In [1]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

In [ ]:

In [2]:
```
df = pd.read_csv("Credit_Risk_XTrain (1).csv")
```

In [3]:
```
df.head()
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loa
0	LP001002	Male	No	0	Graduate	No	5849	0.0	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	

In [4]:
```
df.shape
```

Out[4]:
```
(614, 13)
```

In [5]:
```
# Find null values in the dataset
df.isnull().sum()/len(df)*100
```

Out[5]:
```
Loan_ID              0.000000
Gender               2.117264
Married              0.488599
Dependents           2.442997
Education            0.000000
Self_Employed        5.211726
ApplicantIncome      0.000000
CoapplicantIncome    0.000000
LoanAmount           3.583062
Loan_Amount_Term     2.280130
Credit_History       8.143322
Property_Area        0.000000
Loan_Status          0.000000
dtype: float64
```

In [6]:
```
# check dataset information
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 [7]:
```
df.describe()
```

Out[7]:

	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 [8]:
```
# Imputing null value
# pls treate numerical value first and then try below one - most_frequent
from sklearn.impute import SimpleImputer
imp_mode = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
df_imputed = pd.DataFrame(imp_mode.fit_transform(df))
df_imputed.columns = df.columns
df_imputed
```

Out[8]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	L
0	LP001002	Male	No	0	Graduate	No	5849	0.0	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	
...	...	...	...	...	...	...	...	...	
609	LP002978	Female	No	0	Graduate	No	2900	0.0	
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	

614 rows × 13 columns

In [9]:
```
df_imputed.isnull().sum()
```

Out[9]:
```
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
```

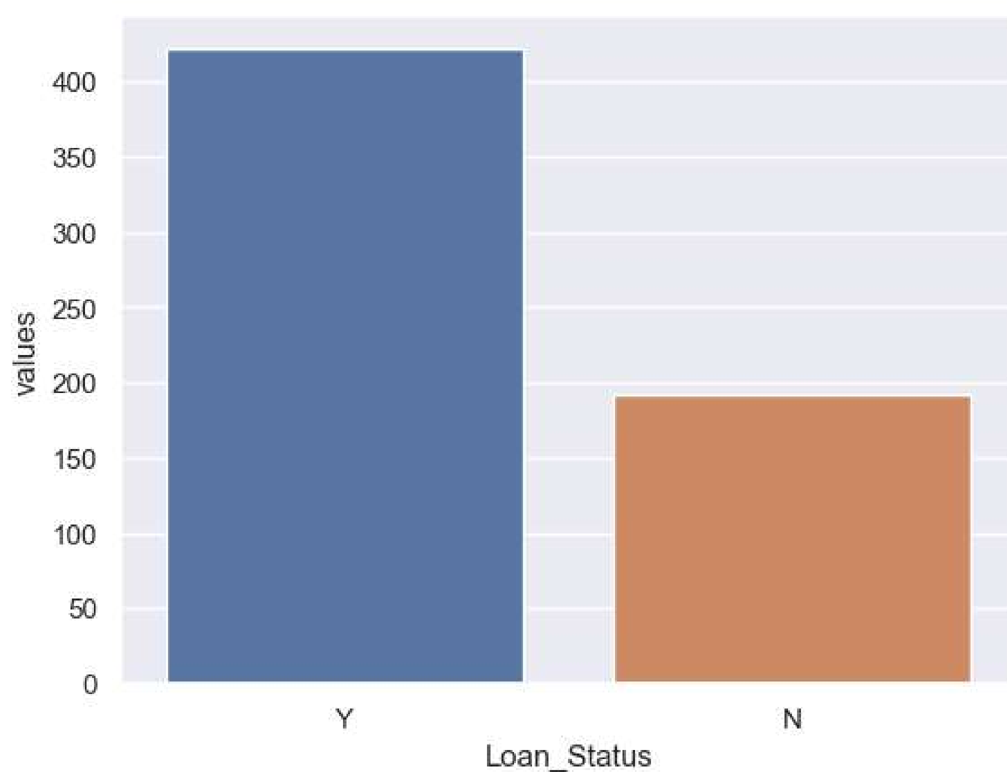
In [10]:
```
# Find the unique values in the columns
for i in df_imputed.columns:
    print("****************************************", i ,
        "***********************************************")
    print()
    print(set(df_imputed[i].tolist()))
    print()
```

```
5, 3675, 3676, 5726, 9833, 7787, 3692, 3691, 5746, 3704, 3707, 3708, 3716, 3717, 1668, 3
727, 5780, 3748, 3750, 5800, 3762, 5815, 5818, 5819, 5821, 3775, 5829, 20166, 3800, 584
9, 7901, 1759, 12000, 3812, 3813, 3814, 3816, 9963, 18165, 1782, 16120, 3833, 7933, 384
6, 1800, 20233, 3850, 7948, 10000, 1809, 3858, 1811, 3859, 3865, 3867, 1820, 3875, 1828,
5923, 1830, 1836, 5935, 3887, 5941, 3900, 1853, 3902, 10047, 8000, 5955, 1863, 3917, 187
5, 3927, 1880, 3941, 63337, 3948, 6000, 1907, 16250, 1916, 1926, 3975, 1928, 8072, 8080,
6033, 3987, 3988, 3992, 3993, 10139, 6045, 18333, 4000, 6050, 4006, 1958, 4009, 1963, 60
65, 1977, 6080, 6083, 1993, 2000, 6096, 4050, 4053, 2014, 6125, 2031, 6133, 2045, 4095}

**************************************** CoapplicantIncome ***************************
*******************

{0.0, 1030.0, 2054.0, 1542.0, 2569.0, 6666.0, 6667.0, 1032.0, 1040.0, 3600.0, 4114.0, 20
67.0, 1041.0, 16.12000084, 5654.0, 2583.0, 1560.0, 536.0, 2079.0, 20000.0, 2042.0, 2083.
0, 11300.0, 2598.0, 2087.0, 4648.0, 7210.0, 33837.0, 1587.0, 2100.0, 1590.0, 1591.0, 108
3.0, 1086.0, 1600.0, 3136.0, 2115.0, 1603.0, 5701.0, 2118.0, 4167.0, 7750.0, 3150.0, 725
0.0, 1619.0, 3667.0, 3666.0, 2134.0, 1625.0, 2138.0, 2142.0, 3166.0, 3167.0, 1632.0, 368
3.0, 4196.0, 1125.0, 1126.0, 1640.0, 6250.0, 2667.0, 1644.0, 1131.0, 2157.0, 2669.0, 216
0.0, 2166.0, 2167.0, 2168.0, 1664.0, 1666.0, 1667.0, 1668.0, 4232.0, 5624.0, 2188.0, 833
3.0, 4750.0, 1167.0, 2064.0, 5266.0, 663.0, 2200.0, 4250.0, 3230.0, 1695.0, 2209.0, 221
```

In [11]:
```
df_imputed['Dependents'] = df_imputed['Dependents'].apply(lambda x : 'no' if x=='+' else x)
```

In [12]:
```
# Check label imbalance
temp = df_imputed['Loan_Status'].value_counts()
temp_df = pd.DataFrame({'Loan_Status' : temp.index, 'values' : temp.values})
print(sns.barplot(x='Loan_Status', y = 'values', data=temp_df))
```

```
Axes(0.125,0.11;0.775x0.77)
```

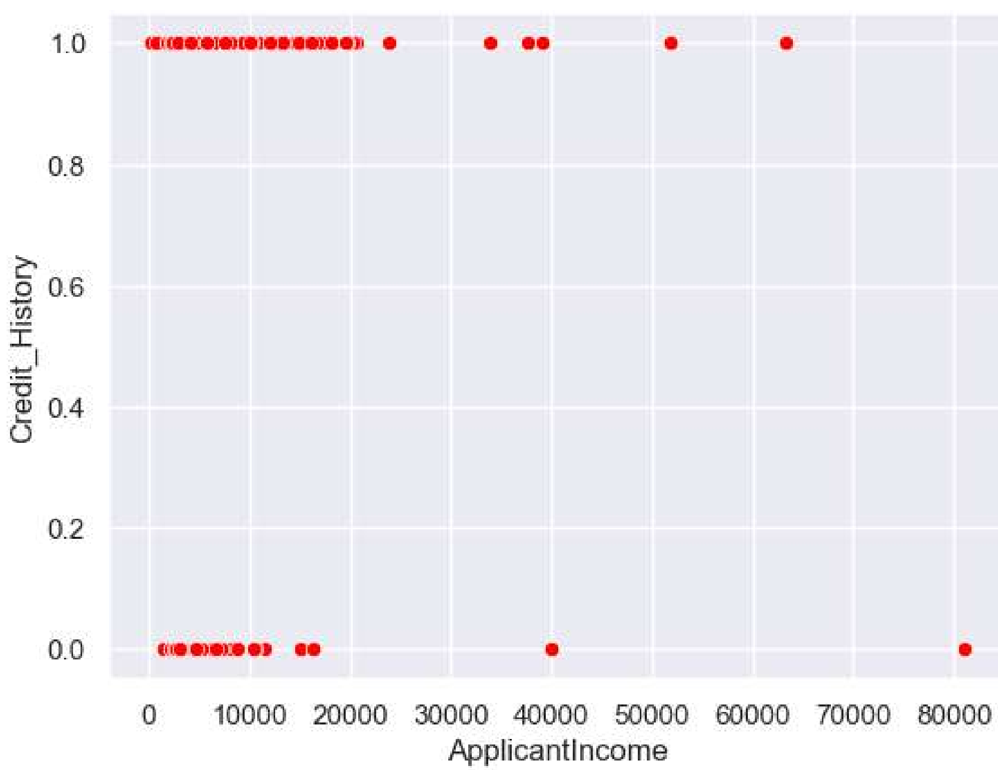


In [60]:
```
sns.scatterplot(x ="Loan_Status",y = "Credit_History",data=df,color="red")
```

Out[60]:
```
<Axes: xlabel='Loan_Status', ylabel='Credit_History'>
```

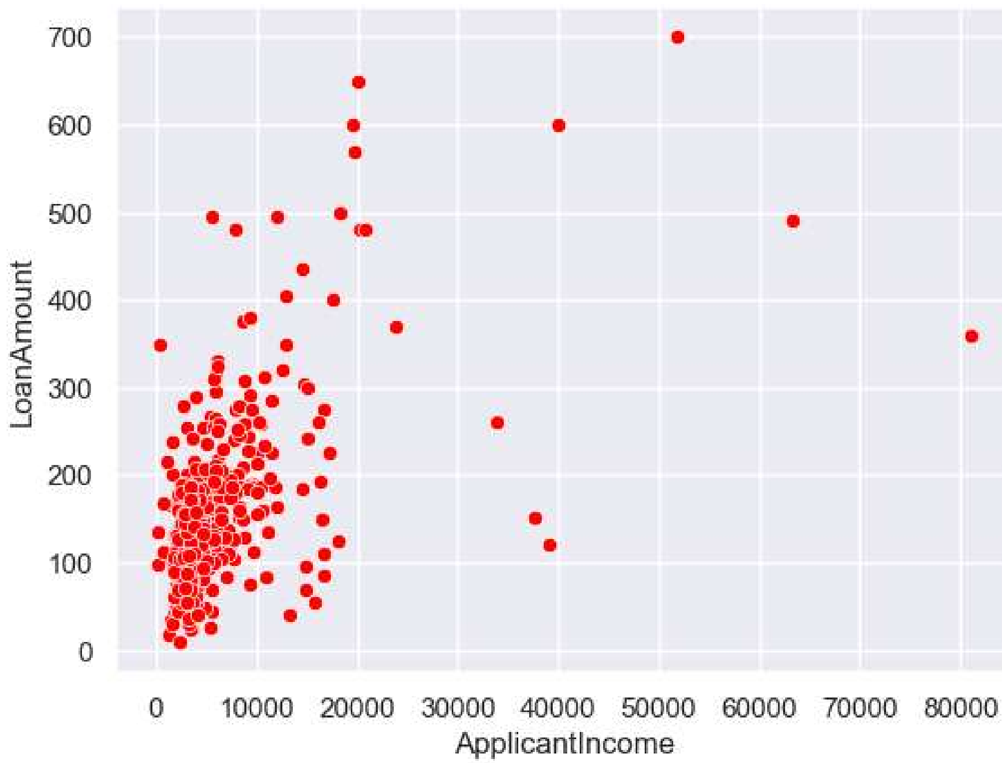
In [61]:
```
sns.scatterplot(x ="ApplicantIncome",y = "Credit_History",data=df,color="red")
```

Out[61]:
```
<Axes: xlabel='ApplicantIncome', ylabel='Credit_History'>
```



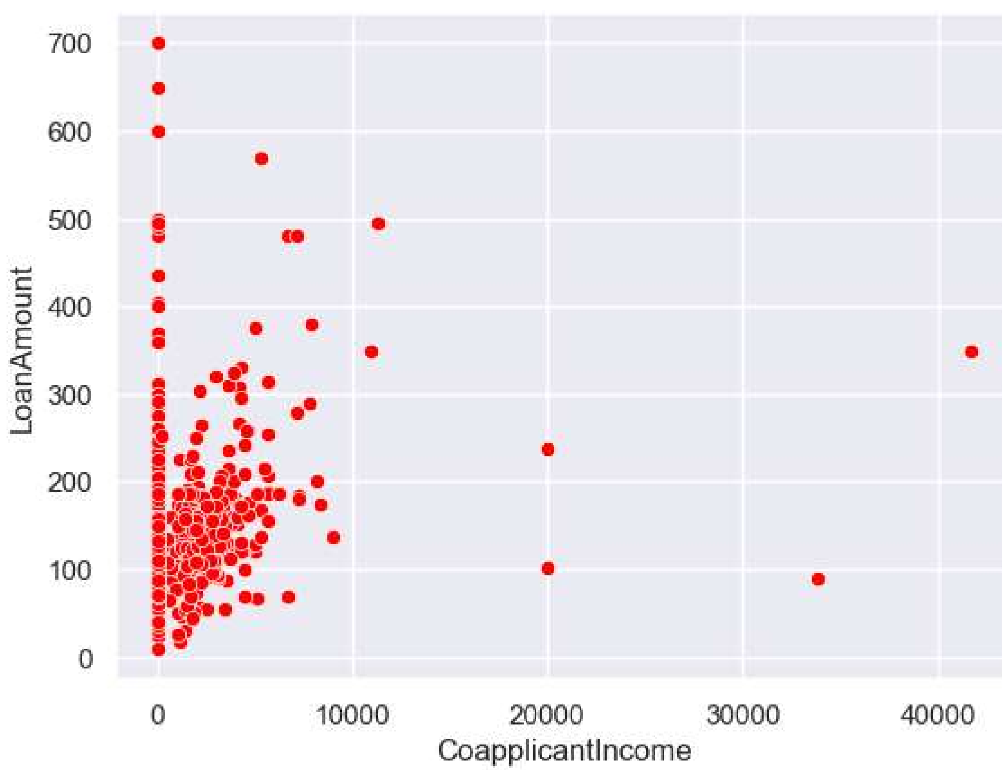
In [62]:
```
sns.scatterplot(x ="ApplicantIncome",y = "LoanAmount",data=df,color="red")
```

Out[62]: <Axes: xlabel='ApplicantIncome', ylabel='LoanAmount'>



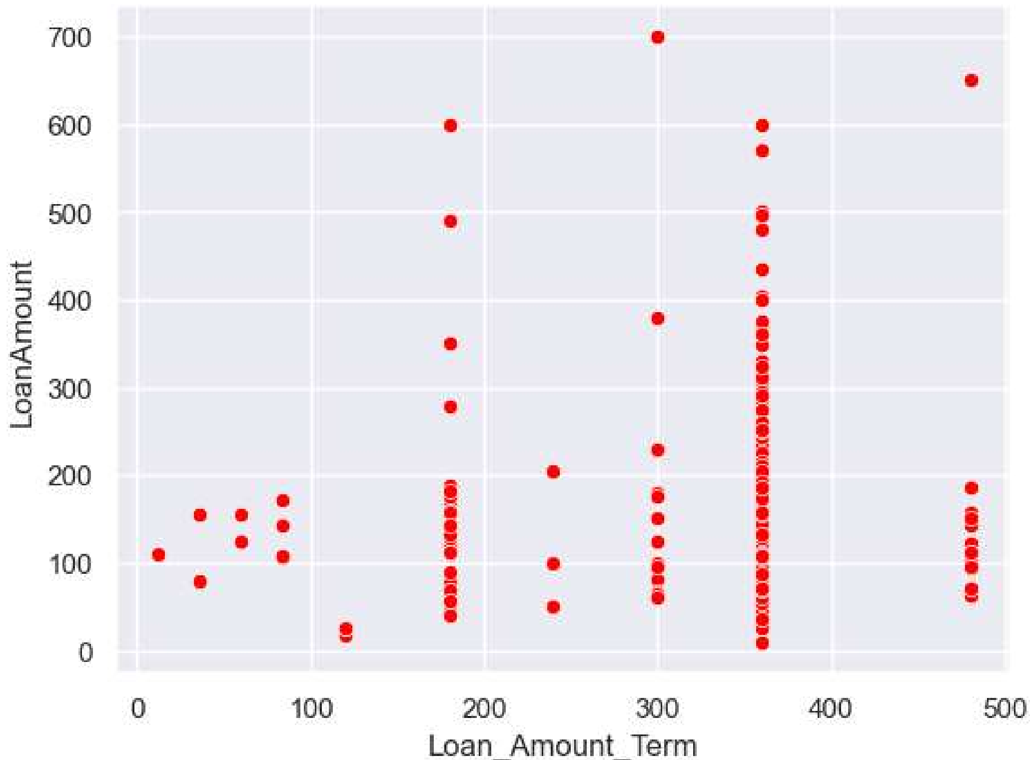
In [63]:
```
sns.scatterplot(x ="CoapplicantIncome",y = "LoanAmount",data=df,color="red")
```

Out[63]: <Axes: xlabel='CoapplicantIncome', ylabel='LoanAmount'>



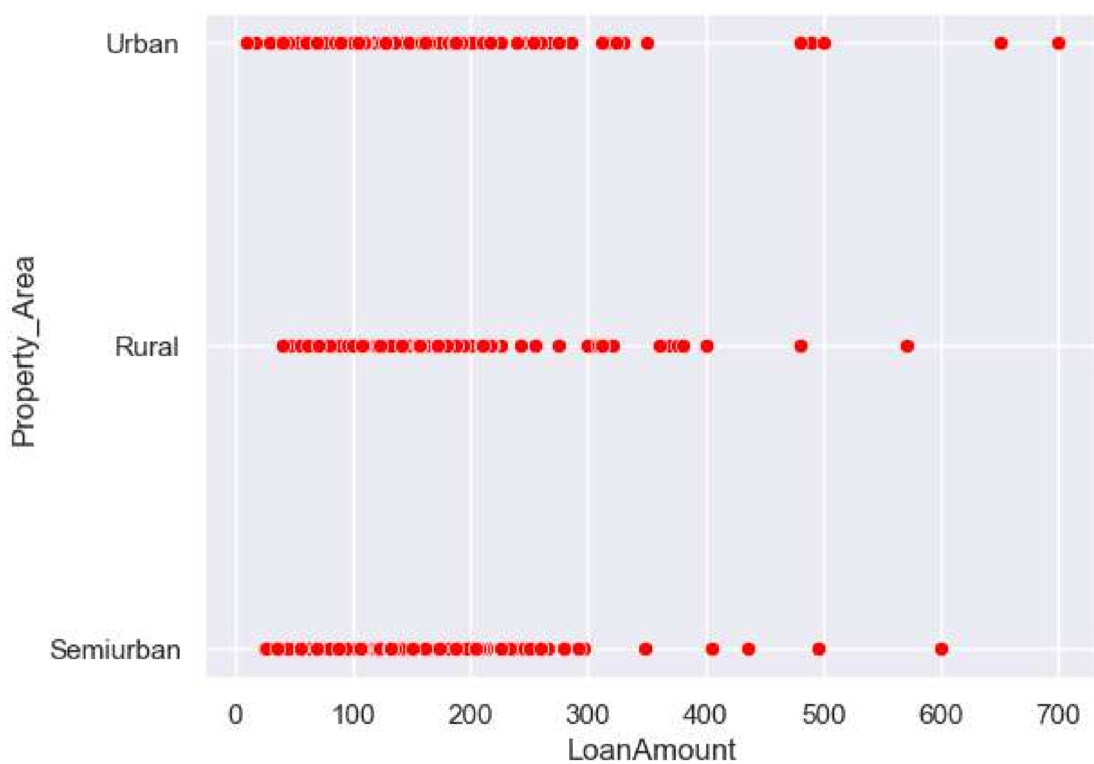
In [64]:
```
sns.scatterplot(x ="Loan_Amount_Term",y = "LoanAmount",data=df,color="red")
```

Out[64]: <Axes: xlabel='Loan_Amount_Term', ylabel='LoanAmount'>



In [67]:
```
sns.scatterplot(x ="LoanAmount",y = "Property_Area",data=df,color="red")
```

Out[67]: <Axes: xlabel='LoanAmount', ylabel='Property_Area'>



In [13]:
```
df_imputed.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             614 non-null    object
 2   Married            614 non-null    object
 3   Dependents         614 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      614 non-null    object
 6   ApplicantIncome    614 non-null    object
 7   CoapplicantIncome  614 non-null    object
 8   LoanAmount         614 non-null    object
 9   Loan_Amount_Term   614 non-null    object
 10  Credit_History     614 non-null    object
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: object(13)
memory usage: 62.5+ KB
```

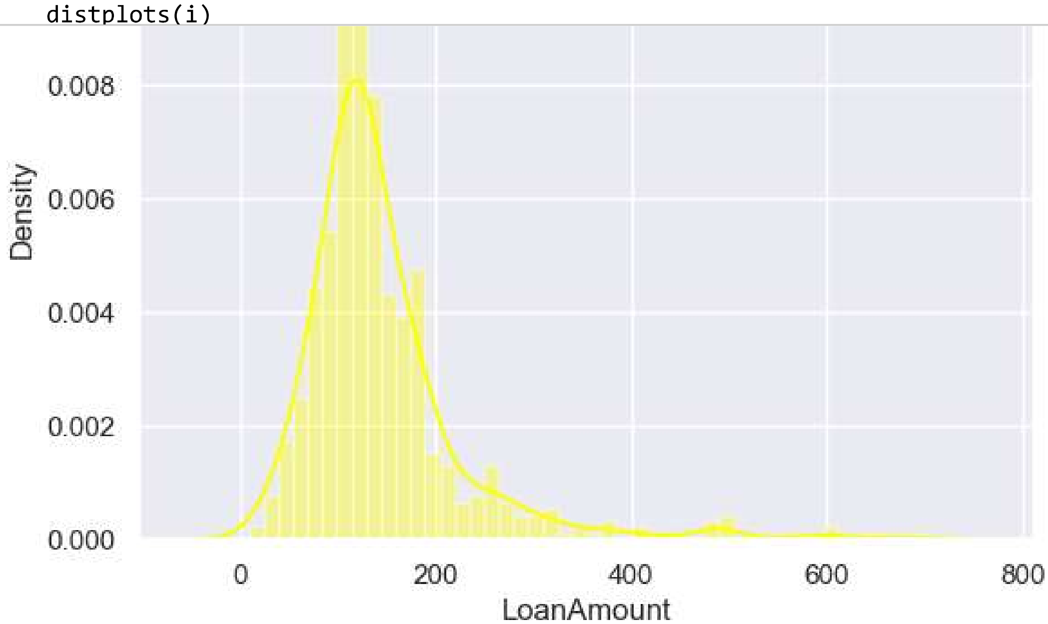
In [14]:
```
for i in df.select_dtypes(exclude=['object']).columns:
    df_imputed[i] = df_imputed[i].apply(lambda x :float(x))
```

In [15]:
```
df_imputed.info()
```

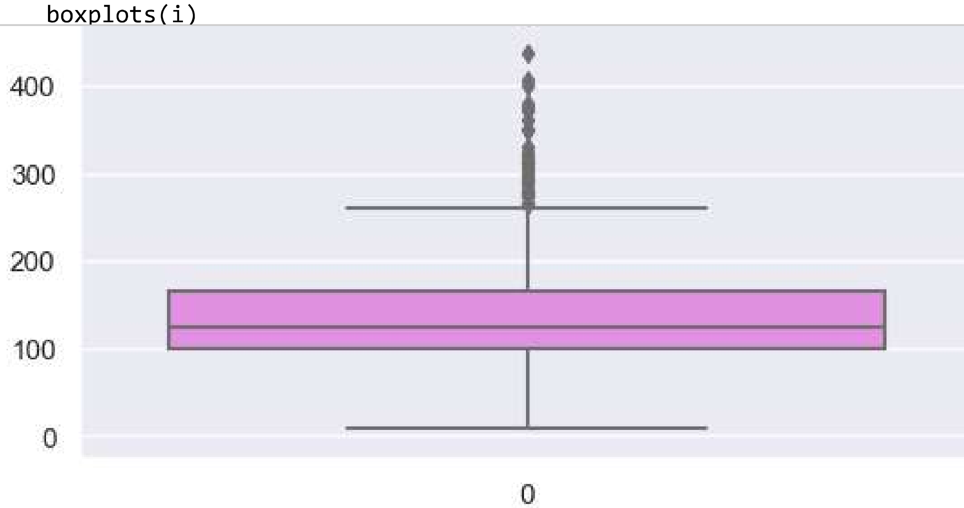
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            614 non-null    object 
 1   Gender             614 non-null    object 
 2   Married            614 non-null    object 
 3   Dependents         614 non-null    object 
 4   Education          614 non-null    object 
 5   Self_Employed      614 non-null    object 
 6   ApplicantIncome    614 non-null    float64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         614 non-null    float64
 9   Loan_Amount_Term   614 non-null    float64
 10  Credit_History     614 non-null    float64
 11  Property_Area      614 non-null    object 
 12  Loan_Status        614 non-null    object 
dtypes: float64(5), object(8)
memory usage: 62.5+ KB
```

In [ ]:

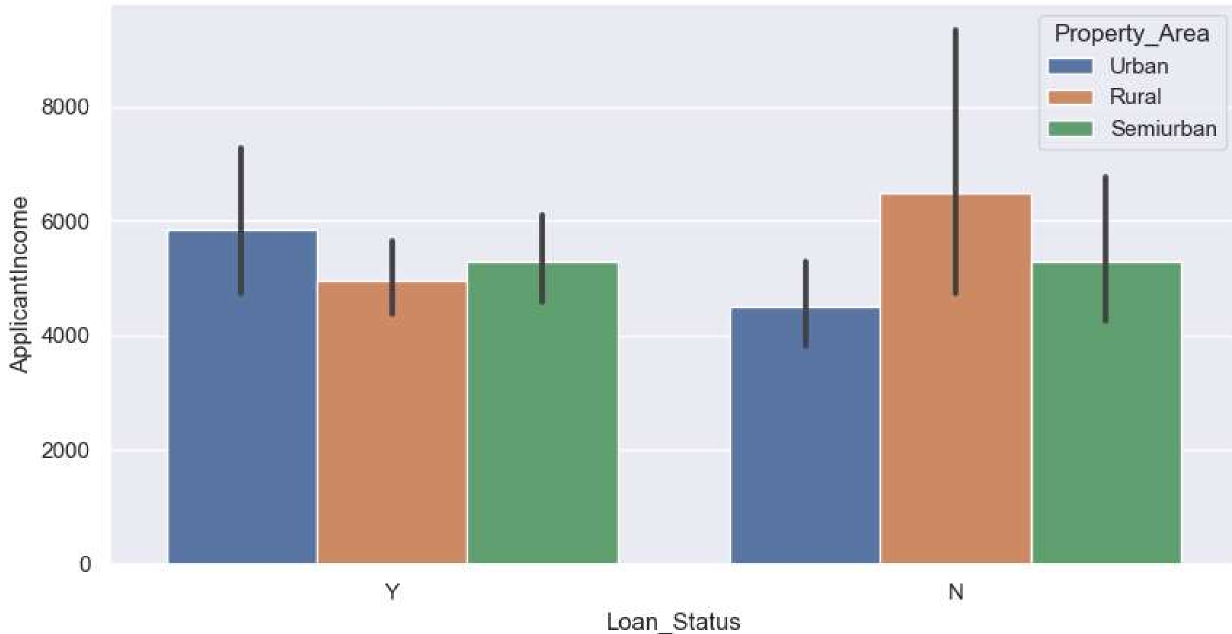
In [16]:
```
# Find the distribution of the dataset
def distplots(col):
    sns.distplot(df_imputed[col],color='yellow')
    plt.show()

for i in list(df_imputed.select_dtypes(exclude=['object']).columns)[0:]:
    distplots(i)
```



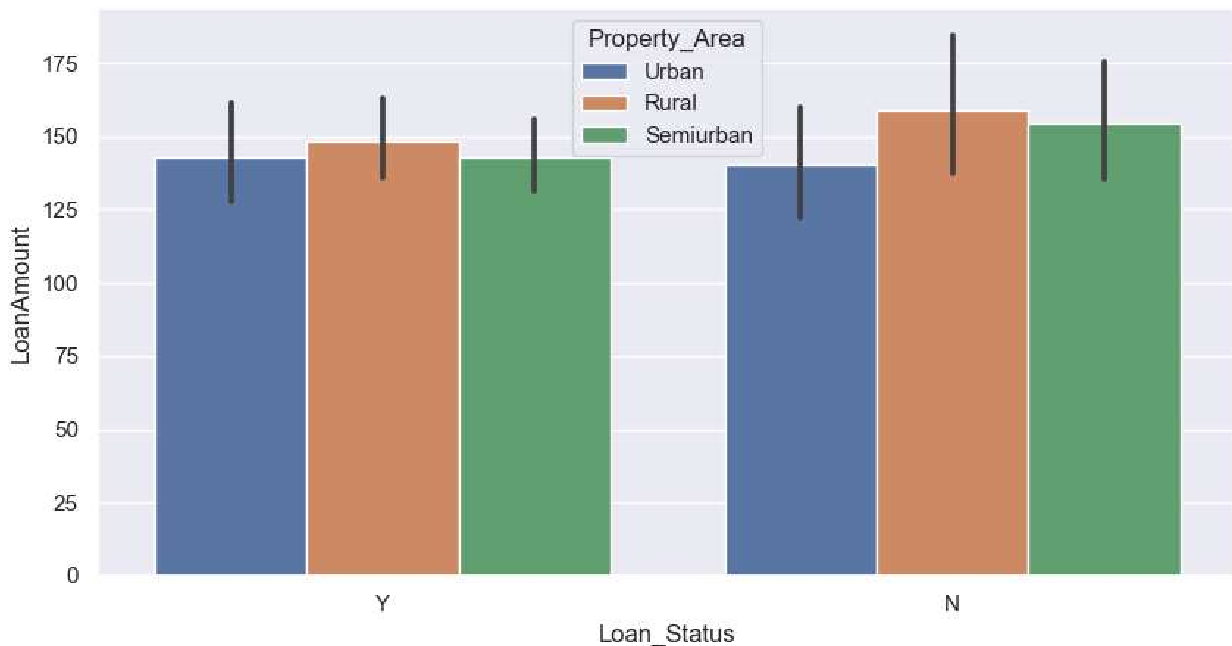
In [17]:
```
# Find the outlier
def boxplots(col):
    sns.boxplot(df_imputed[col],color='violet')
    plt.show()

for i in list(df_imputed.select_dtypes(exclude=['object']).columns)[0:]:
    boxplots(i)
```



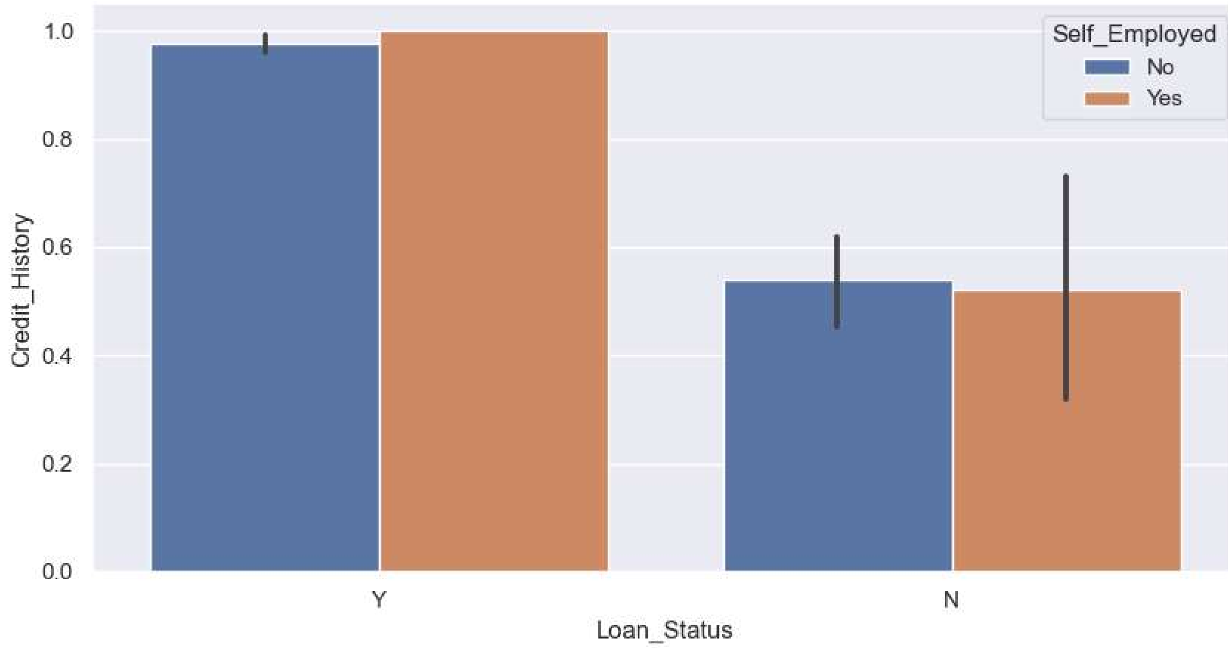
In [18]:
```
plt.figure(figsize=(10,5),dpi=100)
sns.barplot(y='ApplicantIncome',x='Loan_Status',hue='Property_Area',data=df)
plt.show()
```



In [19]:
```
plt.figure(figsize=(10,5),dpi=100)
sns.barplot(y='LoanAmount',x='Loan_Status',hue='Property_Area',data=df)
plt.show()
```



In [20]:
```
plt.figure(figsize=(10,5),dpi=100)
sns.barplot(y='Credit_History',x='Loan_Status',hue='Self_Employed',data=df)
plt.show()
```



In [ ]:

In [21]:
```
# Label encoding to convert categorical values to numerical
from sklearn import preprocessing
```

In [22]:
```
df_enco = df_imputed.apply(preprocessing.LabelEncoder().fit_transform)
df_enco
```

Out[22]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Lo
0	0	1	0	0	0	0	376	0	
1	1	1	1	1	0	0	306	60	
2	2	1	1	0	0	1	139	0	
3	3	1	1	0	1	0	90	160	
4	4	1	0	0	0	0	381	0	
...	...	...	...	...	...	...	...	...	
609	609	0	0	0	0	0	125	0	
610	610	1	1	3	0	0	275	0	
611	611	1	1	1	0	0	431	3	
612	612	1	1	2	0	0	422	0	
613	613	0	0	0	0	1	306	0	

614 rows × 13 columns

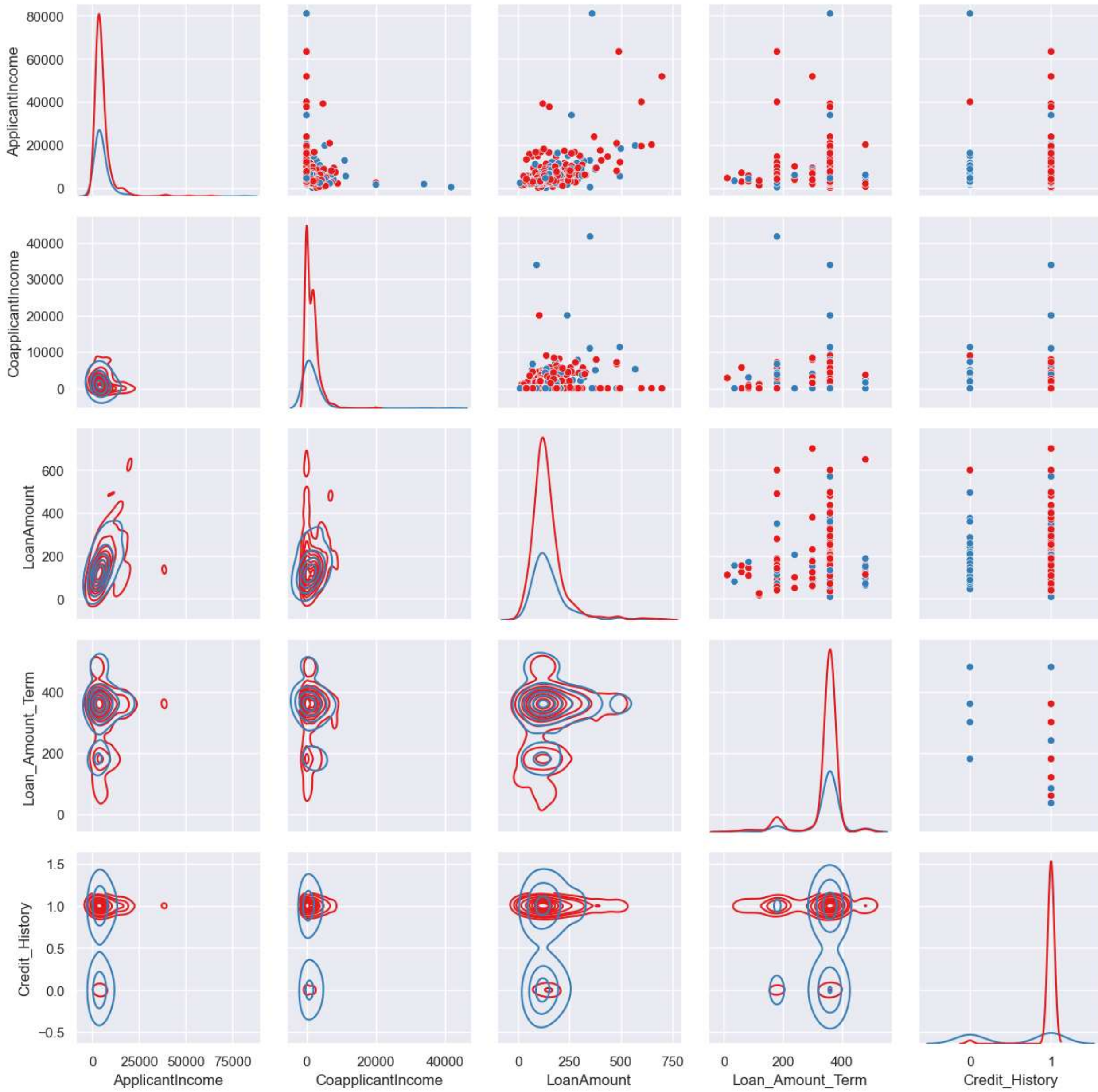
In [23]:
```
df_enco.info()
```

```
<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    int32
 1   Gender             614 non-null    int32
 2   Married            614 non-null    int32
 3   Dependents         614 non-null    int32
 4   Education          614 non-null    int32
 5   Self_Employed      614 non-null    int32
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    int64
 8   LoanAmount         614 non-null    int64
 9   Loan_Amount_Term   614 non-null    int64
 10  Credit_History     614 non-null    int64
 11  Property_Area      614 non-null    int32
 12  Loan_Status        614 non-null    int32
dtypes: int32(8), int64(5)
memory usage: 43.3 KB
```

In [ ]:

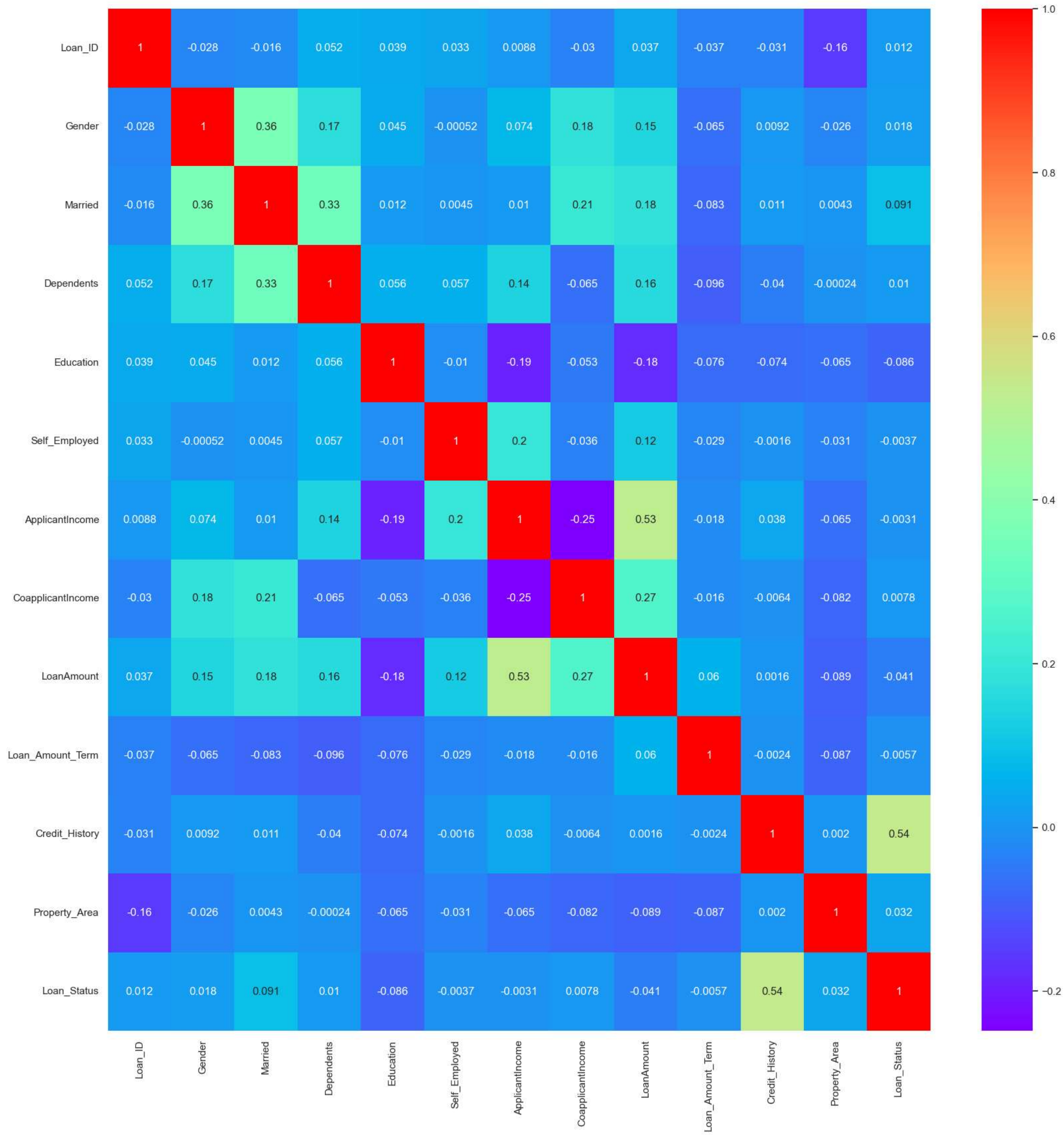
In [ ]:

In [24]:
```
g = sns.PairGrid(df, hue='Loan_Status',palette ='Set1' , diag_sharey=False)
g.map_upper(sns.scatterplot)
g.map_lower(sns.kdeplot)
g.map_diag(sns.kdeplot)
plt.show()
```



In [25]:
```
# Finding correlation
plt.figure(figsize=(20,20))
corr = df_enco.corr()
sns.heatmap(corr, annot=True, cmap='rainbow')
```

Out[25]:
```
<Axes: >
```



In [26]:
```
df_enco.columns
```

Out[26]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [27]:
```
# seperate independent and dependent variables and drop the ID column

x = df_enco.drop(['Loan_ID','Loan_Status'], axis=1)
y = df_enco[['Loan_Status']]
```

In [28]:
```
x.head()
```

Out[28]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount
0	1	0	0	0	0	376	0	73
1	1	1	1	0	0	306	60	81
2	1	1	0	0	1	139	0	26
3	1	1	0	1	0	90	160	73
4	1	0	0	0	0	381	0	94

In [29]:
```
y.head()
```

Out[29]:

	Loan_Status
0	1
1	0
2	1
3	1
4	1

In [30]:
```
y.value_counts()
```

Out[30]:
```
Loan_Status
1              422
0              192
dtype: int64
```

In [31]:
```
y.value_counts()/len(y)*100
```

Out[31]:
```
Loan_Status
1              68.729642
0              31.270358
dtype: float64
```

In [32]:
```
# balance the dataset
import imblearn
from imblearn.over_sampling import RandomOverSampler
from collections import Counter
print(Counter(y))
```

```
Counter({'Loan_Status': 1})
```

In [33]:
```
ros = RandomOverSampler()
x_ros, y_ros = ros.fit_resample(x, y)
print(Counter(y_ros))
```

```
Counter({'Loan_Status': 1})
```

In [34]:
```
print(y.value_counts())
print()
print(y_ros.value_counts())
```

```
Loan_Status
1        422
0        192
dtype: int64

Loan_Status
0        422
1        422
dtype: int64
```

In [35]:
```
x_ros.describe()
```

Out[35]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Lo
count	844.000000	844.000000	844.000000	844.000000	844.000000	844.000000	844.000000	8
mean	0.817536	0.637441	0.734597	0.226303	0.139810	250.021327	75.188389	
std	0.386456	0.481024	1.003286	0.418686	0.346996	144.111577	91.655701	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	0.000000	0.000000	0.000000	0.000000	126.000000	0.000000	
50%	1.000000	1.000000	0.000000	0.000000	0.000000	243.500000	20.500000	
75%	1.000000	1.000000	1.000000	0.000000	0.000000	379.250000	145.250000	1
max	1.000000	1.000000	3.000000	1.000000	1.000000	504.000000	286.000000	2

In [36]:
```
# Feature Scaling - Normalization, Standarisation, MinMax
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler((-1,1))
x = scaler.fit_transform(x_ros)
y = y_ros
```

In [37]:
```
x
```

Out[37]:
```
array([[ 1.        , -1.        , -1.        , ...,  0.77777778,
         1.        ,  1.        ],
       [ 1.        ,  1.        , -0.33333333, ...,  0.77777778,
         1.        , -1.        ],
       [ 1.        ,  1.        , -1.        , ...,  0.77777778,
         1.        ,  1.        ],
       ...,
       [ 1.        ,  1.        , -1.        , ...,  0.77777778,
        -1.        ,  1.        ],
       [ 1.        ,  1.        , -1.        , ...,  0.77777778,
         1.        , -1.        ],
       [ 1.        , -1.        , -1.        , ...,  0.77777778,
         1.        ,  1.        ]])
```

In [38]:
```
y
```

Out[38]:

	Loan_Status
0	1
1	0
2	1
3	1
4	1
...	...
839	0
840	0
841	0
842	0
843	0

844 rows × 1 columns

Dimension Reduction - Principal Component Analysis (PCA)

In [39]:
```
from sklearn.decomposition import PCA
```

In [40]:
```
pca = PCA(0.95)
x_pca = pca.fit_transform(x)
print(x.shape)
print(x_pca.shape)
```

```
(844, 11)
(844, 9)
```

In [41]:
```
# Split the data into training and test for model building
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_pca, y, test_size=0.2, random_state=7
```

In [42]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
#from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB

from sklearn.ensemble import VotingClassifier

from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
```

In [43]:
```
!pip install xgboost
from xgboost import XGBClassifier
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: xgboost in c:\users\vikas\appdata\roaming\python\python310
\site-packages (1.7.6)
Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-packages (from x
gboost) (1.10.0)
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (from x
gboost) (1.23.5)
```

Applying all the model together

In [44]:
```
# LogisticRegression
logistic = LogisticRegression()
lr = logistic.fit(x_train, y_train)
y_pred_lr = logistic.predict(x_test)
accuracy_lr = accuracy_score(y_test, y_pred_lr)

# DecisionTree
dtree = DecisionTreeClassifier()
dt = dtree.fit(x_train, y_train)
y_pred_dt = dtree.predict(x_test)
accuracy_dt = accuracy_score(y_test, y_pred_dt)

# RandomForest
rfmodel = RandomForestClassifier()
rf = rfmodel.fit(x_train, y_train)
y_pred_rf = rfmodel.predict(x_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)

# BaggingClassifier
bagg = BaggingClassifier()
bg = bagg.fit(x_train, y_train)
y_pred_bg = bagg.predict(x_test)
accuracy_bg = accuracy_score(y_test, y_pred_bg)

# AdaBoostClassifier
ada = AdaBoostClassifier()
ad = ada.fit(x_train, y_train)
y_pred_ad = ada.predict(x_test)
accuracy_ad = accuracy_score(y_test, y_pred_ad)

# GradientBoostingClassifier
gdb = GradientBoostingClassifier()
gd = gdb.fit(x_train, y_train)
y_pred_gd = gdb.predict(x_test)
accuracy_gd = accuracy_score(y_test, y_pred_gd)

# XGBClassifier = RF + GDBoosting - lambda - regularisation, gamma - autoprunning, eta - lea
xgb = XGBClassifier()
xg = xgb.fit(x_train, y_train)
y_pred_xg = xgb.predict(x_test)
accuracy_xg = accuracy_score(y_test, y_pred_xg)


# SVM
svc = SVC()
sv = svc.fit(x_train, y_train)
y_pred_sv = svc.predict(x_test)
accuracy_sv = accuracy_score(y_test, y_pred_sv)

# KNN
knn = KNeighborsClassifier()
kn = knn.fit(x_train, y_train)
y_pred_knn = knn.predict(x_test)
accuracy_knn = accuracy_score(y_test, y_pred_knn)

# GaussianNB
naive_gb = GaussianNB()
ngb = naive_gb.fit(x_train, y_train)
y_pred_ngb = naive_gb.predict(x_test)
accuracy_ngb = accuracy_score(y_test, y_pred_ngb)

# BernoulliNB
naive_bn = BernoulliNB()
```

```
nbr = naive_bn.fit(x_train, y_train)
y_pred_nbr = naive_bn.predict(x_test)
accuracy_nbr = accuracy_score(y_test, y_pred_nbr)
```

In [45]:
```
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import StackingClassifier
```

In [46]:
```
evc = VotingClassifier(estimators=[('lr',lr),('dt',dt),('rf', rf),('bg', bg),('ad',ad),
                                  ('gd', gd),('xg', xg),('sv', sv),('kn', kn),
                                  ('ngb', ngb),('nbr', nbr)], voting='hard')


model_evc = evc.fit(x_train, y_train)
pred_evc = evc.predict(x_test)
accuracy_evc = accuracy_score(y_test, pred_evc)
```

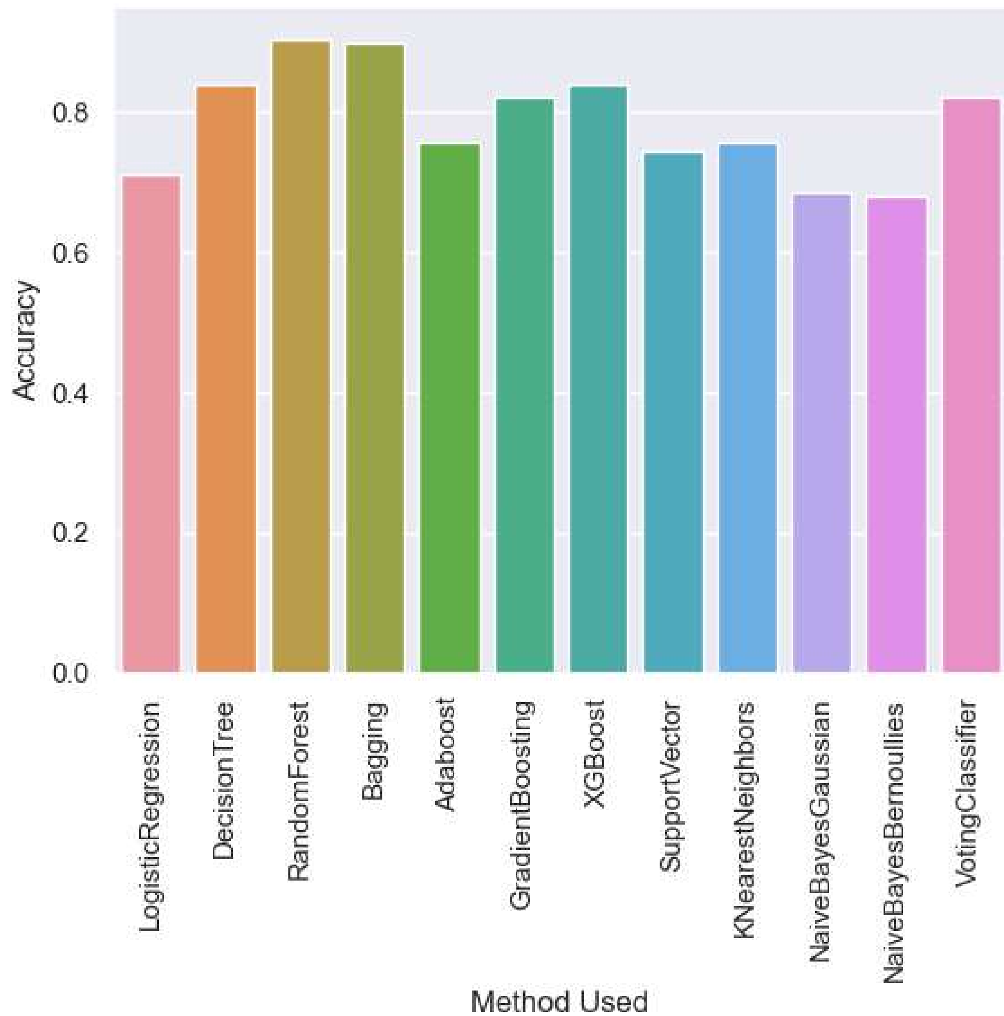
In [47]:
```
list1 = ['LogisticRegression','DecisionTree','RandomForest','Bagging','Adaboost',
         'GradientBoosting', 'XGBoost','SupportVector','KNearestNeighbors',
         'NaiveBayesGaussian','NaiveBayesBernoullies','VotingClassifier']
```

In [48]:
```
list2 = [accuracy_lr, accuracy_dt, accuracy_rf, accuracy_bg,accuracy_ad, accuracy_gd,
         accuracy_xg, accuracy_sv, accuracy_knn, accuracy_ngb, accuracy_nbr, accuracy_evc]
```

In [49]:
```
list3 = [logistic, dtree, rfmodel, bagg, ada, gdb, xgb, svc, knn, naive_gb,naive_bn, evc ]
```

In [50]:
```
final_accuracy = pd.DataFrame({'Method Used': list1, "Accuracy": list2})
print(final_accuracy)
charts = sns.barplot(x="Method Used", y = 'Accuracy', data=final_accuracy)
charts.set_xticklabels(charts.get_xticklabels(), rotation=90)
print(charts)
```

```
              Method Used  Accuracy
0      LogisticRegression  0.710059
1            DecisionTree  0.840237
2            RandomForest  0.905325
3                 Bagging  0.899408
4                Adaboost  0.757396
5        GradientBoosting  0.822485
6                 XGBoost  0.840237
7           SupportVector  0.745562
8        KNearestNeighbors  0.757396
9      NaiveBayesGaussian  0.686391
10  NaiveBayesBernoullies  0.680473
11        VotingClassifier  0.822485
Axes(0.125,0.11;0.775x0.77)
```



In [51]:
```
# GradientBoostingClassifier
gdb = GradientBoostingClassifier()
gd = gdb.fit(x_train, y_train)
y_pred_gd_train = gdb.predict(x_train)
y_pred_gd_test = gdb.predict(x_test)
accuracy_gd_test = accuracy_score(y_test, y_pred_gd_test)
accuracy_gd_train = accuracy_score(y_train, y_pred_gd_train)
print(accuracy_gd_train)
print()
print(accuracy_gd_test)
```

```
0.9392592592592592

0.8224852071005917
```

In [52]:
```
from sklearn.model_selection import cross_val_score
training_accuracy = cross_val_score(gdb, x_train, y_train, cv=15)
test_accuracy = cross_val_score(gdb, x_test, y_test, cv=15)
print(training_accuracy[7])
print(test_accuracy[9])
```

```
0.8222222222222222
0.8181818181818182
```

In [ ]:

In [ ]: