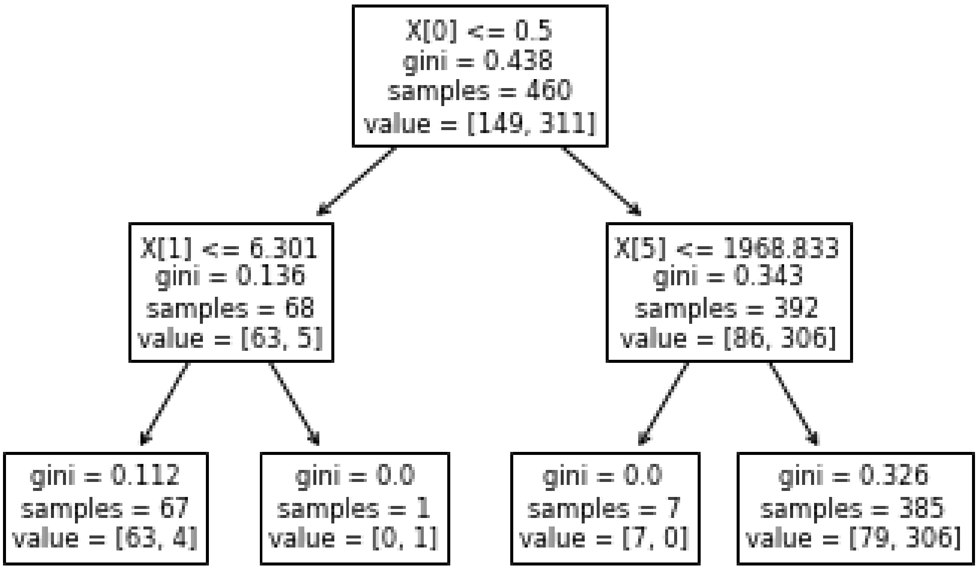
DECISION TREE ALGORITHM



Predicting whether a Customer would be granted a loan or not depending on various parameters.

Date	27/04/2021
Name	Vinit Ravichandran Iyer

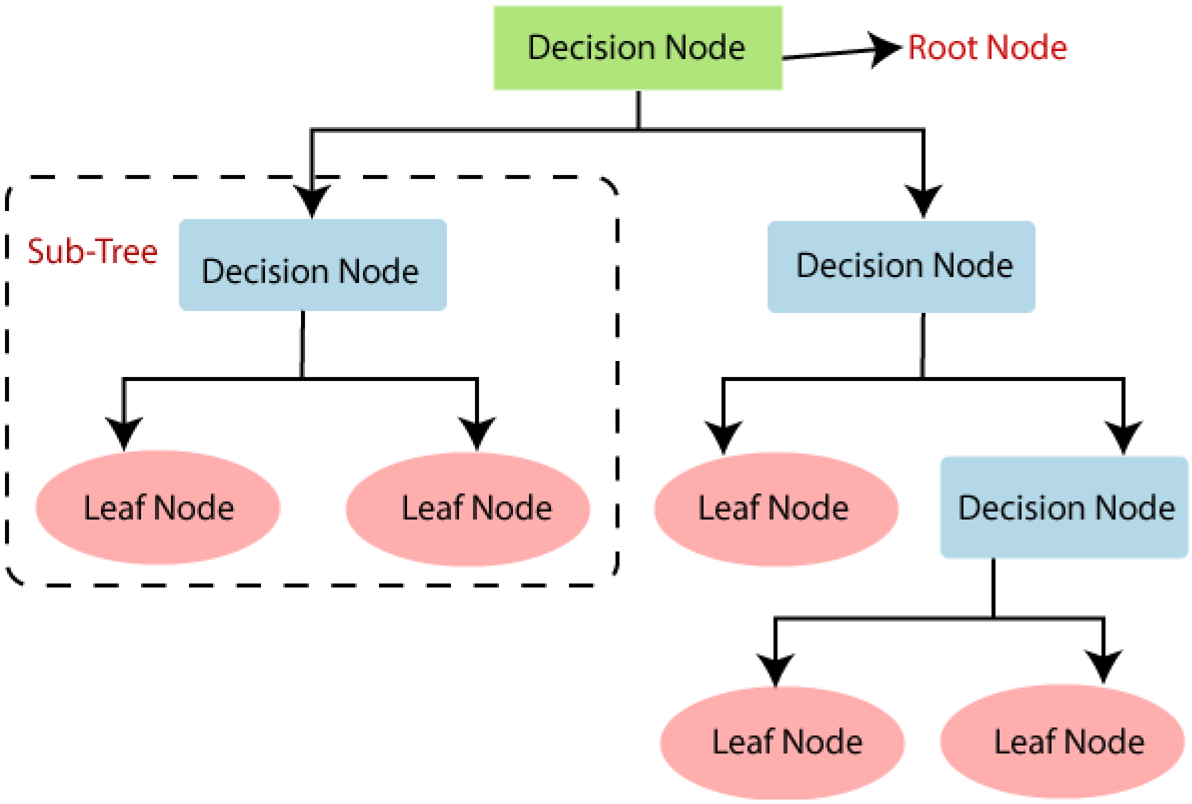
Content List

Introduction

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

The decisions or the test are performed on the basis of features of the given dataset. It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions. It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.



A Decision tree is mainly used over datasets for two reasons. Namely, Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand. The logic behind the decision tree can be easily understood because it shows a tree-like structure. The Following decision tree algorithm “decides” whether a customer will receive a loan from the bank or not.

Source Code

1. Importing Libraries

Input	import numpy as np import matplotlib.pyplot as plt import seaborn as sea import pandas as pd

2. Importing the Dataset

Input	data = pd.read_csv("Loan_Data.csv") data.head()

Output

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0

3. Understanding the Data

Input	data.columns

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

Input	data.dtypes
Output	Loan_ID object Gender object Married object Dependents object Education object Self_Employed object ApplicantIncome int64 CoapplicantIncome float64 LoanAmount float64 Loan_Amount_Term float64 Credit_History float64 Property_Area object Loan_Status object dtype: object
Input	data.shape
Output	(614, 13)

Input	data.isnull().sum()
Output	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 • There are missing values in Gender, Married, Dependents, Self_Employed, LoanAmount, Loan_Amount_Term and Credit_History features. • For numerical variables: imputation using mean or median • For categorical variables: imputation using mode
Input	data.head(2)

Output

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0

Input	data['Gender'].fillna(data['Gender'].mode()[0], inplace=True) data['Married'].fillna(data['Married'].mode()[0], inplace=True) data['Dependents'].fillna(data['Dependents'].mode()[0], inplace=True) data['Self_Employed'].fillna(data['Self_Employed'].mode()[0], inplace=True) data['Credit_History'].fillna(data['Credit_History'].mode()[0], inplace=True) data['Loan_Amount_Term'].value_counts()
Output	360.0 512 180.0 44 480.0 15 300.0 13 84.0 4 240.0 4 120.0 3 36.0 2 60.0 2 12.0 1 Name: Loan_Amount_Term, dtype: int64
Input	data['Loan_Amount_Term'].fillna(data['Loan_Amount_Term'].mode()[0], inplace=True) data['LoanAmount'].fillna(data['LoanAmount'].median(), inplace=True)

Input	data.isnull().sum()
Output	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
Input	data.head(2)

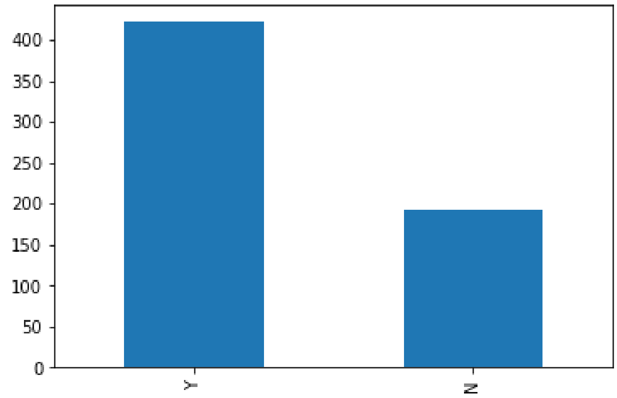
Output

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0

4. Univariate Analysis

Input	data['Loan_Status'].value_counts()
Output	Y 422 N 192 Name: Loan_Status, dtype: int64
Input	data['Loan_Status'].value_counts(normalize=True)
Output	Y 0.687296 N 0.312704 Name: Loan_Status, dtype: float64
Input	data['Loan_Status'].value_counts().plot.bar()

Output

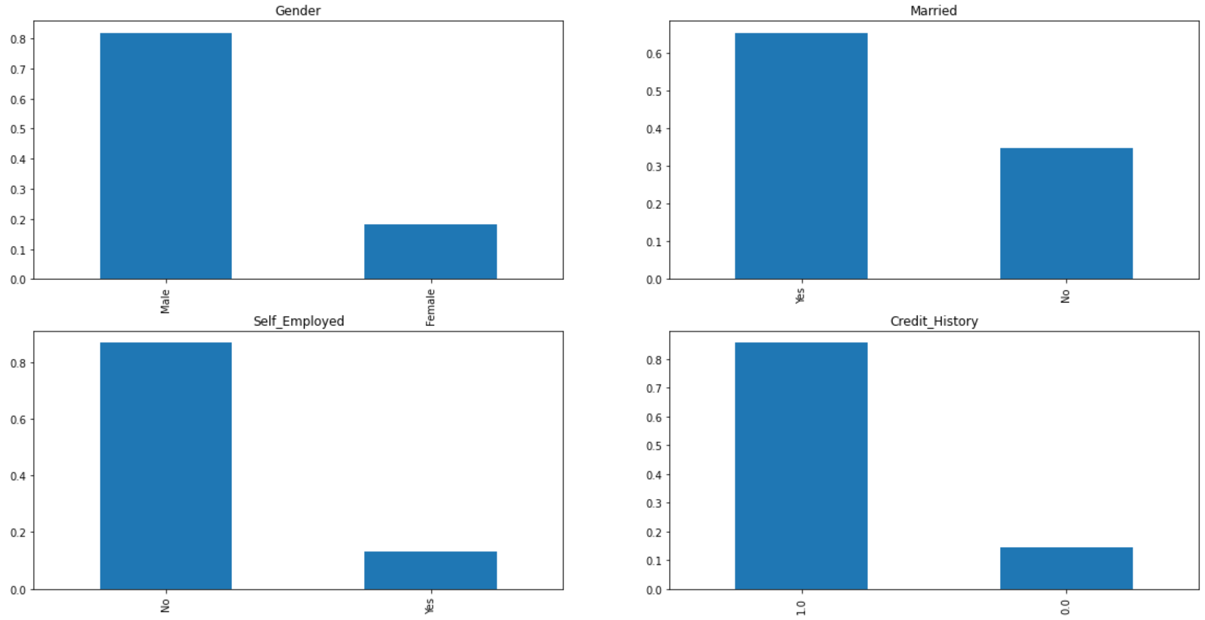


- The loan of 422 people out of 614 was approved
- The approval rate is around 68.73%

- Categorical Variables

Input	plt.figure(1) plt.subplot(221) data['Gender'].value_counts(normalize=True).plot.bar(figsize=(20,10), title= 'Gender') plt.subplot(222) data['Married'].value_counts(normalize=True).plot.bar(title= 'Married') plt.subplot(223) data['Self_Employed'].value_counts(normalize=True).plot.bar(title= 'Self_Employed') plt.subplot(224) data['Credit_History'].value_counts(normalize=True).plot.bar(title= 'Credit_History') plt.show()

Output



It can be inferred from the above bar plots that:

- 81.76% applicants in the dataset are male.
- Around 65.31% of the applicants in the dataset are married.
- Around 13.36% applicants in the dataset are self employed.
- Around 85.5% applicants have repaid their debts.

Input	data['Gender'].value_counts()

Output	`Male      502`
	`Female    112`
	`Name: Gender, dtype: int64`

Input	data['Married'].value_counts()
Output	Yes 401 No 213 Name: Married, dtype: int64

Input	data['Married'].value_counts()
Output	Yes 401 No 213 Name: Married, dtype: int64

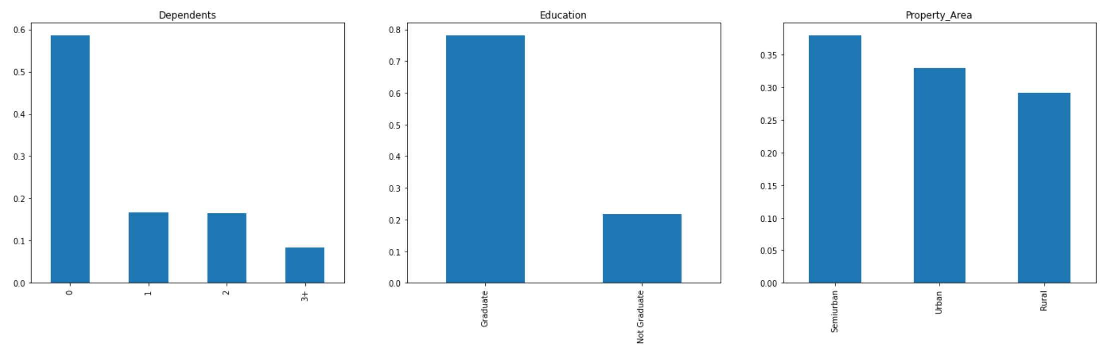
Input	data['Self_Employed'].value_counts()
Output	No 532 Yes 82 Name: Self_Employed, dtype: int64

Input	data['Credit_History'].value_counts()
Output	1.0 525 0.0 89 Name: Credit_History, dtype: int64

- Ordinal Variables

Input	plt.figure(1) plt.subplot(131) data['Dependents'].value_counts(normalize=True).plot.bar(figsize=(24,6), title= 'Dependents') plt.subplot(132) data['Education'].value_counts(normalize=True).plot.bar(title= 'Education') plt.subplot(133) data['Property_Area'].value_counts(normalize=True).plot.bar(title= 'Property_Area') plt.show()

Output



Following inferences can be made from the above bar plots:

- 58.63% of the applicants don't have any dependents.
- 78.18% of the applicants are Graduate.
- Majority (37.95%) of the applicants are from Semiurban area.

Input	data['Dependents'].value_counts()
Output	0 360 1 102 2 101 3+ 51 Name: Dependents, dtype: int64

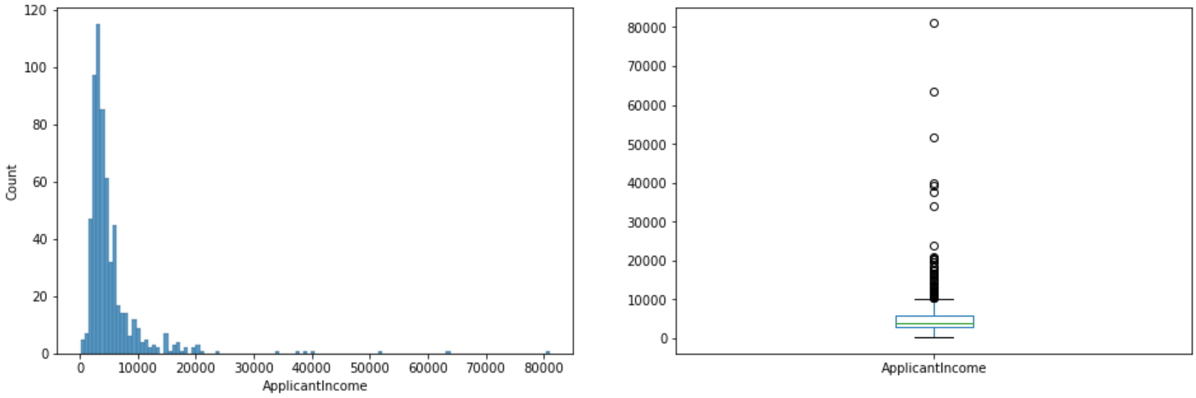
Input	data['Education'].value_counts()
Output	Graduate 480 Not Graduate 134 Name: Education, dtype: int64

Input	data['Property_Area'].value_counts()
Output	Semiurban 233 Urban 202 Rural 179 Name: Property_Area, dtype: int64

- Numericals Variables

Input	plt.figure(1) plt.subplot(121) sea.histplot(data['ApplicantIncome']); plt.subplot(122) data['ApplicantIncome'].plot.box(figsize=(16,5)) plt.show()

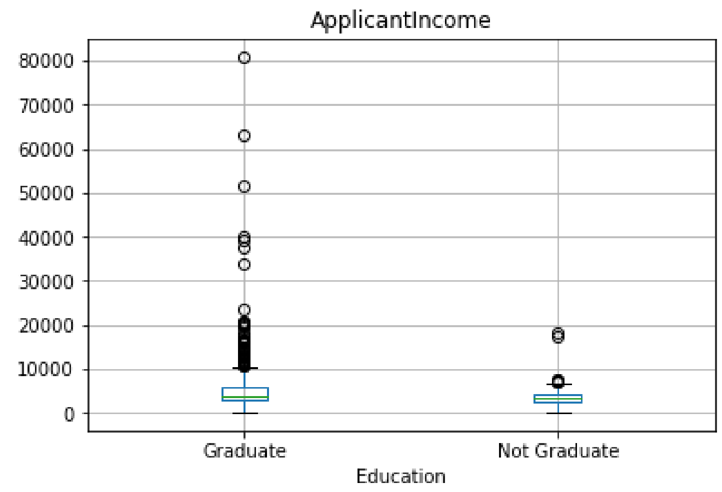
Output



- It can be inferred that most of the data in the distribution of applicant income is towards left which means it is not normally distributed.
- The boxplot confirms the presence of a lot of outliers/extreme values. This can be attributed to the income disparity in the society

Input	data.boxplot(column='ApplicantIncome', by = 'Education') plt.suptitle("")

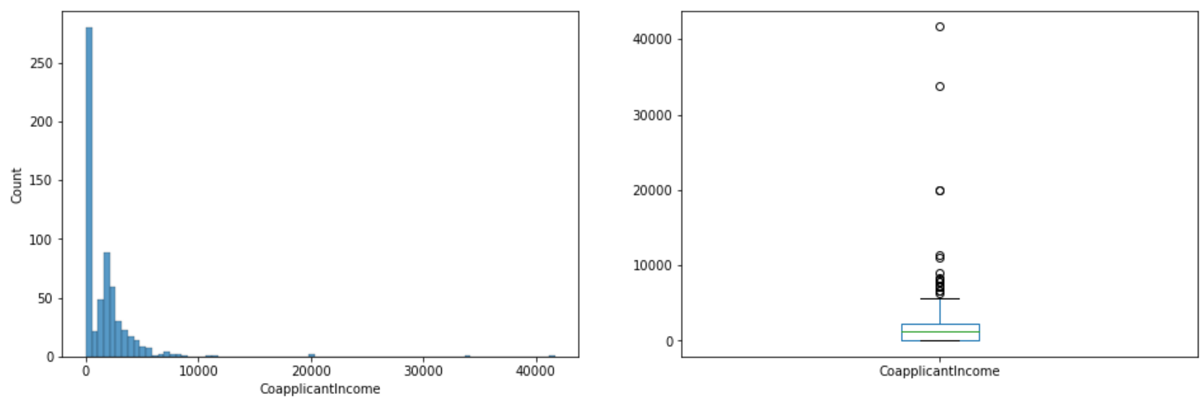
Output



We can see that there are a higher number of graduates with very high incomes, which are appearing to be the outliers.

Input	plt.figure(1) plt.subplot(121) sea.histplot(data['CoapplicantIncome']); plt.subplot(122) data['CoapplicantIncome'].plot.box(figsize=(16,5)) plt.show()

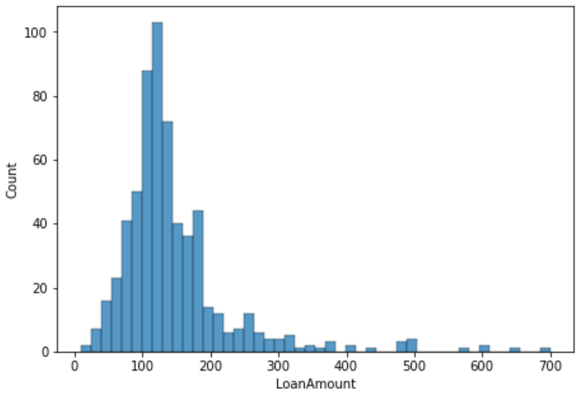
Output

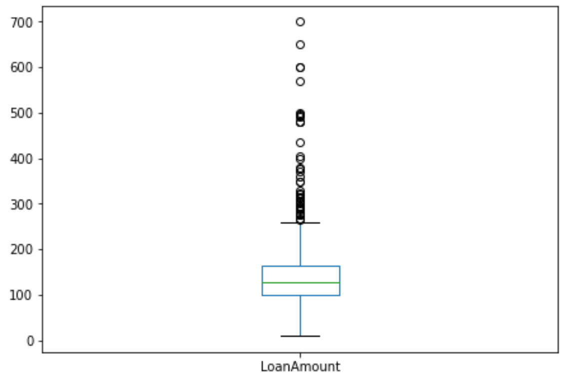


- Majority of coapplicant's income ranges from 0 to 5000.
- We also see a lot of outliers in the coapplicant income and it is not normally distributed.

Input	plt.figure(1) plt.subplot(121) df=data.dropna() sea.histplot(df['LoanAmount']); plt.subplot(122) df['LoanAmount'].plot.box(figsize=(16,5)) plt.show()

Output





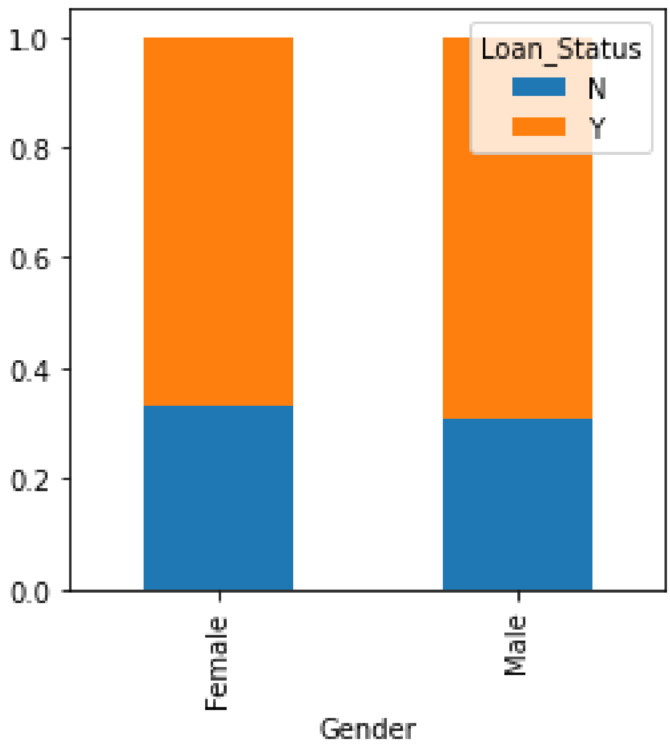
- Outliers are present
- the distribution is fairly normal.

5. Bivariate Analysis

- Categorical Variable vs Target Variable

Input	Gender=pd.crosstab(data['Gender'],data['Loan_Status']) Gender.div(Gender.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True, figsize=(4,4))

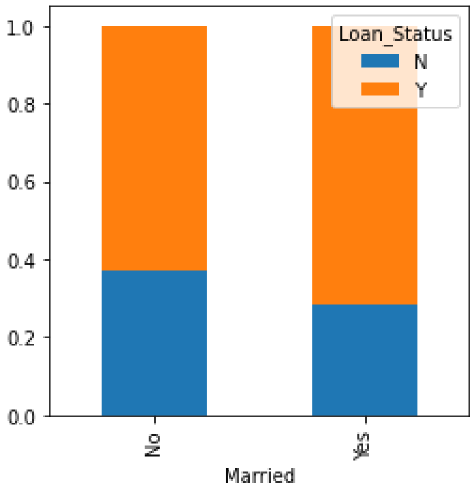
Output

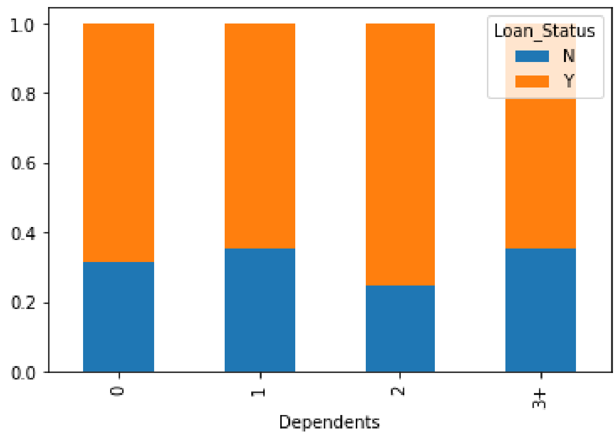


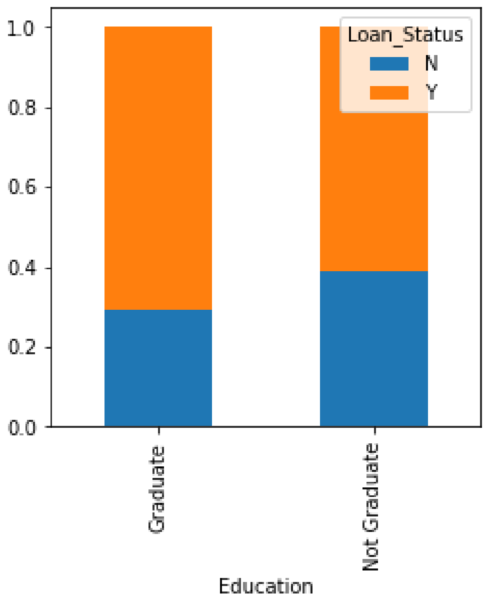
It can be inferred that the proportion of male and female applicants is more or less same for both approved and unapproved loans.

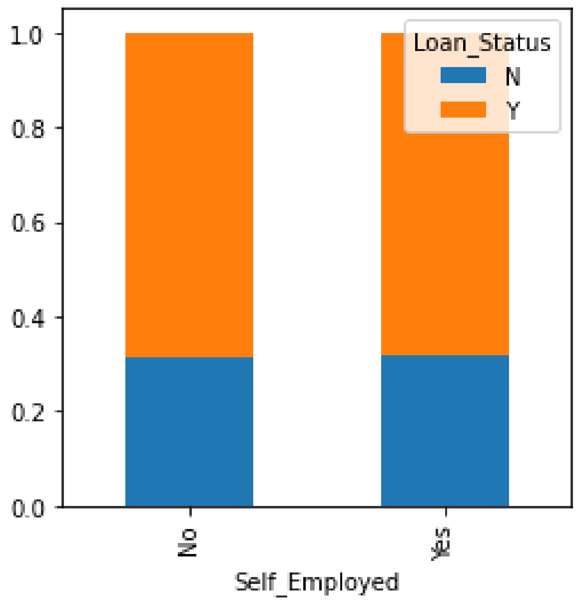
Input	Married=pd.crosstab(data['Married'],data['Loan_Status']) Dependents=pd.crosstab(data['Dependents'],data['Loan_Status']) Education=pd.crosstab(data['Education'],data['Loan_Status']) Self_Employed=pd.crosstab(data['Self_Employed'],data['Loan_Status']) Married.div(Married.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True, figsize=(4,4)) plt.show() Dependents.div(Dependents.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True) plt.show() Education.div(Education.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True, figsize=(4,4)) plt.show() Self_Employed.div(Self_Employed.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True, figsize=(4,4)) plt.show()

Output





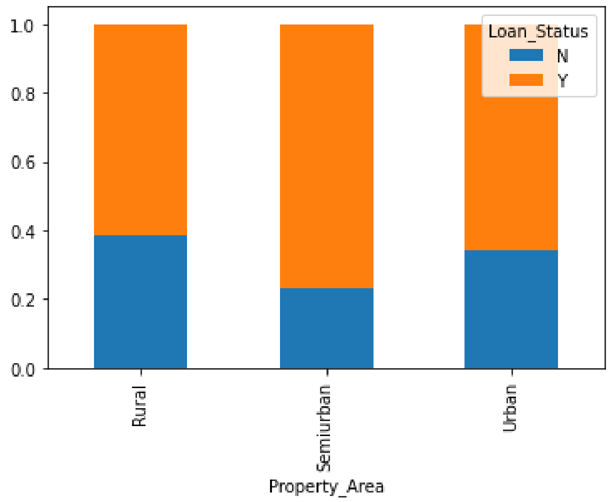




- Proportion of married applicants is higher for the approved loans.
- Distribution of applicants with 1 or 3+ dependents is similar across both the categories of Loan_Status.
- There is nothing significant we can infer from Self_Employed vs Loan_Status plot.

Input	Credit_History=pd.crosstab(data['Credit_History'],data['Loan_Status']) Property_Area=pd.crosstab(data['Property_Area'],data['Loan_Status']) Credit_History.div(Credit_History.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True, figsize=(4,4)) plt.show() Property_Area.div(Property_Area.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True) plt.show()

Output

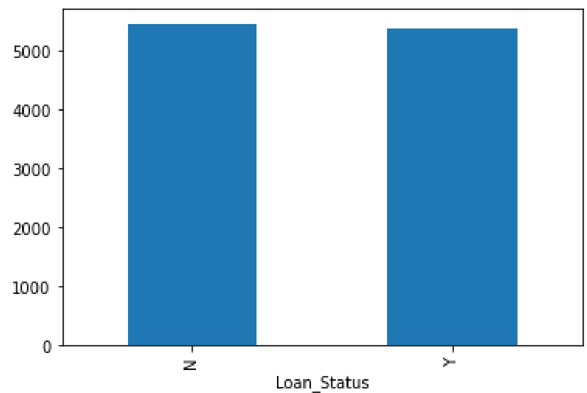


- It seems people with credit history as 1 are more likely to get their loans approved.
- Proportion of loans getting approved in semiurban area is higher as compared to that in rural or urban areas.

- Numerical Variable vs Target Variable

Input	data.groupby('Loan_Status')['ApplicantIncome'].mean().plot.bar()

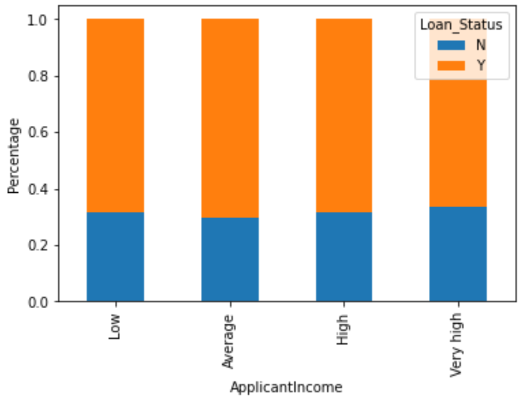
Output



There is not visible difference between the mean income of people for which the loan has been approved vs the mean income of people for which the loan has not been approved.

Input	bins=[0,2500,4000,6000,81000] group=['Low','Average','High', 'Very high'] data['Income_bin']=pd.cut(data['ApplicantIncome'],bins,labels=group) Income_bin=pd.crosstab(data['Income_bin'],data['Loan_Status']) Income_bin.div(Income_bin.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True) plt.xlabel('ApplicantIncome') P = plt.ylabel('Percentage')

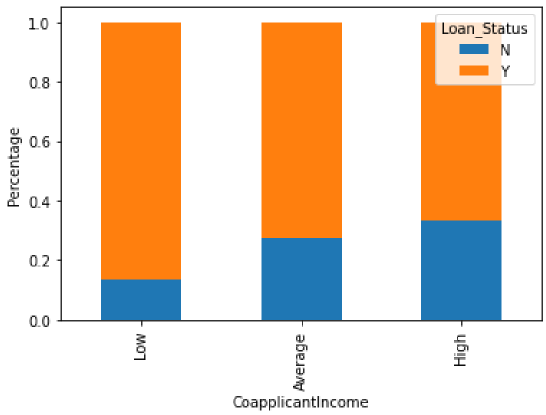
Output



Analysing bins for the applicant income variable based on the values in it and the corresponding loan status for each bin.It can be inferred that Applicant income does not affect the chances of loan approval

Input	bins=[0,1000,3000,42000] group=['Low','Average','High'] data['Coapplicant_Income_bin']=pd.cut(data['CoapplicantIncome'], bins,labels=group) Coapplicant_Income_bin=pd.crosstab(data['Coapplicant_Income_bin'], data['Loan_Status']) Coapplicant_Income_bin.div(Coapplicant_Income_bin.sum(1). astype(float), axis=0).plot(kind="bar", stacked=True) plt.xlabel('CoapplicantIncome') P = plt.ylabel('Percentage')

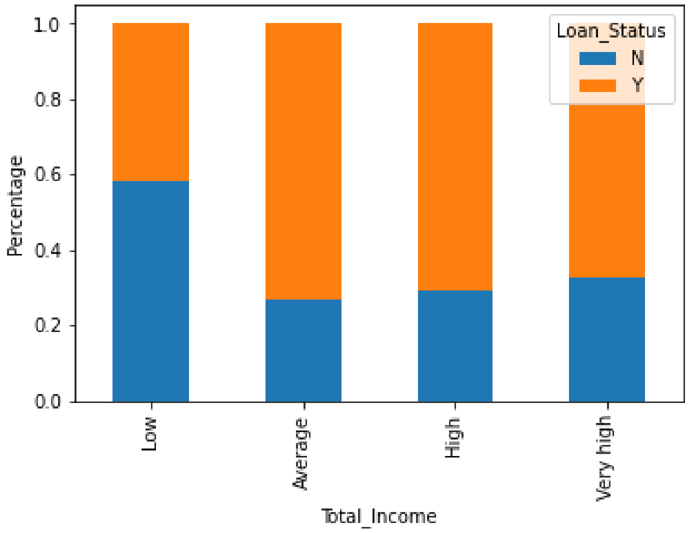
Output



coapplicant's income is less the chances of loan approval are high

Input	data['Total_Income']=data['ApplicantIncome']+data['CoapplicantIncome'] bins=[0,2500,4000,6000,81000] group=['Low','Average','High', 'Very high'] data['Total_Income_bin']=pd.cut(data['Total_Income'],bins,labels=group) Total_Income_bin=pd.crosstab(data['Total_Income_bin'],data['Loan_Status']) Total_Income_bin.div(Total_Income_bin.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True) plt.xlabel('Total_Income') P = plt.ylabel('Percentage')

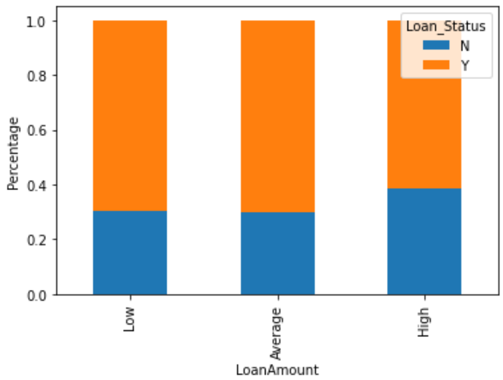
Output



Proportion of loans getting approved for applicants having low Total_Income is very less as compared to that of applicants with Average, High and Very High Income.

Input	bins=[0,100,200,700] group=['Low','Average','High'] data['LoanAmount_bin']=pd.cut(data['LoanAmount'],bins,labels=group) LoanAmount_bin=pd.crosstab(data['LoanAmount_bin'],data['Loan_Status']) LoanAmount_bin.div(LoanAmount_bin.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True) plt.xlabel('LoanAmount') P = plt.ylabel('Percentage')

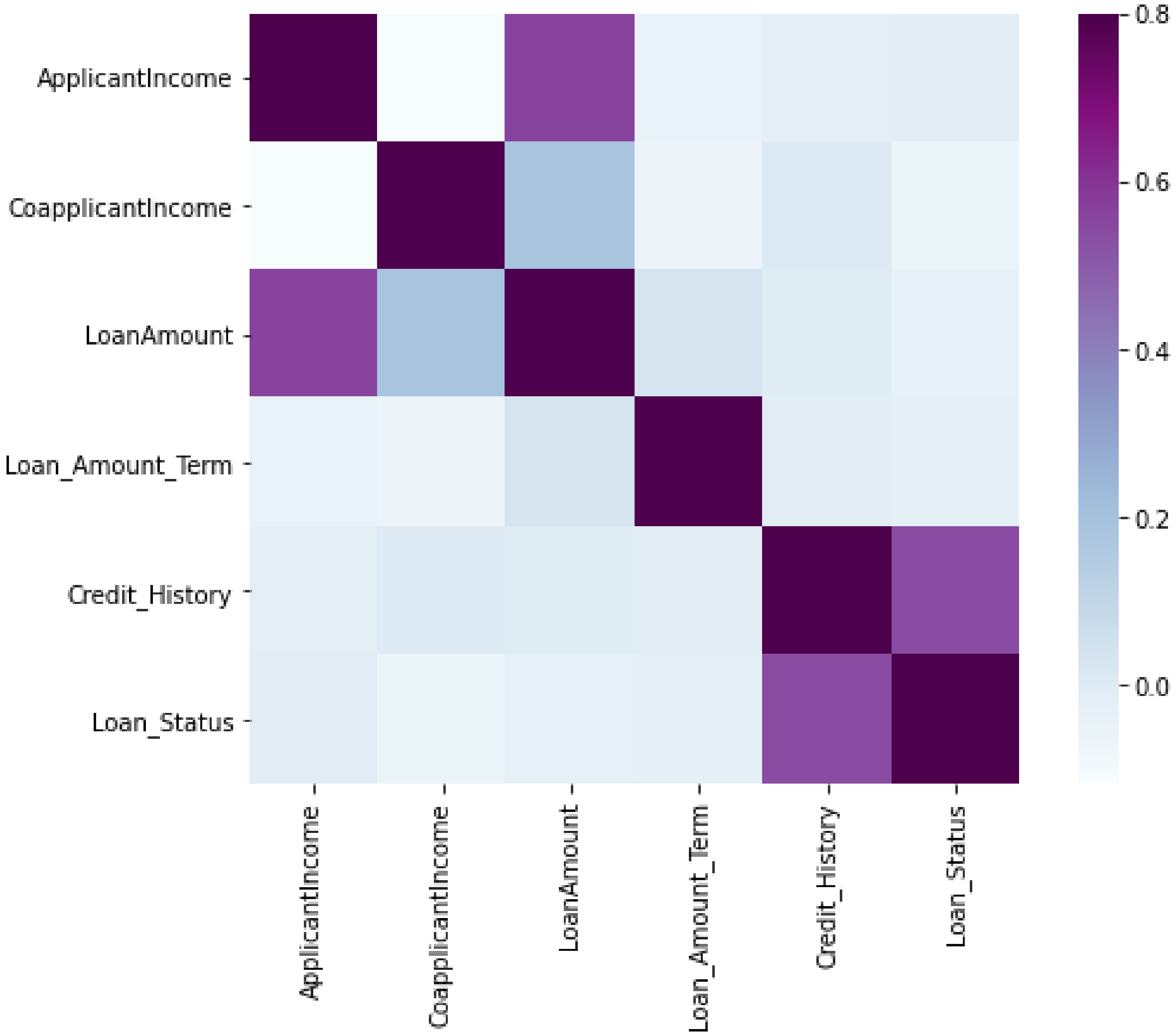
Output



the proportion of approved loans is **higher** for **Low and Average Loan Amount** as compared to that of High Loan Amount

Input	data=data.drop(['Income_bin', 'Coapplicant_Income_bin', 'LoanAmount_bin', 'Total_Income_bin', 'Total_Income'], axis=1) data['Dependents'].replace('3+', 3,inplace=True) data['Loan_Status'].replace('N', 0,inplace=True) data['Loan_Status'].replace('Y', 1,inplace=True) matrix = data.corr() f, ax = plt.subplots(figsize=(9, 6)) sea.heatmap(matrix, vmax=.8, square=True, cmap="BuPu")

Output

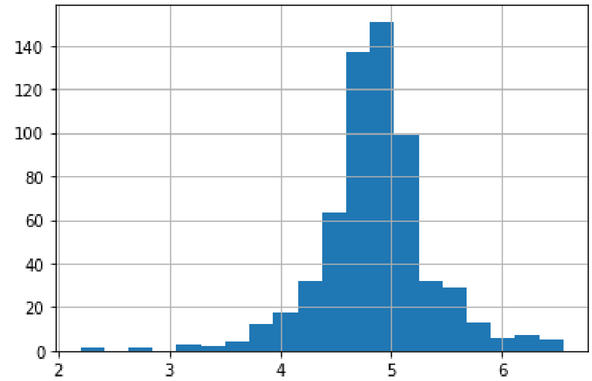


- Highly correlated variables are (ApplicantIncome - LoanAmount)
- Also (Credit_History - Loan_Status). is higly correlated
- LoanAmount is also correlated with CoapplicantIncome.

6. Feature Engineering

Input	data['LoanAmount_log'] = np.log(data['LoanAmount']) data['LoanAmount_log'].hist(bins=20)

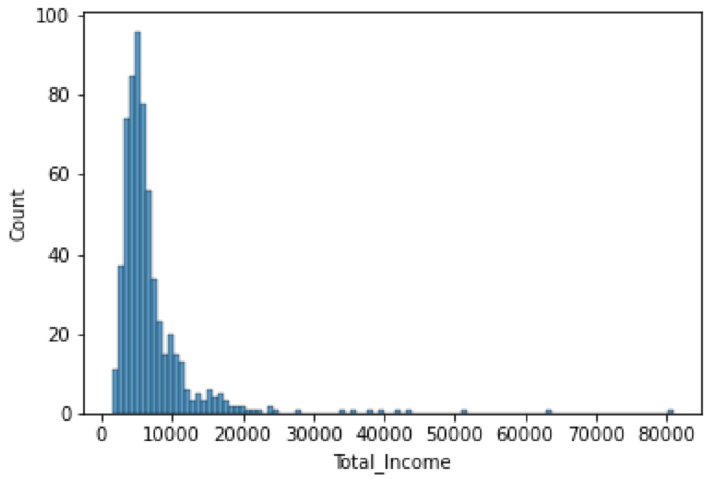
Output



the distribution looks much closer to normal and effect of extreme values has been significantly subsided.

Input	data['Total_Income']=data['ApplicantIncome']+data['CoapplicantIncome'] sea.histplot(data['Total_Income']);

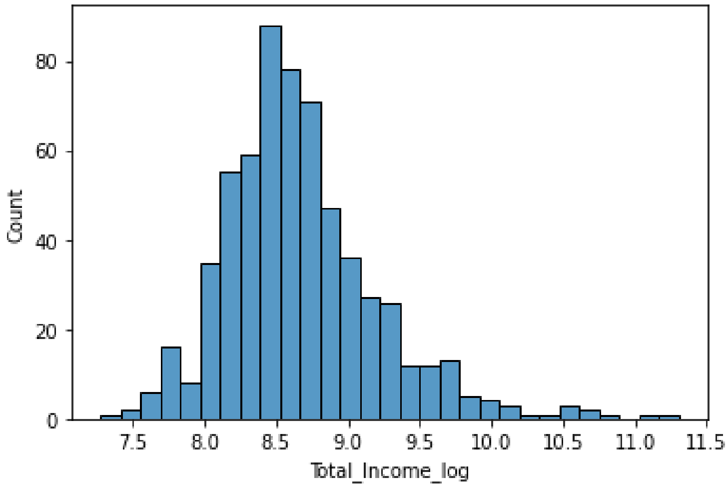
Output



Distribution is shifted towards left, i.e., the distribution is right skewed.

Input	data['Total_Income_log'] = np.log(data['Total_Income']) sea.histplot(data['Total_Income_log']);

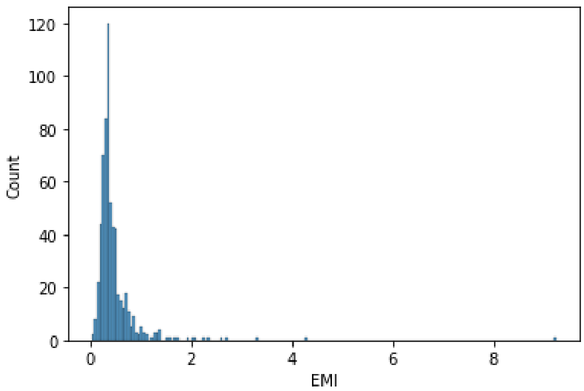
Output



- After taking the log transformation to make the distribution normal.
- Now the distribution looks much closer to normal

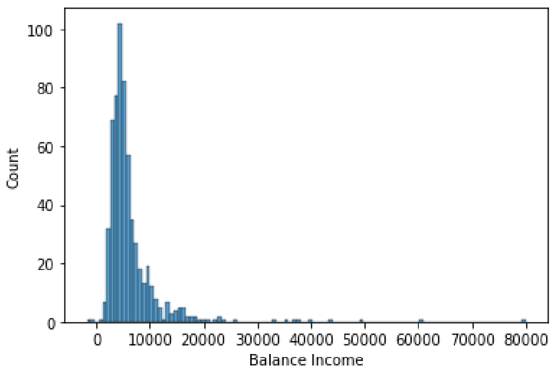
Input	data['EMI']=data['LoanAmount']/data['Loan_Amount_Term'] sea.histplot(data['EMI']);

Output



Input	data['Balance Income']=data['Total_Income']-(data['EMI']*1000) sea.histplot(data['Balance Income']);

Output



Input	data=data.drop(['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term'], axis=1)

- drop the variables which we used to create these new features.
- because the correlation between those old features and these new features will be very high
- removing correlated features will help in reducing the noise too.

Input	data.head()

Output

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Credit_History	Property_Area	Loan_Status	LoanAmount_log	Total_Income	Total_Incoı
0	LP001002	Male	No	0	Graduate	No	1.0	Urban	1	4.852030	5849.0	8.
1	LP001003	Male	Yes	1	Graduate	No	1.0	Rural	0	4.852030	6091.0	8.
2	LP001005	Male	Yes	0	Graduate	Yes	1.0	Urban	1	4.189655	3000.0	8.
3	LP001006	Male	Yes	0	Not Graduate	No	1.0	Urban	1	4.787492	4941.0	8.
4	LP001008	Male	No	0	Graduate	No	1.0	Urban	1	4.948760	6000.0	8.

7. Building the Model

Input	data=data.drop('Loan_ID',axis=1)

Loan Id is not a significant variable and it is not required as a feature for building model

Input	X = data.drop('Loan_Status',1) y = data.Loan_Status

Loan Status is target variabel so seggregating it

Input	X=pd.get_dummies(X) data=pd.get_dummies(data)

Dummy variables for Categorical Variable so each category can be given as a seperate feature to the model

Input

```
from sklearn import tree
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score
import graphviz
i=1
kf = StratifiedKFold(n_splits=5,random_state=3,shuffle=True)
accuracy_list = []
for train_index,test_index in kf.split(X,y):
    print('\n{} of kfold {}'.format(i,kf.n_splits))
    xtr,xvl = X.loc[train_index],X.loc[test_index]
    ytr,yvl = y[train_index],y[test_index]
    model = tree.DecisionTreeClassifier(random_state=1)
```

	model.fit(xtr, ytr) pred_test = model.predict(xvl) score = accuracy_score(yvl,pred_test) accuracy_list.append(score) print('accuracy_score',score) i+=1
Output	1 of kfold 5 accuracy_score 0.6829268292682927 2 of kfold 5 accuracy_score 0.6422764227642277 3 of kfold 5 accuracy_score 0.7642276422764228 4 of kfold 5 accuracy_score 0.7479674796747967 5 of kfold 5 accuracy_score 0.6721311475409836
Input	mean_accuracy = sum(accuracy_list)/ len(accuracy_list) print(mean_accuracy)
Output	0.7019059043049447

Mean Accuracy for the model is around 0.70

Input	data.info()
Output	<class 'pandas.core.frame.DataFrame'> RangeIndex: 614 entries, 0 to 613 Data columns (total 22 columns): # Column Non-Null Count Dtype --- ------ -------------- ----- 0 Credit_History 614 non-null float64 1 Loan_Status 614 non-null int64 2 LoanAmount_log 614 non-null float64 3 Total_Income 614 non-null float64 4 Total_Income_log 614 non-null float64 5 EMI 614 non-null float64 6 Balance Income 614 non-null float64 7 Gender_Female 614 non-null uint8 8 Gender_Male 614 non-null uint8 9 Married_No 614 non-null uint8 10 Married_Yes 614 non-null uint8 11 Dependents_3 614 non-null uint8 12 Dependents_0 614 non-null uint8 13 Dependents_1 614 non-null uint8 14 Dependents_2 614 non-null uint8 15 Education_Graduate 614 non-null uint8 16 Education_Not Graduate 614 non-null uint8 17 Self_Employed_No 614 non-null uint8 18 Self_Employed_Yes 614 non-null uint8 19 Property_Area_Rural 614 non-null uint8 20 Property_Area_Semiurban 614 non-null uint8 21 Property_Area_Urban 614 non-null uint8 dtypes: float64(6), int64(1), uint8(15) memory usage: 42.7 KB

Input	X.head()

Output

	Credit_History	LoanAmount_log	Total_Income	Total_Income_log	EMI	Balance Income	Gender_Female	Gender_Male	Married_No	Married_Yes	...	Depen
0	1.0	4.852030	5849.0	8.674026	0.355556	5493.444444	0	1	1	0	...	
1	1.0	4.852030	6091.0	8.714568	0.355556	5735.444444	0	1	0	1	...	
2	1.0	4.189655	3000.0	8.006368	0.183333	2816.666667	0	1	0	1	...	
3	1.0	4.787492	4941.0	8.505323	0.333333	4607.666667	0	1	0	1	...	
4	1.0	4.948760	6000.0	8.699515	0.391667	5608.333333	0	1	1	0	...	

5 rows × 21 columns

<table>
<tr><td>Input</td><td>y.head()</td></tr>
<tr><td>Output</td><td><pre>
0   1
1   0
2   1
3   1
4   1
Name: Loan_Status, dtype: int64
</pre></td></tr>
<tr><td>Input</td><td><pre>
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, y,
random_state=0)

clf = DecisionTreeClassifier(max_depth = 2, random_state = 0)
clf.fit(X_train, Y_train)
clf.predict(X_test)
</pre></td></tr>
<tr><td>Output</td><td><pre>
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, 0, 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, 1, 1, 0,
1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
0, 0, 1, 1, 1, 0],
      dtype=int64)
</pre></td></tr>
</table>

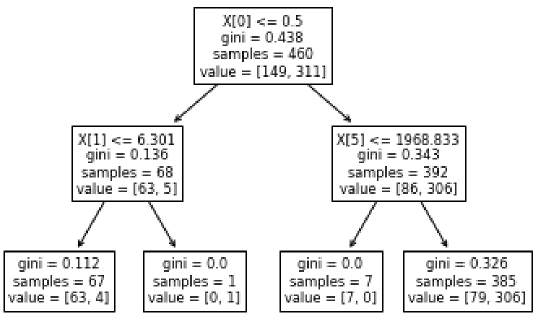
Output

Input	from sklearn import tree
	from sklearn.model_selection import StratifiedKFold
	from sklearn.metrics import accuracy_score
	import graphviz
	i=1
	kf = StratifiedKFold(n_splits=5,random_state=3,shuffle=True)
	accuracy_list = []
	for train_index,test_index in kf.split(X,y):
	    print('\n{} of kfold {}'.format(i,kf.n_splits))
	    xtr,xvl = X.loc[train_index],X.loc[test_index]
	    ytr,yvl = y[train_index],y[test_index]
	    model = tree.DecisionTreeClassifier(random_state=1)
	    model.fit(xtr, ytr)
	    pred_test = model.predict(xvl)
	    score = accuracy_score(yvl,pred_test)
	    accuracy_list.append(score)
	    print('accuracy_score',score)
	    i+=1

Output	1 of kfold 5 accuracy_score 0.6829268292682927 2 of kfold 5 accuracy_score 0.6422764227642277 3 of kfold 5 accuracy_score 0.7642276422764228 4 of kfold 5 accuracy_score 0.7479674796747967 5 of kfold 5 accuracy_score 0.6721311475409836
Input	mean_accuracy = sum(accuracy_list)/ len(accuracy_list) print(mean_accuracy)
Output	0.7019059043049447

Input	tree.plot_tree(clf);

Output



Inference

The given decision tree clearly shows the classification and probability of customers being granted a loan based on the various factors which affect the decision as seen by the nodes of the Decision tree.

The kfold method also determines how accurate the decision tree is based on the test data after working on the training data set. Accordingly the given output Decision Tree has a high accuracy of 70% thus justifying the accuracy of the given decision tree.