I will be trying to analyse the given loan data to find which all factors are impacting for loan approval

In [1177]:
```
#import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import chi2_contingency,ttest_ind,f_oneway
from sklearn.preprocessing import LabelEncoder,MinMaxScaler
from category_encoders import TargetEncoder
```

In [1178]:
```
#load the data set
df=pd.read_csv("loan.csv")
dependent_att=list()
#Viewing the content
df.head()
```

Out[1178]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coappli
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 [1179]:
```
#understanding the data types of the columns
print(df.shape)
df.info()
```

```
(614, 13)
<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 [1180]:
```
df.describe()
```

Out[1180]:

	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 [1181]:
```
#We can see that there are null values in some of the columns
#Finding how much of the values are null
null_data=pd.DataFrame({"column_name":df.isnull().sum().index,
                        "Null_Value":(df.isnull().sum())/df.shape[0]*100
                       }).set_index('column_name').sort_values('Null_Value',ascending=Fa
null_data
```

Out[1181]:

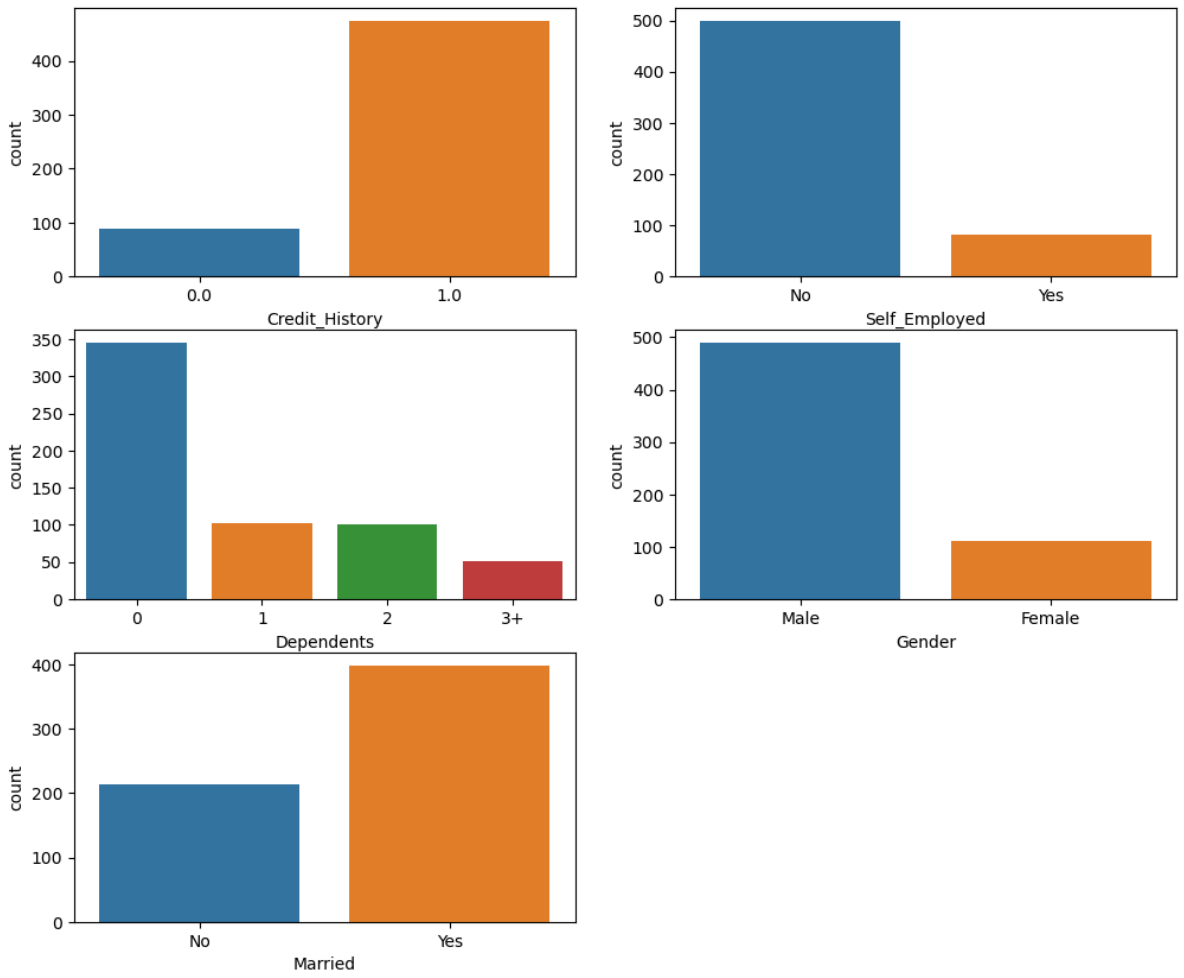
	column_name	Null_Value
0	Credit_History	8.143322
1	Self_Employed	5.211726
2	LoanAmount	3.583062
3	Dependents	2.442997
4	Loan_Amount_Term	2.280130
5	Gender	2.117264
6	Married	0.488599
7	Loan_ID	0.000000
8	Education	0.000000
9	ApplicantIncome	0.000000
10	CoapplicantIncome	0.000000
11	Property_Area	0.000000
12	Loan_Status	0.000000

In [1182]:
```
#We can see the columns having null value and their %. So now lets try to impu
# Before doing so we need to analyse whether they have outliers or not that mi
nul_val_col=null_data[(null_data['Null_Value']!=0) & ((null_data['column_name'
nul_val_col
```

Out[1182]:

	column_name	Null_Value
0	Credit_History	8.143322
1	Self_Employed	5.211726
3	Dependents	2.442997
5	Gender	2.117264
6	Married	0.488599

In [1183]:
```
x=0
plt.figure(figsize=(12,10))
for col in list(nul_val_col.column_name):
    x+=1
    plt.subplot(3,2,x)
    sns.countplot(data=df,x=col)
```



In [1184]:
```
#All the the above ones are categorical befor imputing these I will check if t
#Loan status colum using hypothesis testing
#We will tale alpha(type 1 error) as 5% with 95% confidence level
alpha=.05
#H0:Attributes are independent
#Ha:Attributes are dependent
col=['Credit_History','Self_Employed','Dependents','Gender','Married']
for col_name in col:
    stat,p_val,df1,expe=chi2_contingency(pd.crosstab(index=df['Loan_Status'],c
    if p_val<alpha:
        print(f'Reject H0:Attributes {col_name} is dependent')
        dependent_att.append(col_name)
    else:
        print(f'Accept H0:Attributes {col_name} is not dependent')
```

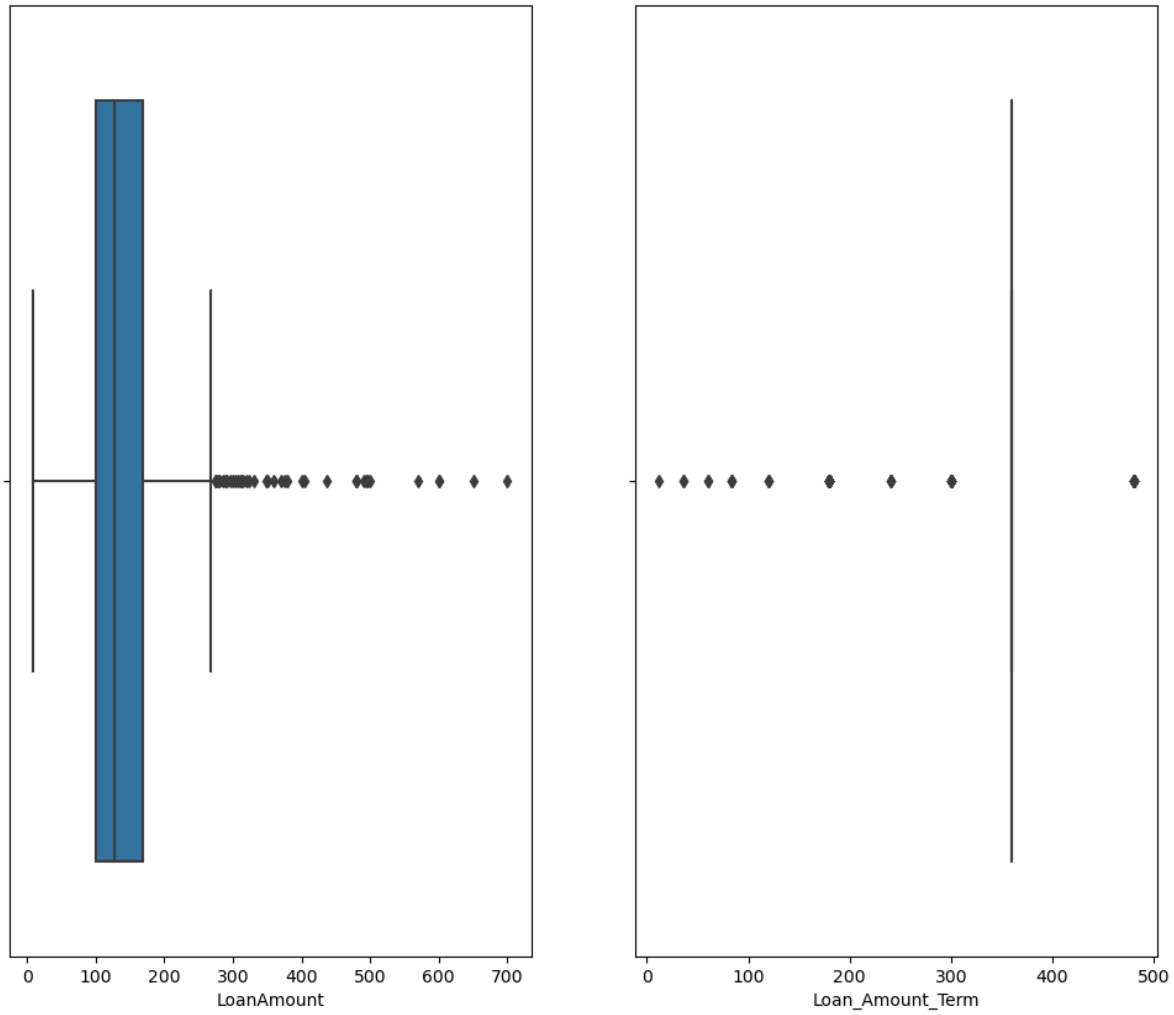
```
Reject H0:Attributes Credit_History is dependent
Accept H0:Attributes Self_Employed is not dependent
Accept H0:Attributes Dependents is not dependent
Accept H0:Attributes Gender is not dependent
Reject H0:Attributes Married is dependent
```

In [1185]:
```
#So from hypothesis testing we were able to observe that credithistory and Mar
#Since these column are have an effect on the tarfet colum we will not be impu
#we will replace it with another value so that we know it represents null
li=list(nul_val_col.column_name)
li.remove('Credit_History')
li
df['Credit_History'].fillna(2,inplace=True)

for i in li:
    df[i].fillna(df[i].mode()[0],inplace=True)
```

In [1186]:
```
(df.isna().sum())/df.shape[0]*100
```

Out[1186]:
```
Loan_ID              0.000000
Gender               0.000000
Married              0.000000
Dependents           0.000000
Education            0.000000
Self_Employed        0.000000
ApplicantIncome      0.000000
CoapplicantIncome    0.000000
LoanAmount           3.583062
Loan_Amount_Term     2.280130
Credit_History       0.000000
Property_Area        0.000000
Loan_Status          0.000000
dtype: float64
```

In [1187]:
```
# we will deal with the null in LoanAmount and Loan_Amount_Term column
#Usually we replace the numerical data with the mean if there are no outliers
x=0
plt.figure(figsize=(12,10))
for col in ['LoanAmount','Loan_Amount_Term']:
    x+=1
    plt.subplot(1,2,x)
    sns.boxplot(data=df,x=col)
```



In [1188]:
```
#both the attributes have outliers so will replace null values with median of
df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mode()[0],inplace=True)
df['LoanAmount'].fillna(df['LoanAmount'].median(),inplace=True)
```

In [1189]:
```
(df.isna().sum())/df.shape[0]*100
```

Out[1189]:
```
Loan_ID              0.0
Gender               0.0
Married              0.0
Dependents           0.0
Education            0.0
Self_Employed        0.0
ApplicantIncome      0.0
CoapplicantIncome    0.0
LoanAmount           0.0
Loan_Amount_Term     0.0
Credit_History       0.0
Property_Area        0.0
Loan_Status          0.0
dtype: float64
```

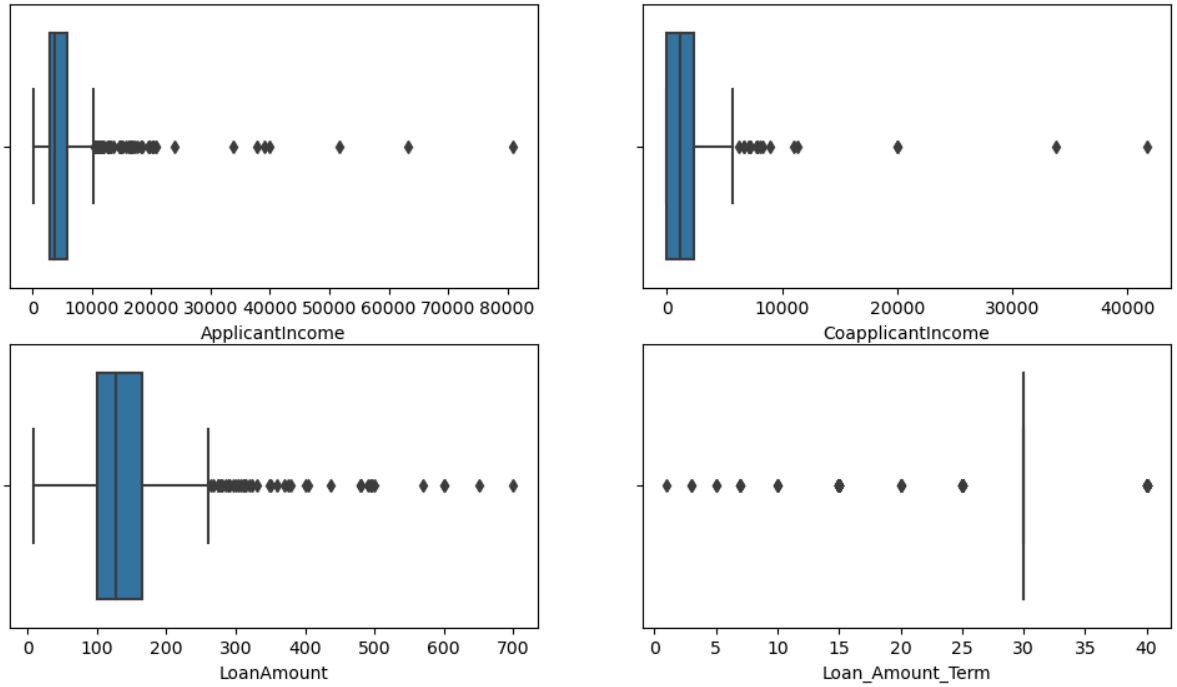
In [1190]:
```
#We have reomve all teh null values from the data set
#lets check for duplicates
df.duplicated().sum()
#There are no duplicates in the data set
```

Out[1190]:
```
0
```

In [1191]:
```
df.head()
df['Credit_History']=df['Credit_History'].astype("object")
df['Loan_Amount_Term']=(df['Loan_Amount_Term']/12).astype('float')
df['Loan_Amount_Term'].value_counts()
```

Out[1191]:
```
30.0    526
15.0     44
40.0     15
25.0     13
20.0      4
7.0       4
10.0      3
5.0       2
3.0       2
1.0       1
Name: Loan_Amount_Term, dtype: int64
```

In [1192]:
```
# Lets deal with outliers now.First we will seperate categorical and numerical
df_num=df.select_dtypes(include=np.number)
df_num

x=0
plt.figure(figsize=(12,10))
for i in list(df_num.columns):
    x+=1
    plt.subplot(3,2,x)
    sns.boxplot(data=df,x=i)
```



In [ ]:

In [1193]:
```
df_cat=df.select_dtypes(include='object')
df_cat
```

Out[1193]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Credit_History	Property
0	LP001002	Male	No	0	Graduate	No	1.0	
1	LP001003	Male	Yes	1	Graduate	No	1.0	
2	LP001005	Male	Yes	0	Graduate	Yes	1.0	
3	LP001006	Male	Yes	0	Not Graduate	No	1.0	
4	LP001008	Male	No	0	Graduate	No	1.0	
...	...	...	...	...	...	...	...	
609	LP002978	Female	No	0	Graduate	No	1.0	
610	LP002979	Male	Yes	3+	Graduate	No	1.0	
611	LP002983	Male	Yes	1	Graduate	No	1.0	
612	LP002984	Male	Yes	2	Graduate	No	1.0	
613	LP002990	Female	No	0	Graduate	Yes	0.0	Sem

614 rows × 9 columns

In [1194]:
```
#We are removing the outliers using the IQR method
q1=df_num.quantile(.25)
q2=df_num.quantile(.75)
iqr=q2-q1
iqr
lower=q1-1.5*iqr
upper=q2+1.5*iqr
clean_data=df[~((df<lower)|(df>upper)).any(axis=1)]
clean_data['EMI_per_month']=((clean_data['LoanAmount']/clean_data['Loan_Amount
clean_data['EMI_payable']=((clean_data['CoapplicantIncome'])*0.3>clean_data['E
clean_data
```

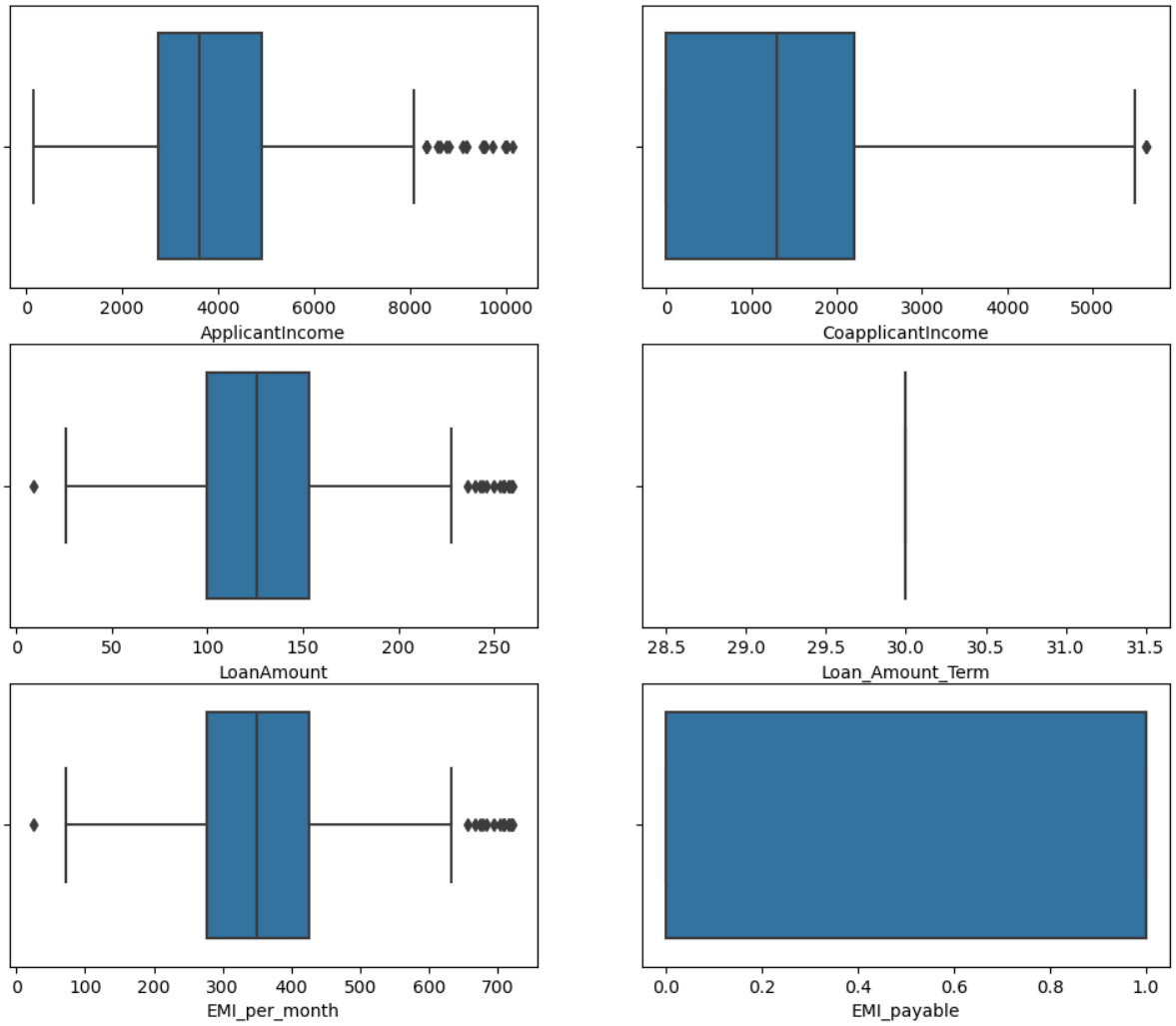
```
C:\Users\denms\AppData\Local\Temp\ipykernel_40344\4294283588.py:8: FutureWarn
ing: Automatic reindexing on DataFrame vs Series comparisons is deprecated an
d will raise ValueError in a future version. Do `left, right = left.align(rig
ht, axis=1, copy=False)` before e.g. `left == right`
  clean_data=df[~((df<lower)|(df>upper)).any(axis=1)]
C:\Users\denms\AppData\Local\Temp\ipykernel_40344\4294283588.py:9: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  clean_data['EMI_per_month']=((clean_data['LoanAmount']/clean_data['Loan_Amo
unt_Term']).astype('float')*1000/12)
C:\Users\denms\AppData\Local\Temp\ipykernel_40344\4294283588.py:10: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  clean_data['EMI_payable']=((clean_data['CoapplicantIncome'])*0.3>clean_data
['EMI_per_month']).astype('int')
```

Out[1194]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coap
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	
4	LP001008	Male	No	0	Graduate	No	6000	
...	...	...	...	...	...	...	...	
608	LP002974	Male	Yes	0	Graduate	No	3232	
609	LP002978	Female	No	0	Graduate	No	2900	
611	LP002983	Male	Yes	1	Graduate	No	8072	
612	LP002984	Male	Yes	2	Graduate	No	7583	
613	LP002990	Female	No	0	Graduate	Yes	4583	

459 rows × 15 columns

In [1195]:
```
clean_data
df1_num=clean_data.select_dtypes(include=np.number)
x=0
plt.figure(figsize=(12,10))
for i in list(df1_num.columns):
    x+=1
    plt.subplot(3,2,x)
    sns.boxplot(data=clean_data,x=i)
```



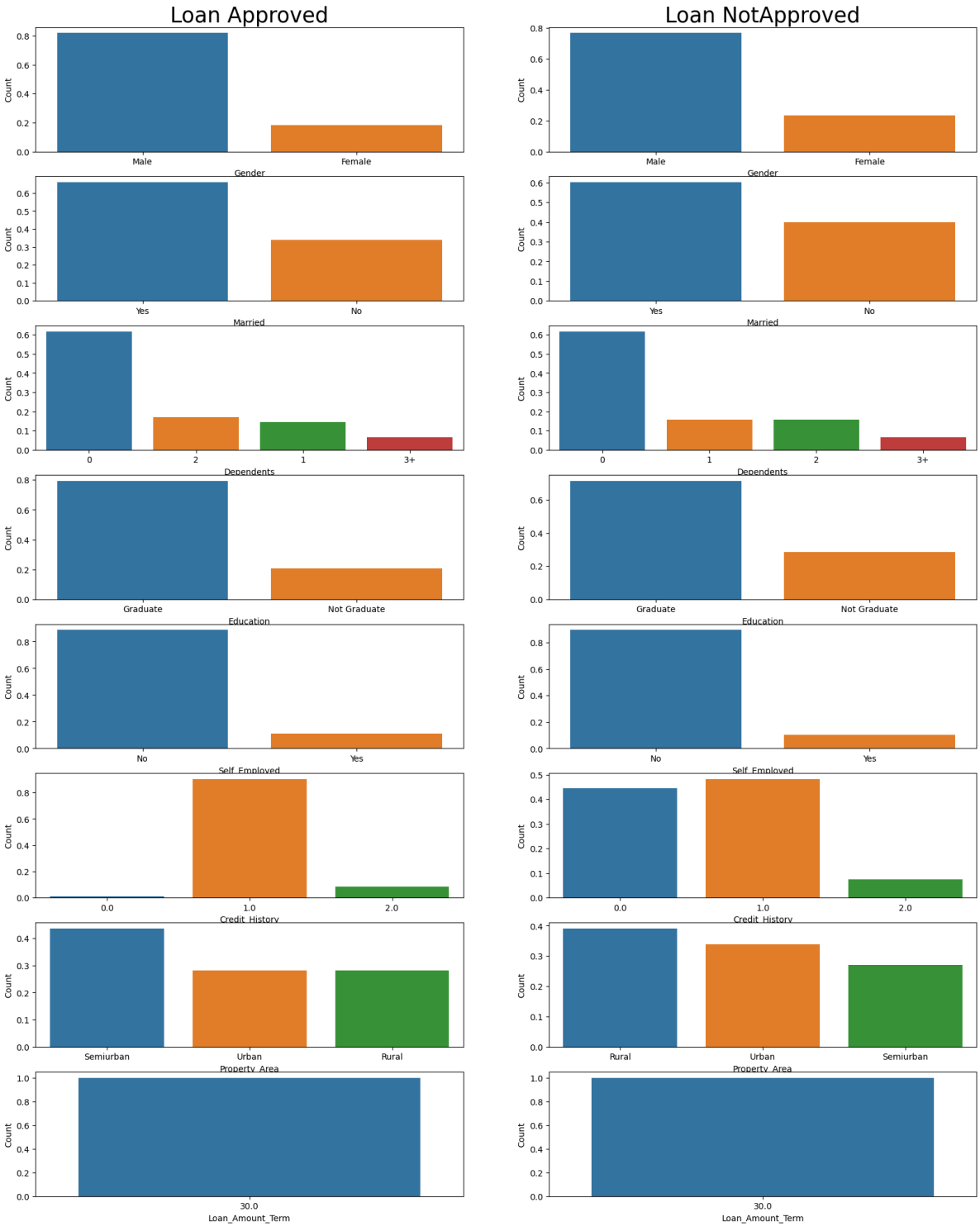
In [1196]:
```
#Compared to the previous data set we have removed most of the outliers and we
```

In [1197]:
```
#NOw lets sepereate the data based on loan status and start our analysis
cat_col=list(clean_data.select_dtypes(include='object').columns)[1:-1]
num_col=list(clean_data.select_dtypes(include=np.number).columns)
loan_y=clean_data[clean_data["Loan_Status"]=='Y']
loan_n=clean_data[clean_data["Loan_Status"]=='N']
```

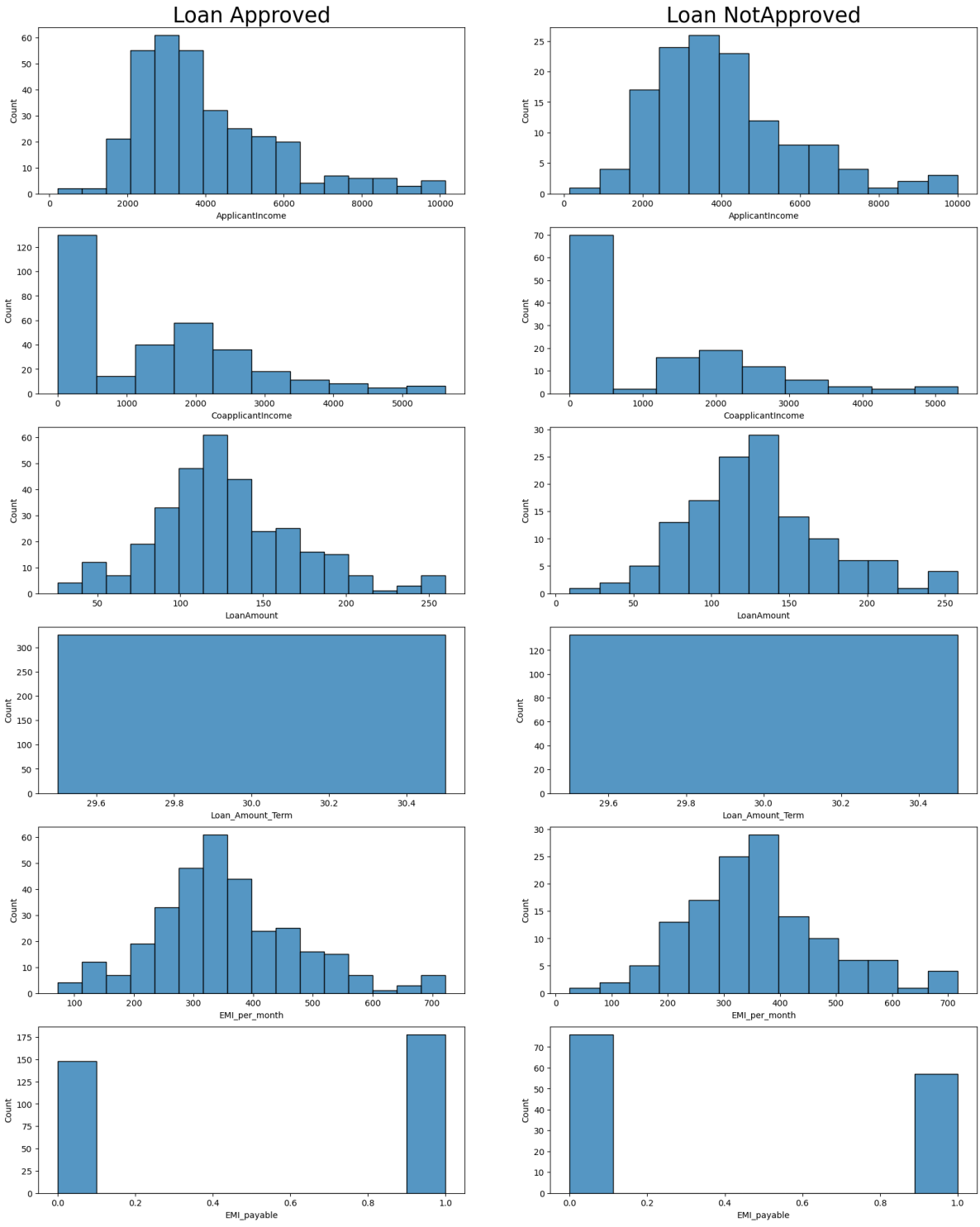
In [1198]:
```
#Will do hypothesis testing on categorical data to check for dependency on tar
alpha=.05
#H0:Attributes are independent
#Ha:Attributes are dependent
for col_name in cat_col:
    stat,p_val,df1,expe=chi2_contingency(pd.crosstab(index=clean_data['Loan_St
    if p_val<alpha:
        print(f'Reject H0:Attributes {col_name} is dependent')
        dependent_att.append(col_name)
    else:
        print(f'Accept H0:Attributes {col_name} is not dependent')
```

```
Accept H0:Attributes Gender is not dependent
Accept H0:Attributes Married is not dependent
Accept H0:Attributes Dependents is not dependent
Accept H0:Attributes Education is not dependent
Accept H0:Attributes Self_Employed is not dependent
Reject H0:Attributes Credit_History is dependent
Reject H0:Attributes Property_Area is dependent
```

In [1199]:
```
#Univariate analysis-categorical
x=0
plt.figure(figsize=(20,25))
cat_col.append('Loan_Amount_Term')
for col in cat_col:
    x+=1
    plt.subplot(8,2,x)
    plt.ylabel("Count")
    plt.xlabel(col)
    if x<=2:
        plt.title("Loan Approved",fontsize=25)
    sns.barplot(x=loan_y[col].value_counts().index,y=loan_y[col].value_counts(

    x+=1
    plt.subplot(8,2,x)
    plt.ylabel("Count")
    plt.xlabel(col)
    if x<=2:
        plt.title("Loan NotApproved",fontsize=25)
    sns.barplot(x=loan_n[col].value_counts().index,y=loan_n[col].value_counts(
```


Loan Approved
Loan NotApproved
Count
Male
Female
Gender
Yes
No
Married
0
2
1
3+
0
1
2
3+
Dependents
Graduate
Not Graduate
Education
No
Yes
Self_Employed
0.0
1.0
2.0
Credit_History
Semiurban
Urban
Rural
Property_Area
30.0
Loan_Amount_Term

In [1200]:
```
#Univariate analysis-numerical
x=0
plt.figure(figsize=(20,25))
for col in num_col:
    x+=1
    plt.subplot(len(num_col),2,x)
    plt.ylabel("Count")
    plt.xlabel(col)
    if x<=2:
        plt.title("Loan Approved",fontsize=25)
    sns.histplot(loan_y[col])

    x+=1
    plt.subplot(len(num_col),2,x)
    plt.ylabel("Count")
    plt.xlabel(col)
    if x<=2:
        plt.title("Loan NotApproved",fontsize=25)
    sns.histplot(loan_n[col])
```


Loan Approved
Loan NotApproved
Count
ApplicantIncome
CoapplicantIncome
LoanAmount
Loan_Amount_Term
EMI_per_month
EMI_payable

In [1201]:
```
#Will do hypothesis testing on numerical data to check for dependency on targe
alpha=.05
#H0:Attributes are independent
#Ha:Attributes are dependent
for col_name in num_col[:-3]:
    stat,p_val=ttest_ind(loan_y[col_name],loan_n[col_name])
    if p_val<alpha:
        print(f'Reject H0:Attributes {col_name} is dependent')
        dependent_att.append(col_name)
    else:
        print(f'Accept H0:Attributes {col_name} is not dependent')
```

```
Accept H0:Attributes ApplicantIncome is not dependent
Accept H0:Attributes CoapplicantIncome is not dependent
Accept H0:Attributes LoanAmount is not dependent
```

In [1202]:
```
from scipy.stats import f_oneway

for col_name in num_col[:-3]:
    stat,p_val=f_oneway(loan_y[col_name],loan_n[col_name])
    if p_val<alpha:
        print(f'Reject H0:Attributes {col_name} is dependent')
        dependent_att.append(col_name)
    else:
        print(f'Accept H0:Attributes {col_name} is not dependent')
```

```
Accept H0:Attributes ApplicantIncome is not dependent
Accept H0:Attributes CoapplicantIncome is not dependent
Accept H0:Attributes LoanAmount is not dependent
```

In [1203]:
```
#normality check
from scipy.stats import shapiro
#H0:Attributes is Gaussian
#Ha:Attributes is not Gaussian
for col_name in num_col[:-3]:
    stat,p_val=shapiro(clean_data[col_name])
    if p_val<alpha:
        print(f'Attributes is not Gaussian {col_name} ')
    else:
        print(f'Attributes is  Gaussian {col_name} ')
```

```
Attributes is not Gaussian ApplicantIncome
Attributes is not Gaussian CoapplicantIncome
Attributes is not Gaussian LoanAmount
```

In [1204]:
```
from scipy.stats import kruskal
#Since data is not Gaussian we do kruskal test
for col_name in num_col[:-3]:
    stat,p_val=kruskal(loan_y[col_name],loan_n[col_name])
    if p_val<alpha:
        print(f'Reject H0:Attributes {col_name} is dependent')
        dependent_att.append(col_name)
    else:
        print(f'Accept H0:Attributes {col_name} is not dependent')
```

```
Accept H0:Attributes ApplicantIncome is not dependent
Reject H0:Attributes CoapplicantIncome is dependent
Accept H0:Attributes LoanAmount is not dependent
```

In [1205]:
```
### lets categorize few columns
#bins = [0,3000,7000,10200]
#group = ['Low','Average','High']
#clean_data["AppliIncome_bin"] = pd.cut(clean_data["ApplicantIncome"],bins,lab
#bins = [-1,2000,4000,6000]
#group = ['Low','Average','High']
#clean_data["Co_AppliIncome_bin"] = pd.cut(clean_data["CoapplicantIncome"],bin
```

In [1206]:
```
clean_data
```

Out[1206]:

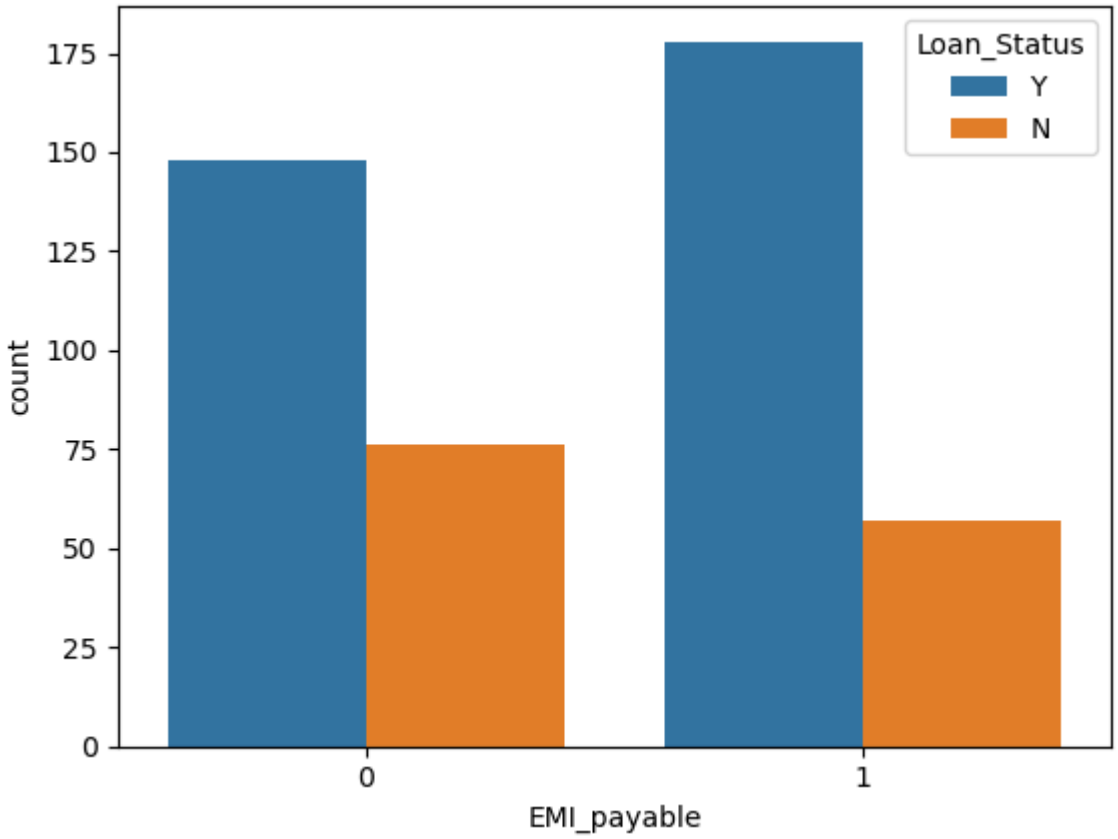
	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coap
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	
4	LP001008	Male	No	0	Graduate	No	6000	
...	...	...	...	...	...	...	...	...
608	LP002974	Male	Yes	0	Graduate	No	3232	
609	LP002978	Female	No	0	Graduate	No	2900	
611	LP002983	Male	Yes	1	Graduate	No	8072	
612	LP002984	Male	Yes	2	Graduate	No	7583	
613	LP002990	Female	No	0	Graduate	Yes	4583	

459 rows × 15 columns

In [1207]:
```
#will assume that 30% of applicants salary shoul cover emi
```

In [1208]:
```
sns.countplot(data=clean_data,x='EMI_payable',hue='Loan_Status')
```

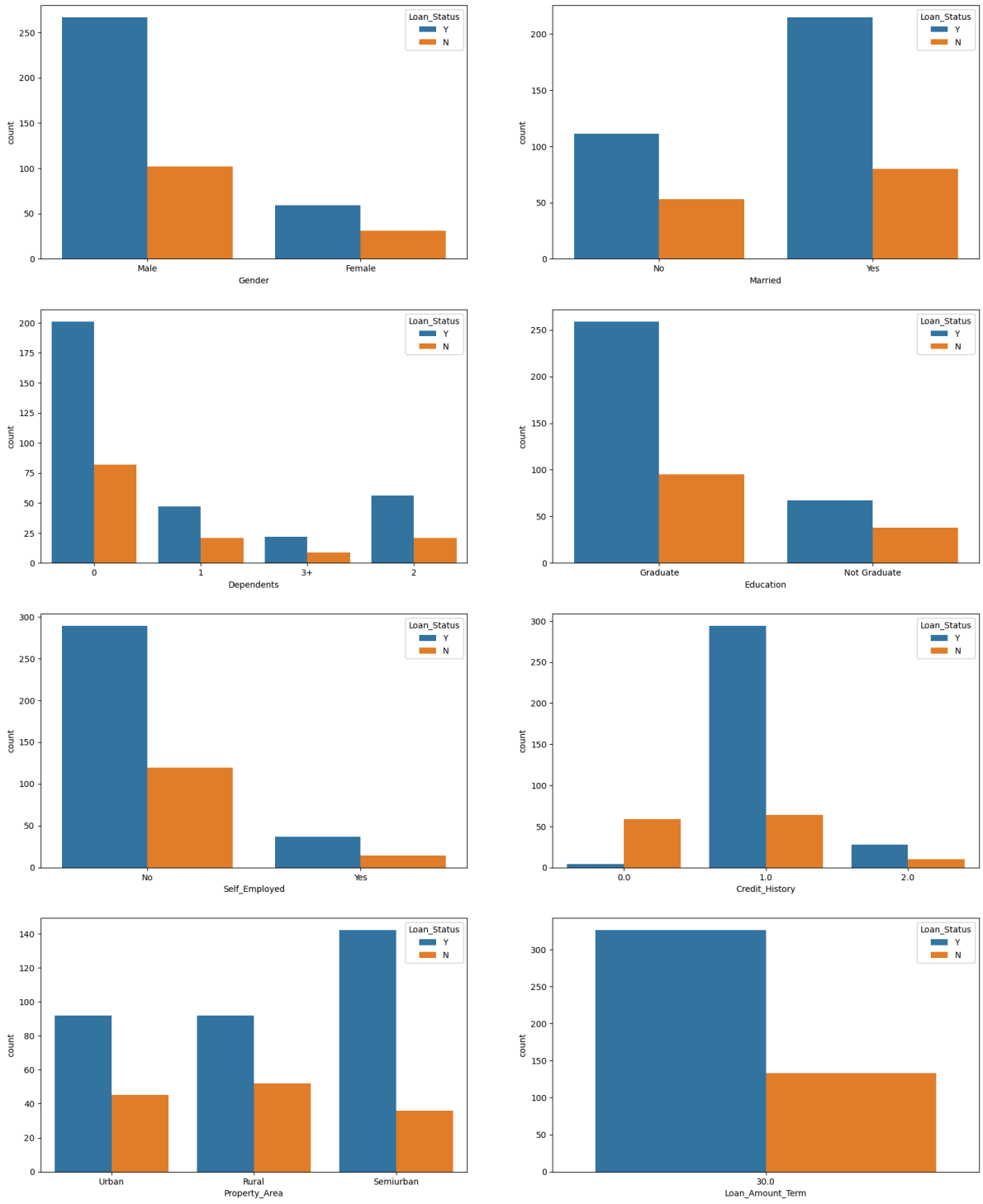
Out[1208]:
```
<AxesSubplot:xlabel='EMI_payable', ylabel='count'>
```



In [1209]:
```
col_name='EMI_payable'
stat,p_val,df1,expe=chi2_contingency(pd.crosstab(index=clean_data['Loan_Status
if p_val<alpha:
    print(f'Reject H0:Attributes EMI_payable is dependent')
    dependent_att.append(col_name)
else:
    print(f'Accept H0:Attributes EMI_payable is not dependent')
```

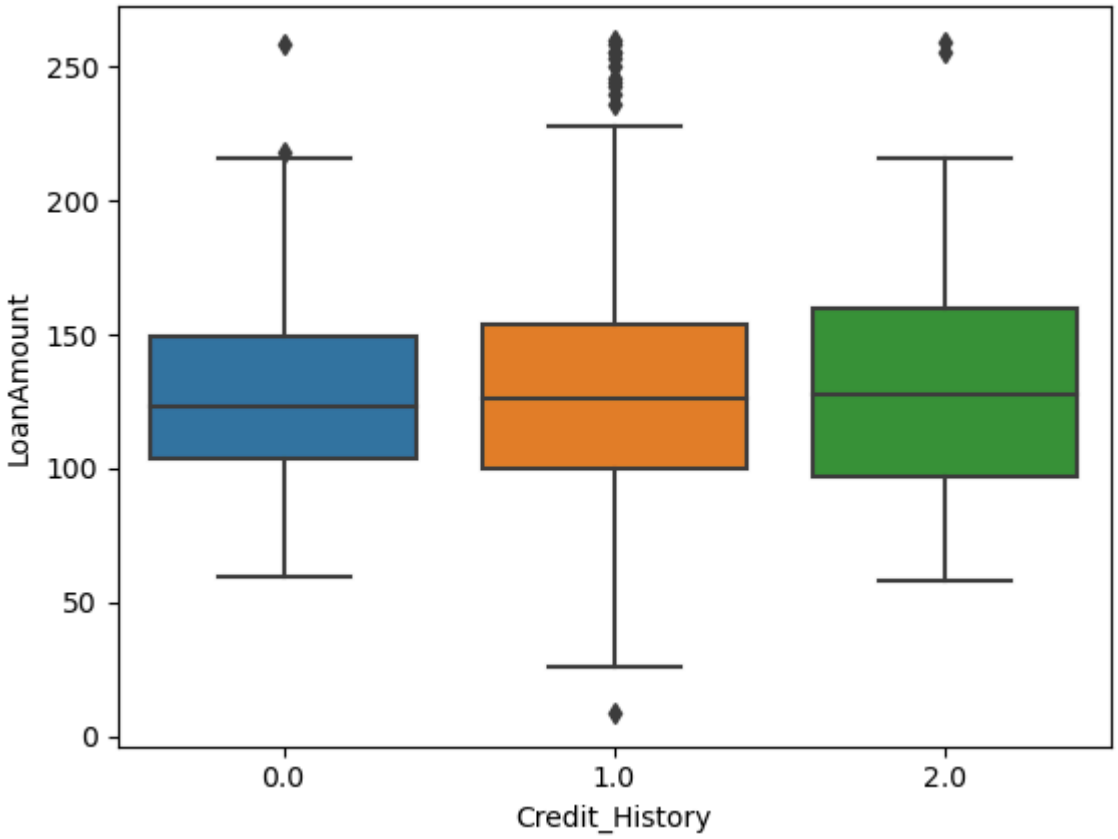
```
Reject H0:Attributes EMI_payable is dependent
```

In [1210]:
```
#Bi variate analysis-cat
x=0
plt.figure(figsize=(20,25))
for col in cat_col:
    x+=1
    plt.subplot(4,2,x)
    plt.ylabel("Count")
    plt.xlabel(col)
    sns.countplot(data=clean_data,x=col,hue='Loan_Status')
```



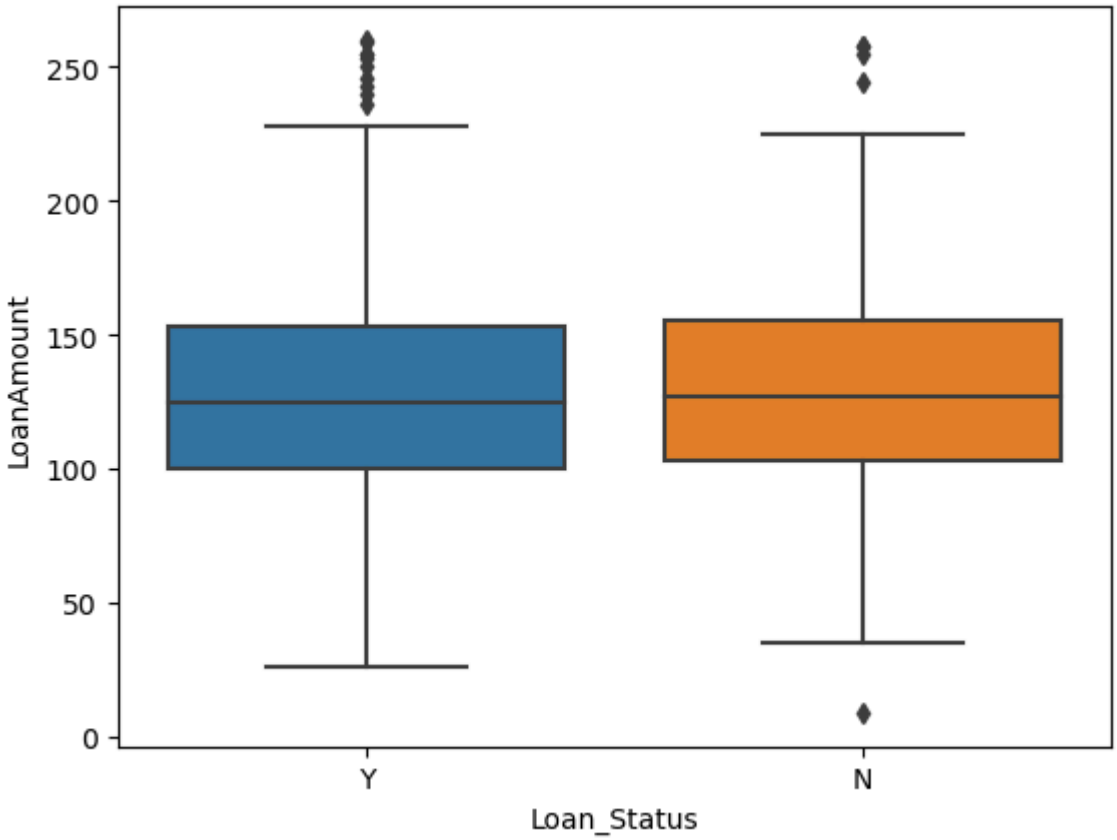
In [1211]:
```
sns.boxplot(data=clean_data,x='Credit_History',y='LoanAmount')
```

Out[1211]:
```
<AxesSubplot:xlabel='Credit_History', ylabel='LoanAmount'>
```



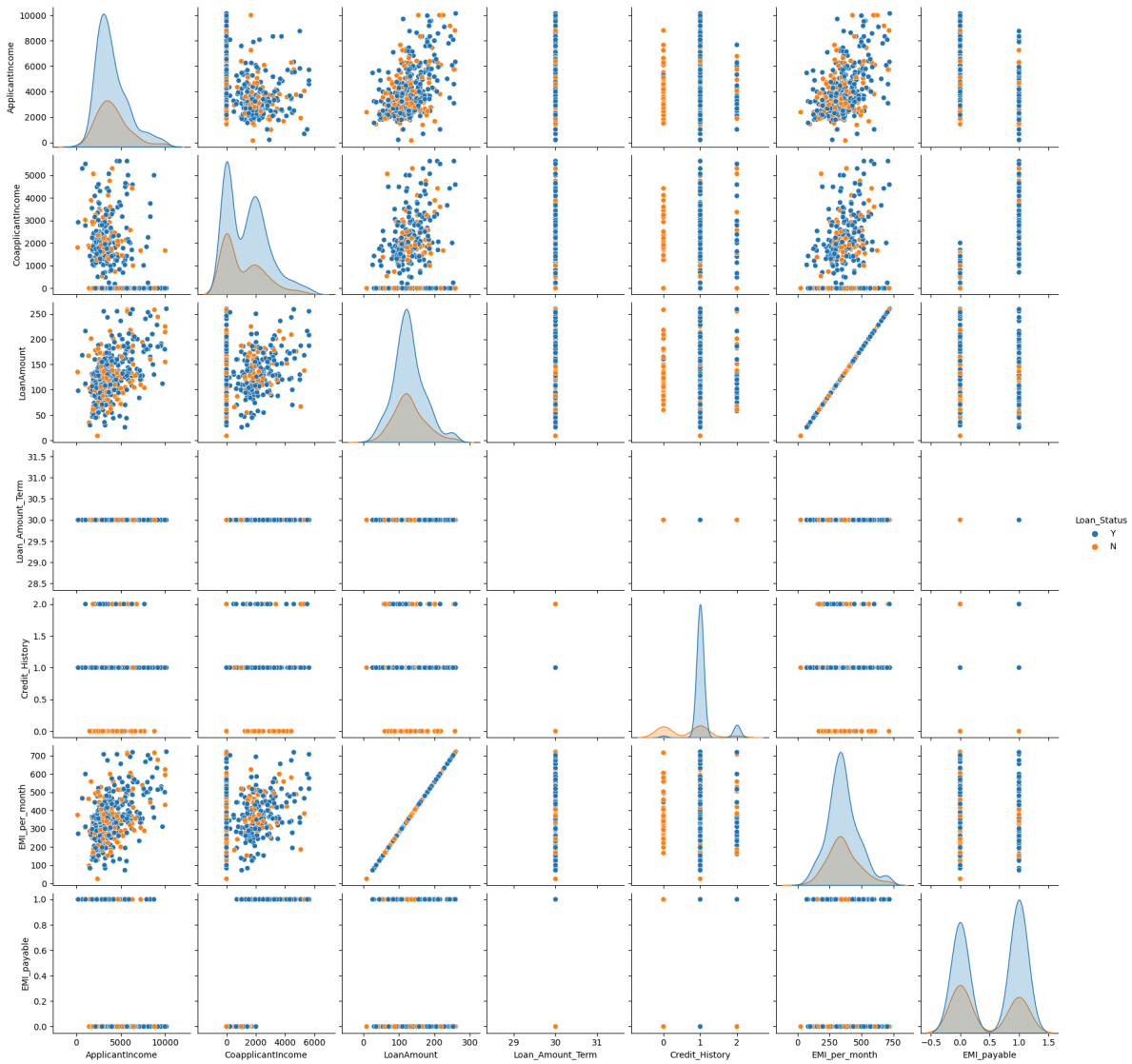
In [1212]:
```
sns.boxplot(data=clean_data,x='Loan_Status',y='LoanAmount')
```

Out[1212]:
```
<AxesSubplot:xlabel='Loan_Status', ylabel='LoanAmount'>
```



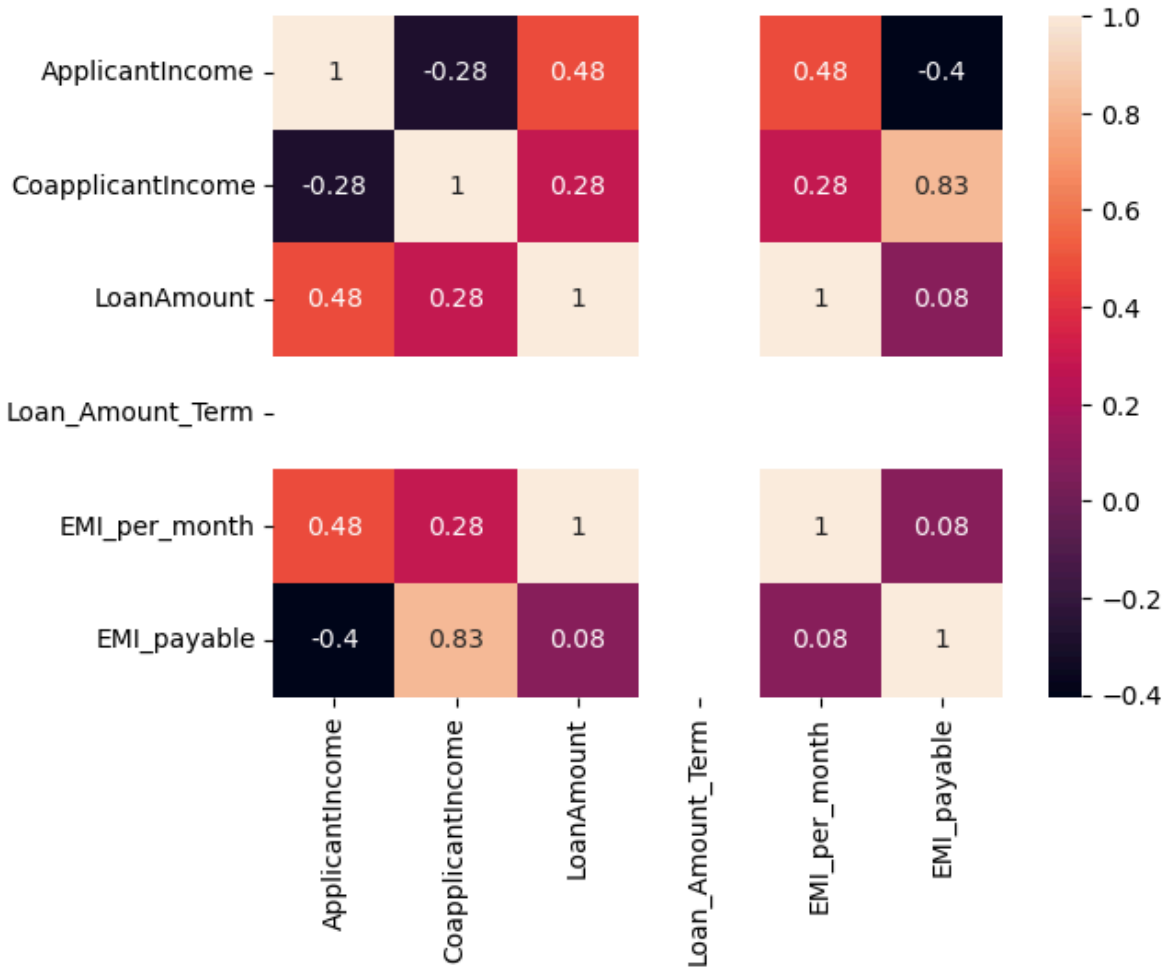
In [1213]:
```
sns.pairplot(data=clean_data,hue='Loan_Status')
```

Out[1213]: <seaborn.axisgrid.PairGrid at 0x1dd4fcda2b0>



In [1214]:
```
sns.heatmap(clean_data.corr(),annot=True)
```

Out[1214]: <AxesSubplot:>



In [1215]:
```
#So from the analysis we have done the columns that impact th eloans status ar
```

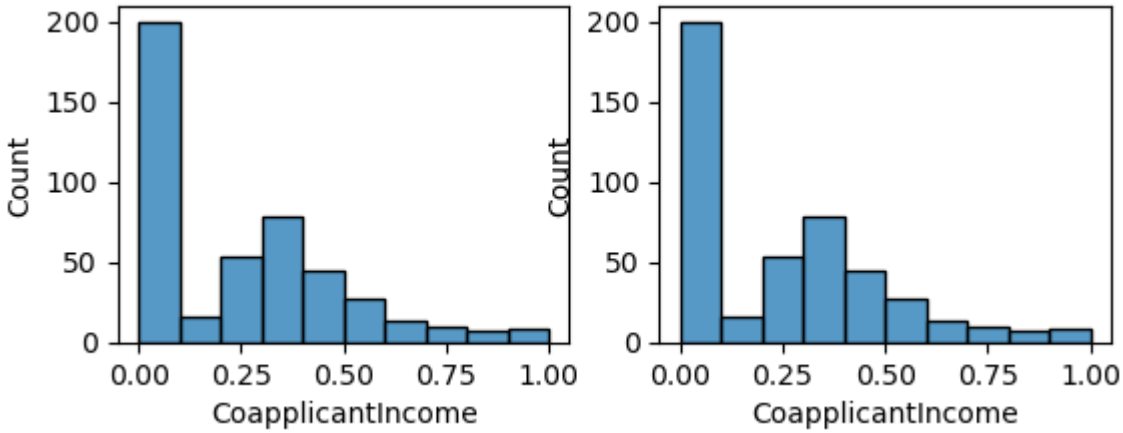
In [ ]:
```

```

In [1216]:
```
#Now lets prepare the data from ML .we will normalize the data and convert cat
```

In [1217]:
```
min_max = MinMaxScaler()
clean_data['CoapplicantIncome']=min_max.fit_transform(clean_data[['Coapplicant

plt.subplot(2,2,1)
sns.histplot(clean_data['CoapplicantIncome'])

plt.subplot(2,2,2)
sns.histplot(procesed_data['CoapplicantIncome'])
```

```
C:\Users\denms\AppData\Local\Temp\ipykernel_40344\3979801236.py:2: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  clean_data['CoapplicantIncome']=min_max.fit_transform(clean_data[['Coapplic
antIncome']])
```

Out[1217]:
```
<AxesSubplot:xlabel='CoapplicantIncome', ylabel='Count'>
```



In [1218]:
```
lab_en=LabelEncoder()
clean_data['Married']=lab_en.fit_transform(clean_data['Married'])
```

```
C:\Users\denms\AppData\Local\Temp\ipykernel_40344\4164503690.py:2: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  clean_data['Married']=lab_en.fit_transform(clean_data['Married'])
```

In [1219]:
```
pip install --upgrade category_encoders
```

```
Requirement already satisfied: category_encoders in c:\users\denms\anaconda3
\lib\site-packages (2.6.3)
Requirement already satisfied: scipy>=1.0.0 in c:\users\denms\anaconda3\lib\s
ite-packages (from category_encoders) (1.9.1)
Requirement already satisfied: numpy>=1.14.0 in c:\users\denms\anaconda3\lib
\site-packages (from category_encoders) (1.21.5)
Requirement already satisfied: pandas>=1.0.5 in c:\users\denms\anaconda3\lib
\site-packages (from category_encoders) (1.4.4)
Requirement already satisfied: scikit-learn>=0.20.0 in c:\users\denms\anacond
a3\lib\site-packages (from category_encoders) (1.0.2)
Requirement already satisfied: patsy>=0.5.1 in c:\users\denms\anaconda3\lib\s
ite-packages (from category_encoders) (0.5.2)
Requirement already satisfied: statsmodels>=0.9.0 in c:\users\denms\anaconda3
\lib\site-packages (from category_encoders) (0.13.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\denms\anaconda3\lib\s
ite-packages (from pandas>=1.0.5->category_encoders) (2022.1)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\denms\anaco
nda3\lib\site-packages (from pandas>=1.0.5->category_encoders) (2.8.2)
Requirement already satisfied: six in c:\users\denms\anaconda3\lib\site-packa
ges (from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=0.11 in c:\users\denms\anaconda3\lib\s
ite-packages (from scikit-learn>=0.20.0->category_encoders) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\denms\anacond
a3\lib\site-packages (from scikit-learn>=0.20.0->category_encoders) (2.2.0)
Requirement already satisfied: packaging>=21.3 in c:\users\denms\anaconda3\li
b\site-packages (from statsmodels>=0.9.0->category_encoders) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\denms\ana
conda3\lib\site-packages (from packaging>=21.3->statsmodels>=0.9.0->category_
encoders) (3.0.9)
Note: you may need to restart the kernel to use updated packages.
```

In [1220]:
```
clean_data['Loan_Status'].replace(['N','Y'],[0,1],inplace=True)
tar_en=TargetEncoder()
clean_data['Property_Area']=tar_en.fit_transform(clean_data['Property_Area'],c
```

```
C:\Users\denms\AppData\Local\Temp\ipykernel_40344\418446286.py:1: SettingWith
CopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  clean_data['Loan_Status'].replace(['N','Y'],[0,1],inplace=True)
C:\Users\denms\AppData\Local\Temp\ipykernel_40344\418446286.py:3: SettingWith
CopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy (https://panda
s.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ver
sus-a-copy)
  clean_data['Property_Area']=tar_en.fit_transform(clean_data['Property_Are
a'],clean_data['Loan_Status'])
```

In [1221]:
```
procesed_data=clean_data[list(set(dependent_att))]
procesed_data
```

Out[1221]:

	CoapplicantIncome	Married	Credit_History	EMI_payable	Property_Area
0	0.000000	0	1.0	0	0.671533
1	0.268089	1	1.0	1	0.638889
2	0.000000	1	1.0	0	0.671533
3	0.419200	1	1.0	1	0.671533
4	0.000000	0	1.0	0	0.671533
...	...	...	...	...	...
608	0.346667	1	1.0	1	0.638889
609	0.000000	0	1.0	0	0.638889
611	0.042667	1	1.0	0	0.671533
612	0.000000	1	1.0	0	0.671533
613	0.000000	0	0.0	0	0.797753

459 rows × 5 columns

From my analysis the above mentioned attributes impact the target colum(Loan

In [ ]: