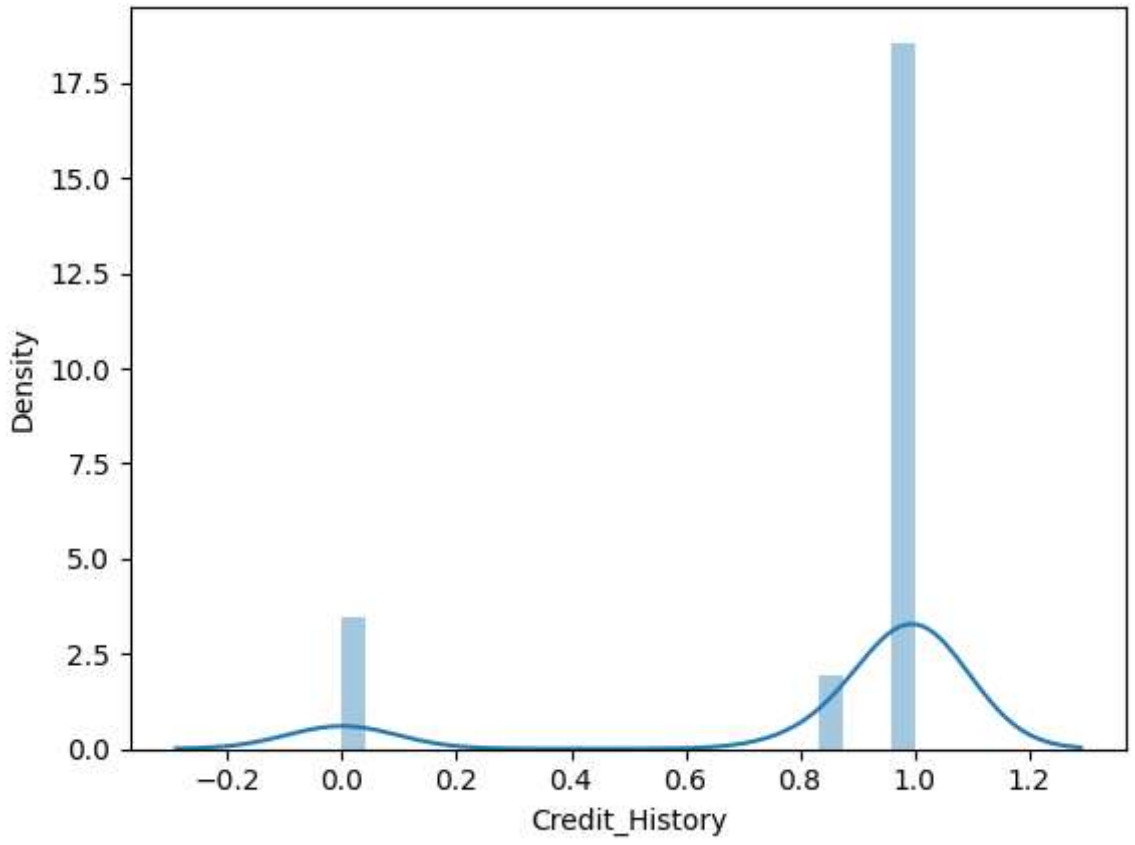
In [ ]:
```
sns.distplot(df['Loan_Amount_Term'])
```

Out[ ]:
```
<Axes: xlabel='Loan_Amount_Term', ylabel='Density'>
```



In [ ]:
```
sns.distplot(df['Credit_History'])
```

Out[ ]:
```
<Axes: xlabel='Credit_History', ylabel='Density'>
```



In [ ]:

Creation of new attributes

In [ ]:
```
# total income
df['Total_Income'] = df['ApplicantIncome'] + df['CoapplicantIncome']
df.head()
```

Out[ ]:

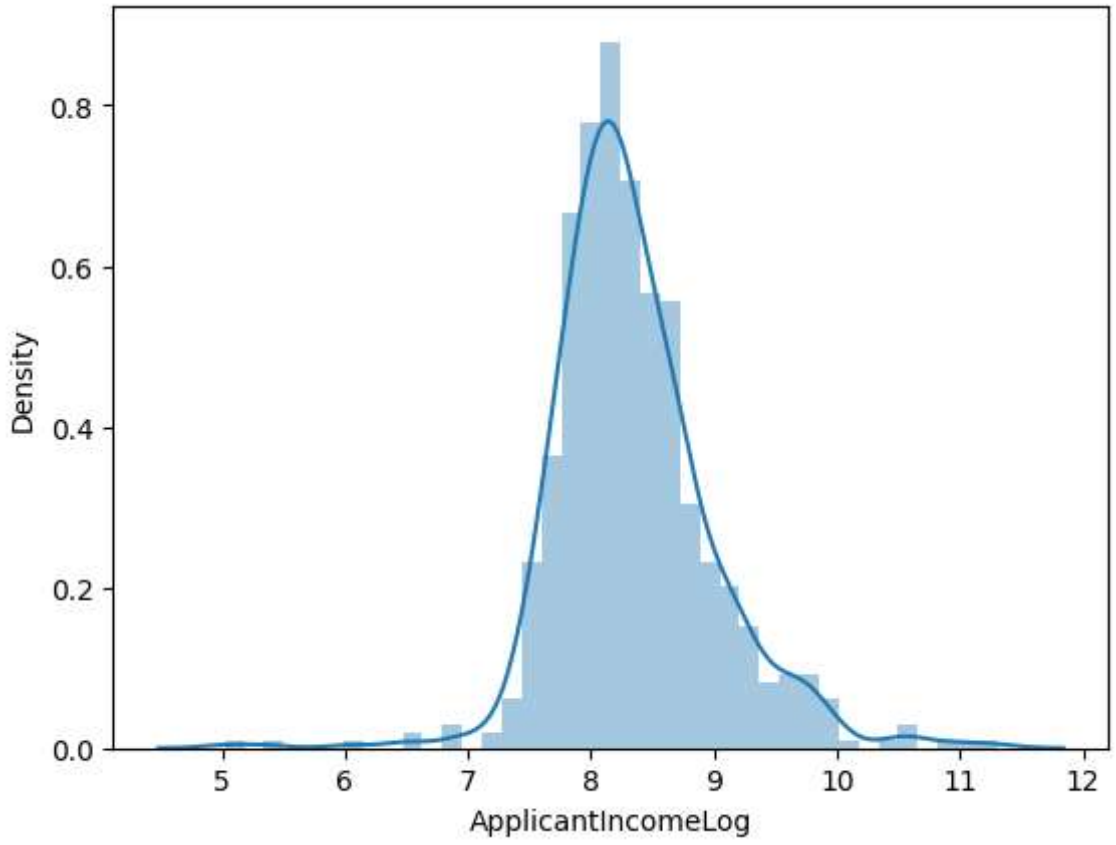
	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coap
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	
4	LP001008	Male	No	0	Graduate	No	6000	

Log Transformation

Log transformation is performed to make the data more symmetric , normalize and it can be used for any statistical method , before performing of log transformation data is completely skwed distribution while it becomes normal distribution now

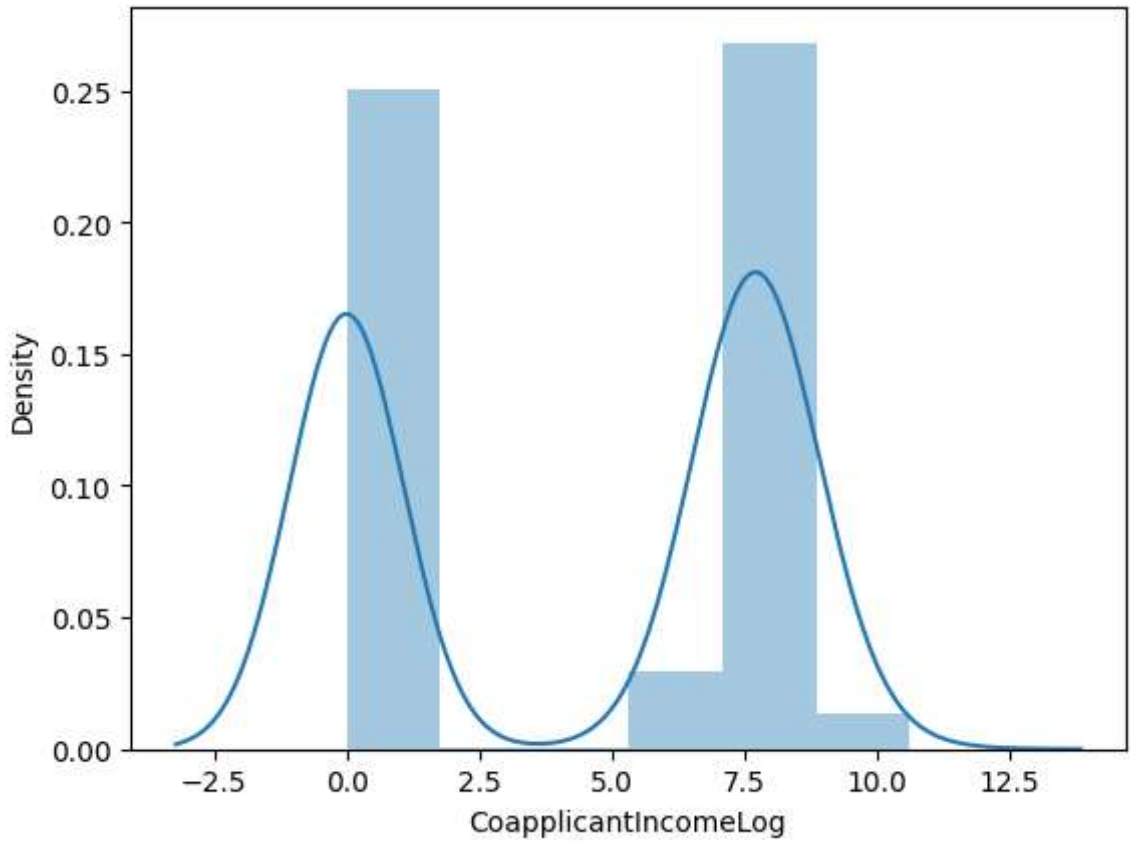
In [ ]:
```
# apply log transformation to the attribute
df['ApplicantIncomeLog'] = np.log(df['ApplicantIncome']+1)
sns.distplot(df["ApplicantIncomeLog"])
```

Out[ ]:
```
<Axes: xlabel='ApplicantIncomeLog', ylabel='Density'>
```



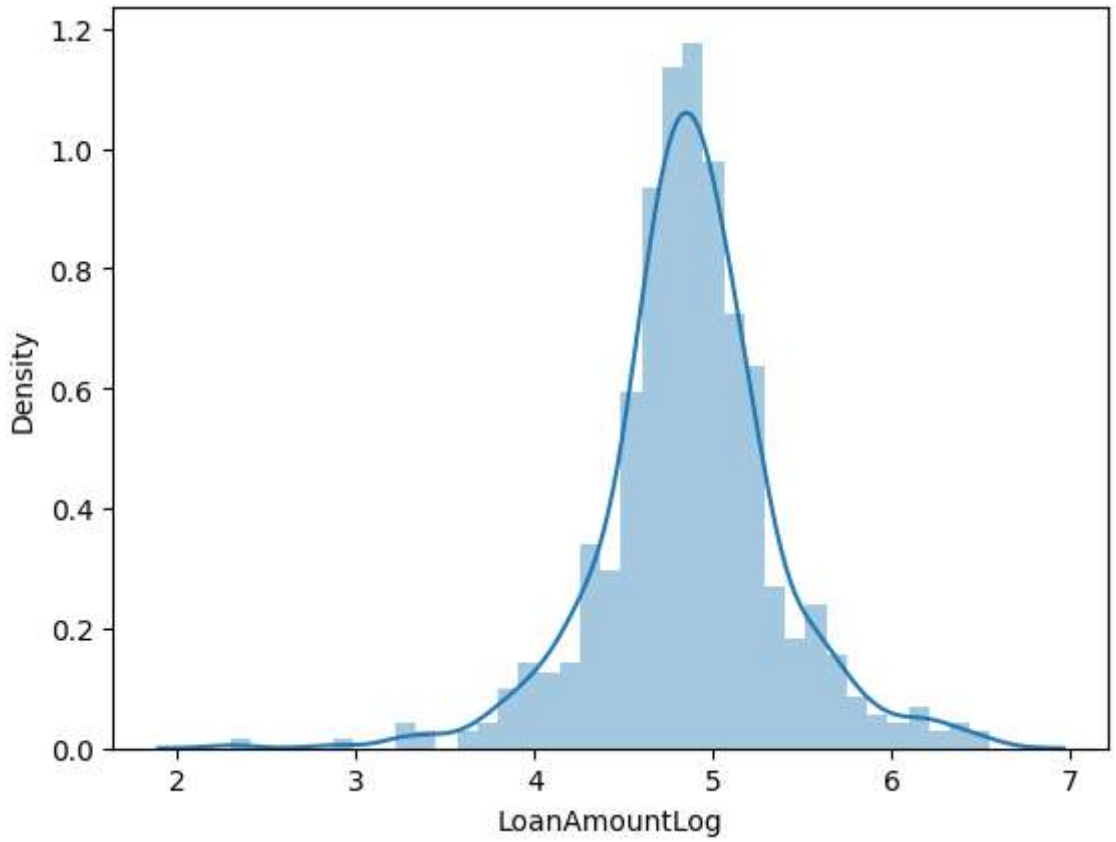
In [ ]:
```
df['CoapplicantIncomeLog'] = np.log(df['CoapplicantIncome']+1)
sns.distplot(df["CoapplicantIncomeLog"])
```

Out[ ]:
```
<Axes: xlabel='CoapplicantIncomeLog', ylabel='Density'>
```



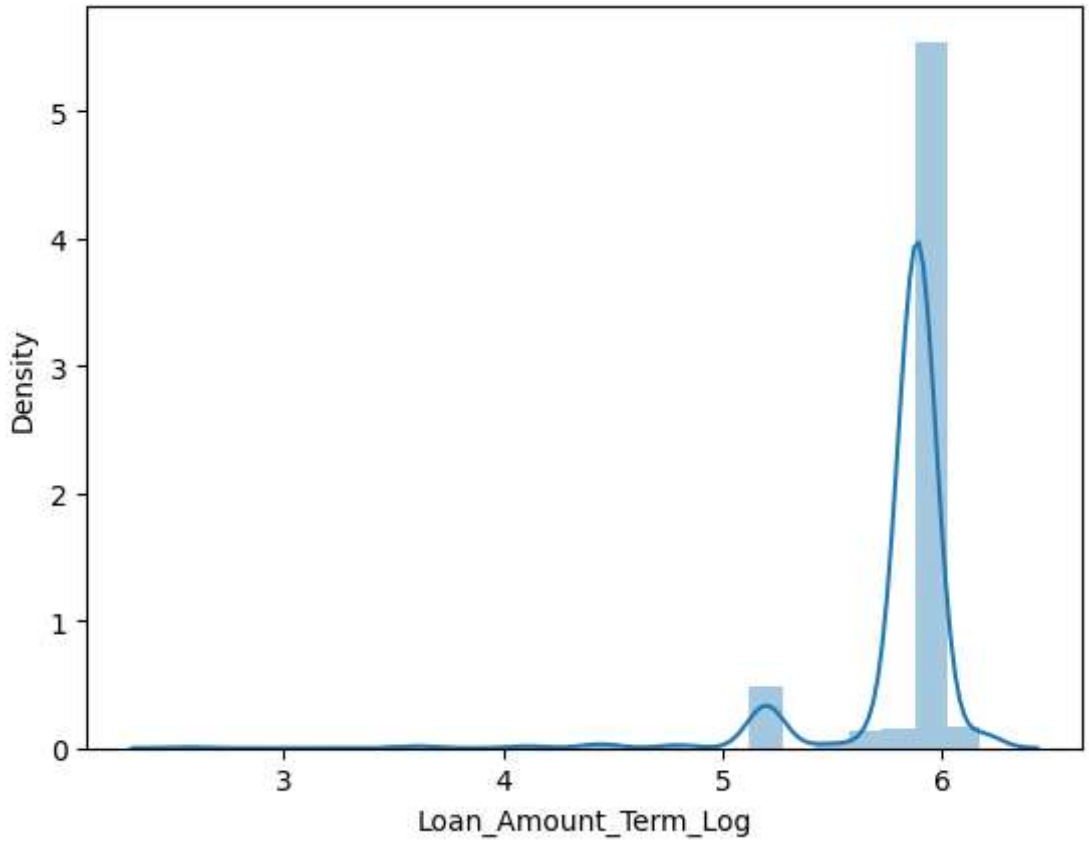
In [ ]:
```
df['LoanAmountLog'] = np.log(df['LoanAmount']+1)
sns.distplot(df["LoanAmountLog"])
```

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



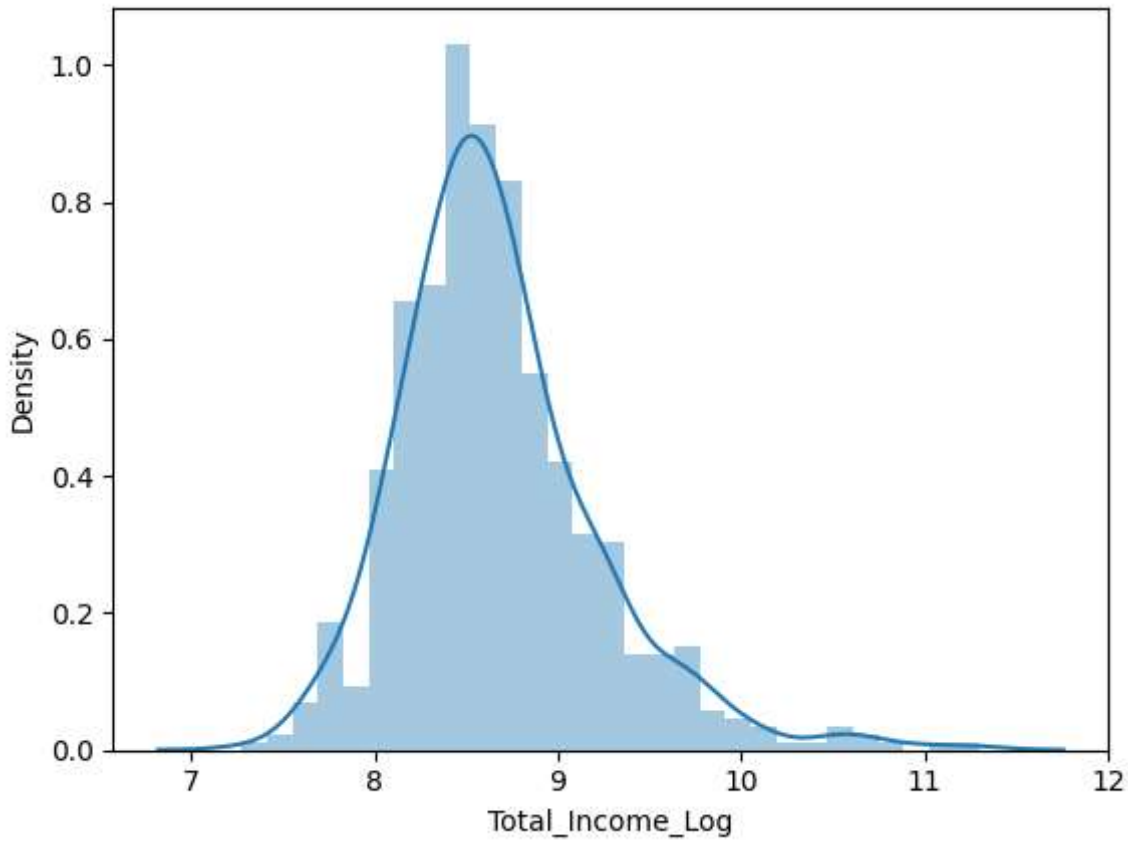
In [ ]:
```
df['Loan_Amount_Term_Log'] = np.log(df['Loan_Amount_Term']+1)
sns.distplot(df["Loan_Amount_Term_Log"])
```

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



In [ ]:
```
df['Total_Income_Log'] = np.log(df['Total_Income']+1)
sns.distplot(df["Total_Income_Log"])
```

Out[ ]:
```
<Axes: xlabel='Total_Income_Log', ylabel='Density'>
```



Class Imbalancing

In [ ]:
```
import pandas as pd
from imblearn.over_sampling import SMOTE
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming df is your dataset with 'X' as features and 'y' as the target variabl
# X = df.drop('target', axis=1)
# y = df['target']

# Initialize and apply SMOTEa
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Visualize the class distribution before and after SMOTE
plt.figure(figsize=(6, 3))
plt.subplot(1, 2, 1)
sns.countplot(x=y, data=df)
plt.title("Class Distribution Before SMOTE")

plt.subplot(1, 2, 2)
sns.countplot(x=y_resampled, data=pd.DataFrame({'target': y_resampled}))
plt.title("Class Distribution After SMOTE")

plt.tight_layout()
```

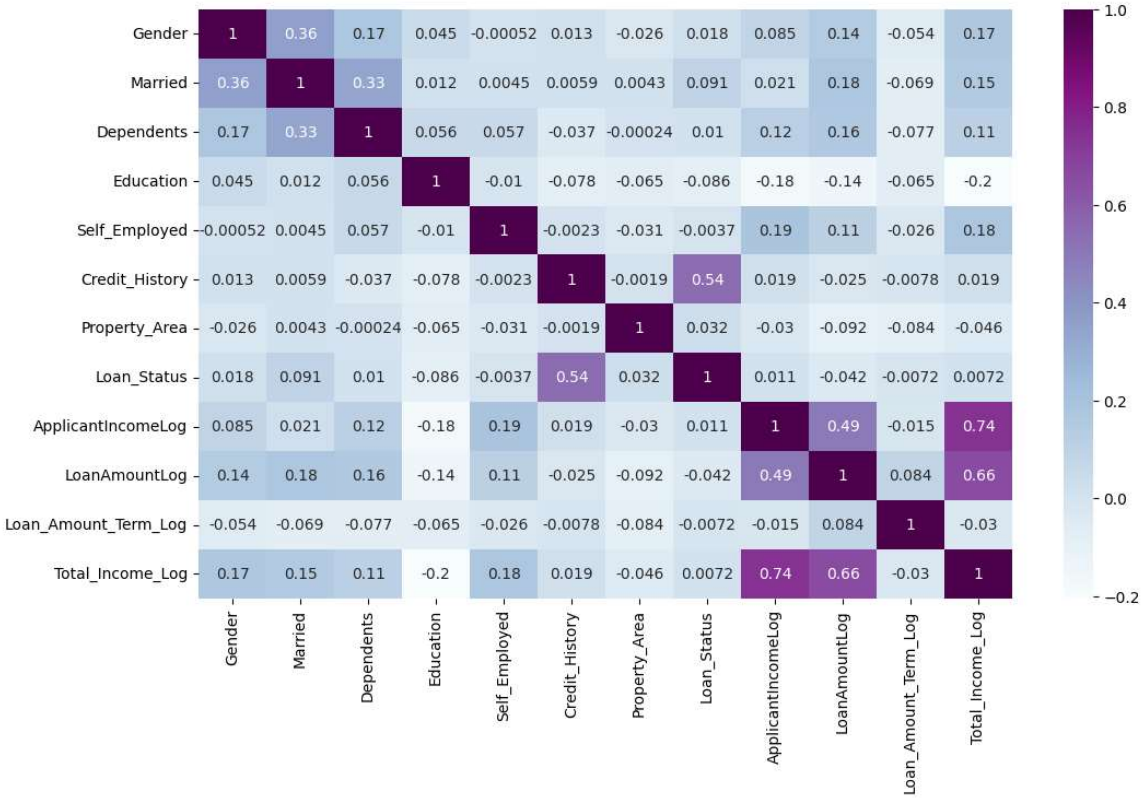
```
# plt.figure(figsize=(3, 2))  # Set the figure size to 8 inches in width and 4 i
plt.show()
```

Coorelation Matrix

A correlation matrix, or correlation matrix, is a table or matrix that displays the pairwise correlations between multiple variables in a dataset . It is used to identify how the column attribute is getting changes as compared to other columns .

In [ ]:
```
corr = df.corr()
plt.figure(figsize=(12,7))
sns.heatmap(corr, annot = True, cmap="BuPu")
```

Out[ ]: <Axes: >



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

Out[ ]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coap
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 [ ]:
```
# drop unnecessary columns
cols = ['ApplicantIncome', 'CoapplicantIncome', "LoanAmount", "Loan_Amount_Term"
df = df.drop(columns=cols, axis=1)
df.head()
```

Out[ ]:

	Gender	Married	Dependents	Education	Self_Employed	Credit_History	Property_Area	Lo
0	Male	No	0	Graduate	No	1.0	Urban	
1	Male	Yes	1	Graduate	No	1.0	Rural	
2	Male	Yes	0	Graduate	Yes	1.0	Urban	
3	Male	Yes	0	Not Graduate	No	1.0	Urban	
4	Male	No	0	Graduate	No	1.0	Urban	

Label Encoding

Label encoding is a technique used in machine learning and data preprocessing to convert categorical data (non-numeric data) into numerical format.

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

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

Out[ ]:

	Gender	Married	Dependents	Education	Self_Employed	Credit_History	Property_Area	Lo
0	1	0	0	0	0	1.0	2	
1	1	1	1	0	0	1.0	0	
2	1	1	0	0	1	1.0	2	
3	1	1	0	1	0	1.0	2	
4	1	0	0	0	0	1.0	2	

Train-Test Split

In [ ]:
```
# specify input and output attributes
X = df.drop(columns=['Loan_Status'], axis=1)
y = df['Loan_Status']
```

In [ ]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random
```

Model Training

In [ ]:
```
# classify function
from sklearn.model_selection import cross_val_score
def classify(model, x, y):
    x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.25, ra
    model.fit(x_train, y_train)
    print("Accuracy is", model.score(x_test, y_test)*100)
    # cross validation - it is used for better validation of model
    # eg: cv-5, train-4, test-1
    score = cross_val_score(model, x, y, cv=5)
    print("Cross validation is",np.mean(score)*100)
```

In [ ]:
```
from sklearn.linear_model import LogisticRegression
tmodel = LogisticRegression()
classify(tmodel, X, y)
```

```
Accuracy is 77.27272727272727
Cross validation is 80.9462881514061
```

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
classify(model, X, y)
```

```
Accuracy is 72.72727272727273
Cross validation is 70.19992003198719
```

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier,ExtraTreesClassifier
model = RandomForestClassifier()
classify(model, X, y)
```

```
Accuracy is 77.92207792207793
Cross validation is 78.01412768226042
```

In [ ]:
```
model = ExtraTreesClassifier()
classify(model, X, y)
```

```
Accuracy is 72.72727272727273
Cross validation is 77.68892443022791
```

In [ ]:

In [ ]:

Performnce Matrix

In [ ]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc

# Define a function to evaluate and print classification performance metrics
def evaluate_model(model, x_test, y_test):
    y_pred = model.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)
    report = classification_report(y_test, y_pred)

    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1 Score:", f1)
    print("Confusion Matrix:\n", conf_matrix)
    print("Classification Report:\n", report)

# Assuming you have already trained your models and have x_test and y_test
# For each model, you can evaluate and print the performance metrics
evaluate_model(tmodel, x_test, y_test)  # Logistic Regression
evaluate_model(model, x_test, y_test)  # Decision Tree
evaluate_model(model, x_test, y_test)  # Random Forest
evaluate_model(model, x_test, y_test)  # Extra Trees
```

```
Accuracy: 0.7727272727272727
Precision: 0.7480916030534351
Recall: 0.98
F1 Score: 0.8484848484848484
Confusion Matrix:
 [[21 33]
 [ 2 98]]
Classification Report:
              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

Accuracy: 0.7727272727272727
Precision: 0.7642276422764228
Recall: 0.94
F1 Score: 0.8430493273542602
Confusion Matrix:
 [[25 29]
 [ 6 94]]
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.46      0.59        54
           1       0.76      0.94      0.84       100

    accuracy                           0.77       154
   macro avg       0.79      0.70      0.72       154
weighted avg       0.78      0.77      0.75       154

Accuracy: 0.7727272727272727
Precision: 0.7642276422764228
Recall: 0.94
F1 Score: 0.8430493273542602
Confusion Matrix:
 [[25 29]
 [ 6 94]]
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.46      0.59        54
           1       0.76      0.94      0.84       100

    accuracy                           0.77       154
   macro avg       0.79      0.70      0.72       154
weighted avg       0.78      0.77      0.75       154

Accuracy: 0.7727272727272727
Precision: 0.7642276422764228
Recall: 0.94
F1 Score: 0.8430493273542602
Confusion Matrix:
 [[25 29]
 [ 6 94]]
Classification Report:
              precision    recall  f1-score   support
```

0	0.81	0.46	0.59	54
1	0.76	0.94	0.84	100
accuracy			0.77	154
macro avg	0.79	0.70	0.72	154
weighted avg	0.78	0.77	0.75	154

Conclusion:

The Random Forest algorithm, boasting a respectable accuracy rate of 77.9%, emerges as a powerful tool for classifying data efficiently. Its strength lies in its ability to make accurate predictions. Particularly, this algorithm proves invaluable in the context of loan status prediction.

When it comes to obtaining a loan, users can rely on this method to gauge their chances of approval. This predictive tool is a valuable resource for individuals seeking financial assistance. Its accuracy empowers users to make informed decisions and increases their chances of successfully securing a loan. In summary, the Random Forest algorithm provides a dependable means of forecasting loan outcomes, offering peace of mind and confidence to applicants.