LOAN AMOUT PREDICTION

In [1]:
```
# import libraries

import numpy as np # linear algebra
import pandas as pd # data processing
import os
for dirname, _, filenames in os.walk('/jaya/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
%matplotlib inline
```

In [3]:
```
df = pd.read_csv("train_data.csv")
df
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplica
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	
...	...	...	...	...	...	...	...	
609	LP002978	Female	No	0	Graduate	No	2900	
610	LP002979	Male	Yes	3+	Graduate	No	4106	
611	LP002983	Male	Yes	1	Graduate	No	8072	
612	LP002984	Male	Yes	2	Graduate	No	7583	
613	LP002990	Female	No	0	Graduate	Yes	4583	

614 rows × 13 columns

In [4]:
```
df.head()
# 1st 5 rows of data
```

Out[4]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantI
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 [5]:
```
df.tail()# last 5 rows of data
```

Out[5]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplica
609	LP002978	Female	No	0	Graduate	No	2900	
610	LP002979	Male	Yes	3+	Graduate	No	4106	
611	LP002983	Male	Yes	1	Graduate	No	8072	
612	LP002984	Male	Yes	2	Graduate	No	7583	
613	LP002990	Female	No	0	Graduate	Yes	4583	

In [6]:
```
print("No.of Rows and Columns:",df.shape)
```

```
No.of Rows and Columns: (614, 13)
```

In [7]:
```
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
```

In [8]:
```
df.describe()
```

Out[8]:

	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 [9]:
```
df.isnull().sum()
```

Out[9]:
```
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
```

Analysis of ecah columns and also Data cleaning

DAta cleaning removing null values replacing duplicate values

In [10]:
```
df['Gender'].value_counts(dropna=False)
```

Out[10]:
```
Male      489
Female    112
NaN        13
Name: Gender, dtype: int64
```

In [11]:
```
# Here we can replace the null values with Male
df['Gender']=df['Gender'].replace(np.nan,'Male')
```

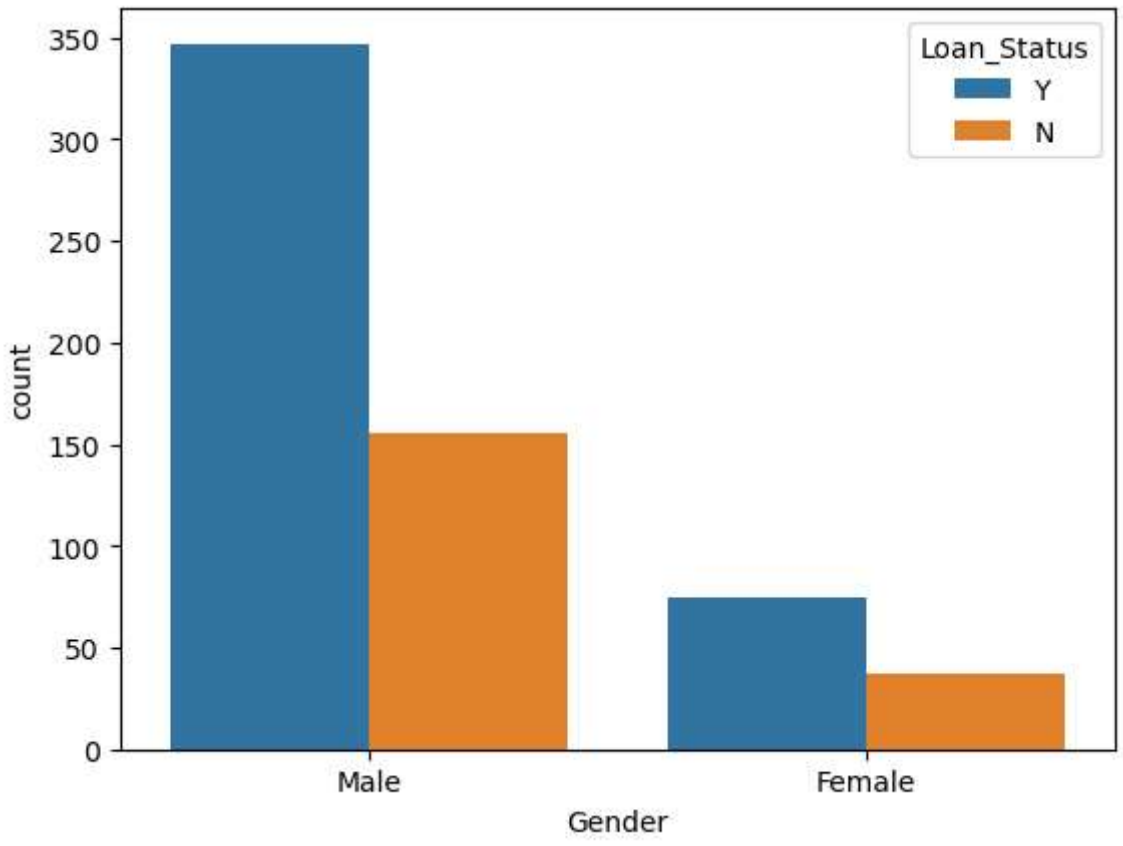
visulization

using countplot

In [12]:
```
# COUNT PLOTE USING SEABORN IN GENDER COLUMN

sns.countplot(x=df['Gender'],hue=df['Loan_Status'])
```

Out[12]: <AxesSubplot:xlabel='Gender', ylabel='count'>



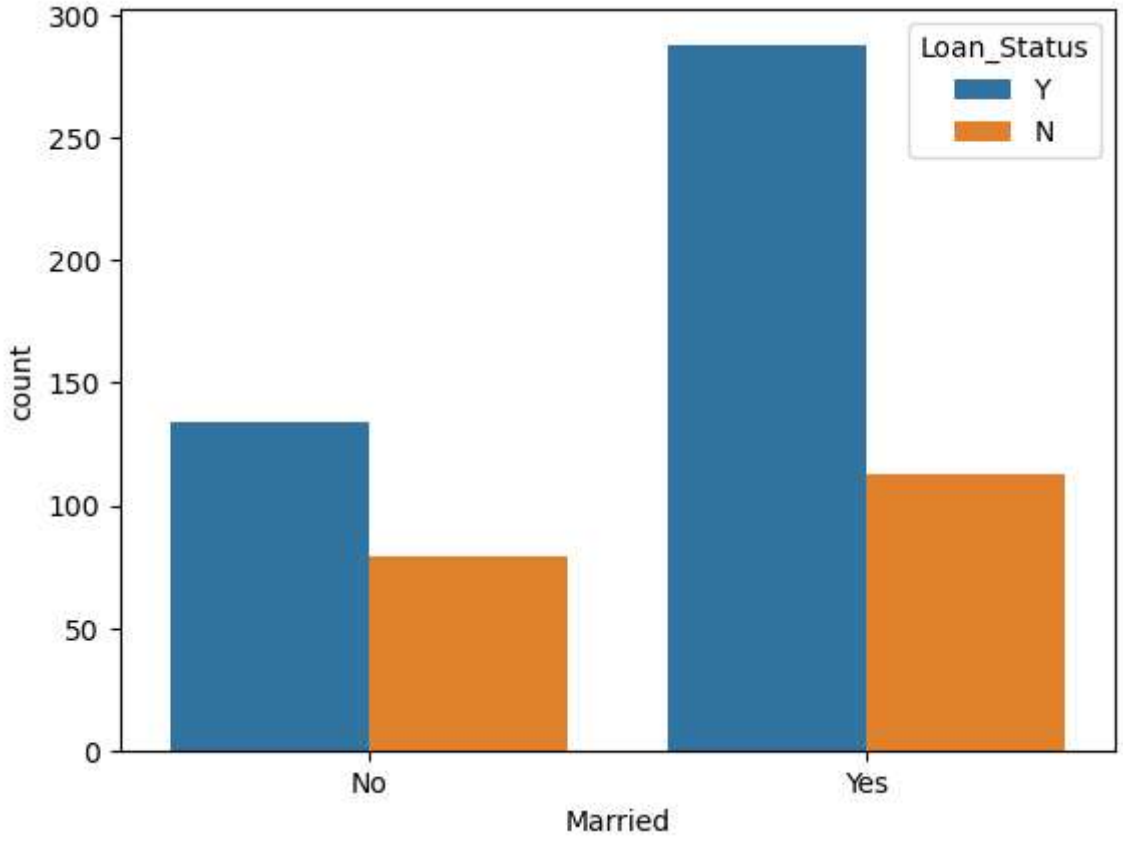
In [13]:
```
df['Married'].value_counts(dropna=False)
```

Out[13]:
```
Yes    398
No     213
NaN      3
Name: Married, dtype: int64
```

In [14]:
```
# here we replace the null values with Yes
df['Married']=df['Married'].replace(np.nan,'Yes')
```

In [15]:
```
sns.countplot(x=df['Married'],hue=df['Loan_Status'])
```

Out[15]: <AxesSubplot:xlabel='Married', ylabel='count'>



we can observe married applicants have more chances of getting loan

In [16]:
```
df['Dependents'].value_counts(dropna=False)
```

Out[16]:
```
0      345
1      102
2      101
3+      51
NaN     15
Name: Dependents, dtype: int64
```

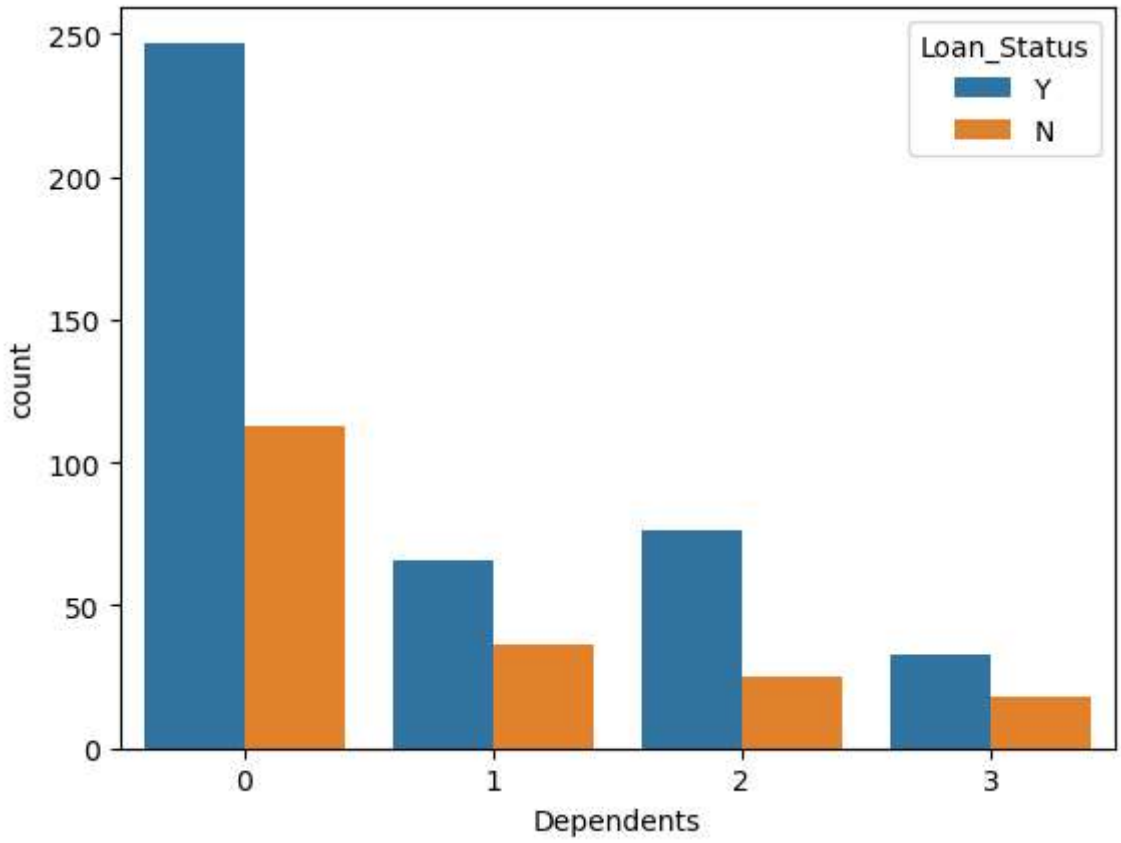
In [17]:
```
df['Dependents']=df['Dependents'].replace(np.nan,'0')
df['Dependents']=df['Dependents'].replace('3+' , '3')
```

In [18]:
```
df['Dependents'].value_counts(dropna=False)
```

Out[18]:
```
0    360
1    102
2    101
3     51
Name: Dependents, dtype: int64
```

In [19]:
```
sns.countplot(x=df['Dependents'],hue=df['Loan_Status'])
```

Out[19]:
```
<AxesSubplot:xlabel='Dependents', ylabel='count'>
```

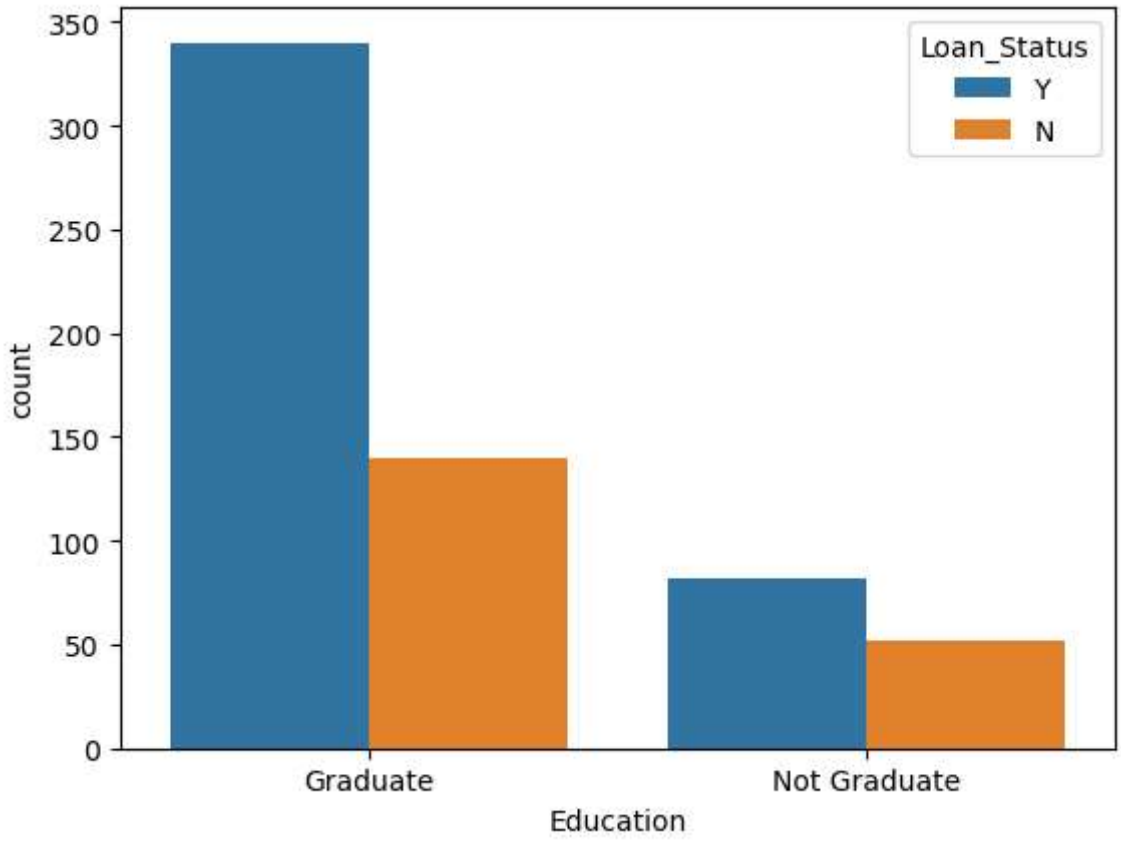


In [20]:
```
df['Education'].value_counts(dropna=False)
```

Out[20]:
```
Graduate        480
Not Graduate    134
Name: Education, dtype: int64
```

In [21]:
```
sns.countplot(x=df['Education'],hue=df['Loan_Status'])
```

Out[21]:
```
<AxesSubplot:xlabel='Education', ylabel='count'>
```



we can see graduates have higher chances of getting loan

In [22]:
```
df['Self_Employed'].value_counts(dropna=False)
```

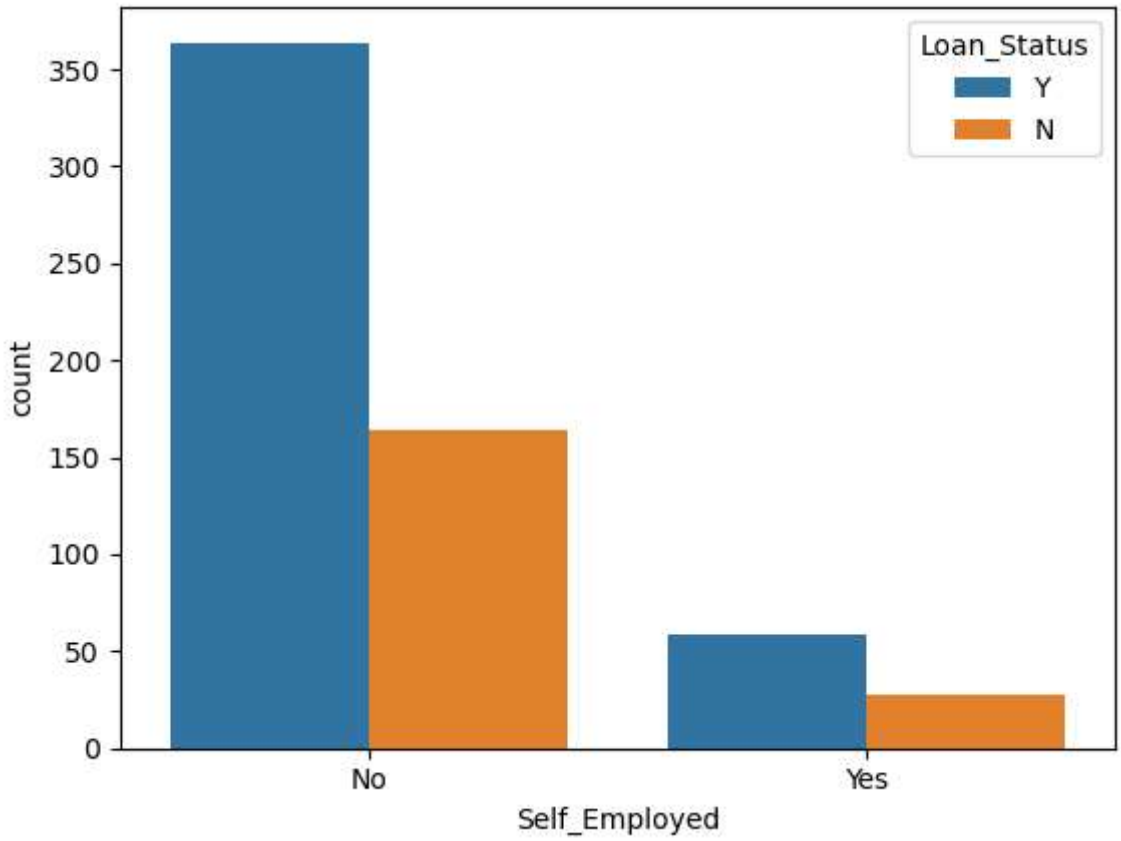
Out[22]:
```
No     500
Yes     82
NaN     32
Name: Self_Employed, dtype: int64
```

bfill used for filling null values

In [23]:
```
# bfill used for filling null values
df['Self_Employed']=df['Self_Employed'].bfill()
```

In [24]:
```
sns.countplot(x=df['Self_Employed'],hue=df['Loan_Status'])
```

Out[24]:
```
<AxesSubplot:xlabel='Self_Employed', ylabel='count'>
```



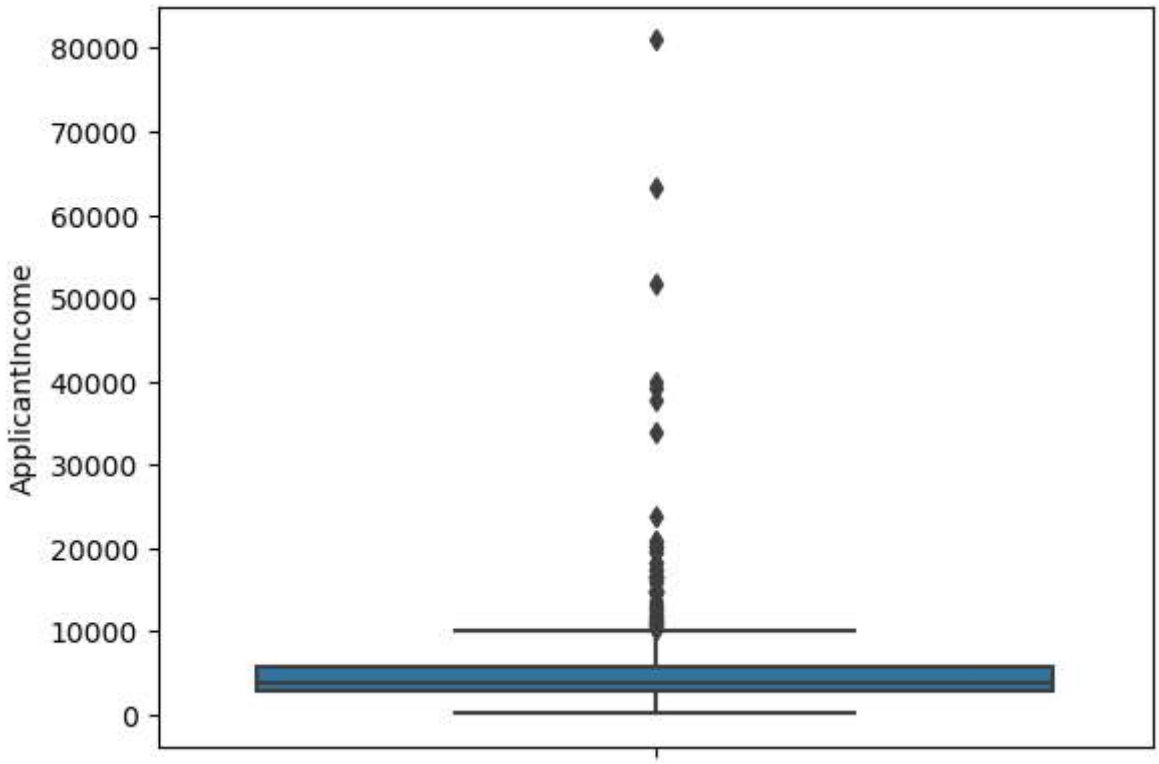
for self emloyed coustomers have less change of loan approval

In [25]:
```
df['ApplicantIncome'].value_counts(dropna=False)
```

Out[25]:
```
2500    9
4583    6
6000    6
2600    6
3333    5
       ..
3244    1
4408    1
3917    1
3992    1
7583    1
Name: ApplicantIncome, Length: 505, dtype: int64
```

In [26]:
```
#visualizing  ApplicantIncome numeric variable
sns.boxplot(y=df['ApplicantIncome'])
```

Out[26]:
```
<AxesSubplot:ylabel='ApplicantIncome'>
```



In [27]:
```
#checking percentile values
df['ApplicantIncome'].describe(percentiles=(0.01,0.05,0.1,0.25,0.5,0.75,0.9,0.95,0.99)
```

Out[27]:
```
count      614.000000
mean      5403.459283
std       6109.041673
min        150.000000
1%        1025.000000
5%        1897.550000
10%       2216.100000
25%       2877.500000
50%       3812.500000
75%       5795.000000
90%       9459.900000
95%      14583.000000
99%      32540.410000
max      81000.000000
Name: ApplicantIncome, dtype: float64
```

capping

capping is an outlier technic

uses for removing outliers

removes top&bottom 1 %

In [28]:
```
Q1=df['ApplicantIncome'].quantile(0.01)
Q3=df['ApplicantIncome'].quantile(0.99)
```

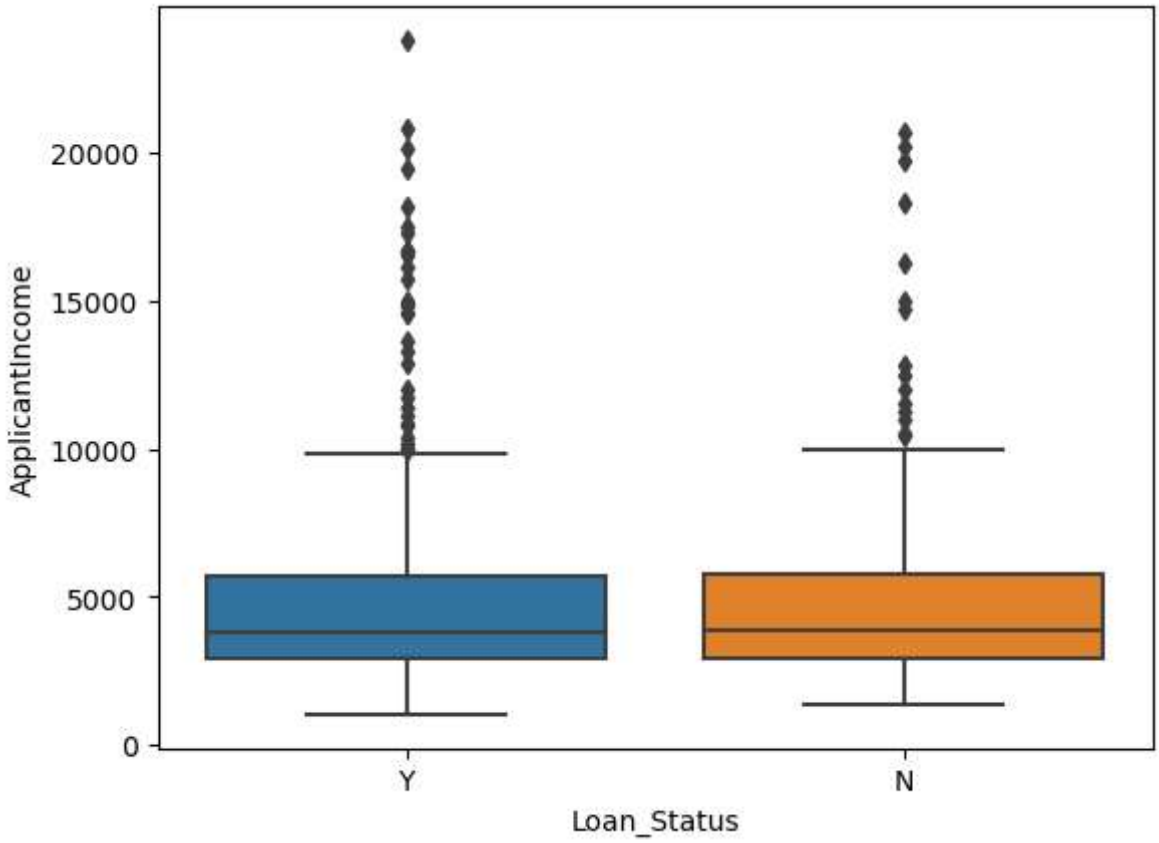
In [29]:
```
df=df[df['ApplicantIncome']<=Q3]
df=df[df['ApplicantIncome']>=Q1]
```

In [30]:
```
df.shape
```

Out[30]:
```
(601, 13)
```

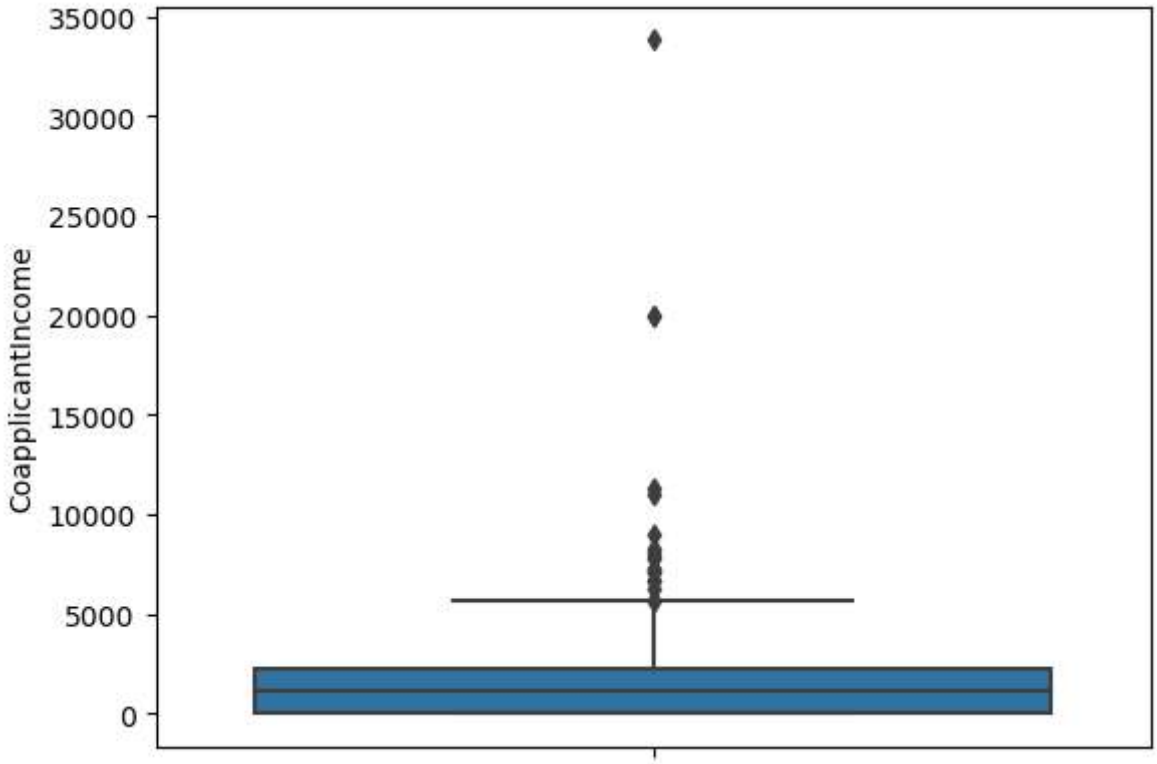
In [31]:
```
#checking Spread of ApplicantIncome vs Loan_Status variable
sns.boxplot(y=df['ApplicantIncome'],x=df['Loan_Status'])
```

Out[31]:
```
<AxesSubplot:xlabel='Loan_Status', ylabel='ApplicantIncome'>
```



In [32]:
```
#visualizing spread of CoapplicantIncome numeric variable
sns.boxplot(y=df['CoapplicantIncome'])
```

Out[32]:
```
<AxesSubplot:ylabel='CoapplicantIncome'>
```



In [33]:
```
#checking percentiles for CoapplicantIncome
df['CoapplicantIncome'].describe(percentiles=(0.01,0.05,0.1,0.25,0.5,0.75,0.9,0.95,0.9
```

Out[33]:
```
count      601.000000
mean      1551.264426
std       2447.952413
min          0.000000
1%           0.000000
5%           0.000000
10%          0.000000
25%          0.000000
50%       1167.000000
75%       2254.000000
90%       3750.000000
95%       4983.000000
99%       8333.000000
max      33837.000000
Name: CoapplicantIncome, dtype: float64
```

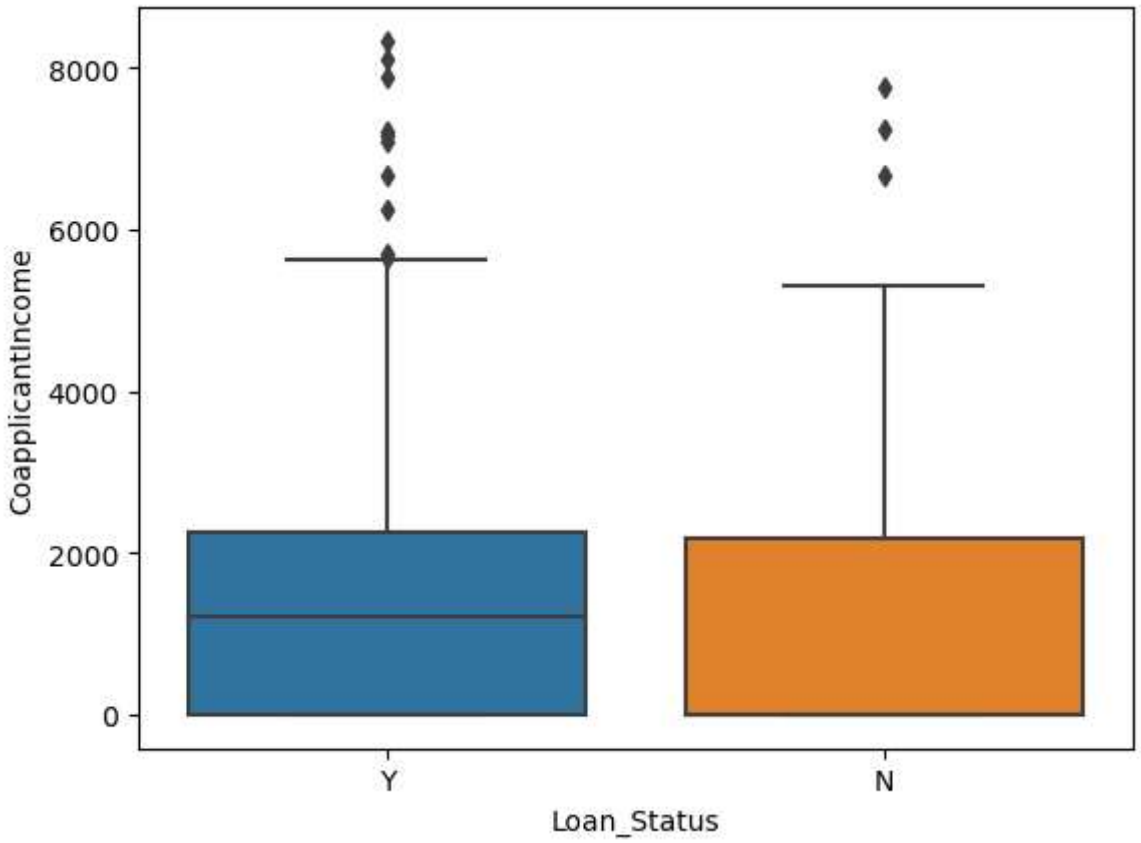
In [34]:
```
#outlier treatment - Remove top&bottom 1%
Q1=df['CoapplicantIncome'].quantile(0.01)
Q3=df['CoapplicantIncome'].quantile(0.99)
df=df[df['CoapplicantIncome']<=Q3]
df=df[df['CoapplicantIncome']>=Q1]
```

In [35]:
```
df.shape
```

Out[35]:
```
(595, 13)
```

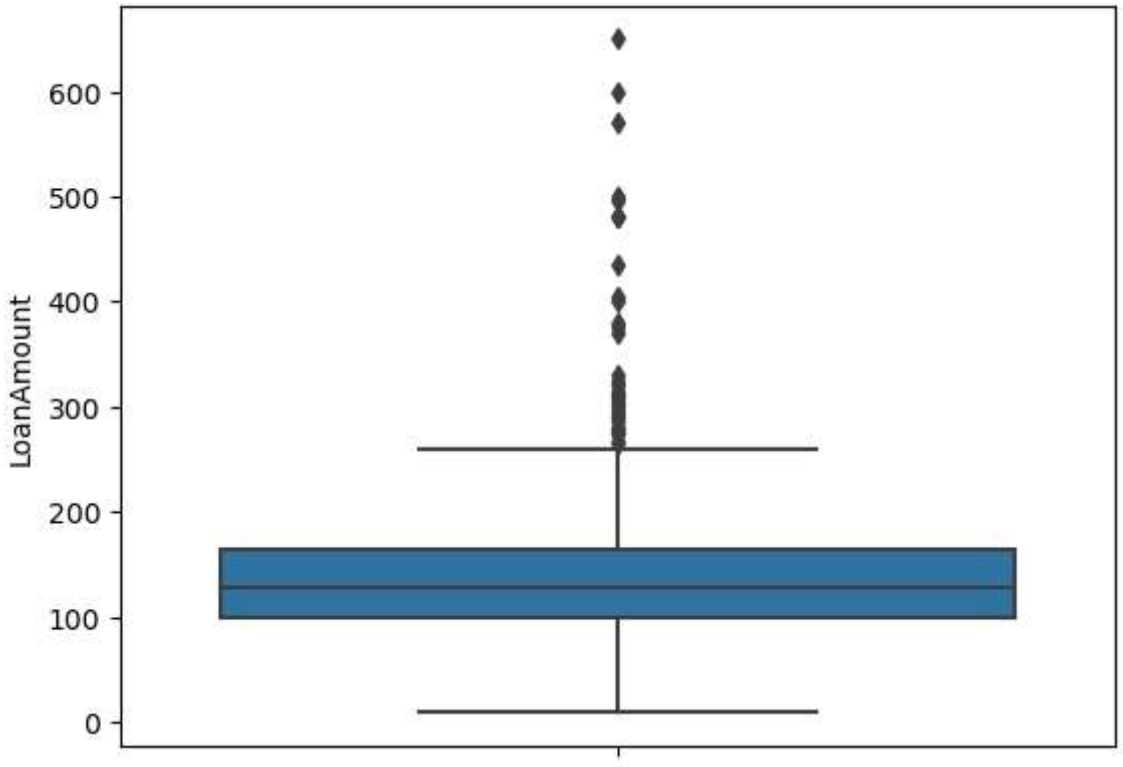
In [36]:
```
#boxplot of  CoapplicantIncome vs Loan_Status variable
sns.boxplot(y=df['CoapplicantIncome'],x=df['Loan_Status'])
```

Out[36]:
```
<AxesSubplot:xlabel='Loan_Status', ylabel='CoapplicantIncome'>
```



In [37]:
```
#visualizing spread of LoanAmount numeric variable
sns.boxplot(y=df['LoanAmount'])
```

Out[37]:
```
<AxesSubplot:ylabel='LoanAmount'>
```



In [38]:
```
df['LoanAmount'].isnull().sum()
```

Out[38]:
```
22
```

In [39]:
```
#checking percentiles for "LoanAmount"
df['LoanAmount'].describe(percentiles=(0.01,0.05,0.1,0.25,0.5,0.75,0.9,0.95,0.99))
```

Out[39]:
```
count    573.000000
mean     142.420593
std       77.182008
min        9.000000
1%        30.000000
5%        55.600000
10%       71.000000
25%      100.000000
50%      127.000000
75%      165.000000
90%      217.600000
95%      276.600000
99%      480.000000
max      650.000000
Name: LoanAmount, dtype: float64
```

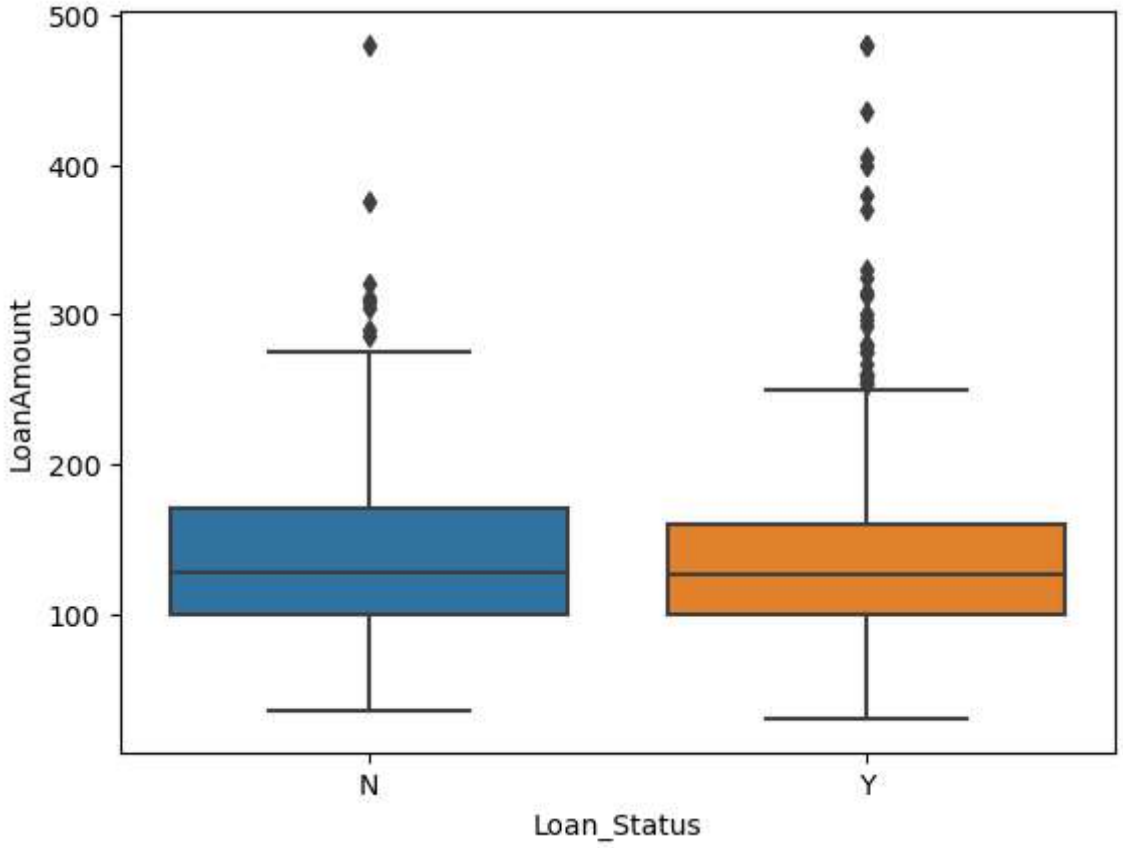
In [40]:
```
#Outlier Treatment: Remove top & bottom 1%
Q1=df['LoanAmount'].quantile(0.01)
Q3=df['LoanAmount'].quantile(0.99)
df=df[df['LoanAmount']<=Q3]
df=df[df['LoanAmount']>=Q1]
```

In [41]:
```
df.shape
```

Out[41]:
```
(563, 13)
```

In [42]:
```
#checking Spread of "LoanAmount " vs Loan_Status variable
sns.boxplot(y=df['LoanAmount'],x=df['Loan_Status'])
```

Out[42]:
```
<AxesSubplot:xlabel='Loan_Status', ylabel='LoanAmount'>
```



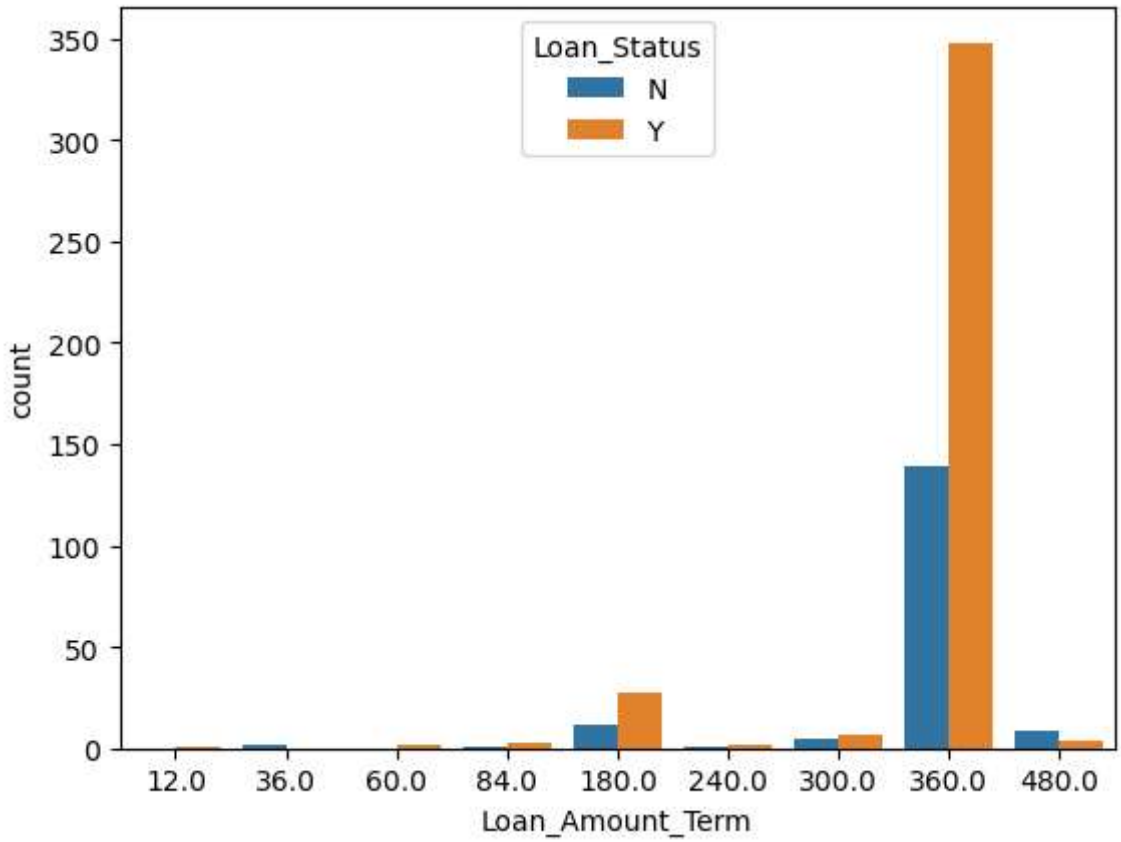
In [43]:
```
df['Loan_Amount_Term'].value_counts(dropna=False)
```

Out[43]:
```
360.0    473
180.0     39
NaN       14
480.0     13
300.0     12
84.0       4
240.0      3
60.0       2
36.0       2
12.0       1
Name: Loan_Amount_Term, dtype: int64
```

In [44]:
```
df['Loan_Amount_Term']=df['Loan_Amount_Term'].bfill()
```

In [45]:
```
sns.countplot(x=df['Loan_Amount_Term'],hue=df['Loan_Status'])
```

Out[45]:
```
<AxesSubplot:xlabel='Loan_Amount_Term', ylabel='count'>
```



In [46]:
```
#columns in the dataset
df.columns
```

Out[46]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [47]:
```
df['Credit_History'].value_counts(dropna=False,normalize=True)
```

Out[47]:
```
1.0    0.772647
0.0    0.143872
NaN    0.083481
Name: Credit_History, dtype: float64
```

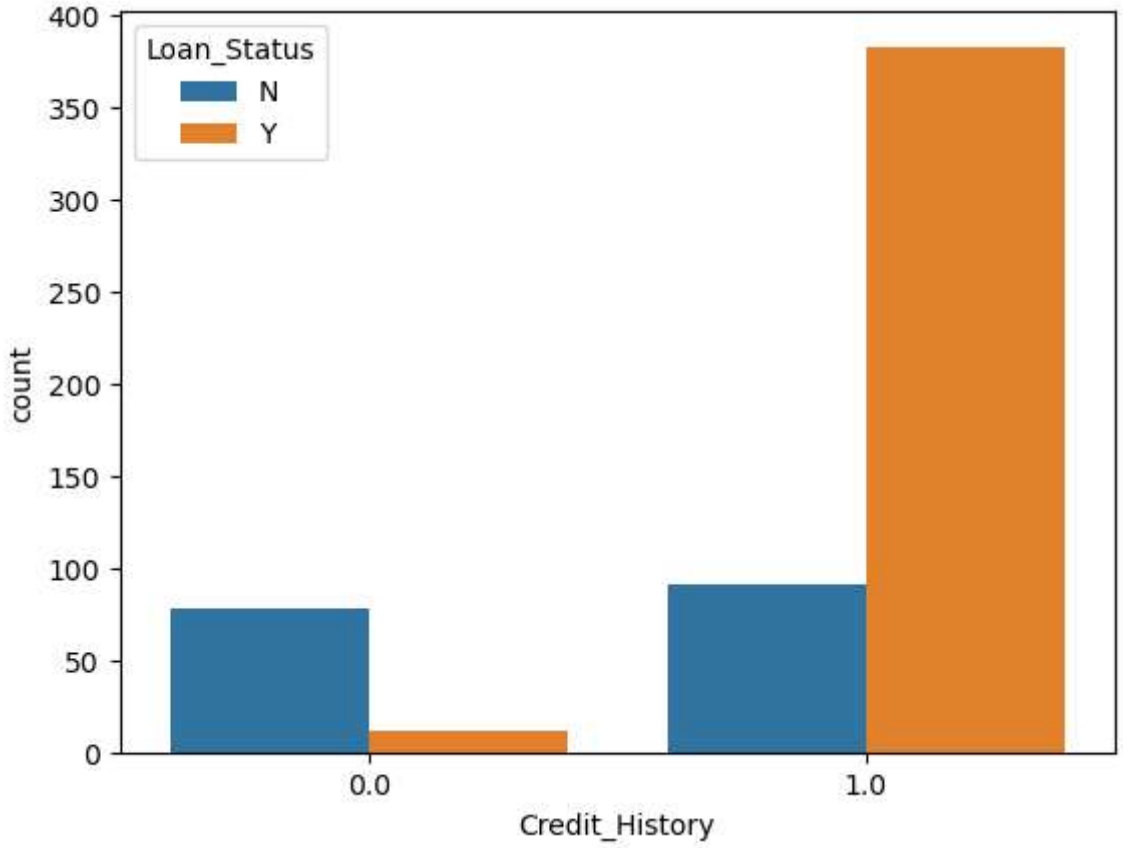
In [48]:
```
df['Credit_History']=df['Credit_History'].bfill()
```

In [49]:
```
df['Credit_History'].value_counts(dropna=False,normalize=True)
```

Out[49]:
```
1.0    0.841918
0.0    0.158082
Name: Credit_History, dtype: float64
```

In [50]:
```
sns.countplot(x=df['Credit_History'],hue=df['Loan_Status'])
```

Out[50]:
```
<AxesSubplot:xlabel='Credit_History', ylabel='count'>
```



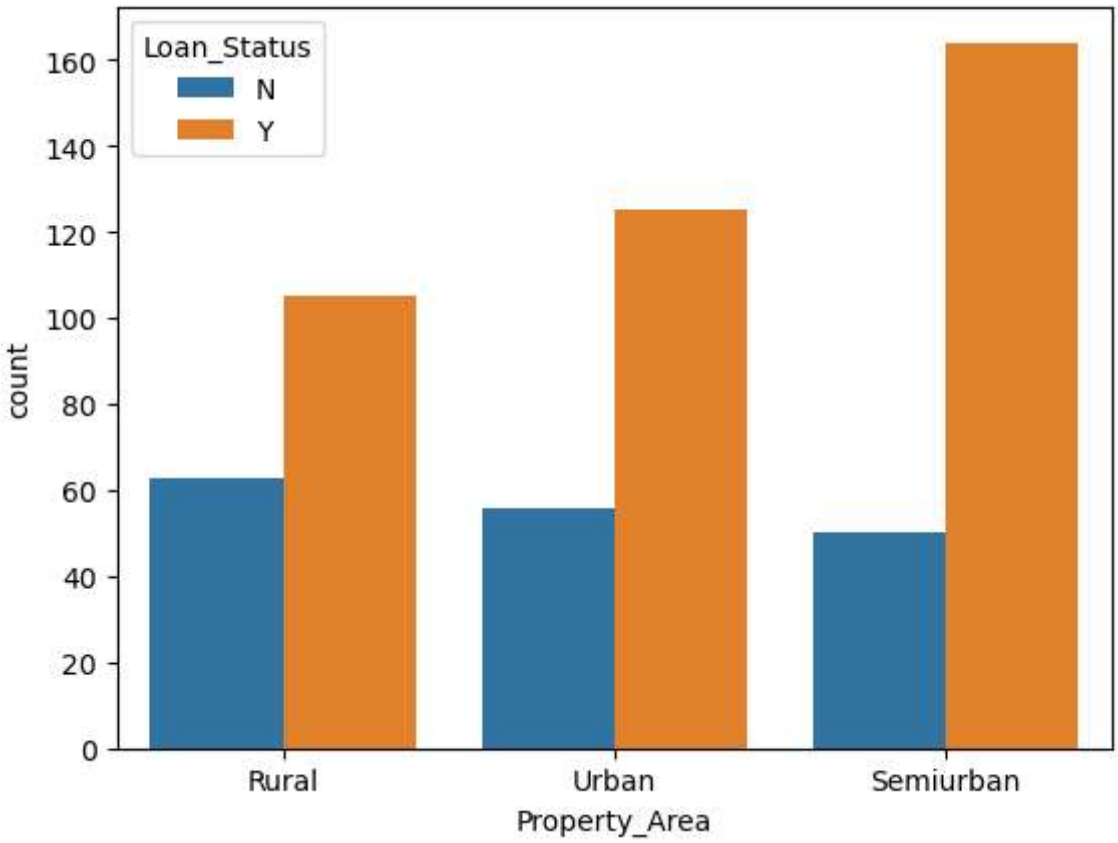
creditscore having 1 have high chances of getting approval

In [51]:
```
df['Property_Area'].value_counts(dropna=False,normalize=True)
```

Out[51]:
```
Semiurban    0.380107
Urban        0.321492
Rural        0.298401
Name: Property_Area, dtype: float64
```

In [52]:
```
sns.countplot(x=df['Property_Area'],hue=df['Loan_Status'])
```

Out[52]:
```
<AxesSubplot:xlabel='Property_Area', ylabel='count'>
```



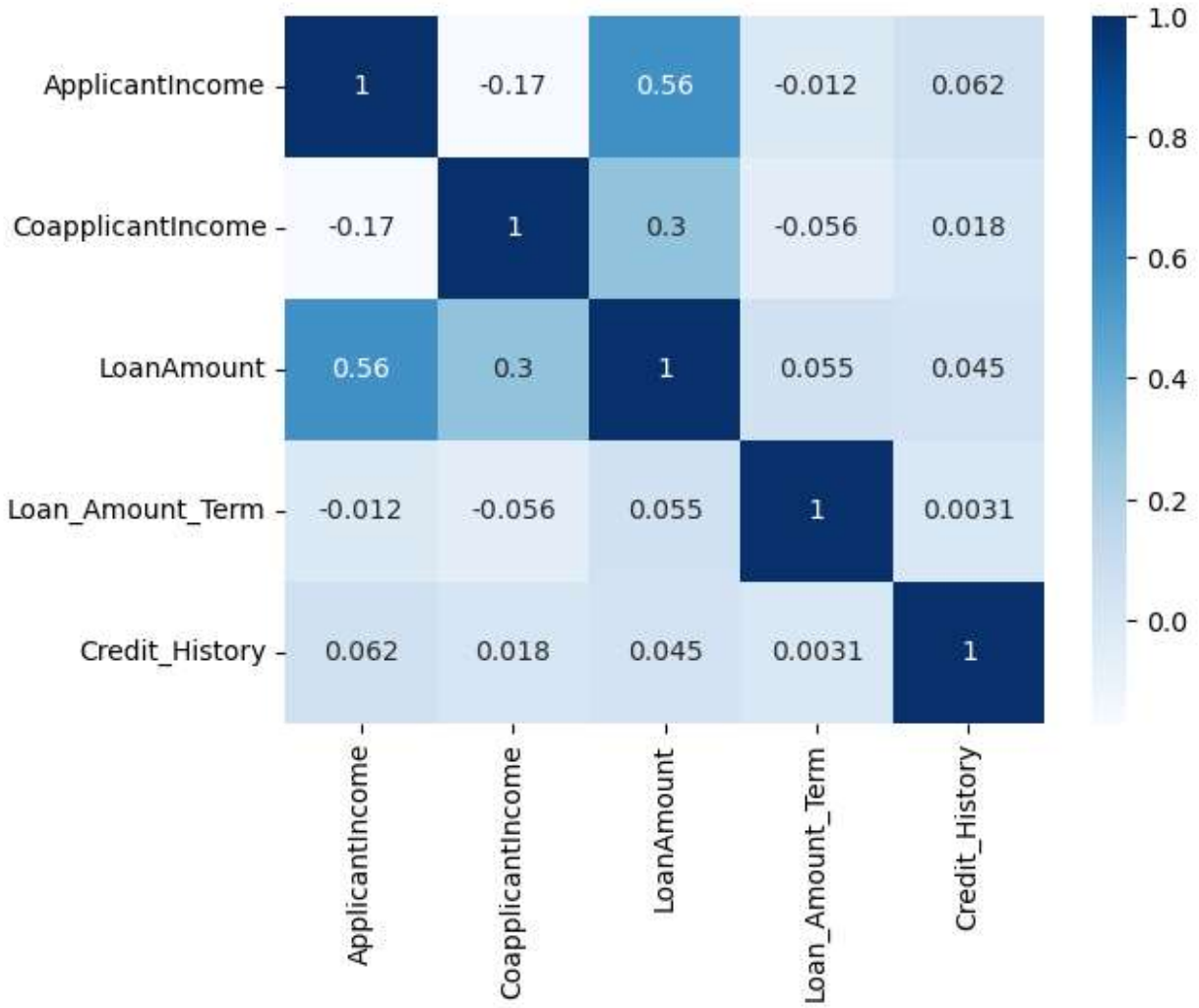
we can see people who are living in semiurban have high chance of getting loan approval

In [53]:
```
df['LoanAmount'].value_counts(dropna=False,normalize=True)
```

Out[53]:
```
120.0    0.033748
110.0    0.028419
100.0    0.026643
160.0    0.021314
187.0    0.021314
           ...
89.0     0.001776
54.0     0.001776
78.0     0.001776
436.0    0.001776
253.0    0.001776
Name: LoanAmount, Length: 187, dtype: float64
```

In [54]:
```
#Visualization of correlation using Heat map
sns.heatmap(df.corr(),annot=True,cmap='Blues')
```

Out[54]:
```
<AxesSubplot:>
```



In [55]:
```
#creating dummy
df.select_dtypes(include=object).columns
```

Out[55]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [56]:
```
dummy=pd.get_dummies(df['Property_Area'],prefix='Property_Area')
dummy
```

Out[56]:

	Property_Area_Rural	Property_Area_Semiurban	Property_Area_Urban
1	1	0	0
2	0	0	1
3	0	0	1
4	0	0	1
5	0	0	1
...	...	...	...
609	1	0	0
610	1	0	0
611	0	0	1
612	0	0	1
613	0	1	0

563 rows × 3 columns

In [57]:
```
df=pd.concat([df,dummy],1)
df=df.drop('Property_Area',1)
df
```

```
C:\Users\gunnu\AppData\Local\Temp\ipykernel_3032\2188141727.py:1: FutureWarning: In a
future version of pandas all arguments of concat except for the argument 'objs' will
be keyword-only.
  df=pd.concat([df,dummy],1)
C:\Users\gunnu\AppData\Local\Temp\ipykernel_3032\2188141727.py:2: FutureWarning: In a
future version of pandas all arguments of DataFrame.drop except for the argument 'lab
els' will be keyword-only.
  df=df.drop('Property_Area',1)
```

Out[57]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplica
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	
5	LP001011	Male	Yes	2	Graduate	Yes	5417	
...	...	...	...	...	...	...	...	
609	LP002978	Female	No	0	Graduate	No	2900	
610	LP002979	Male	Yes	3	Graduate	No	4106	
611	LP002983	Male	Yes	1	Graduate	No	8072	
612	LP002984	Male	Yes	2	Graduate	No	7583	
613	LP002990	Female	No	0	Graduate	Yes	4583	

563 rows × 15 columns

In [58]:

```
dummy1=pd.get_dummies(df['Gender'],prefix='gender')
df=pd.concat([df,dummy1],1)
dummy2=pd.get_dummies(df['Education'],prefix='Education')
df=pd.concat([df,dummy2],1)
df
```

```
C:\Users\gunnu\AppData\Local\Temp\ipykernel_3032\1941449558.py:2: FutureWarning: In a
future version of pandas all arguments of concat except for the argument 'objs' will
be keyword-only.
  df=pd.concat([df,dummy1],1)
C:\Users\gunnu\AppData\Local\Temp\ipykernel_3032\1941449558.py:4: FutureWarning: In a
future version of pandas all arguments of concat except for the argument 'objs' will
be keyword-only.
  df=pd.concat([df,dummy2],1)
```

Out[58]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplica
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	
5	LP001011	Male	Yes	2	Graduate	Yes	5417	
...	...	...	...	...	...	...	...	
609	LP002978	Female	No	0	Graduate	No	2900	
610	LP002979	Male	Yes	3	Graduate	No	4106	
611	LP002983	Male	Yes	1	Graduate	No	8072	
612	LP002984	Male	Yes	2	Graduate	No	7583	
613	LP002990	Female	No	0	Graduate	Yes	4583	

563 rows × 19 columns

In [59]:
```
#dropping the original columns after dummy creation
d_col=['Gender','Education','Loan_ID']
df.drop(d_col,axis=1,inplace=True)
```

In [60]:
```
v =  ['Loan_Status']
def binary_map(x):
    return x.map({'Y': 1, "N": 0})
df[v] = df[v].apply(binary_map)
```

In [61]:
```
varlist =  ['Married','Self_Employed']
def binary_map(x):
    return x.map({'Yes': 1, "No": 0})
df[varlist] = df[varlist].apply(binary_map)
```

In [62]:
```
df
```

Out[62]:

	Married	Dependents	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_
1	1	1	0	4583	1508.0	128.0	
2	1	0	1	3000	0.0	66.0	
3	1	0	0	2583	2358.0	120.0	
4	0	0	0	6000	0.0	141.0	
5	1	2	1	5417	4196.0	267.0	
...	...	...	...	...	...	...	
609	0	0	0	2900	0.0	71.0	
610	1	3	0	4106	0.0	40.0	
611	1	1	0	8072	240.0	253.0	
612	1	2	0	7583	0.0	187.0	
613	0	0	1	4583	0.0	133.0	

563 rows × 16 columns

In [63]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 563 entries, 1 to 613
Data columns (total 16 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Married                  563 non-null    int64  
 1   Dependents               563 non-null    object 
 2   Self_Employed            563 non-null    int64  
 3   ApplicantIncome          563 non-null    int64  
 4   CoapplicantIncome        563 non-null    float64
 5   LoanAmount               563 non-null    float64
 6   Loan_Amount_Term         563 non-null    float64
 7   Credit_History           563 non-null    float64
 8   Loan_Status              563 non-null    int64  
 9   Property_Area_Rural      563 non-null    uint8  
 10  Property_Area_Semiurban  563 non-null    uint8  
 11  Property_Area_Urban      563 non-null    uint8  
 12  gender_Female            563 non-null    uint8  
 13  gender_Male              563 non-null    uint8  
 14  Education_Graduate       563 non-null    uint8  
 15  Education_Not Graduate   563 non-null    uint8  
dtypes: float64(4), int64(4), object(1), uint8(7)
memory usage: 47.8+ KB
```

In [64]:
```
df['Dependents']=pd.to_numeric(df['Dependents'])
df['Credit_History']=pd.to_numeric(df['Credit_History'])
```

In [65]:
```
from sklearn.model_selection import train_test_split
y = df['Loan_Status']
X = df.drop('Loan_Status',axis=1)
```

In [66]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,train_size=0.7,random_state=50)
```

In [67]:
```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 394 entries, 34 to 521
Data columns (total 15 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Married                  394 non-null    int64  
 1   Dependents               394 non-null    int64  
 2   Self_Employed            394 non-null    int64  
 3   ApplicantIncome          394 non-null    int64  
 4   CoapplicantIncome        394 non-null    float64
 5   LoanAmount               394 non-null    float64
 6   Loan_Amount_Term         394 non-null    float64
 7   Credit_History           394 non-null    float64
 8   Property_Area_Rural      394 non-null    uint8  
 9   Property_Area_Semiurban  394 non-null    uint8  
 10  Property_Area_Urban      394 non-null    uint8  
 11  gender_Female            394 non-null    uint8  
 12  gender_Male              394 non-null    uint8  
 13  Education_Graduate       394 non-null    uint8  
 14  Education_Not Graduate   394 non-null    uint8  
dtypes: float64(4), int64(4), uint8(7)
memory usage: 30.4 KB
```

In [68]:
```
from sklearn.preprocessing import StandardScaler
Scaler = StandardScaler()
```

In [69]:
```
num_cols =X_train.select_dtypes(include=['float64', 'int64']).columns
num_cols
```

Out[69]:
```
Index(['Married', 'Dependents', 'Self_Employed', 'ApplicantIncome',
       'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term',
       'Credit_History'],
      dtype='object')
```

In [70]:
```
X_train[num_cols] = Scaler.fit_transform(X_train[num_cols])
X_train
```

Out[70]:

	Married	Dependents	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan
34	-1.416910	2.208703	-0.402793	2.432051	0.918465	2.808206	
390	-1.416910	2.208703	-0.402793	1.369937	-0.863334	0.695737	
457	0.705761	-0.761362	-0.402793	-0.369661	0.662480	0.507962	
295	0.705761	2.208703	-0.402793	-0.144045	-0.120918	-0.477857	
482	0.705761	-0.761362	-0.402793	-0.887493	1.007555	-0.196195	
...	...	...	...	...	...	...	
75	-1.416910	-0.761362	-0.402793	-0.356277	-0.863334	-0.430914	
148	-1.416910	-0.761362	-0.402793	1.635386	0.126158	1.321653	
315	0.705761	0.228660	-0.402793	-0.468128	0.110716	-0.462209	
121	-1.416910	-0.761362	-0.402793	-0.223712	-0.863334	-1.510620	
521	-1.416910	-0.761362	-0.402793	-0.754609	-0.863334	-1.338493	

394 rows × 15 columns

In [71]:
```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 394 entries, 34 to 521
Data columns (total 15 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Married                  394 non-null    float64
 1   Dependents               394 non-null    float64
 2   Self_Employed            394 non-null    float64
 3   ApplicantIncome          394 non-null    float64
 4   CoapplicantIncome        394 non-null    float64
 5   LoanAmount               394 non-null    float64
 6   Loan_Amount_Term         394 non-null    float64
 7   Credit_History           394 non-null    float64
 8   Property_Area_Rural      394 non-null    uint8  
 9   Property_Area_Semiurban  394 non-null    uint8  
 10  Property_Area_Urban      394 non-null    uint8  
 11  gender_Female            394 non-null    uint8  
 12  gender_Male              394 non-null    uint8  
 13  Education_Graduate       394 non-null    uint8  
 14  Education_Not Graduate   394 non-null    uint8  
dtypes: float64(8), uint8(7)
memory usage: 30.4 KB
```

In [72]:
```
X_train.isnull().sum()
```

Out[72]:
```
Married                    0
Dependents                 0
Self_Employed              0
ApplicantIncome            0
CoapplicantIncome          0
LoanAmount                 0
Loan_Amount_Term           0
Credit_History             0
Property_Area_Rural        0
Property_Area_Semiurban    0
Property_Area_Urban        0
gender_Female              0
gender_Male                0
Education_Graduate         0
Education_Not Graduate     0
dtype: int64
```

In [73]:
```
import statsmodels.api as sm
from sklearn.linear_model import LogisticRegression
logreg= LogisticRegression()
```

In [74]:
```
from sklearn.feature_selection import RFE
rfe = RFE(estimator=logreg, n_features_to_select=15) # running RFE with 15 variables c
rfe = rfe.fit(X_train,y_train)
```

In [75]:
```
col = X_train.columns[rfe.support_]
col
```

Out[75]:
```
Index(['Married', 'Dependents', 'Self_Employed', 'ApplicantIncome',
       'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term', 'Credit_History',
       'Property_Area_Rural', 'Property_Area_Semiurban', 'Property_Area_Urban',
       'gender_Female', 'gender_Male', 'Education_Graduate',
       'Education_Not Graduate'],
      dtype='object')
```

In [76]:
```
#BUILDING MODEL #1

X_train_sm = sm.add_constant(X_train[col])
logm1 = sm.GLM(y_train,X_train_sm, family = sm.families.Binomial())
res = logm1.fit()
res.summary()
```

Out[76]:

Generalized Linear Model Regression Results

Dep. Variable:	Loan_Status	**No. Observations:**	394
Model:	GLM	**Df Residuals:**	381
Model Family:	Binomial	**Df Model:**	12
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-180.60
Date:	Sun, 13 Aug 2023	**Deviance:**	361.20
Time:	13:13:40	**Pearson chi2:**	396.
No. Iterations:	12	**Pseudo R-squ. (CS):**	0.2622
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.3996	0.088	4.545	0.000	0.227	0.572
Married	0.2617	0.151	1.728	0.084	-0.035	0.558
Dependents	0.0572	0.147	0.388	0.698	-0.232	0.346
Self_Employed	-0.0506	0.141	-0.360	0.719	-0.326	0.225
ApplicantIncome	0.1236	0.192	0.645	0.519	-0.252	0.499
CoapplicantIncome	0.0673	0.165	0.409	0.682	-0.255	0.390
LoanAmount	-0.1932	0.187	-1.034	0.301	-0.559	0.173
Loan_Amount_Term	0.0202	0.137	0.147	0.883	-0.249	0.290
Credit_History	1.2422	0.147	8.440	0.000	0.954	1.531
Property_Area_Rural	-0.1250	0.189	-0.663	0.507	-0.495	0.245
Property_Area_Semiurban	0.5162	0.193	2.672	0.008	0.138	0.895
Property_Area_Urban	0.0084	0.199	0.042	0.966	-0.381	0.398
gender_Female	0.2339	0.222	1.054	0.292	-0.201	0.669
gender_Male	0.1657	0.166	0.995	0.320	-0.161	0.492
Education_Graduate	0.2295	0.148	1.553	0.121	-0.060	0.519
Education_Not Graduate	0.1701	0.186	0.916	0.359	-0.194	0.534

In [77]:
```
# Check for the VIF values of the feature variables.
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [78]:
```
# Create a dataframe that will contain the names of all the feature variables and thei
vif = pd.DataFrame()
vif['Features'] = X_train[col].columns
vif['VIF'] = [variance_inflation_factor(X_train[col].values, i) for i in range(X_trair
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

```
C:\Users\gunnu\anaconda3\lib\site-packages\statsmodels\stats\outliers_influence.py:19
5: RuntimeWarning: divide by zero encountered in double_scalars
  vif = 1. / (1. - r_squared_i)
```

Out[78]:

	Features	VIF
8	Property_Area_Rural	inf
9	Property_Area_Semiurban	inf
10	Property_Area_Urban	inf
11	gender_Female	inf
12	gender_Male	inf
13	Education_Graduate	inf
14	Education_Not Graduate	inf
5	LoanAmount	1.93
3	ApplicantIncome	1.91
4	CoapplicantIncome	1.49
0	Married	1.34
1	Dependents	1.20
2	Self_Employed	1.12
6	Loan_Amount_Term	1.05
7	Credit_History	1.02

In [79]:
```
col=col.drop('Property_Area_Rural',1)
```

In [80]:
```
#BUILDING MODEL #2
X_train_sm = sm.add_constant(X_train[col])
logm1 = sm.GLM(y_train,X_train_sm, family = sm.families.Binomial())
res = logm1.fit()
res.summary()
```

Out[80]:

Generalized Linear Model Regression Results

Dep. Variable:	Loan_Status	**No. Observations:**	394
Model:	GLM	**Df Residuals:**	381
Model Family:	Binomial	**Df Model:**	12
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-180.60
Date:	Sun, 13 Aug 2023	**Deviance:**	361.20
Time:	13:13:40	**Pearson chi2:**	396.
No. Iterations:	80	**Pseudo R-squ. (CS):**	0.2622
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.3371	0.135	2.503	0.012	0.073	0.601
Married	0.2617	0.151	1.728	0.084	-0.035	0.558
Dependents	0.0572	0.147	0.388	0.698	-0.232	0.346
Self_Employed	-0.0506	0.141	-0.360	0.719	-0.326	0.225
ApplicantIncome	0.1236	0.192	0.645	0.519	-0.252	0.499
CoapplicantIncome	0.0673	0.165	0.409	0.682	-0.255	0.390
LoanAmount	-0.1932	0.187	-1.034	0.301	-0.559	0.173
Loan_Amount_Term	0.0202	0.137	0.147	0.883	-0.249	0.290
Credit_History	1.2422	0.147	8.440	0.000	0.954	1.531
Property_Area_Semiurban	0.6412	0.327	1.960	0.050	0.000	1.282
Property_Area_Urban	0.1334	0.330	0.404	0.686	-0.514	0.781
gender_Female	0.2027	0.232	0.873	0.383	-0.252	0.658
gender_Male	0.1344	0.168	0.800	0.424	-0.195	0.464
Education_Graduate	0.1982	0.159	1.248	0.212	-0.113	0.510
Education_Not Graduate	0.1389	0.190	0.729	0.466	-0.234	0.512

In [81]:

```
vif = pd.DataFrame()
vif['Features'] = X_train[col].columns
vif['VIF'] = [variance_inflation_factor(X_train[col].values, i) for i in range(X_train
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

```
C:\Users\gunnu\anaconda3\lib\site-packages\statsmodels\stats\outliers_influence.py:19
5: RuntimeWarning: divide by zero encountered in double_scalars
  vif = 1. / (1. - r_squared_i)
```

Out[81]:

	Features	VIF
10	gender_Female	inf
11	gender_Male	inf
12	Education_Graduate	inf
13	Education_Not Graduate	inf
5	LoanAmount	1.93
3	ApplicantIncome	1.91
4	CoapplicantIncome	1.49
8	Property_Area_Semiurban	1.41
9	Property_Area_Urban	1.41
0	Married	1.34
1	Dependents	1.20
2	Self_Employed	1.12
6	Loan_Amount_Term	1.05
7	Credit_History	1.02

In [82]:

```
#BUILDING MODEL #3
col=col.drop('gender_Female',1)

X_train_sm = sm.add_constant(X_train[col])
logm1 = sm.GLM(y_train,X_train_sm, family = sm.families.Binomial())
res = logm1.fit()
res.summary()
```

Out[82]:

Generalized Linear Model Regression Results

Dep. Variable:	Loan_Status	**No. Observations:**	394
Model:	GLM	**Df Residuals:**	381
Model Family:	Binomial	**Df Model:**	12
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-180.60
Date:	Sun, 13 Aug 2023	**Deviance:**	361.20
Time:	13:13:40	**Pearson chi2:**	396.
No. Iterations:	6	**Pseudo R-squ. (CS):**	0.2622
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.4722	0.267	1.769	0.077	-0.051	0.995
Married	0.2617	0.151	1.728	0.084	-0.035	0.558
Dependents	0.0572	0.147	0.388	0.698	-0.232	0.346
Self_Employed	-0.0506	0.141	-0.360	0.719	-0.326	0.225
ApplicantIncome	0.1236	0.192	0.645	0.519	-0.252	0.499
CoapplicantIncome	0.0673	0.165	0.409	0.682	-0.255	0.390
LoanAmount	-0.1932	0.187	-1.034	0.301	-0.559	0.173
Loan_Amount_Term	0.0202	0.137	0.147	0.883	-0.249	0.290
Credit_History	1.2422	0.147	8.440	0.000	0.954	1.531
Property_Area_Semiurban	0.6412	0.327	1.960	0.050	0.000	1.282
Property_Area_Urban	0.1334	0.330	0.404	0.686	-0.514	0.781
gender_Male	-0.0682	0.382	-0.179	0.858	-0.817	0.681
Education_Graduate	0.2658	0.188	1.415	0.157	-0.102	0.634
Education_Not Graduate	0.2064	0.230	0.898	0.369	-0.244	0.657

In [83]:

```
vif = pd.DataFrame()
vif['Features'] = X_train[col].columns
vif['VIF'] = [variance_inflation_factor(X_train[col].values, i) for i in range(X_train
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[83]:

	Features	VIF
11	Education_Graduate	7.36
12	Education_Not Graduate	3.08
5	LoanAmount	1.93
3	ApplicantIncome	1.91
4	CoapplicantIncome	1.49
8	Property_Area_Semiurban	1.41
9	Property_Area_Urban	1.41
0	Married	1.34
10	gender_Male	1.24
1	Dependents	1.20
2	Self_Employed	1.12
6	Loan_Amount_Term	1.05
7	Credit_History	1.02

In [84]:

```
#BUILDING MODEL #3
col=col.drop('Education_Graduate',1)

X_train_sm = sm.add_constant(X_train[col])
logm1 = sm.GLM(y_train,X_train_sm, family = sm.families.Binomial())
res = logm1.fit()
res.summary()
```

Out[84]:

Generalized Linear Model Regression Results

Dep. Variable:	Loan_Status	**No. Observations:**	394
Model:	GLM	**Df Residuals:**	381
Model Family:	Binomial	**Df Model:**	12
Link Function:	Logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-180.60
Date:	Sun, 13 Aug 2023	**Deviance:**	361.20
Time:	13:13:40	**Pearson chi2:**	396.
No. Iterations:	5	**Pseudo R-squ. (CS):**	0.2622
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.7380	0.400	1.844	0.065	-0.047	1.523
Married	0.2617	0.151	1.728	0.084	-0.035	0.558
Dependents	0.0572	0.147	0.388	0.698	-0.232	0.346
Self_Employed	-0.0506	0.141	-0.360	0.719	-0.326	0.225
ApplicantIncome	0.1236	0.192	0.645	0.519	-0.252	0.499
CoapplicantIncome	0.0673	0.165	0.409	0.682	-0.255	0.390
LoanAmount	-0.1932	0.187	-1.034	0.301	-0.559	0.173
Loan_Amount_Term	0.0202	0.137	0.147	0.883	-0.249	0.290
Credit_History	1.2422	0.147	8.440	0.000	0.954	1.531
Property_Area_Semiurban	0.6412	0.327	1.960	0.050	0.000	1.282
Property_Area_Urban	0.1334	0.330	0.404	0.686	-0.514	0.781
gender_Male	-0.0682	0.382	-0.179	0.858	-0.817	0.681
Education_Not Graduate	-0.0593	0.324	-0.183	0.855	-0.694	0.575

In [85]:

```
vif = pd.DataFrame()
vif['Features'] = X_train[col].columns
vif['VIF'] = [variance_inflation_factor(X_train[col].values, i) for i in range(X_trai
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[85]:

	Features	VIF
10	gender_Male	2.75
5	LoanAmount	1.93
3	ApplicantIncome	1.91
8	Property_Area_Semiurban	1.70
9	Property_Area_Urban	1.69
4	CoapplicantIncome	1.48
11	Education_Not Graduate	1.38
0	Married	1.25
1	Dependents	1.19
2	Self_Employed	1.12
6	Loan_Amount_Term	1.05
7	Credit_History	1.02

In [86]:
```
y_train_pred=res.predict(X_train_sm)
y_train_pred[:10]
```

Out[86]:
```
34     0.695057
390    0.727285
457    0.805940
295    0.853134
482    0.883600
552    0.823216
329    0.751549
3      0.813366
497    0.824725
389    0.802651
dtype: float64
```

In [87]:
```
y_train_pred=y_train_pred.values.reshape(-1)
y_train_pred[:10]
```

Out[87]:
```
array([0.69505722, 0.7272854 , 0.80593984, 0.853134  , 0.88359987,
       0.82321638, 0.75154945, 0.81336584, 0.82472469, 0.80265075])
```

In [88]:
```
y_train_pred[:10]
```

Out[88]:
```
array([0.69505722, 0.7272854 , 0.80593984, 0.853134  , 0.88359987,
       0.82321638, 0.75154945, 0.81336584, 0.82472469, 0.80265075])
```

In [89]:
```
y_train_pred_final=pd.DataFrame({'Status':y_train.values,'Status_prob':y_train_pred})
y_train_pred_final['Prospect ID'] = y_train.index
y_train_pred_final
```

Out[89]:

	Status	Status_prob	Prospect ID
0	0	0.695057	34
1	1	0.727285	390
2	0	0.805940	457
3	1	0.853134	295
4	1	0.883600	482
...	...	...	...
389	0	0.728456	75
390	0	0.702170	148
391	1	0.817917	315
392	1	0.851934	121
393	1	0.829596	521

394 rows × 3 columns

In [90]:
```
y_train_pred_final['Predicted'] = y_train_pred_final.Status_prob.map(lambda x: 1 if x

# Let's see the head
y_train_pred_final.head(25)
```

Out[90]:

	Status	Status_prob	Prospect ID	Predicted
0	0	0.695057	34	1
1	1	0.727285	390	1
2	0	0.805940	457	1
3	1	0.853134	295	1
4	1	0.883600	482	1
5	1	0.823216	552	1
6	1	0.751549	329	1
7	1	0.813366	3	1
8	1	0.824725	497	1
9	1	0.802651	389	1
10	1	0.838602	394	1
11	1	0.131685	411	0
12	1	0.802965	441	1
13	0	0.094444	412	0
14	1	0.889715	560	1
15	1	0.836452	542	1
16	0	0.097497	396	0
17	1	0.812971	68	1
18	1	0.820237	176	1
19	1	0.904182	335	1
20	1	0.722824	197	1
21	1	0.828069	249	1
22	0	0.226352	7	0
23	0	0.147294	569	0
24	1	0.884396	234	1

In [91]:
```
from sklearn import metrics

# Confusion matrix
confusion = metrics.confusion_matrix(y_train_pred_final.Status, y_train_pred_final.Pre
print(confusion)
```

```
[[ 56  62]
 [  9 267]]
```

In [92]:
```
# Let's check the overall accuracy.
print(metrics.accuracy_score(y_train_pred_final.Status, y_train_pred_final.Predicted))
```

```
0.8197969543147208
```

In [93]:
```
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

In [94]:
```
# Lets  see the sensitivity of our logistic regression model
TP / float(TP+FN)
```

Out[94]:
```
0.967391304347826
```

In [95]:
```
# Let us calculate specificity
TN / float(TN+FP)
```

Out[95]:
```
0.4745762711864407
```

In [96]:
```
# Calculate False Postive Rate - predicting conversion when customer does not have cor
print(FP/ float(TN+FP))
```

```
0.5254237288135594
```

In [97]:
```

print (TP / float(TP+FP))#positive predictive value
```

```
0.8115501519756839
```

In [98]:
```
# Negative predictive value
print (TN / float(TN+ FN))
```

```
0.8615384615384616
```

In [99]:
```
# prediction test
```

In [100…
```
num_cols=X_test.select_dtypes(include=['float64','int64']).columns
num_cols
```

Out[100]:
```
Index(['Married', 'Dependents', 'Self_Employed', 'ApplicantIncome',
       'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term',
       'Credit_History'],
      dtype='object')
```

In [101…
```
X_test[num_cols]=Scaler.transform(X_test[num_cols])
X_test=X_test[col]
X_test
```

Out[101]:

	Married	Dependents	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan
240	0.705761	1.218681	-0.402793	0.303043	2.106331	-0.321378	
123	0.705761	1.218681	-0.402793	-0.608979	-0.863334	-0.931647	
310	-1.416910	-0.761362	-0.402793	-0.621725	-0.863334	-0.884703	
227	0.705761	1.218681	-0.402793	0.440388	0.143382	1.086935	
502	0.705761	1.218681	-0.402793	-0.000964	2.476946	1.055639	
...	...	...	...	...	...	...	...
23	0.705761	1.218681	-0.402793	-0.478963	0.275236	-0.446561	
110	-1.416910	-0.761362	-0.402793	0.142754	-0.863334	-0.071011	
530	0.705761	-0.761362	-0.402793	-1.224641	2.403298	1.180822	
406	0.705761	-0.761362	-0.402793	-0.860725	0.562105	-0.399618	
511	0.705761	0.228660	-0.402793	0.381435	0.326908	1.712851	

169 rows × 12 columns

In [102…
```
X_test_sm=sm.add_constant(X_test)
y_test_pred=res.predict(X_test_sm)
y_test_pred.head()
```

Out[102]:
```
240    0.851226
123    0.899425
310    0.829168
227    0.880611
502    0.891563
dtype: float64
```

In [103…
```
y_test_pred=y_test_pred.values.reshape(-1)
```

In [104…
```
# Converting y_pred to a dataframe which is an array
y_pred_final = pd.DataFrame({'Status':y_test.values,'Status_prob':y_test_pred})
y_pred_final['Prospect ID'] = y_test.index
```

In [105…
```
# Let's see the head of y_pred_final
y_pred_final.head()
```

Out[105]:

	Status	Status_prob	Prospect ID
0	1	0.851226	240
1	1	0.899425	123
2	1	0.829168	310
3	1	0.880611	227
4	1	0.891563	502

In [106…
```
y_pred_final['final_predicted'] = y_pred_final.Status_prob.map(lambda x: 1 if x > 0.3
y_pred_final.head()
```

Out[106]:

	Status	Status_prob	Prospect ID	final_predicted
0	1	0.851226	240	1
1	1	0.899425	123	1
2	1	0.829168	310	1
3	1	0.880611	227	1
4	1	0.891563	502	1

In [107…
```
metrics.accuracy_score(y_pred_final.Status, y_pred_final.final_predicted)
```

Out[107]:
```
0.8165680473372781
```

In [108…
```
confusion2 = metrics.confusion_matrix(y_pred_final.Status, y_pred_final.final_predicte
confusion2
```

Out[108]:
```
array([[ 22,  29],
       [  2, 116]], dtype=int64)
```

In [109…
```
TP = confusion2[1,1] # true positive
TN = confusion2[0,0] # true negatives
FP = confusion2[0,1] # false positives
FN = confusion2[1,0] # false negatives
```

In [110…
```
#Let's see the sensitivity of our logistic regression model
TP / float(TP+FN)
```

Out[110]:
```
0.9830508474576272
```

In [111…
```
TN / float(TN+FP)
#for caclucalation specificity
```

Out[111]:
```
0.43137254901960786
```

In [112…
```
metrics.precision_score(y_pred_final.Status , y_pred_final.final_predicted)
```

Out[112]:
```
0.8
```

In [113…
```
metrics.recall_score(y_pred_final.Status, y_pred_final.final_predicted)
```

Out[113]:
```
0.9830508474576272
```

In [ ]: