About The Data:

Given below is the description for each variable. Variable Description Loan_ID Unique Loan ID Gender Male/ Female Married Applicant married (Y/N) Dependents Number of dependents Education Applicant Education (Graduate/ Under Graduate) Self_Employed Self employed (Y/N) ApplicantIncome Applicant income CoapplicantIncome Coapplicant income LoanAmount Loan amount in thousands Loan_Amount_Term Term of loan in months Credit_History credit history meets guidelines Property_Area Urban/ Semi Urban/ Rural Loan_Status Loan approved (Y/N)

In [171]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [172]:

```
data = pd.read_csv("loan_data_set.csv")
data.head(10)
```

Out[172]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Ap
0	LP001002	Male	No	0	Graduate	No	
1	LP001003	Male	Yes	1	Graduate	No	
2	LP001005	Male	Yes	0	Graduate	Yes	
3	LP001006	Male	Yes	0	Not Graduate	No	
4	LP001008	Male	No	0	Graduate	No	
5	LP001011	Male	Yes	2	Graduate	Yes	
6	LP001013	Male	Yes	0	Not Graduate	No	
7	LP001014	Male	Yes	3+	Graduate	No	
8	LP001018	Male	Yes	2	Graduate	No	
9	LP001020	Male	Yes	1	Graduate	No	

In [173]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
Loan_ID              614 non-null object
Gender               601 non-null object
Married              611 non-null object
Dependents           599 non-null object
Education            614 non-null object
Self_Employed        582 non-null object
ApplicantIncome      614 non-null int64
CoapplicantIncome    614 non-null float64
LoanAmount           592 non-null float64
Loan_Amount_Term     600 non-null float64
Credit_History       564 non-null float64
Property_Area        614 non-null object
Loan_Status          614 non-null object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.4+ KB
```

In [174]:

```
data.isnull().sum().any()
```

Out[174]:

```
True
```

In [175]:

```
data.isnull().sum()
```

Out[175]:

```
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
```

From above table we can see that there are missing values in Gender, Dependants, Self_Employed, Loan Amount, Loan_Amount_term & Credit_History.

In [176]:

```
data.head(5)
```

Out[176]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Ap
0	LP001002	Male	No	0	Graduate	No	
1	LP001003	Male	Yes	1	Graduate	No	
2	LP001005	Male	Yes	0	Graduate	Yes	
3	LP001006	Male	Yes	0	Not Graduate	No	
4	LP001008	Male	No	0	Graduate	No	

In [177]:

```
data.Loan_Status.value_counts()
```

Out[177]:

```
Y    422
N    192
Name: Loan_Status, dtype: int64
```

In [178]:

```
sns.countplot(data['Loan_Status'],label = "Count")
plt.show()
```

Independant Features # Univariate analysis Below are the types of features available in this datasets: Categorical features(Nominal): These variables are categorical without order or ranking. Below are the list of such variables in this data: Gender, Married, Self_Employed, Credit_History Categorical features(Ordinal): These variables in categorical in nature having some order involved. Dependents, Education, Property_Area Numerical features(Ratio): These features have numerical values. ApplicantIncome, CoapplicantIncome, LoanAmount, Loan_Amount_Term

Analysis on "Gender" variable :

In [179]:

```
data["Gender"].count()
```

Out[179]:

```
601
```

In [180]:

```
data["Gender"].value_counts()
```

Out[180]:

```
Male      489
Female    112
Name: Gender, dtype: int64
```

In [181]:

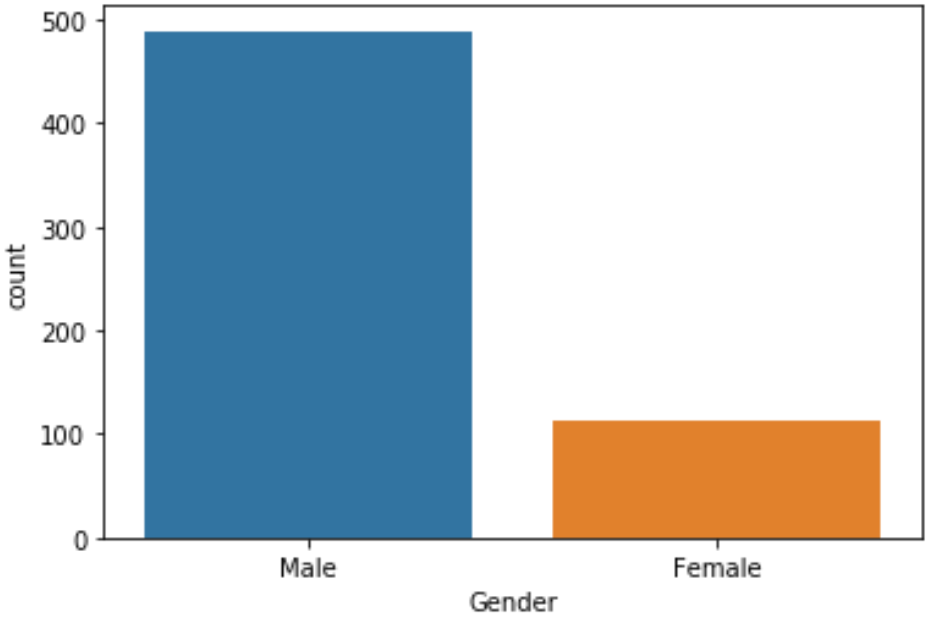
```
data['Gender'].value_counts(normalize=True)*100
```

Out[181]:

```
Male      81.364393
Female    18.635607
Name: Gender, dtype: float64
```

In [182]:

```
sns.countplot(data['Gender'],label = "Gender")
plt.show()
```



Analysis on "Married" variable :

In [183]:

```
data["Married"].count()
```

Out[183]:

```
611
```

In [184]:

```
data["Married"].value_counts()
```

Out[184]:

```
Yes    398
No     213
Name: Married, dtype: int64
```

In [185]:

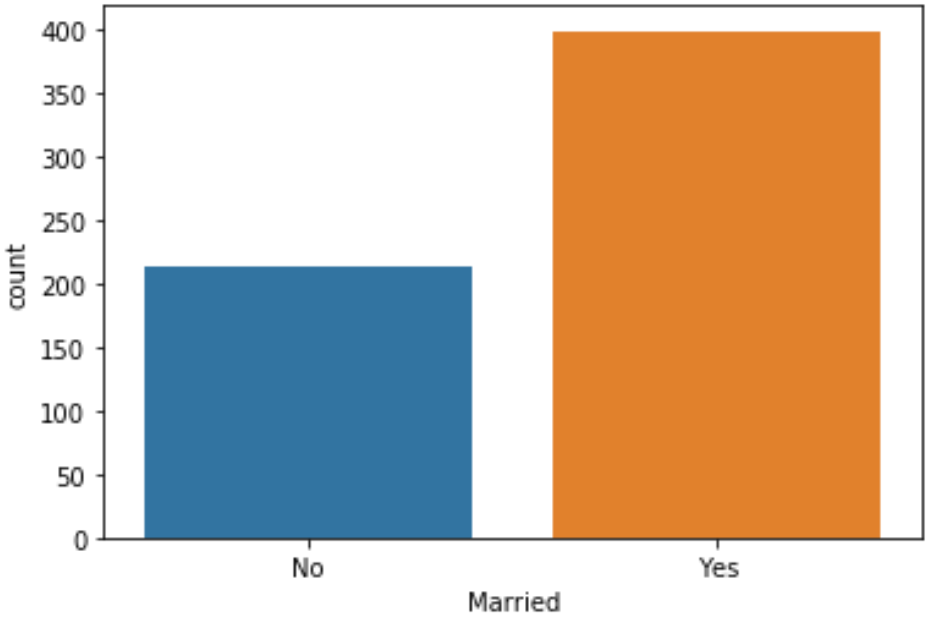
```
data["Married"].value_counts(normalize=True)*100
```

Out[185]:

```
Yes    65.139116
No     34.860884
Name: Married, dtype: float64
```

In [186]:

```
sns.countplot(data["Married"],label="Married Status")
plt.show()
```



Analysis on "Self_Employed" variable :

In [187]:

```
data["Self_Employed"].count()
```

Out[187]:

```
582
```

In [188]:

```
data["Self_Employed"].value_counts()
```

Out[188]:

```
No     500
Yes     82
Name: Self_Employed, dtype: int64
```

In [189]:

```
data["Self_Employed"].value_counts(normalize = True)*100
```

Out[189]:

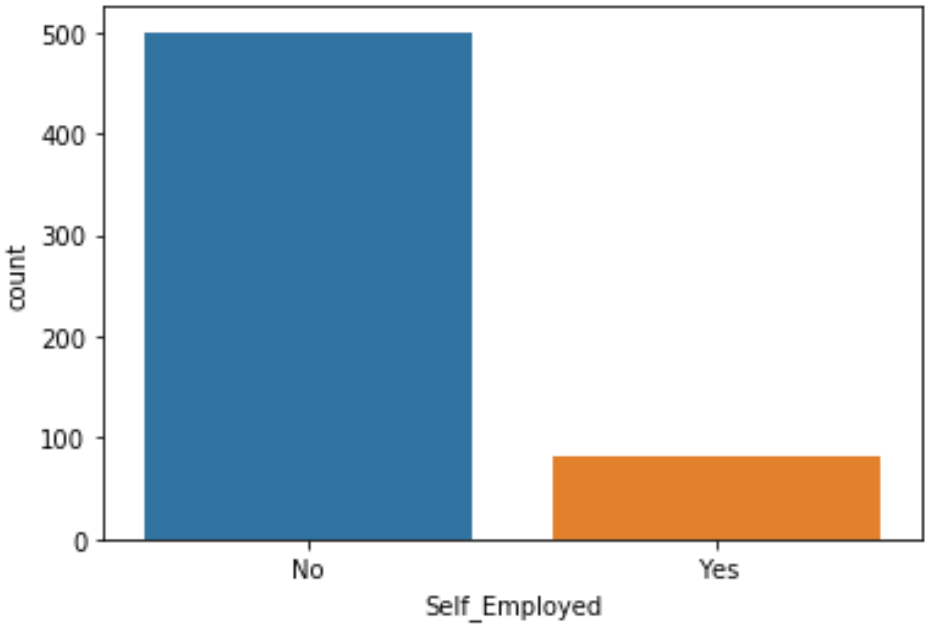
```
No     85.910653
Yes    14.089347
Name: Self_Employed, dtype: float64
```

In [190]:

```
sns.countplot(data["Self_Employed"],label = "Self_Employed")
```

Out[190]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe52f7063d0>
```



Analysis on "Credit_History" variable

In [191]:

```
data["Credit_History"].count()
```

Out[191]:

```
564
```

In [192]:

```
data["Credit_History"].value_counts()
```

Out[192]:

```
1.0    475
0.0     89
Name: Credit_History, dtype: int64
```

In [193]:

```
data["Credit_History"].value_counts(normalize = True)*100
```

Out[193]:

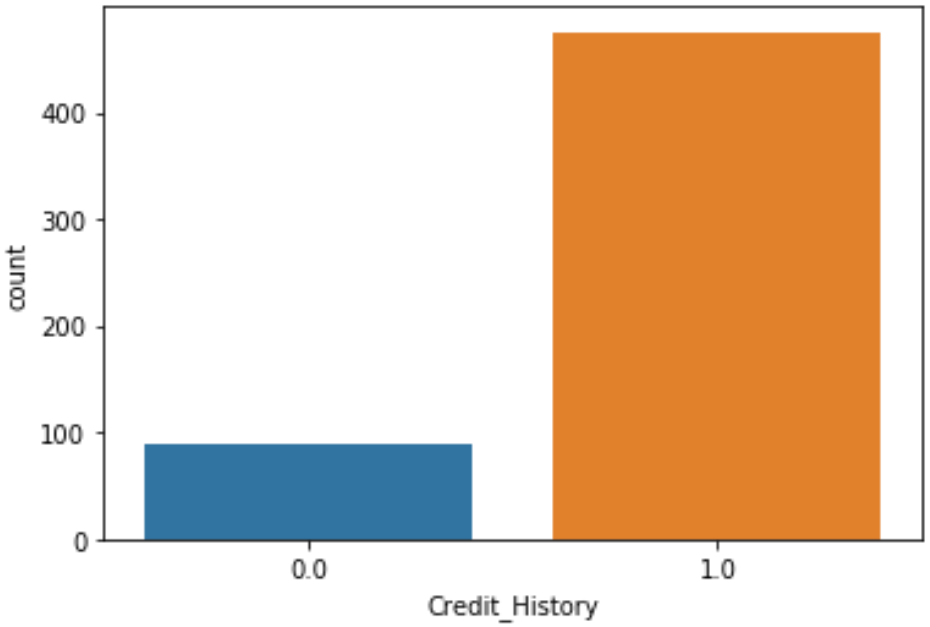
```
1.0    84.219858
0.0    15.780142
Name: Credit_History, dtype: float64
```

In [194]:

```
sns.countplot(data["Credit_History"],label = "Credit_History")
```

Out[194]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe52ef71f50>
```



Analysis on "Dependents" variable :

In [195]:

```
data["Dependents"].count()
```

Out[195]:

```
599
```

In [196]:

```
data["Dependents"].value_counts()
```

Out[196]:

```
0     345
1     102
2     101
3+     51
Name: Dependents, dtype: int64
```

In [197]:

```
data['Dependents'].value_counts(normalize=True)*100
```

Out[197]:

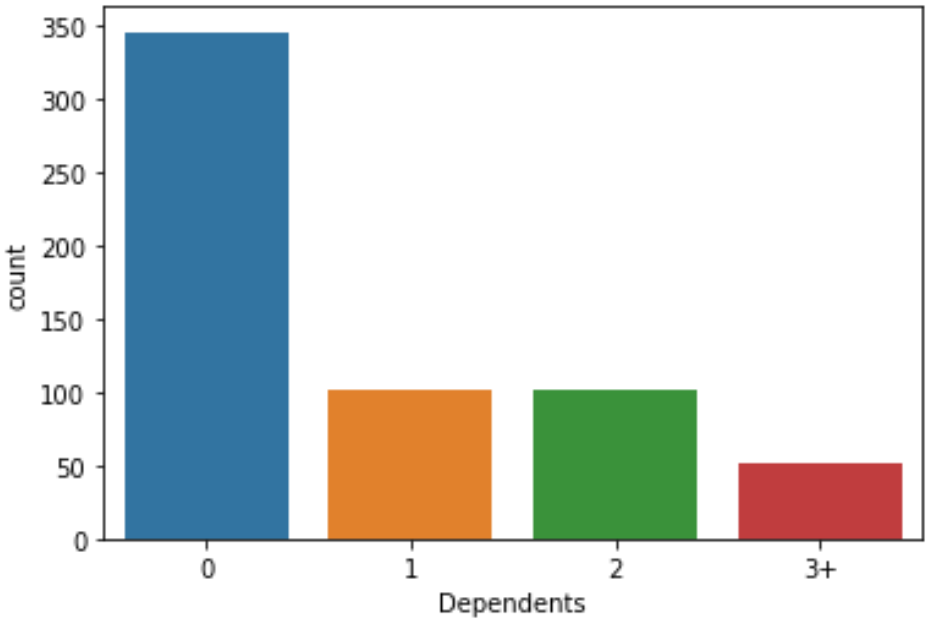
```
0     57.595993
1     17.028381
2     16.861436
3+     8.514190
Name: Dependents, dtype: float64
```

In [198]:

```
sns.countplot(data["Dependents"],label="Dependents")
```

Out[198]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe52ee9b2d0>
```



Analysis on "Education" variable :

In [199]:

```
data["Education"].count()
```

Out[199]:

```
614
```

In [200]:

```
data["Education"].value_counts()
```

Out[200]:

```
Graduate        480
Not Graduate    134
Name: Education, dtype: int64
```

In [201]:

```
data["Education"].value_counts(normalize=True)*100
```

Out[201]:

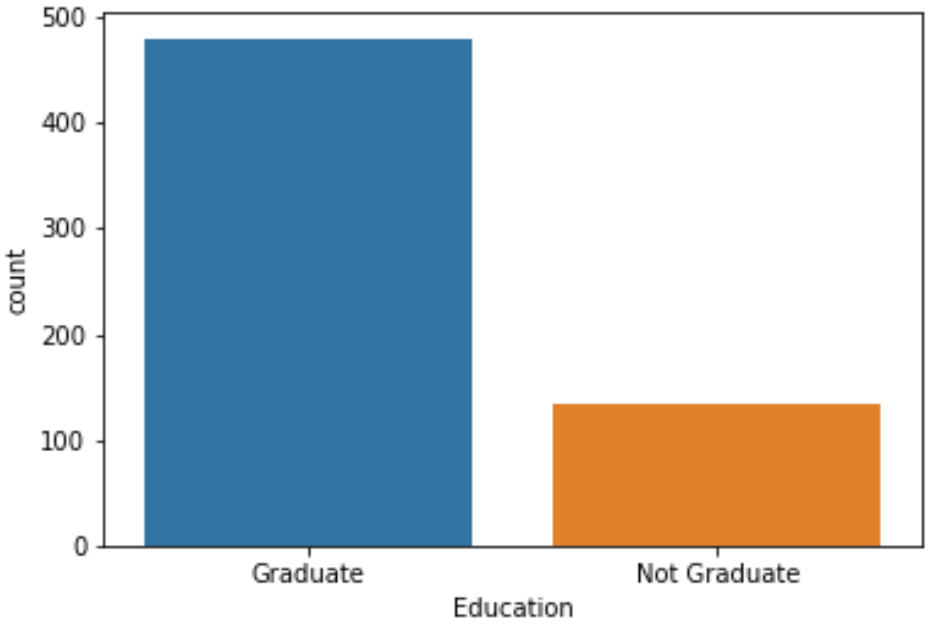
```
Graduate        78.175896
Not Graduate    21.824104
Name: Education, dtype: float64
```

In [202]:

```
sns.countplot(data["Education"],label ="Education")
```

Out[202]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe52f44e950>
```



Analysis on "Property_Area" variable

In [203]:

```
data["Property_Area"].count()
```

Out[203]:

```
614
```

In [204]:

```
data["Property_Area"].value_counts()
```

Out[204]:

```
Semiurban    233
Urban        202
Rural        179
Name: Property_Area, dtype: int64
```

In [205]:

```
data["Property_Area"].value_counts(normalize=True)*100
```

Out[205]:

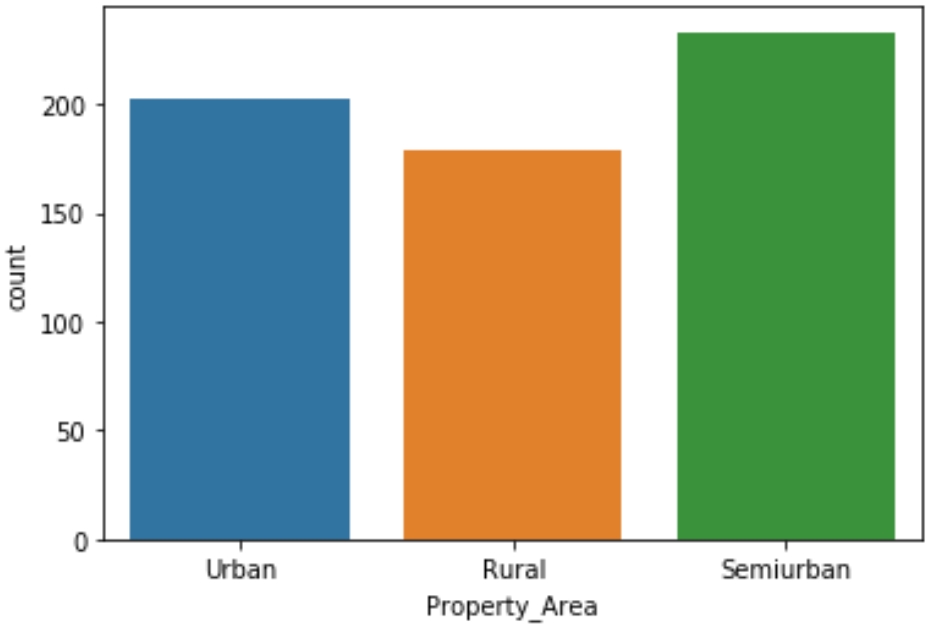
```
Semiurban    37.947883
Urban        32.899023
Rural        29.153094
Name: Property_Area, dtype: float64
```

In [206]:

```
sns.countplot(data["Property_Area"],label ="Property_Area")
```

Out[206]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe52f763510>
```



In [207]:

```
data.describe()
```

Out[207]:

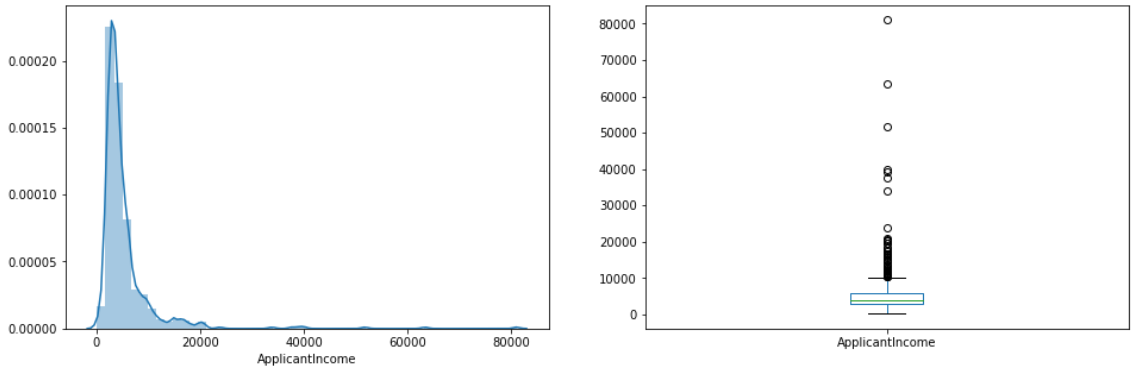
	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Te
count	614.000000	614.000000	592.000000	600.000
mean	5403.459283	1621.245798	146.412162	342.000
std	6109.041673	2926.248369	85.587325	65.120
min	150.000000	0.000000	9.000000	12.000
25%	2877.500000	0.000000	100.000000	360.000
50%	3812.500000	1188.500000	128.000000	360.000
75%	5795.000000	2297.250000	168.000000	360.000
max	81000.000000	41667.000000	700.000000	480.000

From above we can conclude following : Applicantincome, CoapplicantIncome, LoanAmount are left skweed as mean is larger than the median. Loan_Amount_Term is mildly normally distributed. We can't deduce nature of Credit_History as of now.

Distribution Plots

In [208]:

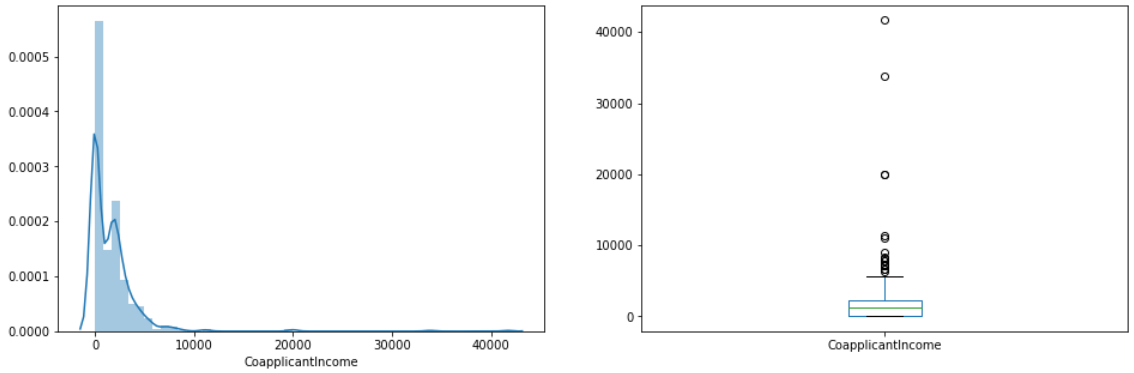
```
plt.figure(1)
plt.subplot(121)
sns.distplot(data["ApplicantIncome"])
plt.subplot(122)
data["ApplicantIncome"].plot.box(figsize=(16,5))
plt.show()
```



From the above plots, we can conclude that distribution is heavily skewed with most of the value is in the left side indicating some outlier values towards higher income side.

In [209]:

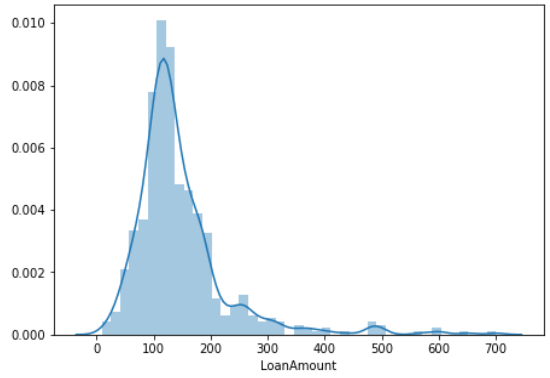
```
plt.figure(1)
plt.subplot(121)
sns.distplot(data["CoapplicantIncome"])
plt.subplot(122)
data["CoapplicantIncome"].plot.box(figsize=(16,5))
plt.show()
```

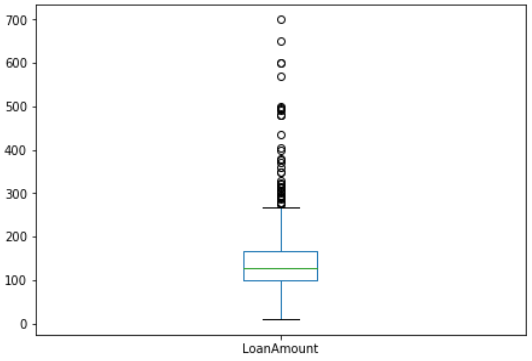


From the above plots, we can conclude that distribution is heavily skewed with most of the value is in the left side indicating some outlier values towards higher income side.

In [210]:

```
plt.figure(1)
plt.subplot(121)
sns.distplot(data["LoanAmount"].dropna())
plt.subplot(122)
data["LoanAmount"].dropna().plot.box(figsize=(16,5))
plt.show()
```





From the above plots, we can conclude that distribution is mildly normal one, however presence of some outlier at higher side is affecting the distribution.

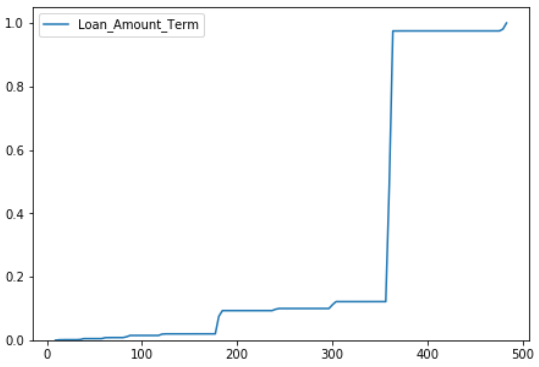
In [211]:

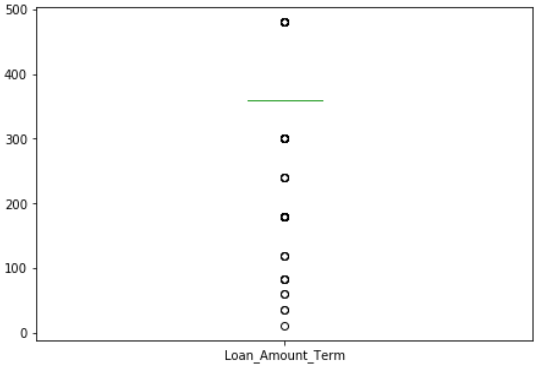
```
print(data["Loan_Amount_Term"].value_counts())
```

```
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
```

In [212]:

```
plt.figure(1)
plt.subplot(121)
sns.kdeplot(data["Loan_Amount_Term"].dropna(), cumulative=True, bw=1)
plt.subplot(122)
data["Loan_Amount_Term"].dropna().plot.box(figsize=(16,5))
plt.show()
```





From the above plots, we can conclude that distribution is bimodal with values centered around 380 and ~180. This indicates most of the loan term are of higher period.

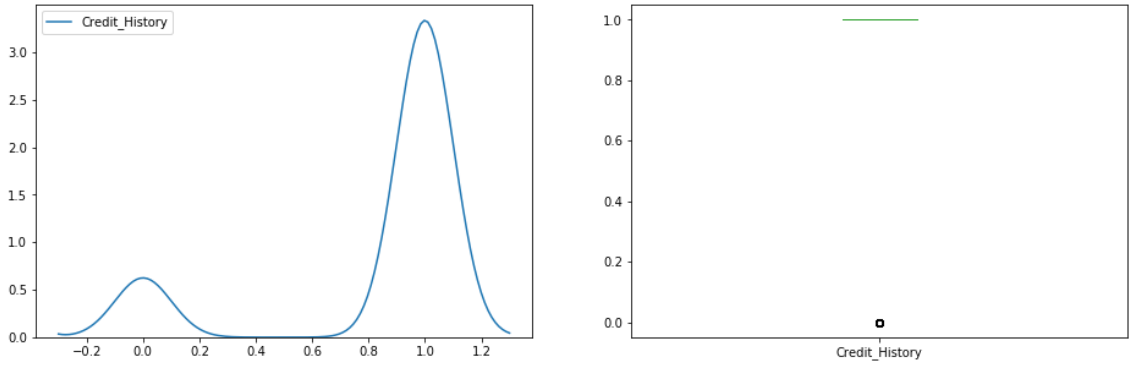
In [213]:

```
print(data["Credit_History"].value_counts())
```

```
1.0    475
0.0     89
Name: Credit_History, dtype: int64
```

In [214]:

```
plt.figure(1)
plt.subplot(121)
sns.kdeplot(data["Credit_History"].dropna(), bw=0.1)
plt.subplot(122)
data["Credit_History"].dropna().plot.box(figsize=(16,5))
plt.show()
```



Though this variable is integer while loading, nature of this variable is binary with most of loan applicant having credit history equal to 1.

Relation between "Loan_Status" and "Gender"

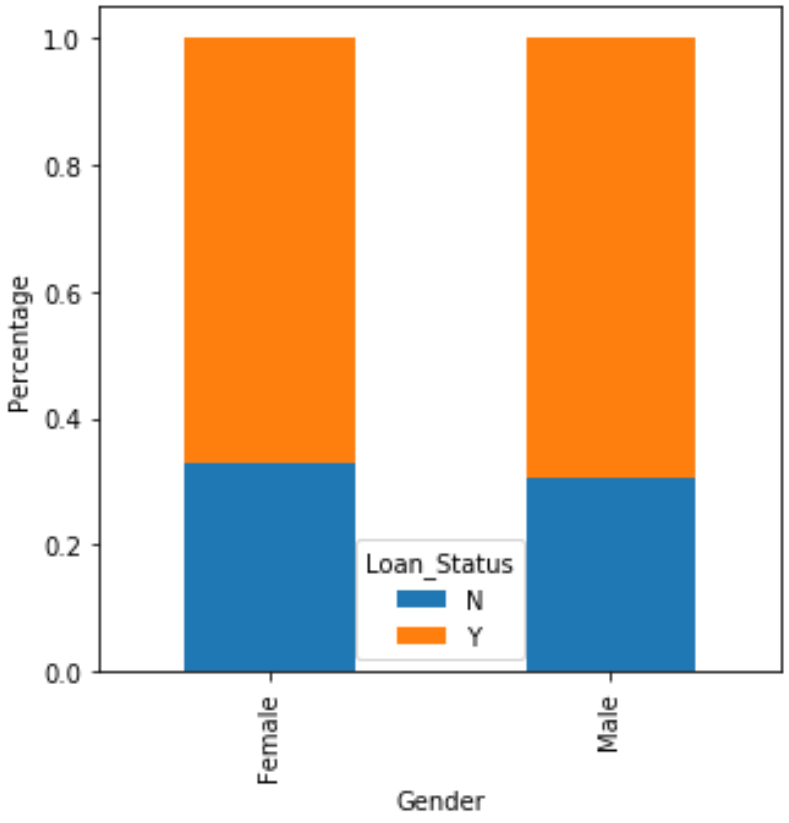
In [215]:

```
print(pd.crosstab(data["Gender"],data["Loan_Status"]))
```

```
Loan_Status    N    Y
Gender
Female        37   75
Male         150  339
```

In [216]:

```
Gender =pd.crosstab(data["Gender"],data["Loan_Status"])
Gender.div(Gender.sum(1).astype(float),axis = 0) .plot(kind="bar",stacked
=True,figsize=(5,5))
plt.xlabel("Gender")
plt.ylabel("Percentage")
plt.show()
```



Male applicants is higher for the approved loans.

Relation between "Loan_Status" and "Married"

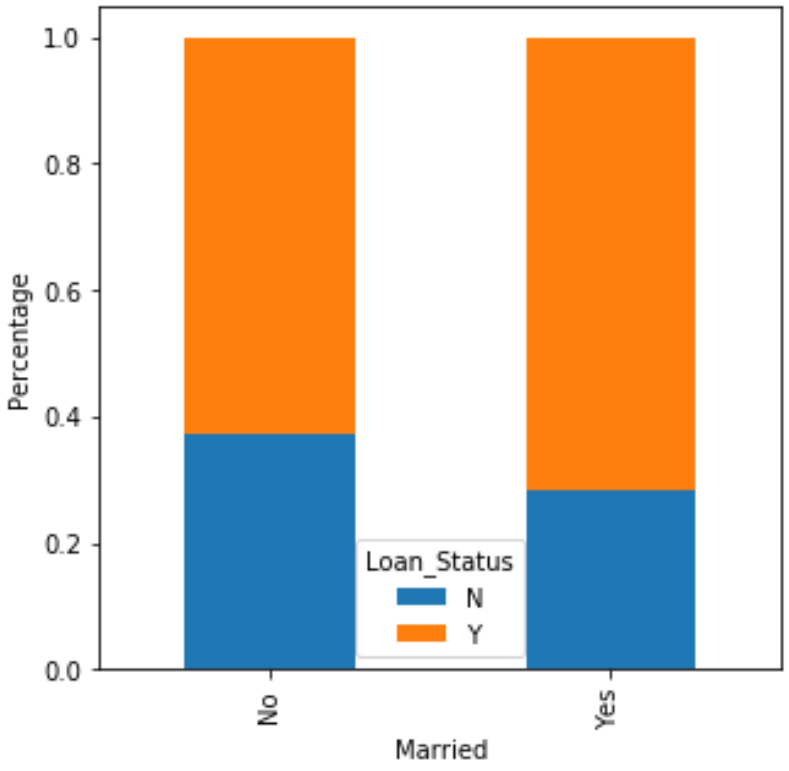
In [217]:

```
print(pd.crosstab(data["Married"],data["Loan_Status"]))
```

```
Loan_Status    N    Y
Married
No            79  134
Yes          113  285
```

In [218]:

```
Married=pd.crosstab(data["Married"],data["Loan_Status"])
Married.div(Married.sum(1).astype(float),axis = 0) .plot(kind="bar",stack
ed =True,figsize=(5,5))
plt.xlabel("Married")
plt.ylabel("Percentage")
plt.show()
```



Married applicants is higher for the approved loans.

Relation between "Loan_Status" and "Dependents"

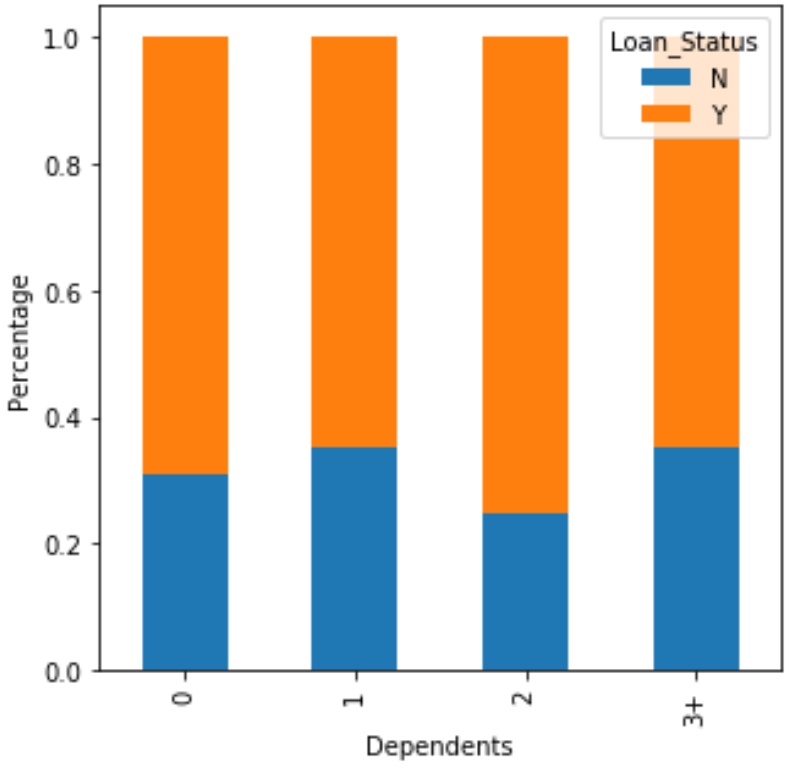
In [219]:

```
print(pd.crosstab(data["Dependents"],data["Loan_Status"]))
```

```
Loan_Status    N    Y
Dependents
0            107  238
1             36   66
2             25   76
3+            18   33
```

In [220]:

```
Dependents =pd.crosstab(data["Dependents"],data["Loan_Status"])
Dependents.div(Dependents.sum(1).astype(float),axis = 0) .plot(kind="bar"
,stacked =True,figsize=(5,5))
plt.xlabel("Dependents")
plt.ylabel("Percentage")
plt.show()
```



1 or 3+ dependents is similar across both the categories of Loan_Status.

Relation between "Loan_Status" and "Education"

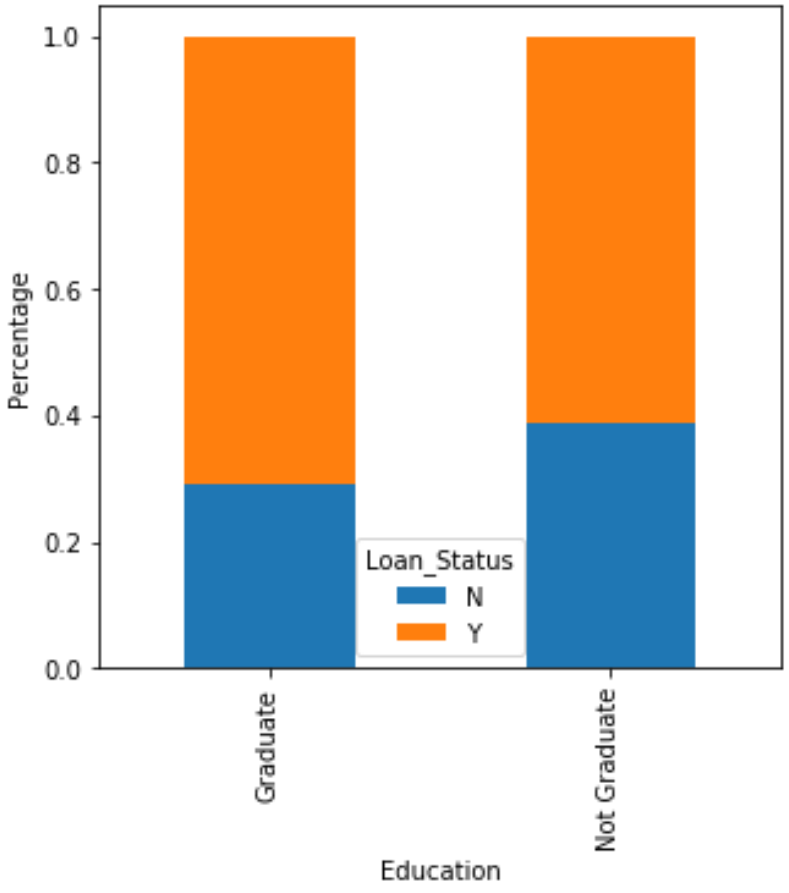
In [221]:

```
print(pd.crosstab(data["Education"],data["Loan_Status"]))
```

```
Loan_Status     N    Y
Education
Graduate      140  340
Not Graduate   52   82
```

In [222]:

```
Education =pd.crosstab(data["Education"],data["Loan_Status"])
Education.div(Education.sum(1).astype(float),axis = 0) .plot(kind="bar",s
tacked =True,figsize=(5,5))
plt.xlabel("Education")
plt.ylabel("Percentage")
plt.show()
```



Graduate applicants is higher for the approved loans.

Relation between "Loan_Status" and "Self_Employed"

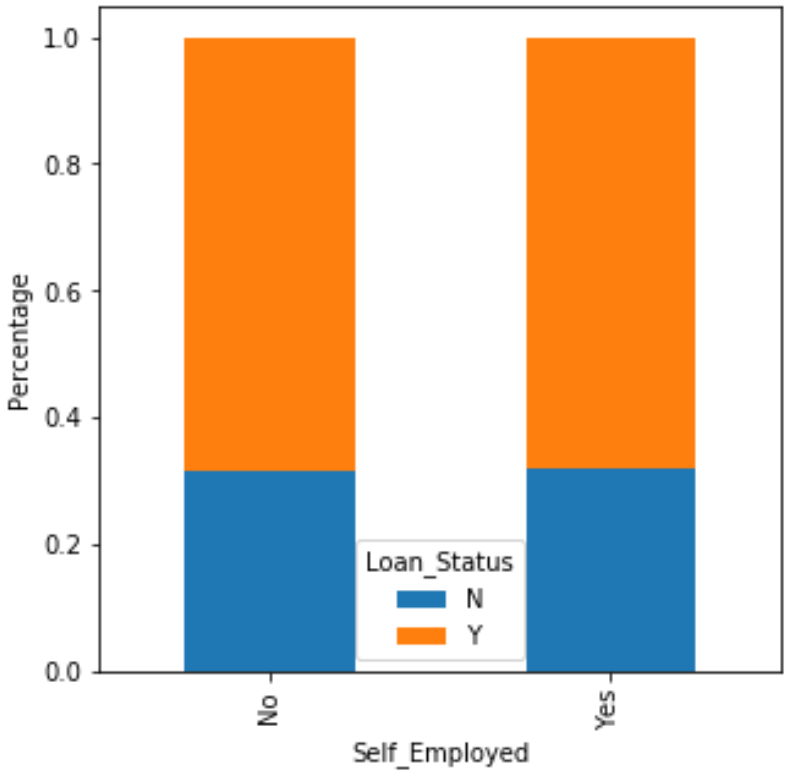
In [223]:

```
print(pd.crosstab(data["Self_Employed"],data["Loan_Status"]))
```

```
Loan_Status       N    Y
Self_Employed
No              157  343
Yes              26   56
```

In [224]:

```
Self_Employed =pd.crosstab(data["Self_Employed"],data["Loan_Status"])
Self_Employed.div(Self_Employed.sum(1).astype(float),axis = 0) .plot(kind
="bar",stacked =True,figsize=(5,5))
plt.xlabel("Self_Employed")
plt.ylabel("Percentage")
plt.show()
```



There is nothing significant we can infer from Self_Employed vs Loan_Status plot.

Relation between "Loan_Status" and "Credit_History"

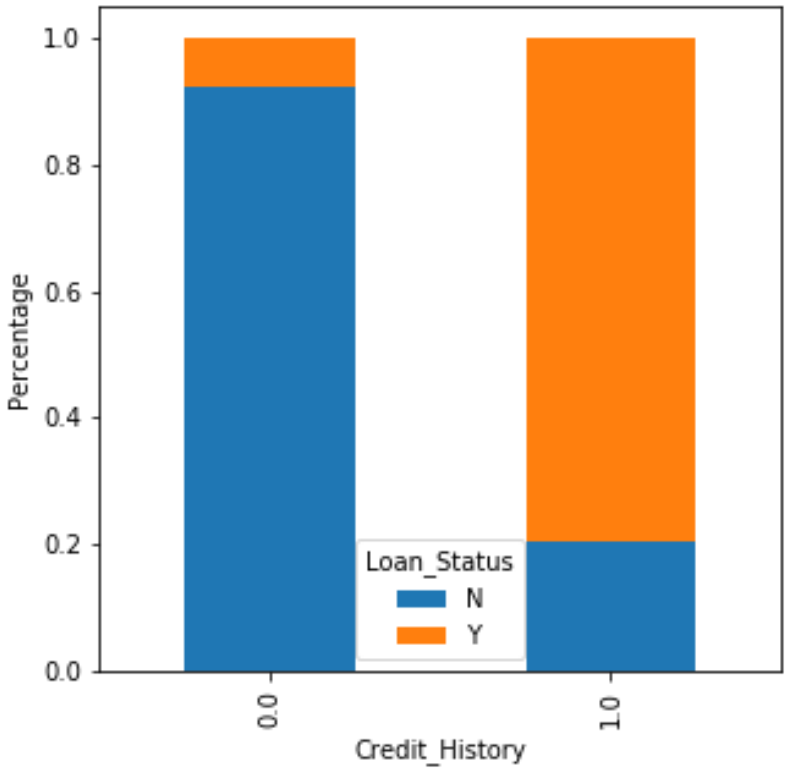
In [225]:

```
print(pd.crosstab(data["Credit_History"],data["Loan_Status"]))
```

```
Loan_Status      N    Y
Credit_History
0.0             82    7
1.0             97  378
```

In [226]:

```
Credit_History =pd.crosstab(data["Credit_History"],data["Loan_Status"])
Credit_History.div(Credit_History.sum(1).astype(float),axis = 0) .plot(ki
nd="bar",stacked =True,figsize=(5,5))
plt.xlabel("Credit_History")
plt.ylabel("Percentage")
plt.show()
```



credit history as 1 are more likely to get their loans approved.

Relation between "Loan_Status" and "Property_Area"

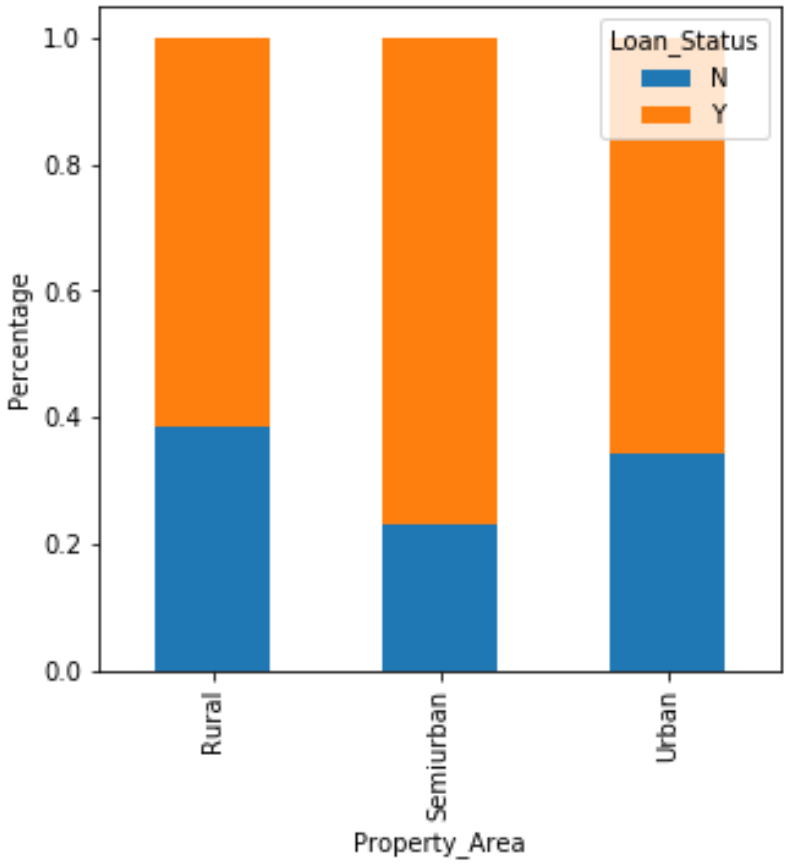
In [227]:

```
print(pd.crosstab(data["Property_Area"],data["Loan_Status"]))
```

```
Loan_Status     N    Y
Property_Area
Rural          69  110
Semiurban      54  179
Urban          69  133
```

In [228]:

```
Property_Area =pd.crosstab(data["Property_Area"],data["Loan_Status"])
Property_Area.div(Property_Area.sum(1).astype(float),axis = 0) .plot(kind
="bar",stacked =True,figsize=(5,5))
plt.xlabel("Property_Area")
plt.ylabel("Percentage")
plt.show()
```



loans getting approved in semiurban area is higher as compared to that in rural or urban areas.

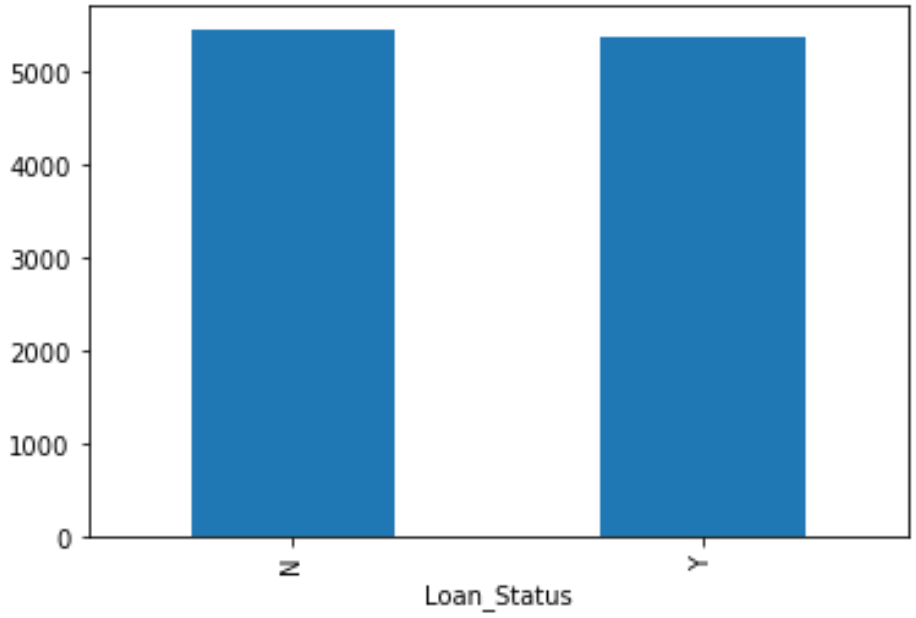
Relation between "Loan_Status" and "Income"

In [229]:

```
data.groupby("Loan_Status")["ApplicantIncome"].mean().plot.bar()
```

Out[229]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe52f010dd0>
```



In [230]:

```
bar = [0,2000,5000,8000,81000]
groups = ["Low","Average","High","Very High"]
data["Income_new"] = pd.cut(data["ApplicantIncome"] , bar , labels = grou
ps)
```

In [231]:

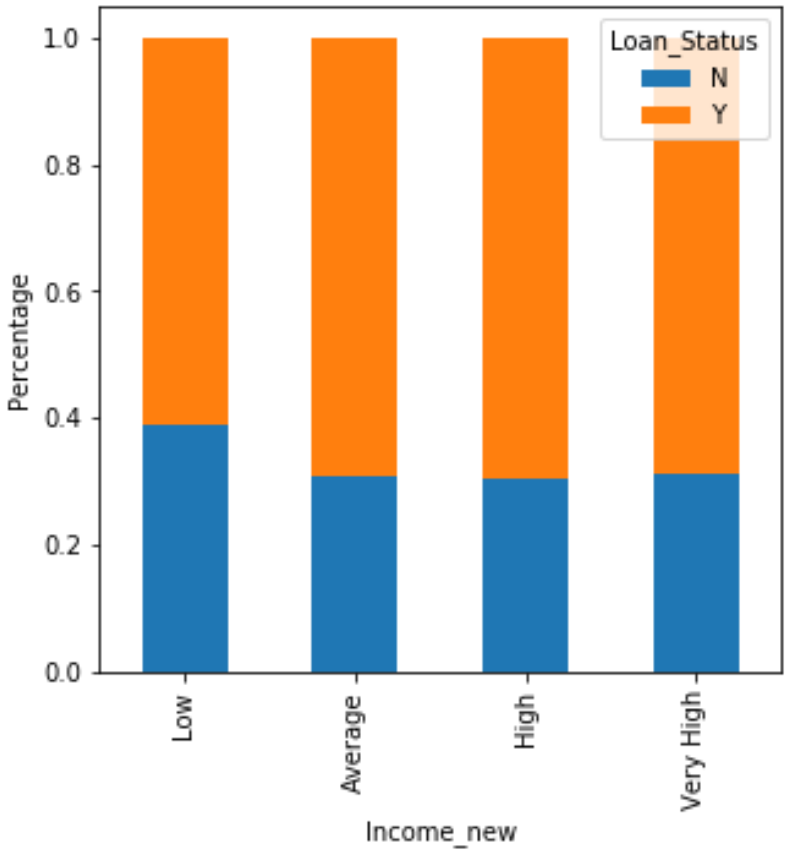
```
print(pd.crosstab(data["Income_new"],data["Loan_Status"]))
```

```
Loan_Status    N    Y
Income_new
Low           16   25
Average      117  265
High          33   75
Very High     26   57
```

if the applicant income is high the chances of loan approval will also be high.

In [232]:

```
Income_new =pd.crosstab(data["Income_new"],data["Loan_Status"])
Income_new.div(Income_new.sum(1).astype(float),axis = 0) .plot(kind="bar"
,stacked =True,figsize=(5,5))
plt.xlabel("Income_new")
plt.ylabel("Percentage")
plt.show()
```



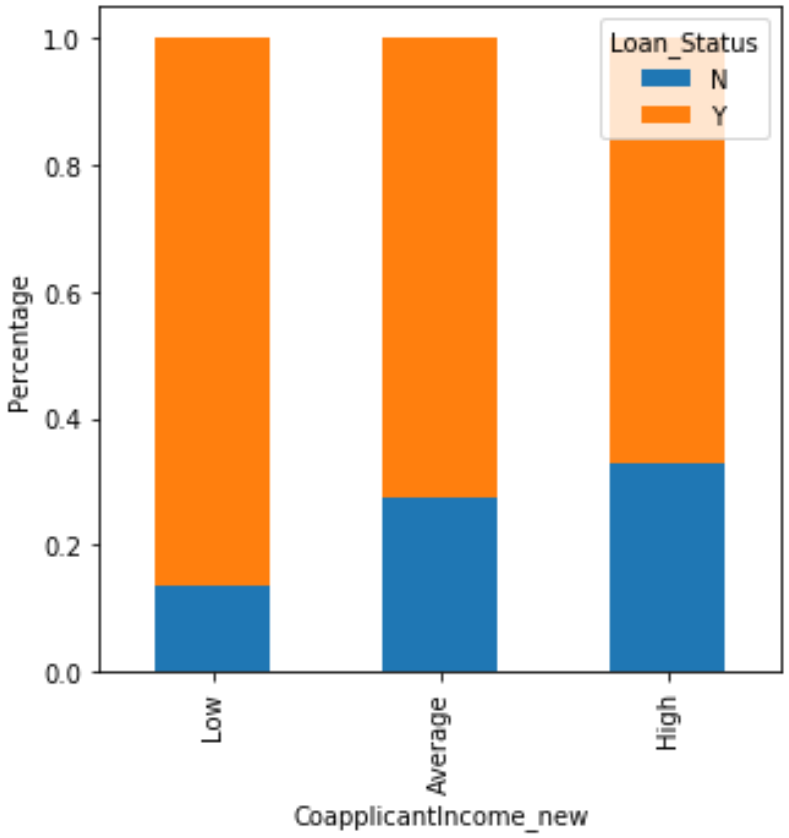
We will analyze the coapplicant income and loan amount variable in similar way.

In [233]:

```
bar = [0,1000,3000,42000]
groups = ["Low","Average","High"]
data["CoapplicantIncome_new"] = pd.cut(data['CoapplicantIncome'], bar ,la
bels= groups)
```

In [234]:

```
CoapplicantIncome_new =pd.crosstab(data["CoapplicantIncome_new"],data["Loan_Status"])
CoapplicantIncome_new.div(CoapplicantIncome_new.sum(1).astype(float),axis = 0) .plot(kind="bar",stacked =True,figsize=(5,5))
plt.xlabel("CoapplicantIncome_new")
plt.ylabel("Percentage")
plt.show()
```



It shows that if coapplicant's income is less the chances of loan approval are high. But this does not look right. The possible reason behind this may be that most of the applicants don't have any coapplicant so the coapplicant income for such applicants is 0 and hence the loan approval is not dependent on it. So we can make a new variable in which we will combine the applicant's and coapplicant's income to visualize the combined effect of income on loan approval.

In [235]:

```
data["TotalIncome"] = data["ApplicantIncome"] + data["CoapplicantIncome"]
```

In [236]:

```
bar = [0,2000,5000,8000,81000]
groups = ["Low","Average","High","Very High"]
data["TotalIncome_new"] = pd.cut(data["TotalIncome"] , bar , labels = gro
ups)
```

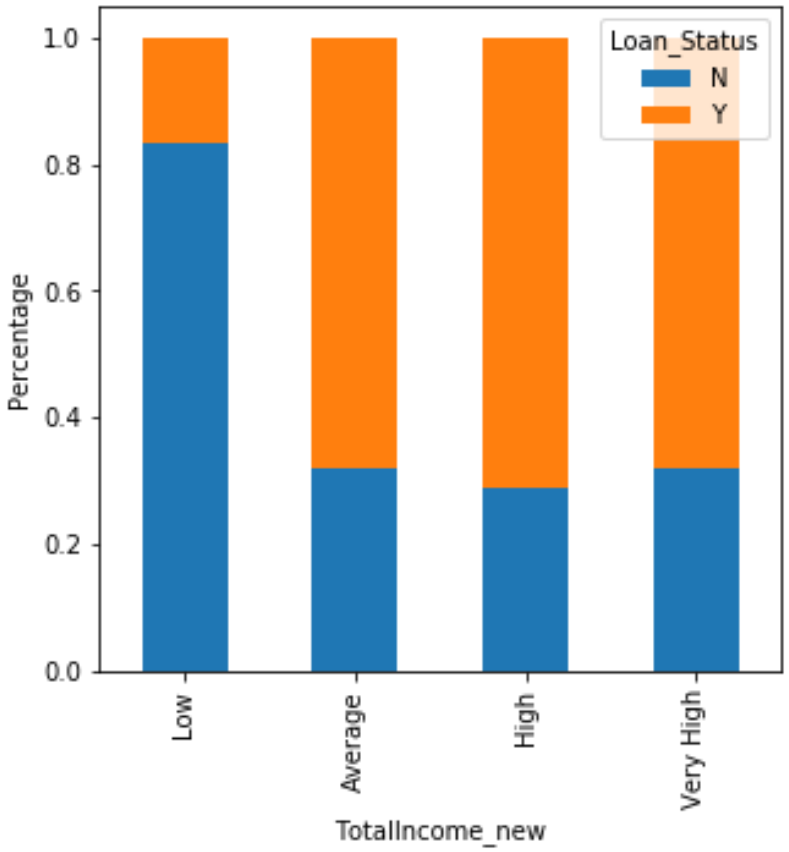
In [237]:

```
print(pd.crosstab(data["TotalIncome_new"],data["Loan_Status"]))
```

```
Loan_Status       N    Y
TotalIncome_new
Low               5    1
Average          84  180
High             61  151
Very High        42   90
```

In [238]:

```
TotalIncome_new =pd.crosstab(data["TotalIncome_new"],data["Loan_Status"])
TotalIncome_new.div(TotalIncome_new.sum(1).astype(float),axis = 0) .plot(
kind="bar",stacked =True,figsize=(5,5))
plt.xlabel("TotalIncome_new")
plt.ylabel("Percentage")
plt.show()
```



We can see that 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.

Relation between "Loan_Status" and "Loan Amount"

In [239]:

```
bar = [0,100,300,800]
groups = ["Low","Average","High"]
data["LoanAmount_new"] = pd.cut(data["LoanAmount"] , bar , labels = group
s)
```

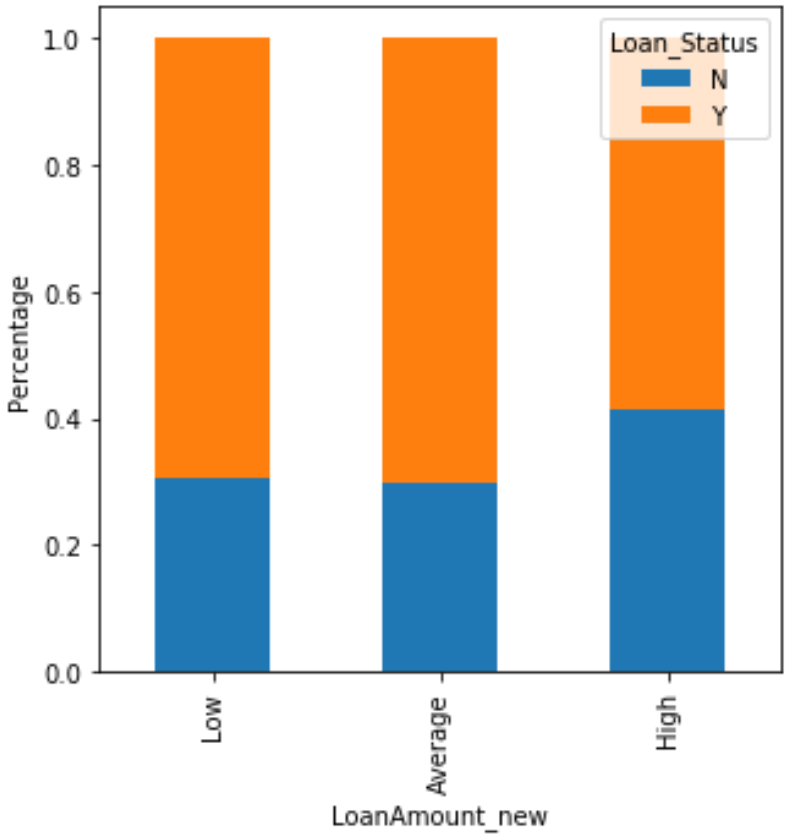
In [240]:

```
print(pd.crosstab(data["LoanAmount_new"],data["Loan_Status"]))
```

```
Loan_Status       N    Y
LoanAmount_new
Low              47  107
Average         122  287
High             12   17
```

In [241]:

```
LoanAmount_new =pd.crosstab(data["LoanAmount_new"],data["Loan_Status"])
LoanAmount_new.div(LoanAmount_new.sum(1).astype(float),axis = 0) .plot(ki
nd="bar",stacked =True,figsize=(5,5))
plt.xlabel("LoanAmount_new")
plt.ylabel("Percentage")
plt.show()
```



It can be seen that the proportion of approved loans is higher for Low and Average Loan Amount as compared to that of High Loan Amount which supports our hypothesis in which we considered that the chances of loan approval will be high when the loan amount is less.

Let's drop the bins which we created for the exploration part. We will change the 3+ in dependents variable to 3 to make it a numerical variable

In [242]:

```
data = data.drop(["Income_new","CoapplicantIncome_new","LoanAmount_new",
"TotalIncome","TotalIncome_new"],axis =1)
```

In [243]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
Loan_ID               614 non-null object
Gender                601 non-null object
Married               611 non-null object
Dependents            599 non-null object
Education             614 non-null object
Self_Employed         582 non-null object
ApplicantIncome       614 non-null int64
CoapplicantIncome     614 non-null float64
LoanAmount            592 non-null float64
Loan_Amount_Term      600 non-null float64
Credit_History        564 non-null float64
Property_Area         614 non-null object
Loan_Status           614 non-null object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.4+ KB
```

In [244]:

```
data["Dependents"].replace('3+' , 3, inplace = True)
data["Loan_Status"].replace('N',0,inplace = True)
data["Loan_Status"].replace('Y',1,inplace = True)
```

In [245]:

```
corr = data.corr()
corr
```

Out[245]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loa
ApplicantIncome	1.000000	-0.116605	0.570909	
CoapplicantIncome	-0.116605	1.000000	0.188619	
LoanAmount	0.570909	0.188619	1.000000	
Loan_Amount_Term	-0.045306	-0.059878	0.039447	
Credit_History	-0.014715	-0.002056	-0.008433	
Loan_Status	-0.004710	-0.059187	-0.037318	

In [246]:

```
f,ax = plt.subplots(figsize=(10,12))
sns.heatmap(corr,vmax = .8 ,square = True , cmap = "BuPu" , annot=True)
```

Out[246]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe52ed09150>
```


ApplicantIncome
CoapplicantIncome
LoanAmount
Loan_Amount_Term
Credit_History
Loan_Status
1 -0.12 0.57 -0.045 -0.015 -0.0047
-0.12 1 0.19 -0.06 -0.0021 -0.059
0.57 0.19 1 0.039 -0.0084 -0.037
-0.045 -0.06 0.039 1 0.0015 -0.021
-0.015 -0.0021 -0.0084 0.0015 1 0.56
-0.0047 -0.059 -0.037 -0.021 0.56 1
0.8
0.6
0.4
0.2
0.0

We see that the most correlated variables are (ApplicantIncome - LoanAmount) and (Credit_History - Loan_Status).

Missing value imputation

In [247]:

```
data.isnull().sum()
```

Out[247]:

```
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 [248]:

```
data.describe()
```

Out[248]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Te
count	614.000000	614.000000	592.000000	600.000
mean	5403.459283	1621.245798	146.412162	342.000
std	6109.041673	2926.248369	85.587325	65.120
min	150.000000	0.000000	9.000000	12.000
25%	2877.500000	0.000000	100.000000	360.000
50%	3812.500000	1188.500000	128.000000	360.000
75%	5795.000000	2297.250000	168.000000	360.000
max	81000.000000	41667.000000	700.000000	480.000

We will treat the missing values in all the features one by one.

We can consider these methods to fill the missing values:

For numerical variables: imputation using mean or median

For categorical variables: imputation using mode

There are very less missing values in Gender, Married, Dependents, Credit_History and Self_Employed features so we can fill them using the mode of the features

In [249]:

```
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 = Tr
ue)
data["Credit_History"].fillna(data["Credit_History"].mode()[0],inplace =
True)
```

In [250]:

```
data["Loan_Amount_Term"].mode()[0]
```

Out[250]:

```
360.0
```

In [251]:

```
data["Loan_Amount_Term"].fillna(data["Loan_Amount_Term"].mode()[0],inplace = True)
```

In [252]:

```
data["Loan_Amount_Term"].value_counts()
```

Out[252]:

```
360.0    526
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
```

In [253]:

```
data.isnull().sum()
```

Out[253]:

```
Loan_ID               0
Gender                0
Married               0
Dependents            0
Education             0
Self_Employed         0
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term      0
Credit_History        0
Property_Area         0
Loan_Status           0
dtype: int64
```

Now we will see the LoanAmount variable. As it is a numerical variable, we can use mean or median to impute the missing values.

We will use median to fill the null values as earlier we saw that loan amount have outliers so the mean will not be the proper approach as it is highly affected by the presence of outliers.

In [254]:

```
data["LoanAmount"].fillna(data["LoanAmount"].median(),inplace =True)
```

In [255]:

```
data.isnull().sum()
```

Out[255]:

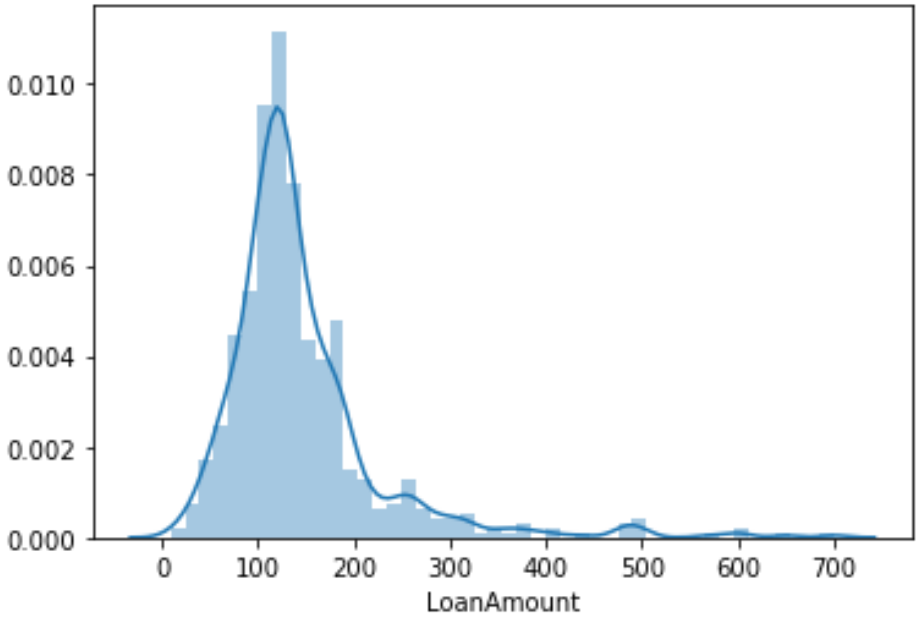
```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status         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

In [256]:

```
sns.distplot(data["LoanAmount"])
```

Out[256]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe52f10ae90>
```

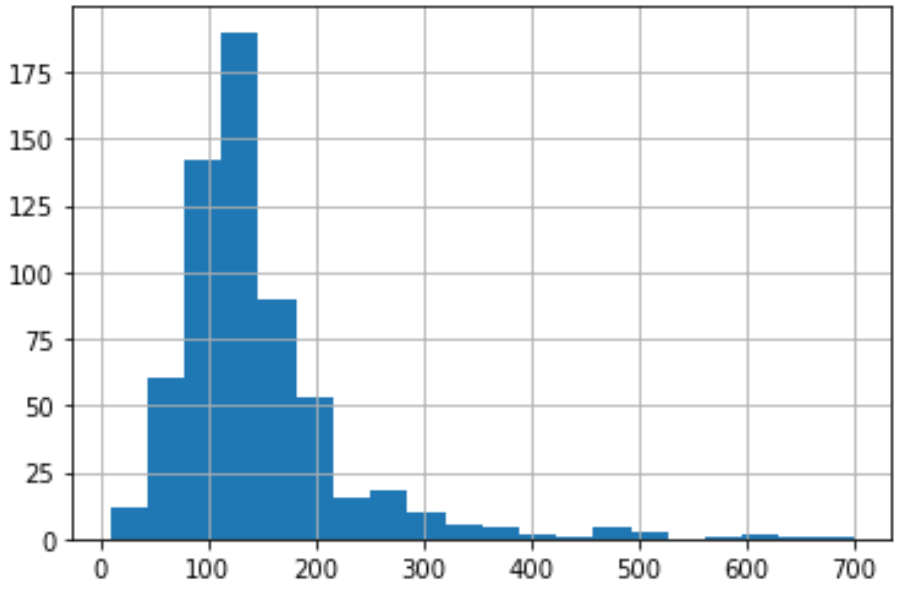


In [257]:

```
data['LoanAmount'].hist(bins =20)
```

Out[257]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe52f3183d0>
```

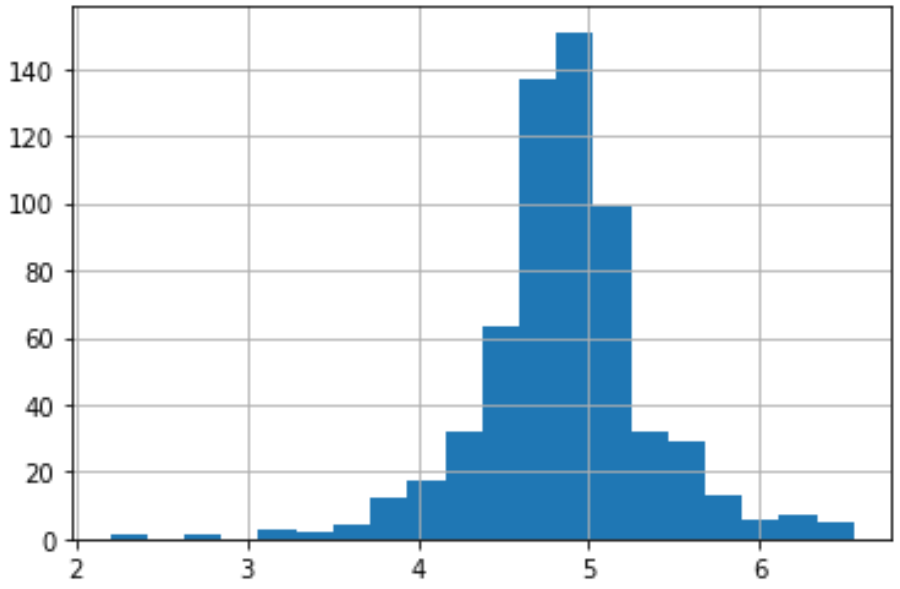


In [258]:

```
data["LoanAmount_log"] = np.log(data["LoanAmount"])
data["LoanAmount_log"].hist(bins=20)
```

Out[258]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe52f9a3dd0>
```



In [259]:

```
data["TotalIncome"] = data["ApplicantIncome"] + data["CoapplicantIncome"]
```

In [260]:

```
data[["TotalIncome"]].head(5)
```

Out[260]:

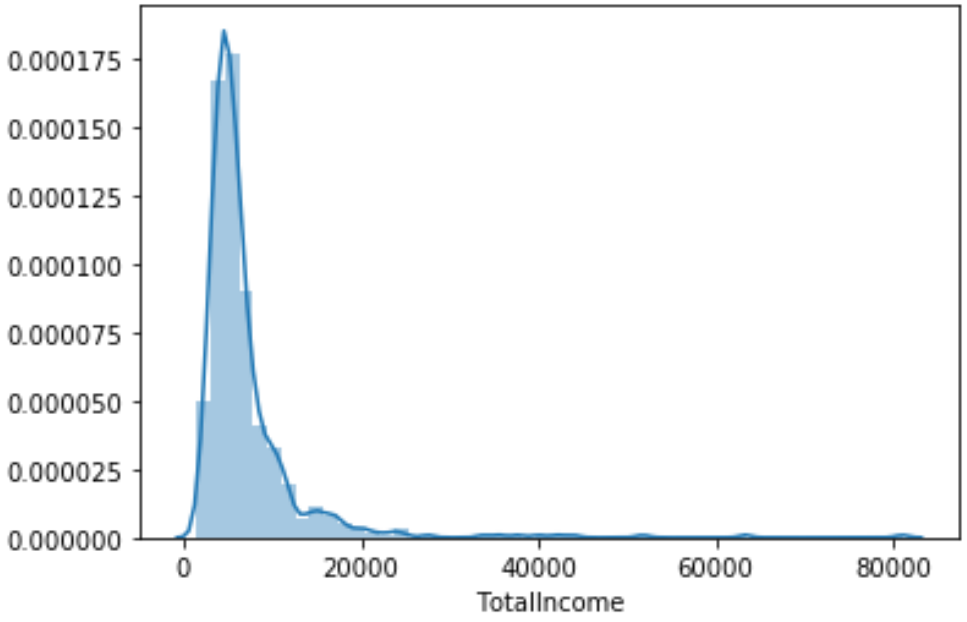
	TotalIncome
0	5849.0
1	6091.0
2	3000.0
3	4941.0
4	6000.0

In [261]:

```
sns.distplot(data["TotalIncome"])
```

Out[261]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe52f50e7d0>
```

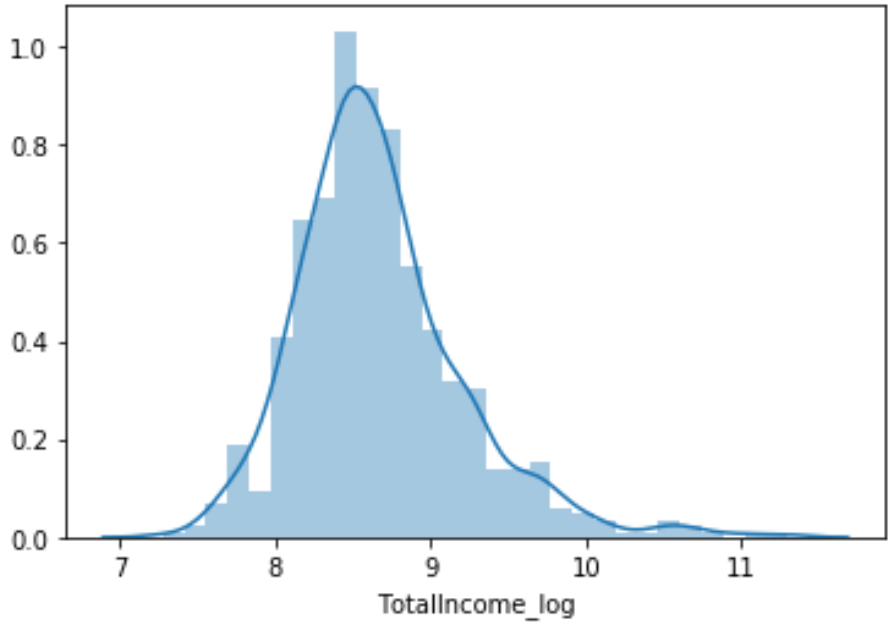


In [262]:

```
data["TotalIncome_log"] = np.log(data["TotalIncome"])
sns.distplot(data["TotalIncome_log"])
```

Out[262]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe52ffc7bd0>
```



Now create the EMI feature.

In [263]:

```
data["EMI"]=data["LoanAmount"]/data["Loan_Amount_Term"]
```

In [264]:

```
data[["EMI"]].head()
```

Out[264]:

	EMI
0	0.355556
1	0.355556
2	0.183333
3	0.333333
4	0.391667

Let's create Balance Income feature now and check its distribution.

In [265]:

```
data["Balance_Income"] = data["TotalIncome"]-data["EMI"]*1000
```

In [266]:

```
data[["Balance_Income"]].head()
```

Out[266]:

	Balance_Income
0	5493.444444
1	5735.444444
2	2816.666667
3	4607.666667
4	5608.333333

Let us now drop the variables which we used to create these new features. Reason for doing this is, the correlation between those old features and these new features will be very high and logistic regression assumes that the variables are not highly correlated. We also wants to remove the noise from the dataset, so removing correlated features will help in reducing the noise too.

In [267]:

```
data = data.drop(["ApplicantIncome" , "CoapplicantIncome" ,"LoanAmount" ,
"Loan_Amount_Term"],axis = 1)
```

In [268]:

```
data.head()
```

Out[268]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Cre
0	LP001002	Male	No	0	Graduate	No	
1	LP001003	Male	Yes	1	Graduate	No	
2	LP001005	Male	Yes	0	Graduate	Yes	
3	LP001006	Male	Yes	0	Not Graduate	No	
4	LP001008	Male	No	0	Graduate	No	

In [269]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 14 columns):
Loan_ID            614 non-null object
Gender             614 non-null object
Married            614 non-null object
Dependents         614 non-null object
Education          614 non-null object
Self_Employed      614 non-null object
Credit_History     614 non-null float64
Property_Area      614 non-null object
Loan_Status        614 non-null int64
LoanAmount_log     614 non-null float64
TotalIncome        614 non-null float64
TotalIncome_log    614 non-null float64
EMI                614 non-null float64
Balance_Income     614 non-null float64
dtypes: float64(6), int64(1), object(7)
memory usage: 67.2+ KB
```

In [270]:

```
loan = data["Loan_ID"]
```

Model Building

After creating new features, we can continue the model building process. So we will start with logistic regression model and then move over to more complex models like RandomForest and Decision tree

In [271]:

```
data = data.drop("Loan_ID",axis =1)
```

In [272]:

```
data.head()
```

Out[272]:

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

In [ ]:

```

```

We will use scikit-learn (sklearn) for making different models which is an open source library for Python. It is one of the most efficient tool which contains many inbuilt functions that can be used for modeling in Python.

Sklearn requires the target variable in a separate dataset. So, we will drop our target variable from the train dataset and save it in another dataset.

droping the target variable "Loan_Status"

In [273]:

```
X = data.drop("Loan_Status", 1)
```

In [274]:

```
X.head()
```

Out[274]:

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

save the target variable "Loan_Status" in another dataset

In [275]:

```
Y = data["Loan_Status"]
```

In [276]:

```
Y.head(5)
```

Out[276]:

```
0    1
1    0
2    1
3    1
4    1
Name: Loan_Status, dtype: int64
```

In [277]:

```
X = pd.get_dummies(X)
```

In [278]:

```
X.head(5)
```

Out[278]:

	Credit_History	LoanAmount_log	TotalIncome	TotalIncome_log	EMI
0	1.0	4.852030	5849.0	8.674026	0.355556
1	1.0	4.852030	6091.0	8.714568	0.355556
2	1.0	4.189655	3000.0	8.006368	0.183333
3	1.0	4.787492	4941.0	8.505323	0.333333
4	1.0	4.948760	6000.0	8.699515	0.391667

5 rows × 21 columns

In [279]:

```
data = pd.get_dummies(data)
```

In [280]:

```
data.head()
```

Out[280]:

	Credit_History	Loan_Status	LoanAmount_log	TotalIncome	TotalIncome_
0	1.0	1	4.852030	5849.0	8.674
1	1.0	0	4.852030	6091.0	8.714
2	1.0	1	4.189655	3000.0	8.006
3	1.0	1	4.787492	4941.0	8.505
4	1.0	1	4.948760	6000.0	8.699

5 rows × 22 columns

In [281]:

```
from sklearn.model_selection import train_test_split
```

In [282]:

```
training_x ,test_x ,training_y ,testing_y = train_test_split(X ,Y ,test_s
ize = 0.3 ,random_state=1)
```

In [ ]:

```

```

Logistic Regression

Let's import LogisticRegression and accuracy_score from sklearn and fit the logistic regression model.

In [283]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

In [284]:

```
logistic_model = LogisticRegression(random_state=1)
```

In [285]:

```
logistic_model.fit(training_x,training_y)
```

```
/srv/conda/envs/notebook/lib/python3.7/site-packages/sklear
n/linear_model/logistic.py:432: FutureWarning: Default solve
r will be changed to 'lbfgs' in 0.22. Specify a solver to si
lence this warning.
  FutureWarning)
```

Out[285]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit
_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_i
ter=100,
                   multi_class='warn', n_jobs=None, penalty
='l2',
                   random_state=1, solver='warn', tol=0.000
1, verbose=0,
                   warm_start=False)
```

In [286]:

```
pred_y0 = logistic_model.predict(test_x)
```

In [287]:

```
pred_y0
```

Out[287]:

```
array([1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 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, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1,
1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 0, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0,
1, 1, 1, 0,
       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, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1,
1, 0, 1, 1,
       1, 0, 1, 1, 0, 1, 1, 1, 1])
```

In [288]:

```
testing_y
```

Out[288]:

```
533    0
544    1
41     1
148    0
111    1
293    0
435    1
407    1
311    1
23     0
242    1
185    1
521    1
496    1
566    1
301    1
375    1
200    1
599    1
306    1
132    1
559    1
0      1
135    0
195    0
386    1
563    1
201    1
520    1
518    0
      ..
385    1
504    1
61     1
159    1
555    1
76     0
581    0
587    1
455    1
68     1
553    0
495    1
4      1
449    0
```

```
536    1
554    0
259    0
19     1
353    0
593    1
117    1
262    0
78     0
482    1
57     0
180    0
491    1
42     1
530    1
457    0
Name: Loan_Status, Length: 185, dtype: int64
```

In [295]:

```
from sklearn.metrics import confusion_matrix
cm1 = confusion_matrix(testing_y , pred_y0)
print(cm1)
```

```
[[ 24  37]
 [  2 122]]
```

In [296]:

```
from sklearn.metrics import accuracy_score
accuracy_pred_y0 = accuracy_score(testing_y , pred_y0)
print(accuracy_pred_y0*100)
```

```
78.91891891891892
```

So our predictions are almost 79% accurate, i.e. we have identified 79% of the loan status correctly for our logistic regression model.

In [ ]:

```

```

In [290]:

```
pred_y = logistic_model.predict(X)
```

In [297]:

```
pred_y
```

Out[297]:

```
array([1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
1, 1, 0, 1,
       0, 0, 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, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
0, 0, 0, 1,
       0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 0, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
0, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1,
1, 1, 1, 1,
       1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 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, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 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, 1,
1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1,
1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
1, 1, 1, 1,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
0, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
       1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1,
```

```
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, 0,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
1, 1, 0, 1,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0,
1, 0, 1, 0,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0,
1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 0])
```

In [298]:

```
Y
```

Out[298]:

```
0      1
1      0
2      1
3      1
4      1
5      1
6      1
7      0
8      1
9      0
10     1
11     1
12     1
13     0
14     1
15     1
16     1
17     0
18     0
19     1
20     0
21     1
22     0
23     0
24     0
25     1
26     1
27     1
28     0
29     1
      ..
584    0
585    0
586    1
587    1
588    1
589    0
590    1
591    0
592    1
593    1
594    1
595    1
596    0
597    0
```

```
598    1
599    1
600    0
601    1
602    1
603    1
604    1
605    0
606    1
607    1
608    1
609    1
610    1
611    1
612    1
613    0
Name: Loan_Status, Length: 614, dtype: int64
```

In [318]:

```
from sklearn.metrics import accuracy_score
accuracy_pred_y = accuracy_score(Y,pred_y)
print(accuracy_pred_y*100)
```

```
80.94462540716613
```

In [319]:

```
from sklearn.metrics import confusion_matrix
cm1 = confusion_matrix(Y , pred_y)
print(cm1)
```

```
[[ 82 110]
 [  7 415]]
```

In [301]:

```
df = pd.DataFrame({}).to_csv("Loan_Output.csv")
```

In [306]:

```
df = pd.read_csv("Loan_Output.csv")
df["Loan_ID"] = loan
df.to_csv("Loan_Output.csv", index=False)
```

In [321]:

```
df.head()
```

Out[321]:

	Loan_ID	Loan_Status	Predict_value_linear_regression
0	LP001002	1	1
1	LP001003	0	1
2	LP001005	1	1
3	LP001006	1	1
4	LP001008	1	1

In [308]:

```
#del df['new_column']
del df['Unnamed: 0']
```

In [ ]:

```

```

In [310]:

```
loan_status =pd.DataFrame({'loan_status':Y})
```

In [312]:

```
df["Loan_Status"] = loan_status
df.to_csv("Loan_Output.csv", index=False)
```

In [ ]:

```

```

In [315]:

```
prediction =pd.DataFrame({'predict':pred_y})
```

In [325]:

```
df["Predict_value_linear_regression"] =prediction
df.to_csv("Loan_Output.csv", index=False)
```

In [324]:

```
df.head()
```

Out[324]:

	Loan_ID	Loan_Status	Predict_value_linear_regression
0	LP001002	1	1
1	LP001003	0	1
2	LP001005	1	1
3	LP001006	1	1
4	LP001008	1	1

In [342]:

```
df["Loan_Status"].replace(0,'No',inplace = True)
df["Loan_Status"].replace(1,'Yes',inplace = True)
df["Predict_value_linear_regression"].replace(0,'No',inplace = True)
df["Predict_value_linear_regression"].replace(1,'Yes',inplace = True)
df.to_csv("Loan_Output.csv", index=False)
```

In [343]:

```
df.head()
```

Out[343]:

	Loan_ID	Loan_Status	Predict_value_linear_regression
0	LP001002	Yes	Yes
1	LP001003	No	Yes
2	LP001005	Yes	Yes
3	LP001006	Yes	Yes
4	LP001008	Yes	Yes

Decision Tree

In [326]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [327]:

```
tree_model = DecisionTreeClassifier(random_state=1)
```

In [328]:

```
tree_model.fit(training_x ,training_y)
```

Out[328]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini',
max_depth=None,
                       max_features=None, max_leaf_nodes=Non
e,
                       min_impurity_decrease=0.0, min_impuri
ty_split=None,
                       min_samples_leaf=1, min_samples_split
=2,
                       min_weight_fraction_leaf=0.0, presort
=False,
                       random_state=1, splitter='best')
```

In [329]:

```
pred_y = tree_model.predict(test_x)
```

In [330]:

```
pred_y
```

Out[330]:

```
array([1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1,
0, 1, 1, 0,
       1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0,
1, 1, 0, 1,
       1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1,
1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
1, 1, 0, 1,
       1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
1, 0, 1, 0,
       1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1,
0, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1,
1, 0, 1, 1,
       0, 0, 1, 1, 0, 0, 0, 1, 1])
```

In [331]:

```
score_tree = accuracy_score(pred_y , testing_y)*100
score_tree
```

Out[331]:

```
71.35135135135135
```

our predictions are almost 71% accurate, i.e. we have identified 71% of the loan status correctly for our Decision tree model.

Random Forest

In [333]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [334]:

```
forest_model = RandomForestClassifier(random_state = 1,max_depth=5 , n_es
timators=50)
```

In [335]:

```
forest_model.fit(training_x,training_y)
```

Out[335]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, cr
iterion='gini',
                       max_depth=5, max_features='auto', max
_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impuri
ty_split=None,
                       min_samples_leaf=1, min_samples_split
=2,
                       min_weight_fraction_leaf=0.0, n_estim
ators=50,
                       n_jobs=None, oob_score=False, random_
state=1, verbose=0,
                       warm_start=False)
```

In [336]:

```
pred_y = forest_model.predict(test_x)
```

In [337]:

```
pred_y
```

Out[337]:

```
array([1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 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, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1,
1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 0, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0,
1, 1, 1, 0,
       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, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1,
1, 0, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1])
```

In [338]:

```
score_forest = accuracy_score(pred_y , testing_y)*100
score_forest
```

Out[338]:

```
77.83783783783784
```

our predictions are almost 78% accurate, i.e. we have identified 78% of the loan status correctly for our Random Forest model.

In [ ]:

In [ ]: