Loan Status Prediction

In [ ]:

Importing the Libraries

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

In [ ]:

Importing the data set

In [2]:
```
data= pd.read_csv('Loan Status Prediction.csv')
```

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

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantI
0	LP001003	Male	Yes	1	Graduate	No	4583	
1	LP001005	Male	Yes	0	Graduate	Yes	3000	
2	LP001006	Male	Yes	0	Not Graduate	No	2583	
3	LP001008	Male	No	0	Graduate	No	6000	
4	LP001013	Male	Yes	0	Not Graduate	No	2333	

In [4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 381 entries, 0 to 380
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            381 non-null    object
 1   Gender             376 non-null    object
 2   Married            381 non-null    object
 3   Dependents         373 non-null    object
 4   Education          381 non-null    object
 5   Self_Employed      360 non-null    object
 6   ApplicantIncome    381 non-null    int64
 7   CoapplicantIncome  381 non-null    float64
 8   LoanAmount         381 non-null    float64
 9   Loan_Amount_Term   370 non-null    float64
 10  Credit_History     351 non-null    float64
 11  Property_Area      381 non-null    object
 12  Loan_Status        381 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 38.8+ KB
```

In [ ]:

Statistical Description of Data

In [5]:
```
data.describe()
```

Out[5]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	381.000000	381.000000	381.000000	370.000000	351.000000
mean	3579.845144	1277.275381	104.986877	340.864865	0.837607
std	1419.813818	2340.818114	28.358464	68.549257	0.369338
min	150.000000	0.000000	9.000000	12.000000	0.000000
25%	2600.000000	0.000000	90.000000	360.000000	1.000000
50%	3333.000000	983.000000	110.000000	360.000000	1.000000
75%	4288.000000	2016.000000	127.000000	360.000000	1.000000
max	9703.000000	33837.000000	150.000000	480.000000	1.000000

In [ ]:

Checking the Null Values

In [6]:
```
data.isnull().sum()
```

Out[6]:
```
Loan_ID               0
Gender                5
Married               0
Dependents            8
Education             0
Self_Employed        21
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            0
Loan_Amount_Term     11
Credit_History       30
Property_Area         0
Loan_Status           0
dtype: int64
```

In [ ]:

Checking the Unique Values

In [7]:
```
data['Education'].unique()
```

Out[7]:
```
array(['Graduate', 'Not Graduate'], dtype=object)
```

In [8]:
```
data['Property_Area'].unique()
```

Out[8]:
```
array(['Rural', 'Urban', 'Semiurban'], dtype=object)
```

In [9]:
```
data['Married'].unique()
```

Out[9]:
```
array(['Yes', 'No'], dtype=object)
```

In [10]:
```
data['Dependents'].unique()
```

Out[10]:
```
array(['1', '0', '2', '3+', nan], dtype=object)
```

In [11]:
```
data['Self_Employed'].unique()
```

Out[11]:
```
array(['No', 'Yes', nan], dtype=object)
```

In [12]:
```
data['Loan_Status'].unique()
```

Out[12]:
```
array(['N', 'Y'], dtype=object)
```

In [ ]:

Droping the Unnecessary Column

In [13]:
```
data.drop(['Loan_ID'],axis=1, inplace=True)
```

In [14]:
```
data.head()
```

Out[14]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Lo
0	Male	Yes	1	Graduate	No	4583	1508.0	
1	Male	Yes	0	Graduate	Yes	3000	0.0	
2	Male	Yes	0	Not Graduate	No	2583	2358.0	
3	Male	No	0	Graduate	No	6000	0.0	
4	Male	Yes	0	Not Graduate	No	2333	1516.0	

In [ ]:

Filling the Null Values

In [15]:
```
data['Gender'].fillna(data['Gender'].mode()[0],inplace=True)
```

In [16]:
```
data.isnull().sum()
```

Out[16]:
```
Gender               0
Married              0
Dependents           8
Education            0
Self_Employed       21
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term    11
Credit_History      30
Property_Area        0
Loan_Status          0
dtype: int64
```

In [17]:
```
data['Dependents'].fillna(data['Dependents'].mode()[0],inplace=True)
```

In [18]:
```
data['Self_Employed'].fillna(data['Self_Employed'].mode()[0],inplace=True)
```

In [19]:
```
data['Loan_Amount_Term'].fillna(data['Loan_Amount_Term'].median(),inplace=True)
```

In [20]:
```
data['Credit_History'].fillna(data['Credit_History'].median(),inplace=True)
```

In [21]:
```
data.isnull().sum()
```

Out[21]:
```
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 [ ]:
```
```

Providing the numerical labels to the category using Label Encoder

In [22]:
```
data.head()
```

Out[22]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Lo
0	Male	Yes	1	Graduate	No	4583	1508.0	
1	Male	Yes	0	Graduate	Yes	3000	0.0	
2	Male	Yes	0	Not Graduate	No	2583	2358.0	
3	Male	No	0	Graduate	No	6000	0.0	
4	Male	Yes	0	Not Graduate	No	2333	1516.0	

In [23]:
```
from sklearn.preprocessing import LabelEncoder
```

In [24]:
```
LE=LabelEncoder()
```

In [25]:
```
data['Loan_Status'] = LE.fit_transform(data['Loan_Status'].astype(str))
```

In [26]:
```
data.head()
```

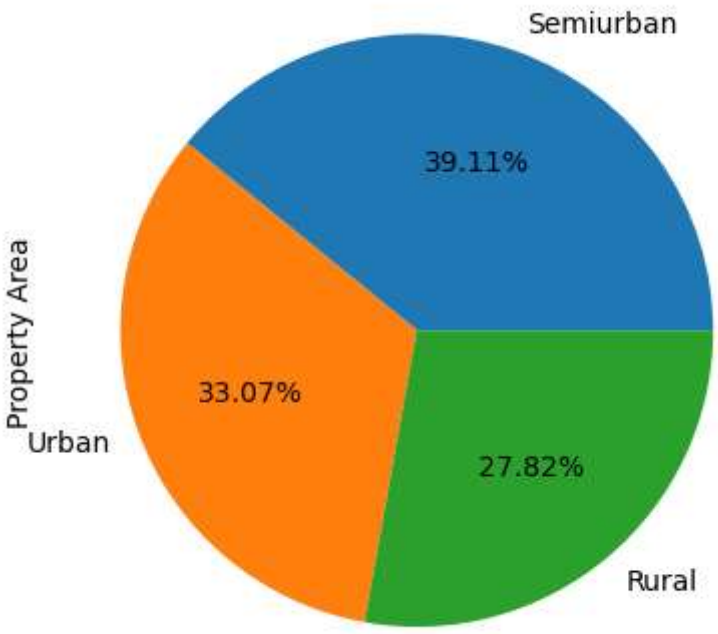
Out[26]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Lo
0	Male	Yes	1	Graduate	No	4583	1508.0	
1	Male	Yes	0	Graduate	Yes	3000	0.0	
2	Male	Yes	0	Not Graduate	No	2583	2358.0	
3	Male	No	0	Graduate	No	6000	0.0	
4	Male	Yes	0	Not Graduate	No	2333	1516.0	

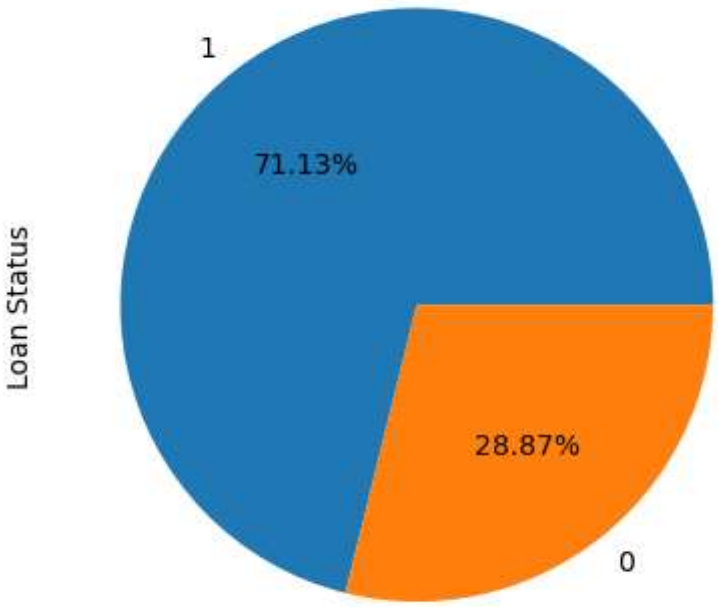
In [ ]:

Data Visualization

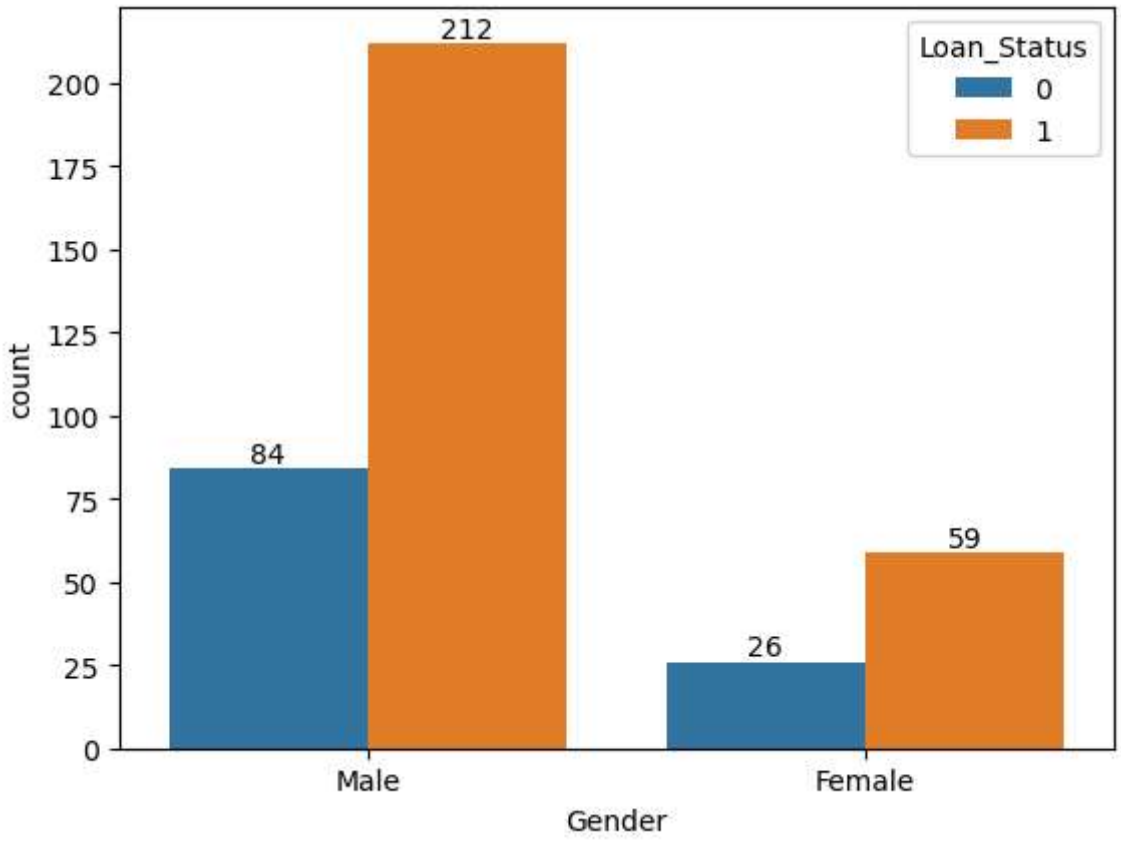
In [27]:
```
data['Property_Area'].value_counts().plot(kind='pie',autopct='%0.2f%%')
plt.ylabel('Property Area')
plt.show()
```



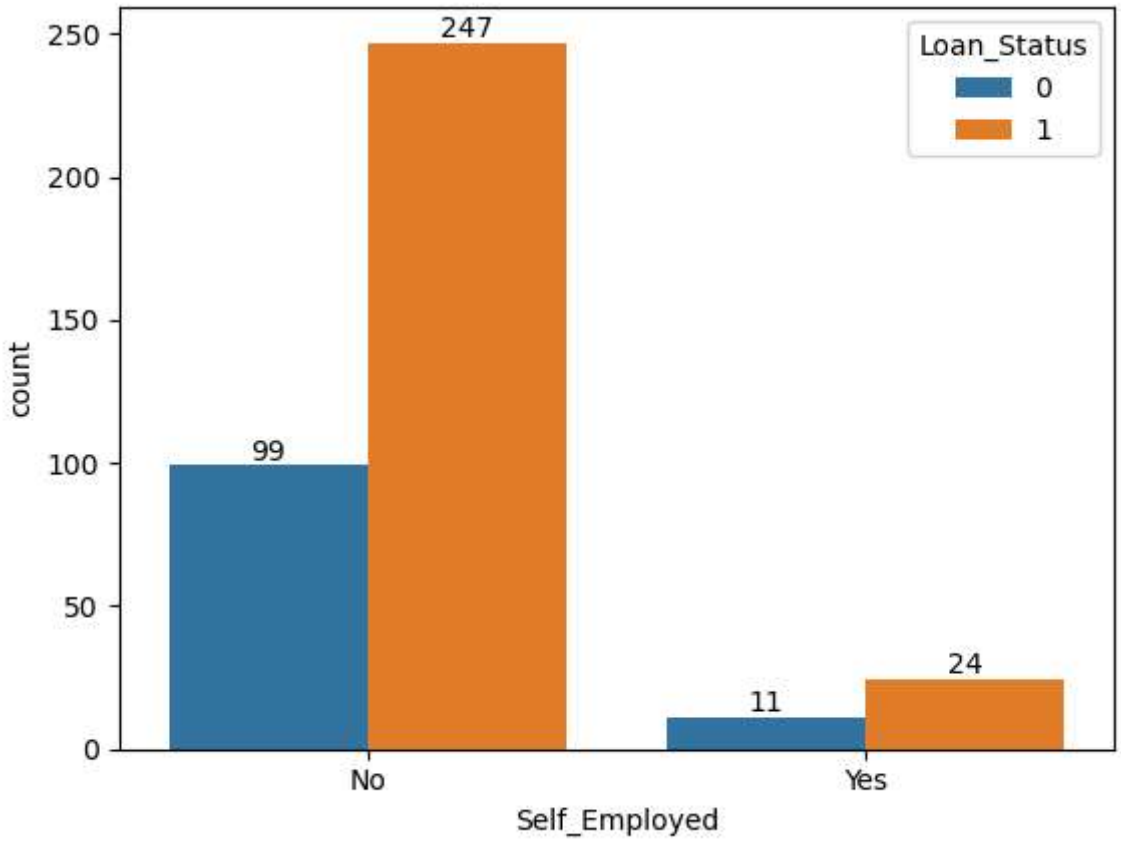
In [28]:
```
data['Loan_Status'].value_counts().plot(kind='pie',autopct='%0.2f%%')
plt.ylabel('Loan Status')
plt.show()
```



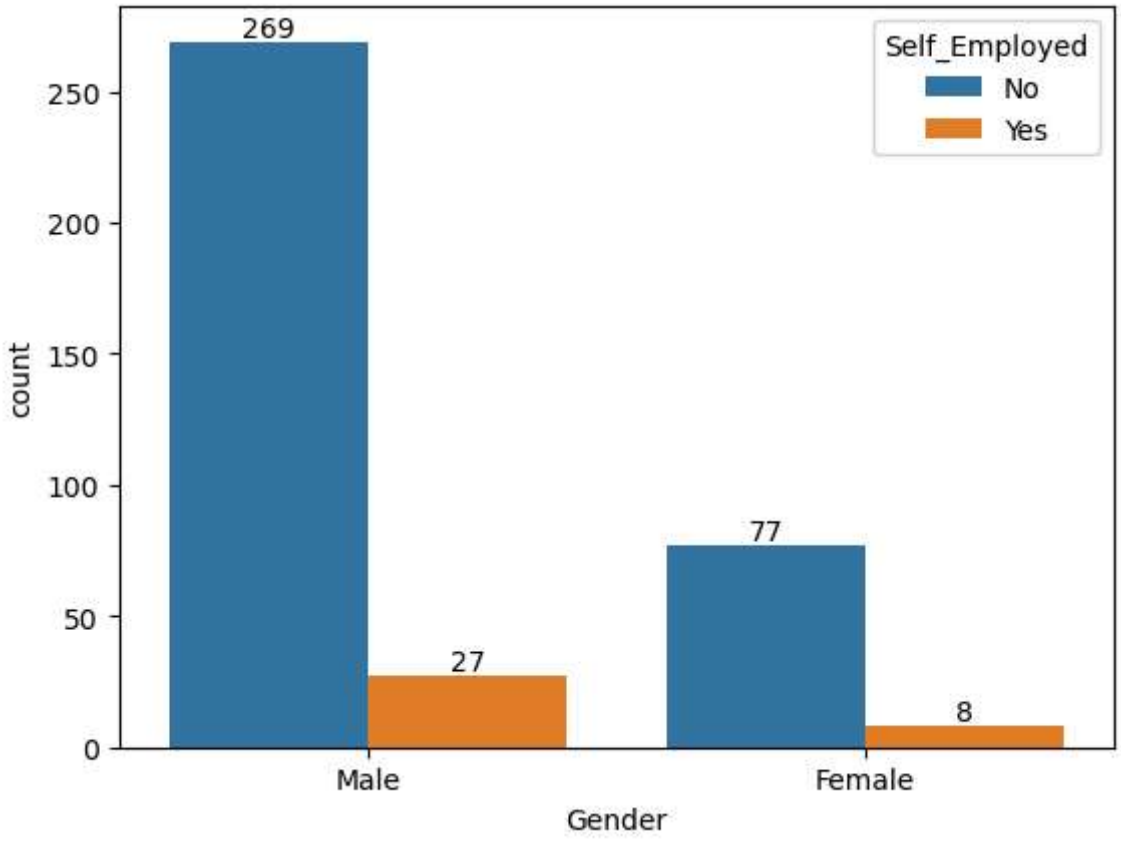
In [29]:
```
Gen=sns.countplot(data=data,x='Gender',hue='Loan_Status')
for i in Gen.containers:
    plt.bar_label(i)
```



In [30]:
```
SelfE=sns.countplot(data=data,x='Self_Employed',hue='Loan_Status')
for i in SelfE.containers:
    plt.bar_label(i)
```



In [31]:
```
A=sns.countplot(data=data,x='Gender',hue='Self_Employed')
for i in A.containers:
    plt.bar_label(i)
```



In [ ]:

Checking for Outliers

In [32]:
```
data.info()
```

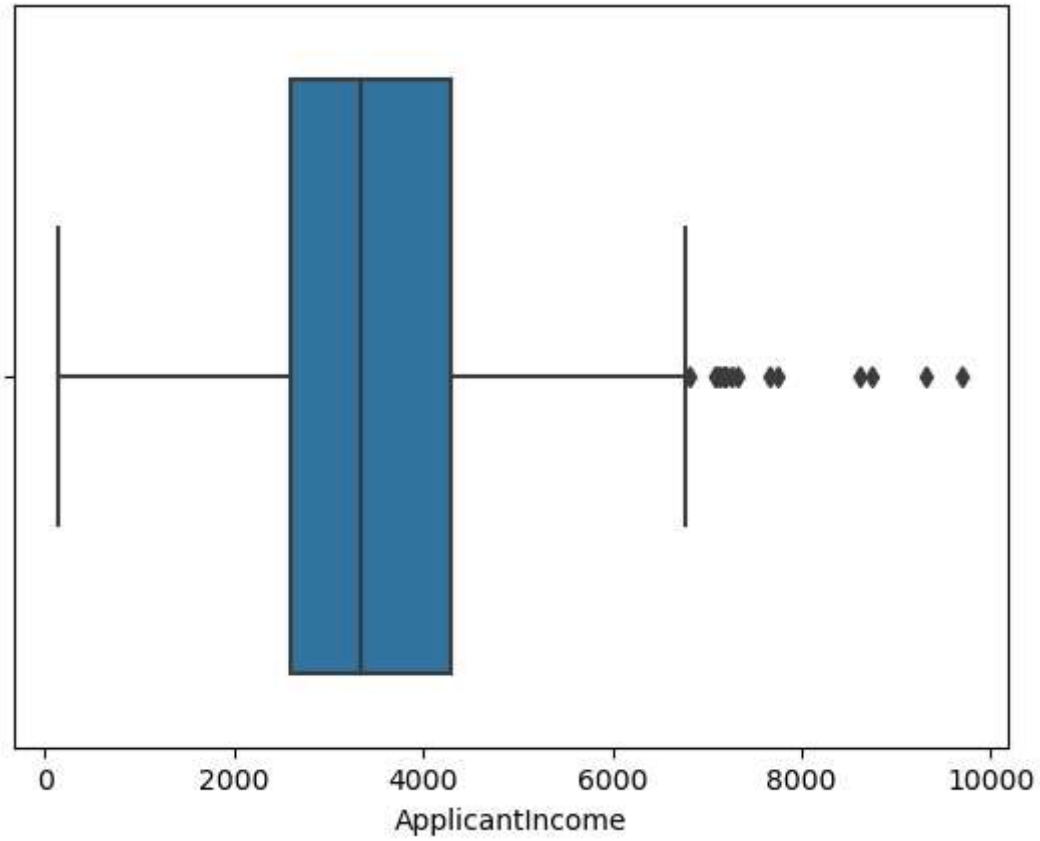
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 381 entries, 0 to 380
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Gender             381 non-null    object 
 1   Married            381 non-null    object 
 2   Dependents         381 non-null    object 
 3   Education          381 non-null    object 
 4   Self_Employed      381 non-null    object 
 5   ApplicantIncome    381 non-null    int64  
 6   CoapplicantIncome  381 non-null    float64
 7   LoanAmount         381 non-null    float64
 8   Loan_Amount_Term   381 non-null    float64
 9   Credit_History     381 non-null    float64
 10  Property_Area      381 non-null    object 
 11  Loan_Status        381 non-null    int32  
dtypes: float64(4), int32(1), int64(1), object(6)
memory usage: 34.4+ KB
```

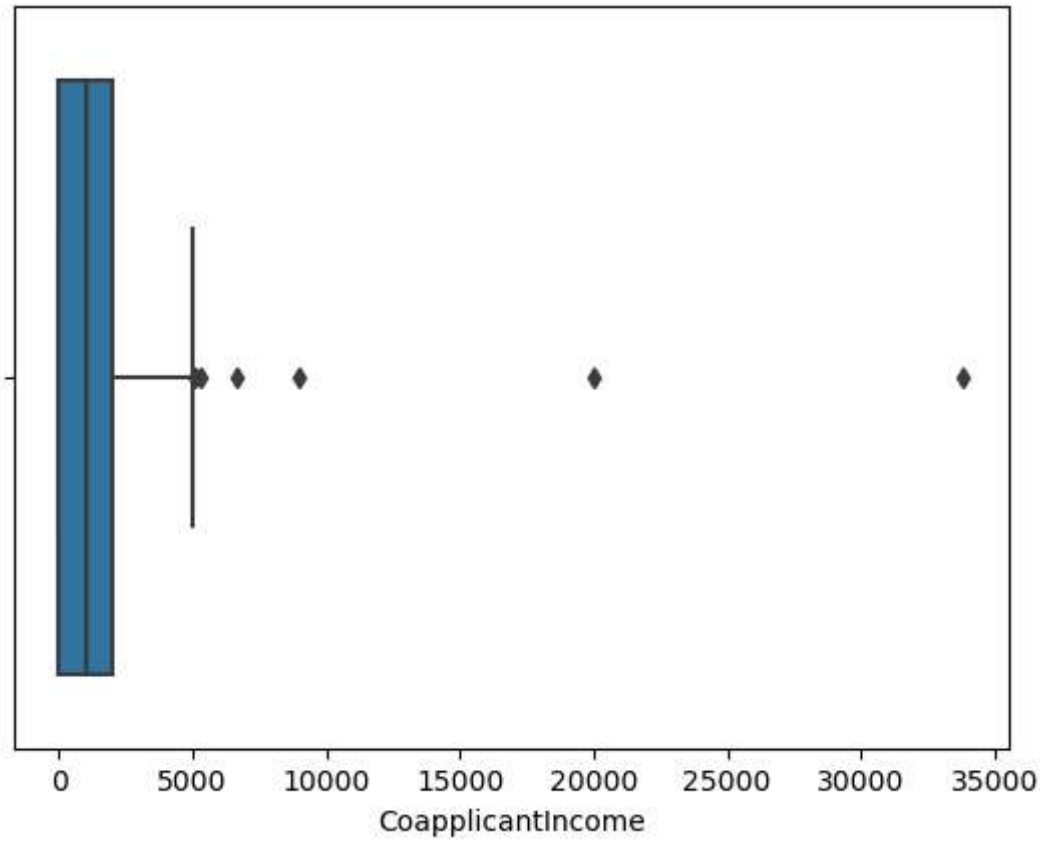
In [33]:
```
data.head()
```

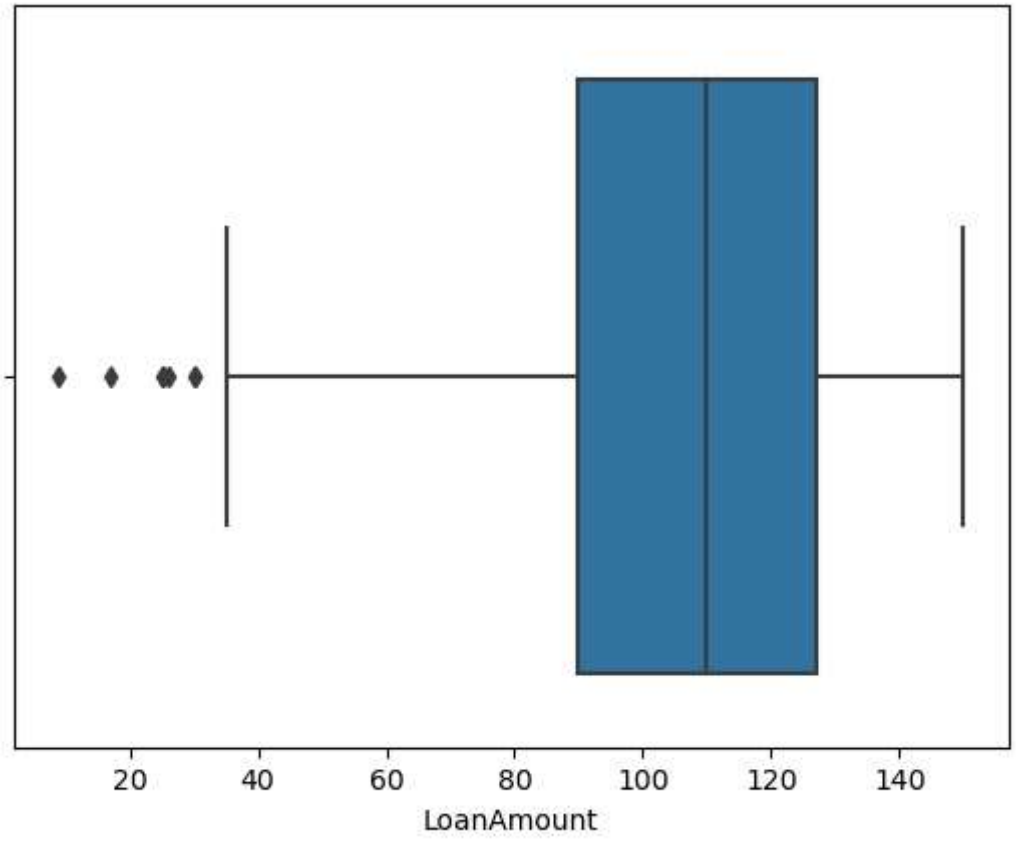
Out[33]:

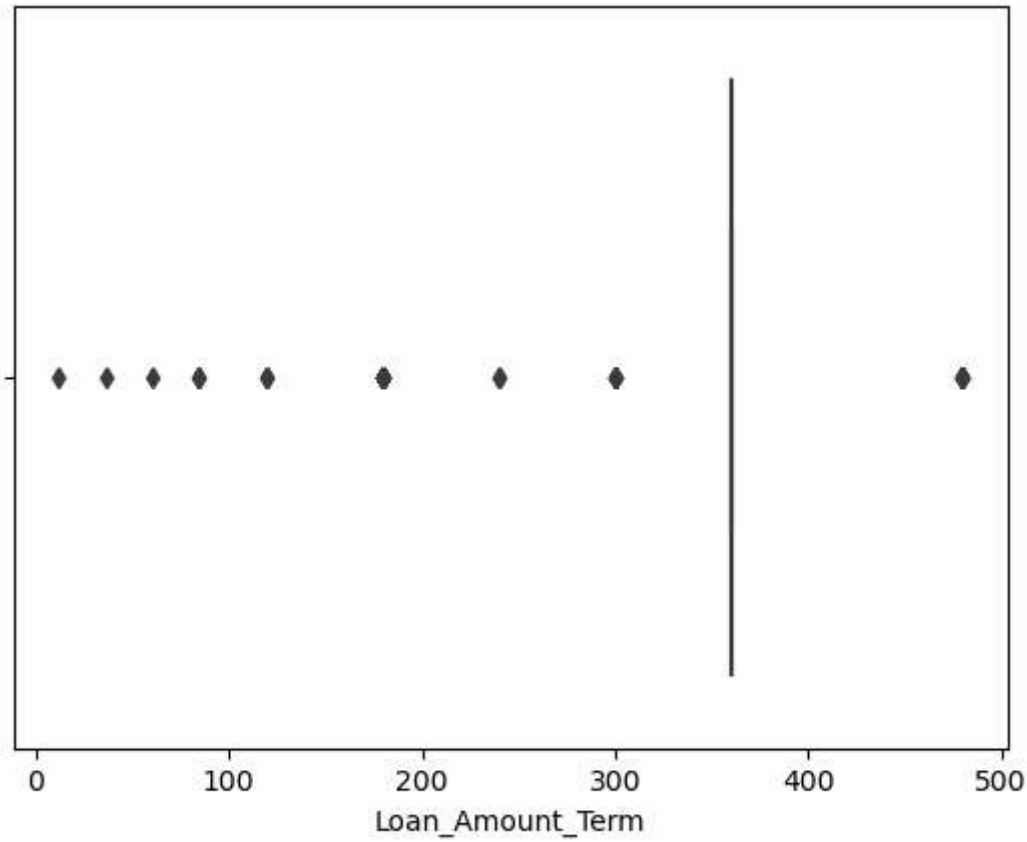
	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Lo
0	Male	Yes	1	Graduate	No	4583	1508.0	
1	Male	Yes	0	Graduate	Yes	3000	0.0	
2	Male	Yes	0	Not Graduate	No	2583	2358.0	
3	Male	No	0	Graduate	No	6000	0.0	
4	Male	Yes	0	Not Graduate	No	2333	1516.0	

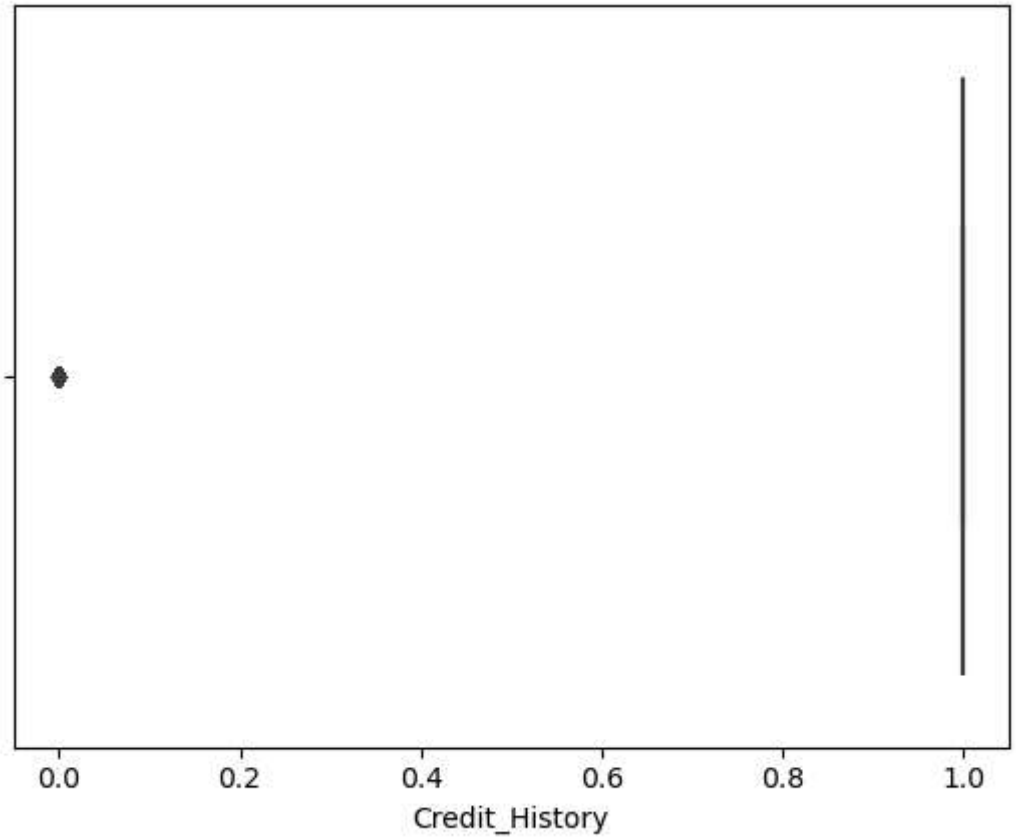
In [34]:
```
for i in data.select_dtypes(['int','float']):
    sns.boxplot(data=data,x=i)
    plt.show()
```

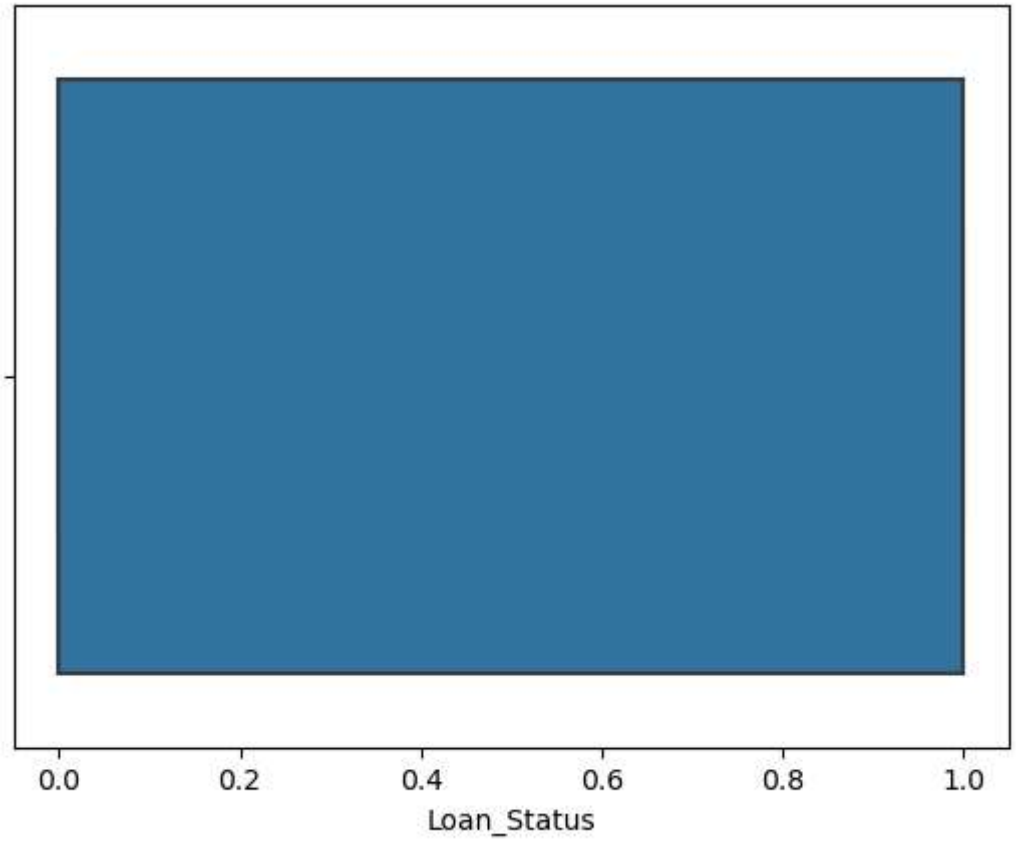

0
2000
4000
6000
8000
10000
ApplicantIncome


0
5000
10000
15000
20000
25000
30000
35000
CoapplicantIncome


20
40
60
80
100
120
140
LoanAmount


0
100
200
300
400
500
Loan_Amount_Term





In [ ]:

Treating the Outliers

In [35]:
```
def outlier(col):
    Q1=data[col].quantile(0.25)
    Q3=data[col].quantile(0.75)
    IQR=Q3-Q1
    upper_limit=Q3+1.5*IQR
    lower_limit=Q1-1.5*IQR
    upper_outlier=data[col]>upper_limit
    lower_outlier=data[col]<lower_limit

    data.loc[upper_outlier,col]=data[col].median()
    data.loc[lower_outlier,col]=data[col].median()
    return data
```

In [36]:
```
data.select_dtypes(['int','float'])
```

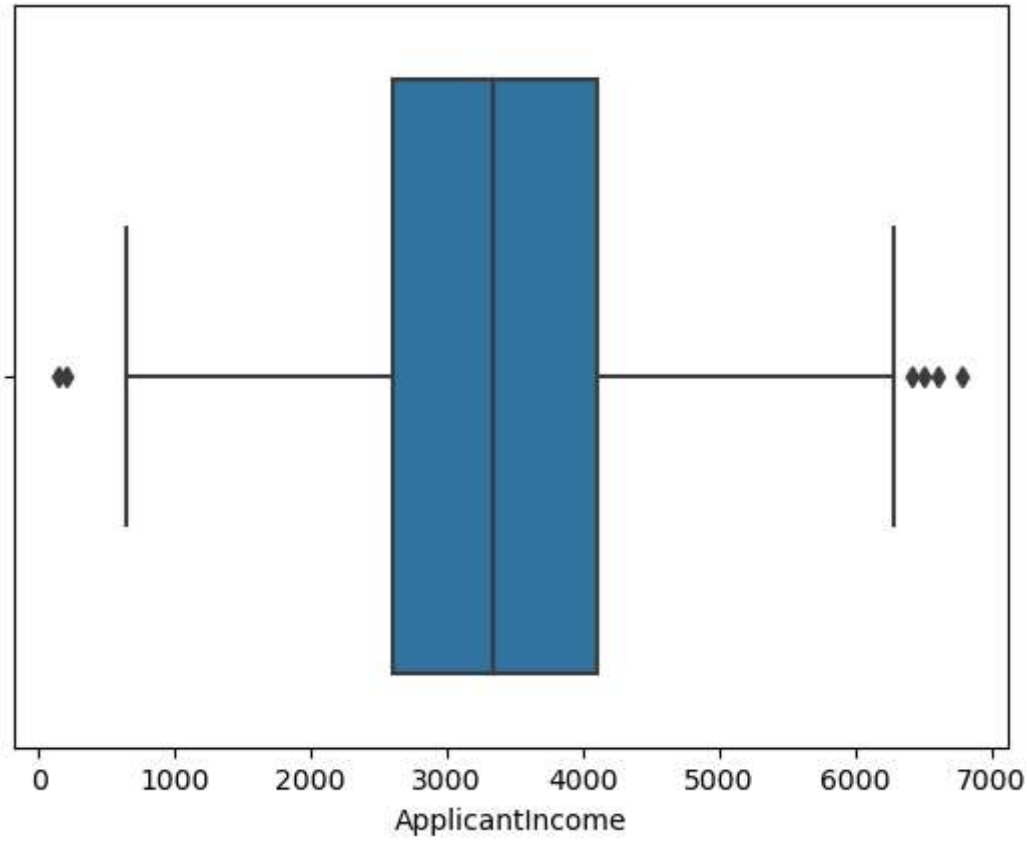
Out[36]:

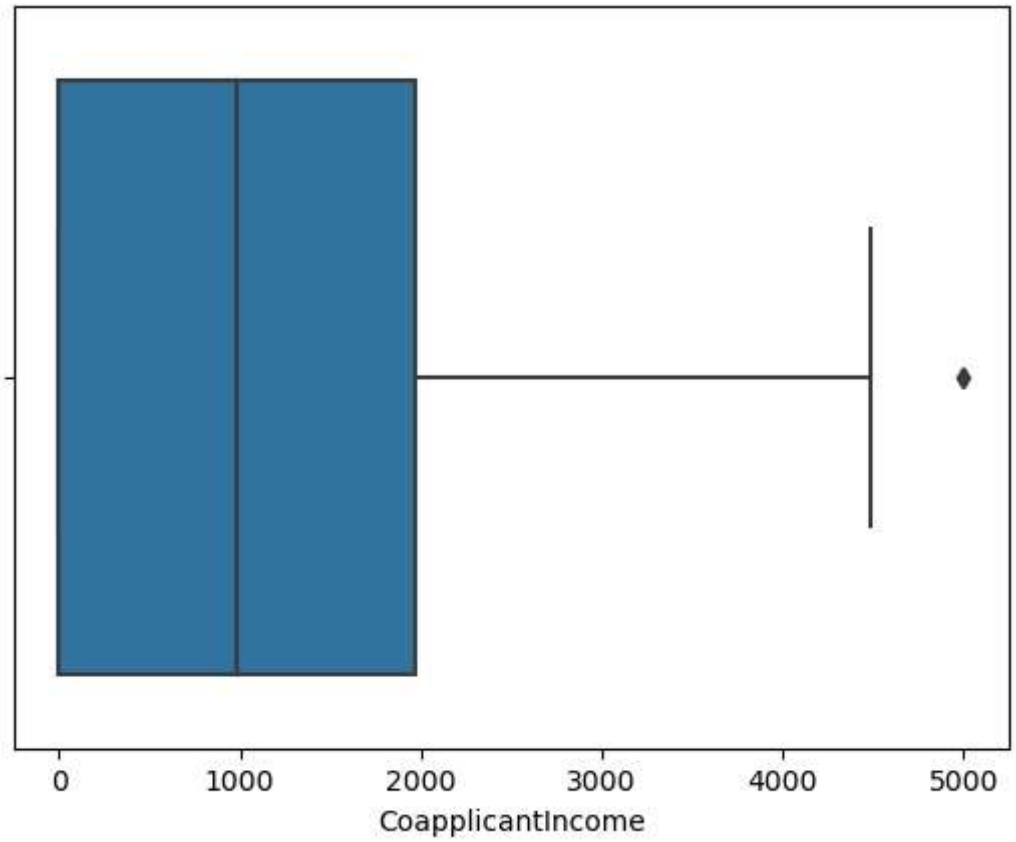
	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Loan_Sta
0	4583	1508.0	128.0	360.0	1.0	
1	3000	0.0	66.0	360.0	1.0	
2	2583	2358.0	120.0	360.0	1.0	
3	6000	0.0	141.0	360.0	1.0	
4	2333	1516.0	95.0	360.0	1.0	
...	...	...	...	...	...	
376	5703	0.0	128.0	360.0	1.0	
377	3232	1950.0	108.0	360.0	1.0	
378	2900	0.0	71.0	360.0	1.0	
379	4106	0.0	40.0	180.0	1.0	
380	4583	0.0	133.0	360.0	0.0	

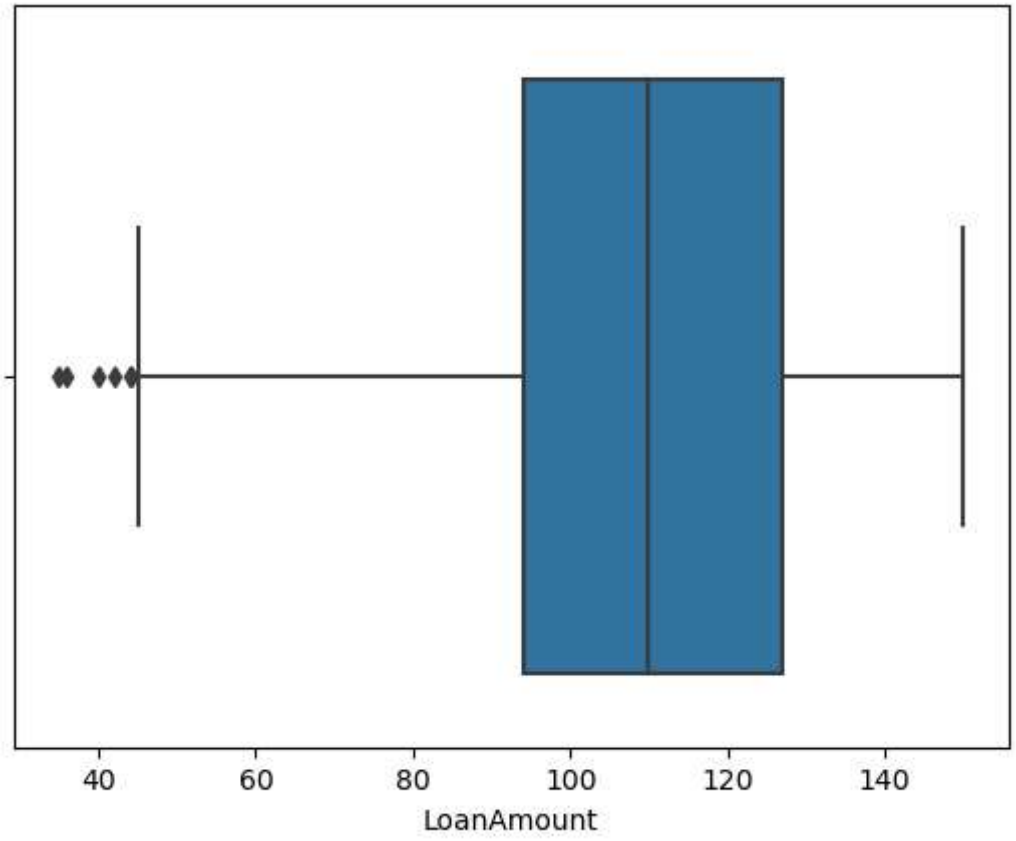
381 rows × 6 columns

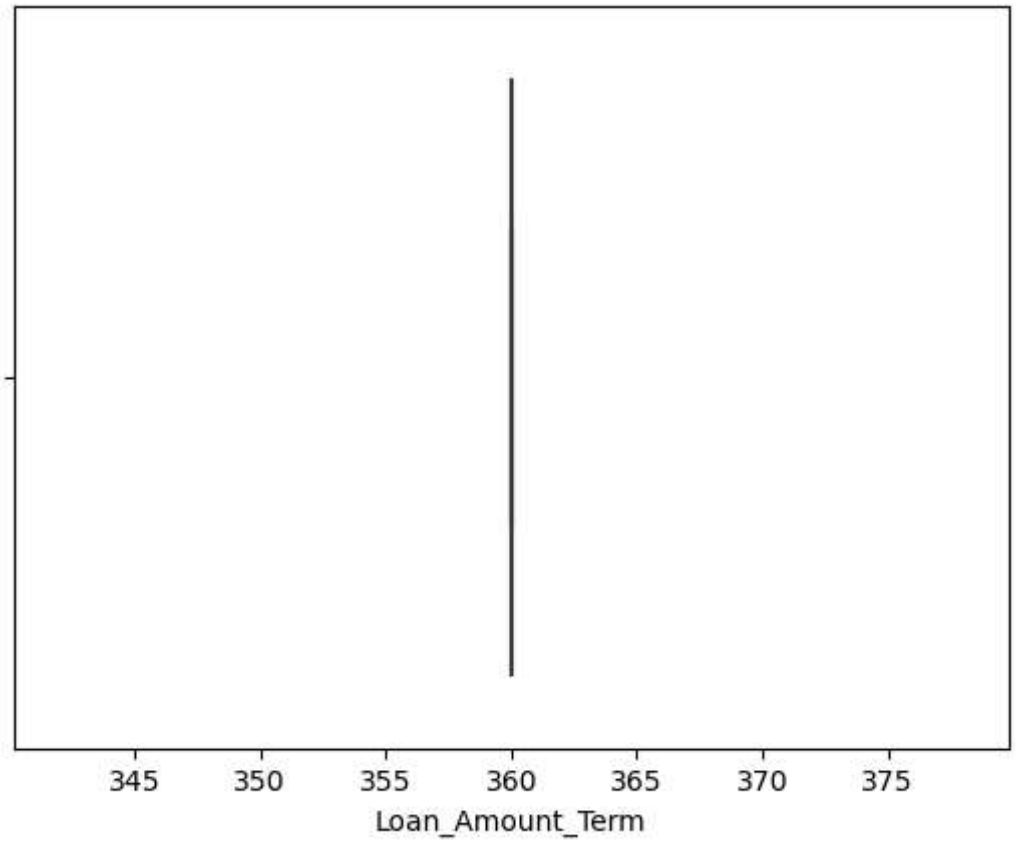
In [37]:
```
for i in data.select_dtypes(['int','float']):
    outlier(i)
```

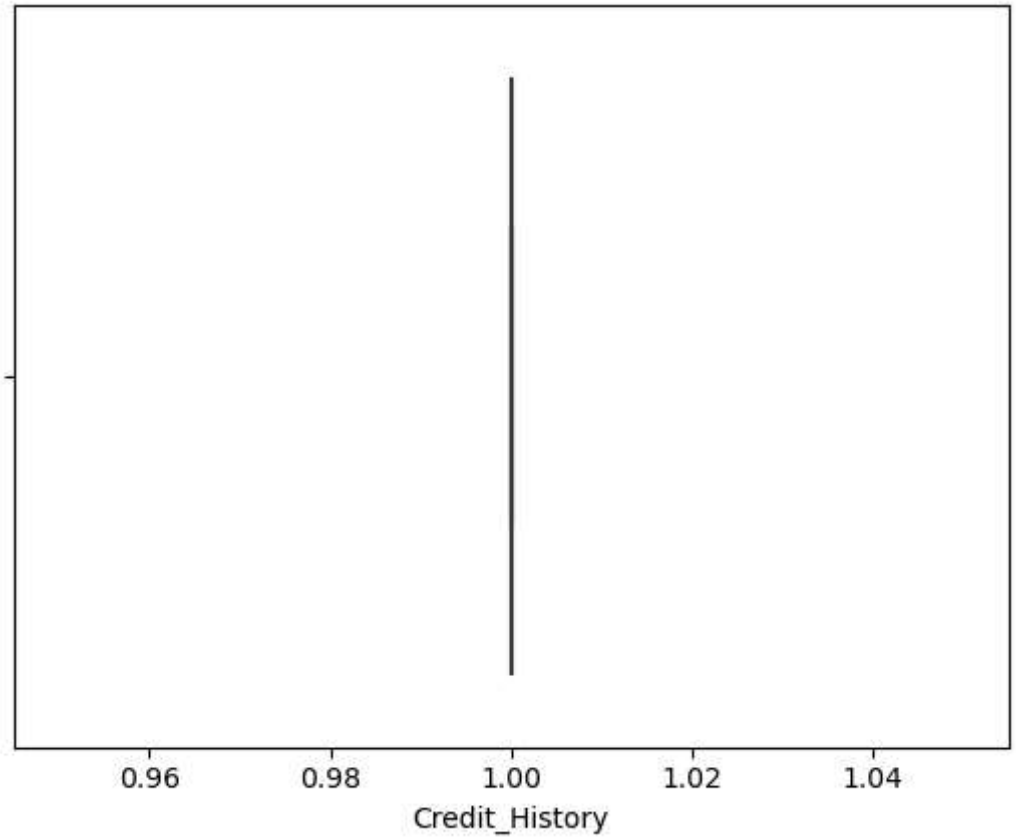
In [38]:
```
for i in data.select_dtypes(['int','float']):
    sns.boxplot(data=data,x=i)
    plt.show()
```

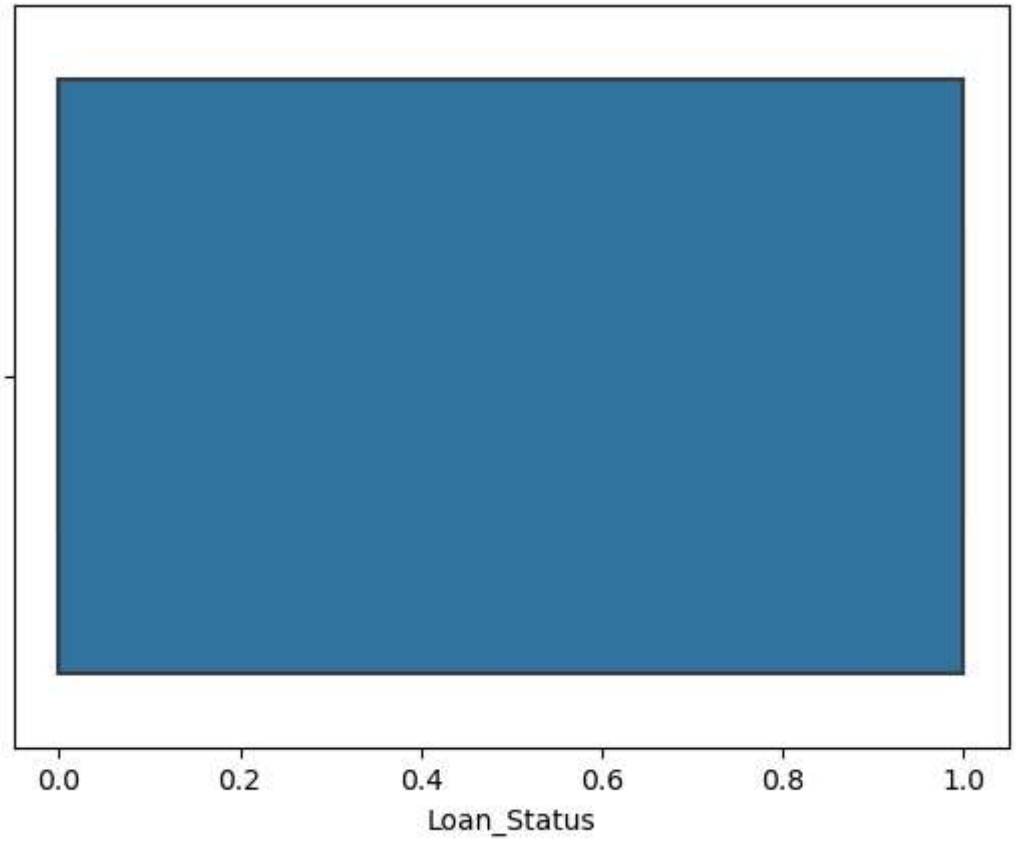

0
1000
2000
3000
4000
5000
6000
7000
ApplicantIncome


0
1000
2000
3000
4000
5000
CoapplicantIncome


40
60
80
100
120
140
LoanAmount


345
350
355
360
365
370
375
Loan_Amount_Term





In [ ]:

Providing the numerical labels to the category using Label Encoder

In [39]:
```
data.head()
```

Out[39]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Lo
0	Male	Yes	1	Graduate	No	4583	1508.0	
1	Male	Yes	0	Graduate	Yes	3000	0.0	
2	Male	Yes	0	Not Graduate	No	2583	2358.0	
3	Male	No	0	Graduate	No	6000	0.0	
4	Male	Yes	0	Not Graduate	No	2333	1516.0	

In [40]:
```
#0 - 0
#1 - 1
#2 - 2
#3+ - 3

data['Dependents'] = LE.fit_transform(data['Dependents'].astype(str))
```

In [41]:
```
data['Gender'] = LE.fit_transform(data['Gender'].astype(str))
```

In [42]:
```
data['Married'] = LE.fit_transform(data['Married'].astype(str))
```

In [43]:
```
# Graduated - 0
# Not Graduated - 1

data['Education'] = LE.fit_transform(data['Education'].astype(str))
```

In [44]:
```
data['Self_Employed'] = LE.fit_transform(data['Self_Employed'].astype(str))
```

In [45]:
```
# Rural - 0
# Urban - 2
# Semiurban - 1

data['Property_Area'] = LE.fit_transform(data['Property_Area'].astype(str))
```

In [46]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 381 entries, 0 to 380
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Gender             381 non-null    int32
 1   Married            381 non-null    int32
 2   Dependents         381 non-null    int32
 3   Education          381 non-null    int32
 4   Self_Employed      381 non-null    int32
 5   ApplicantIncome    381 non-null    int64
 6   CoapplicantIncome  381 non-null    float64
 7   LoanAmount         381 non-null    float64
 8   Loan_Amount_Term   381 non-null    float64
 9   Credit_History     381 non-null    float64
 10  Property_Area      381 non-null    int32
 11  Loan_Status        381 non-null    int32
dtypes: float64(4), int32(7), int64(1)
memory usage: 25.4 KB
```

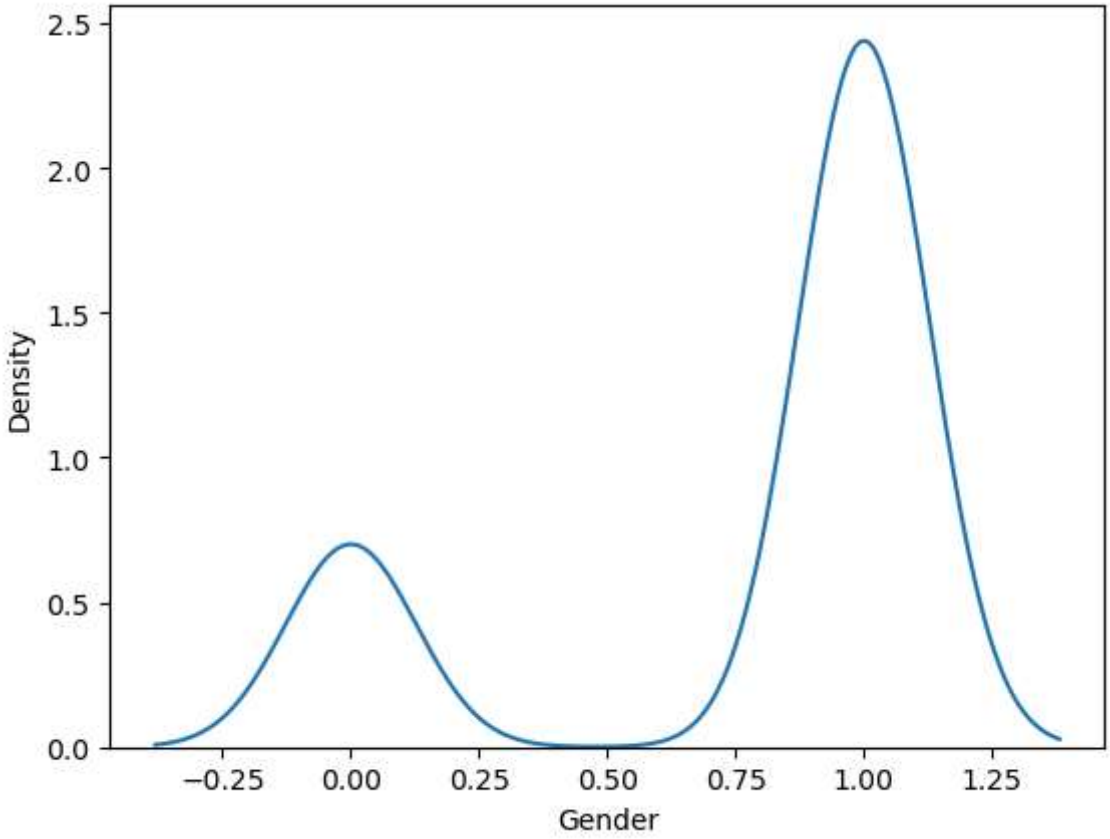
In [ ]:

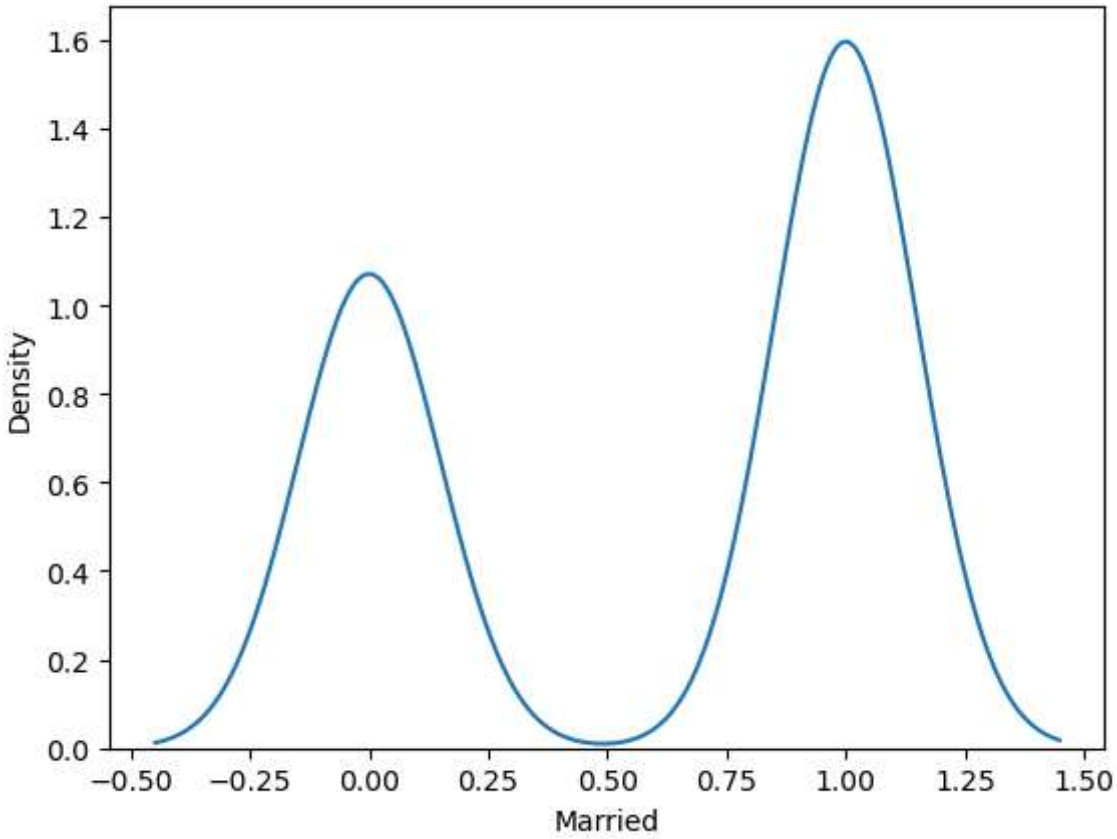
Checking the Skewness

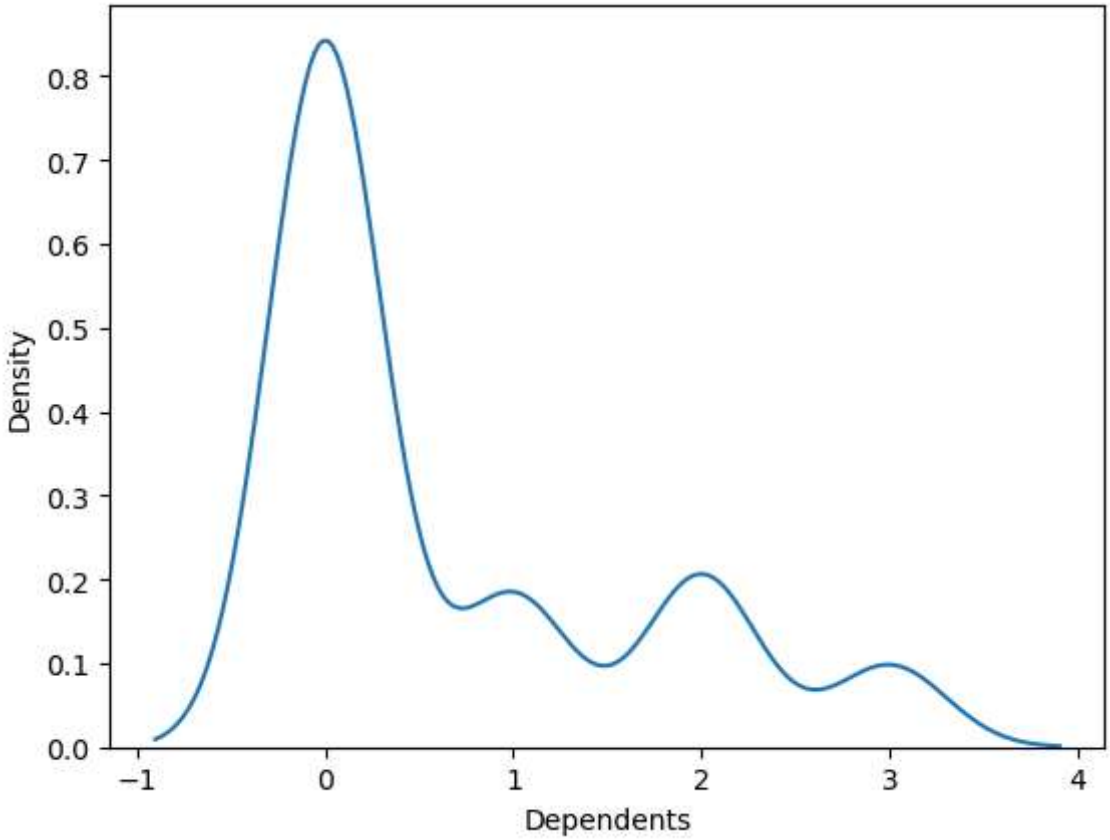
In [47]:
```
data.skew()
```

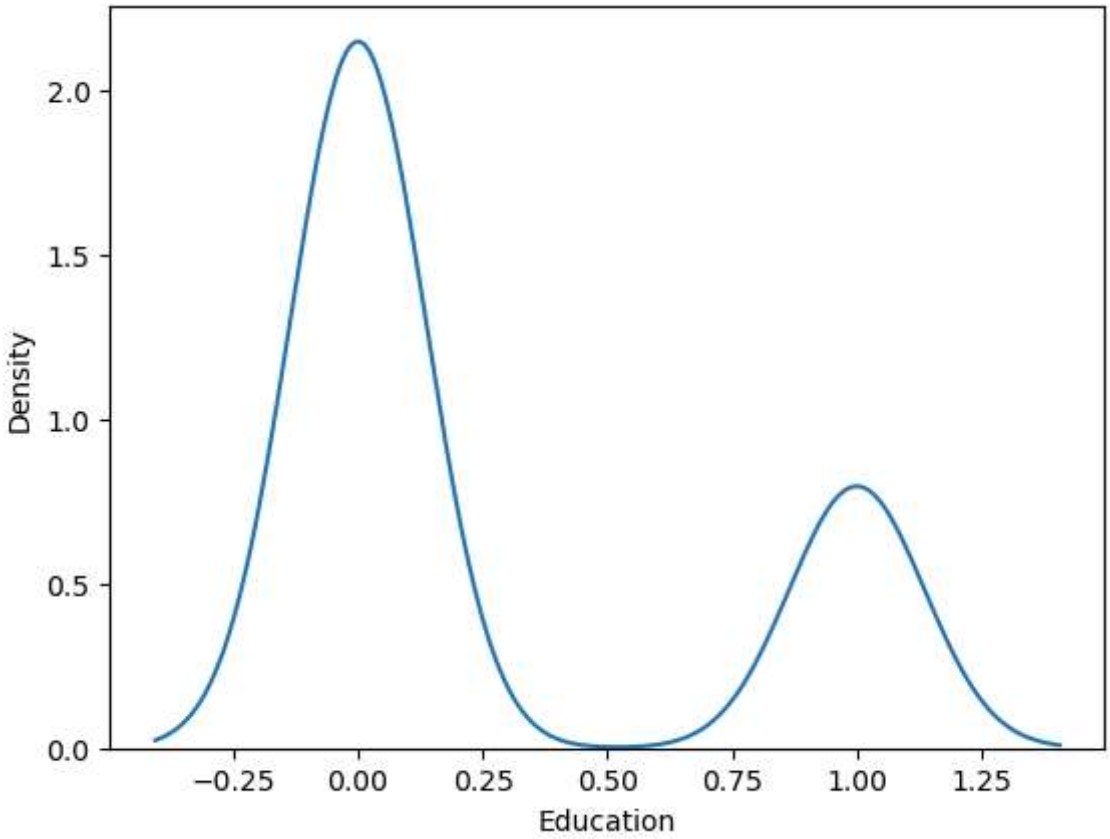
Out[47]:
```
Gender              -1.335495
Married             -0.403147
Dependents           1.169190
Education            1.038274
Self_Employed        2.837288
ApplicantIncome      0.475039
CoapplicantIncome    0.773063
LoanAmount          -0.635957
Loan_Amount_Term     0.000000
Credit_History       0.000000
Property_Area       -0.091709
Loan_Status         -0.936181
dtype: float64
```

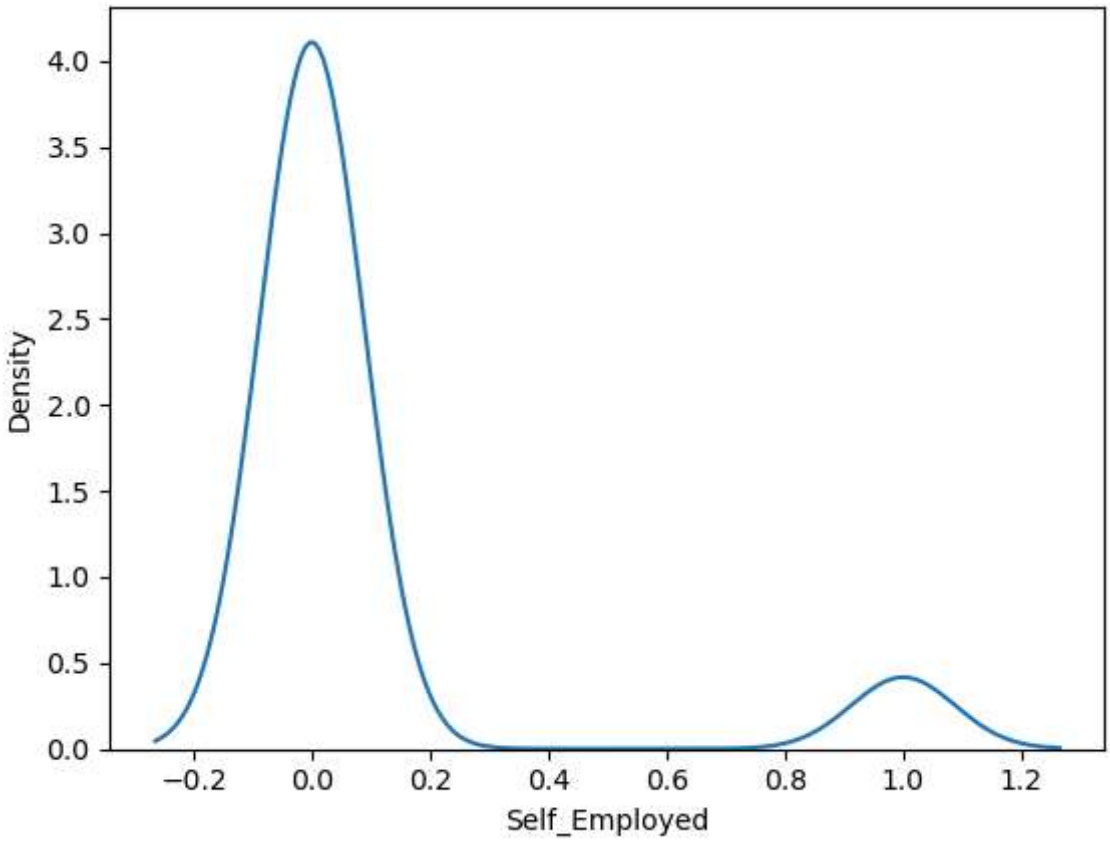
In [48]:
```
for i in data:
    sns.kdeplot(data=data,x=i)
    plt.show()
```

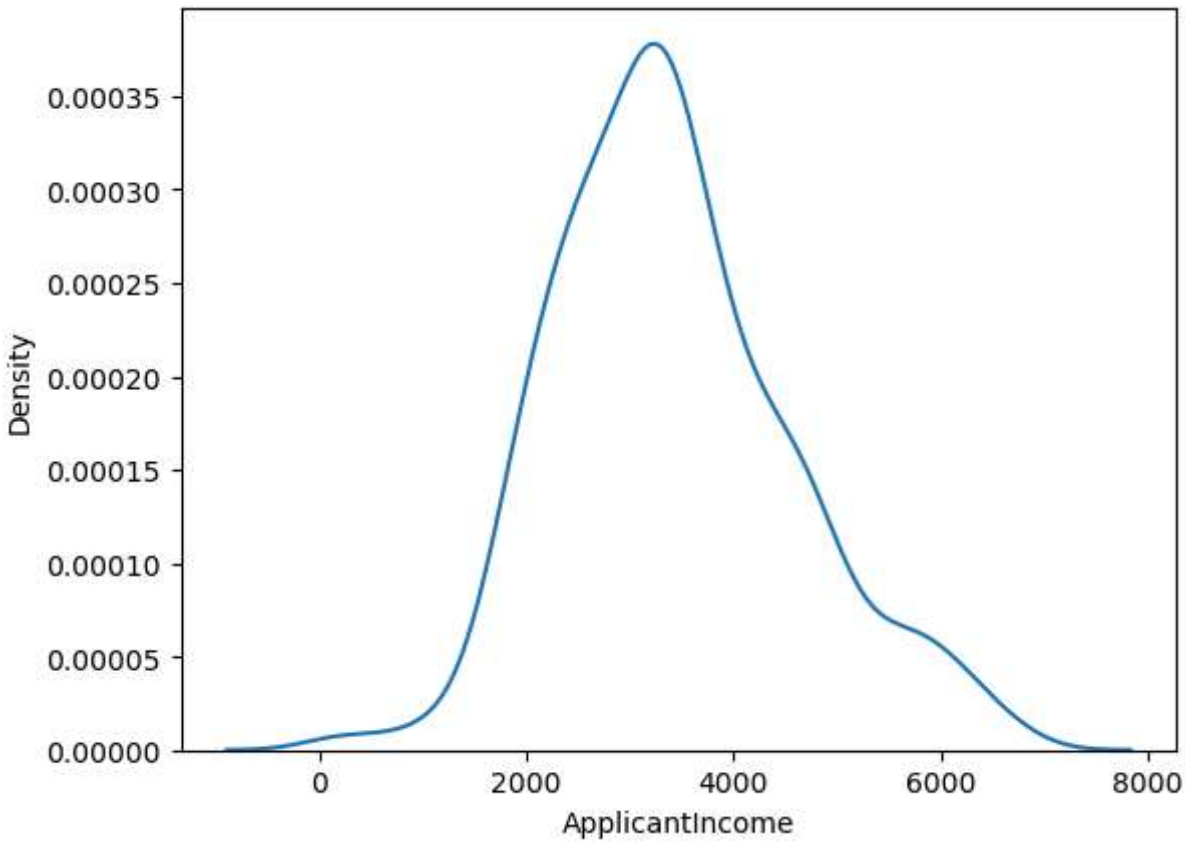

2.5
2.0
1.5
1.0
0.5
0.0
Density
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
Gender

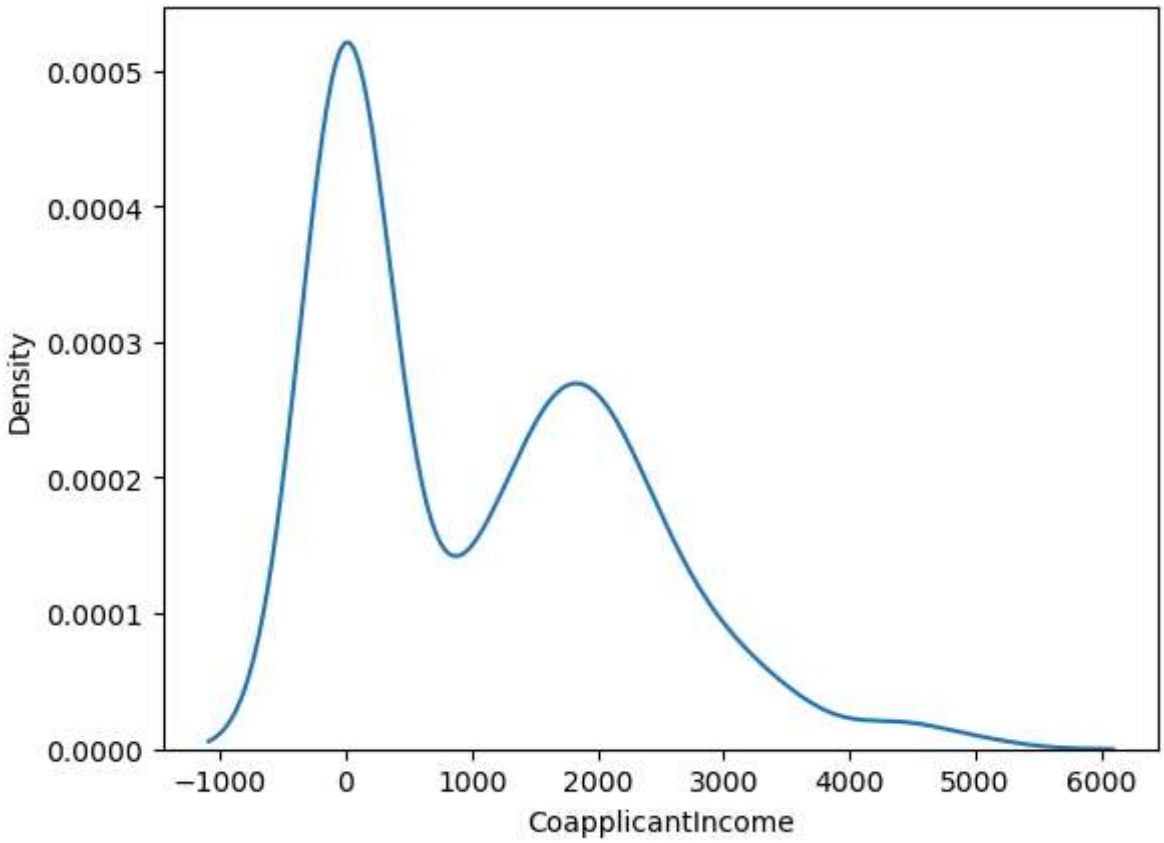

1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Density
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50
Married

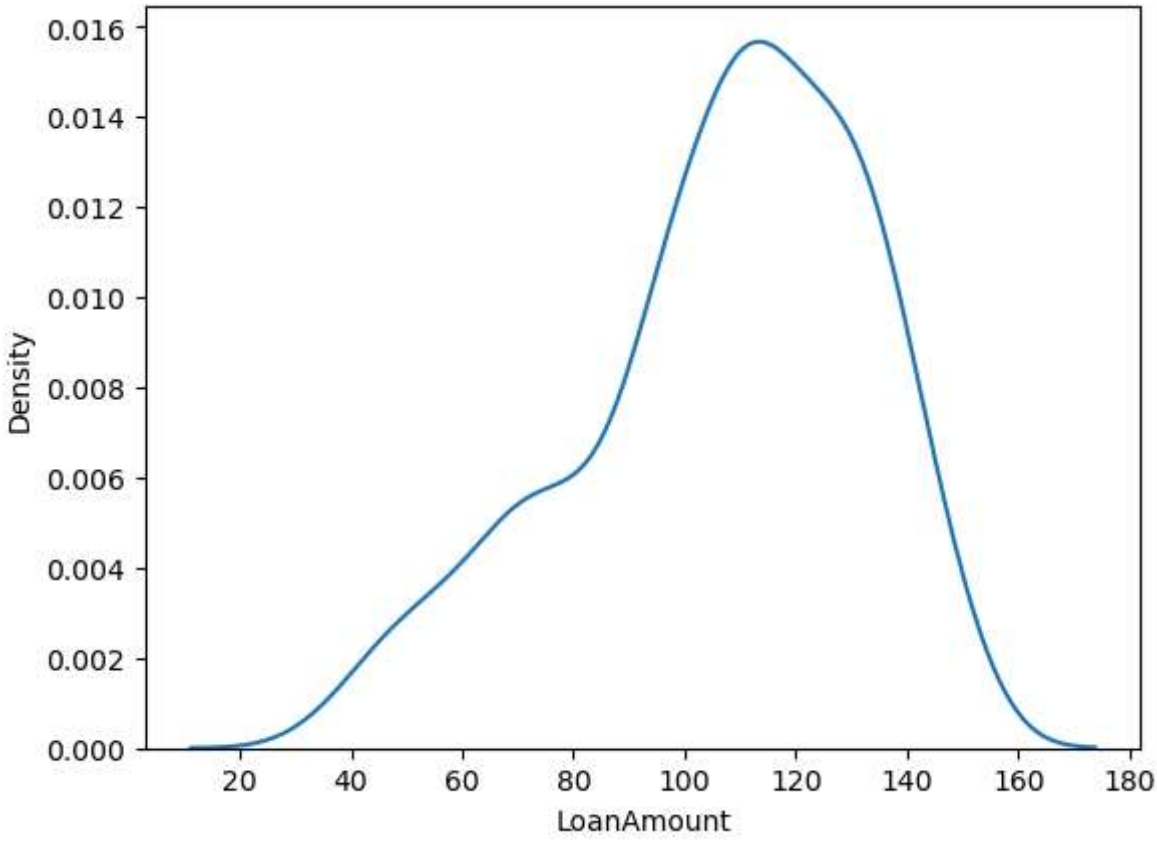

Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
−1
0
1
2
3
4
Dependents


Density
0.0
0.5
1.0
1.5
2.0
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
Education

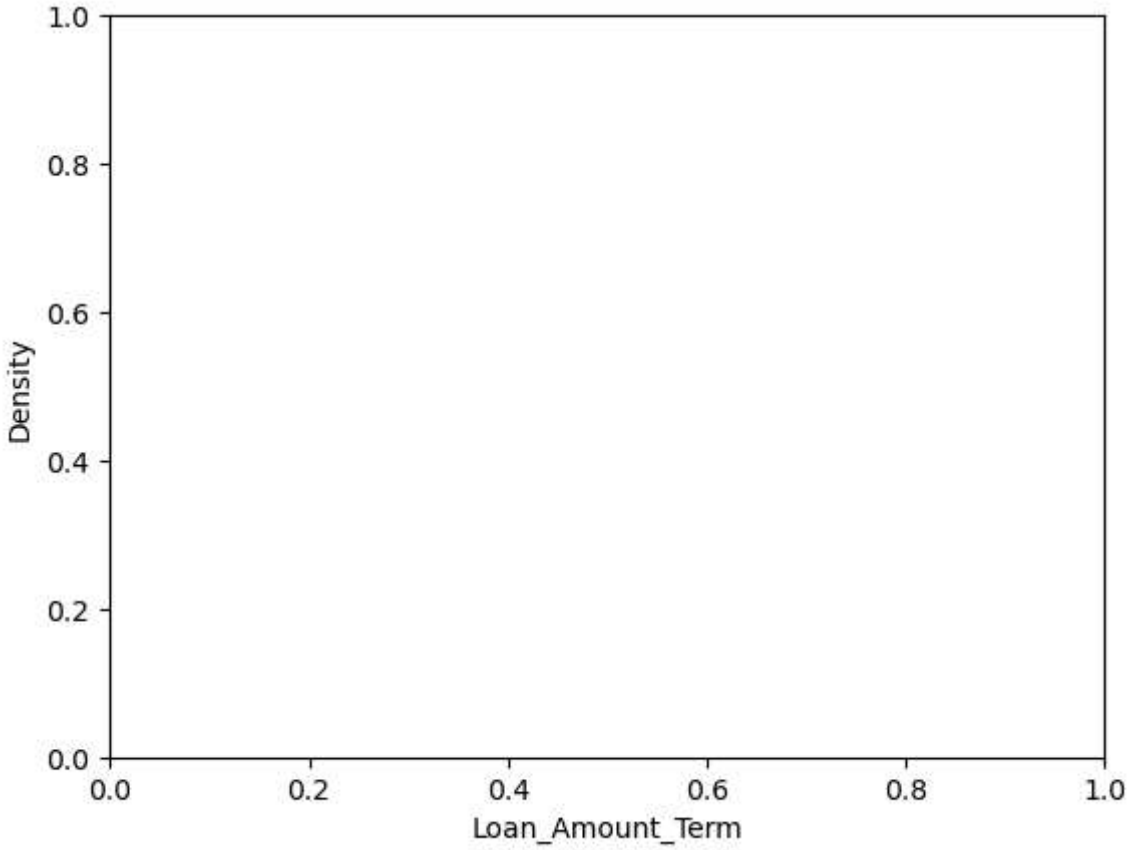

Density
Self_Employed
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2


Density
ApplicantIncome
0.00035
0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
0
2000
4000
6000
8000

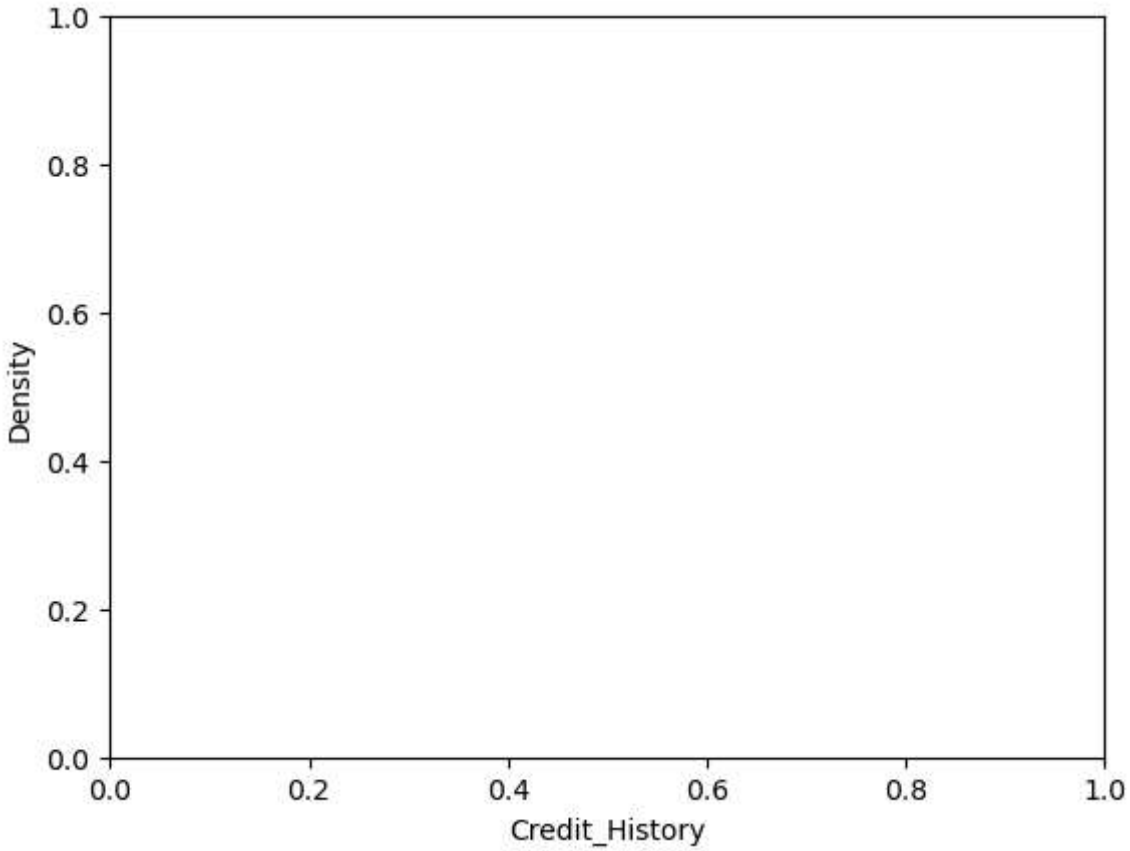


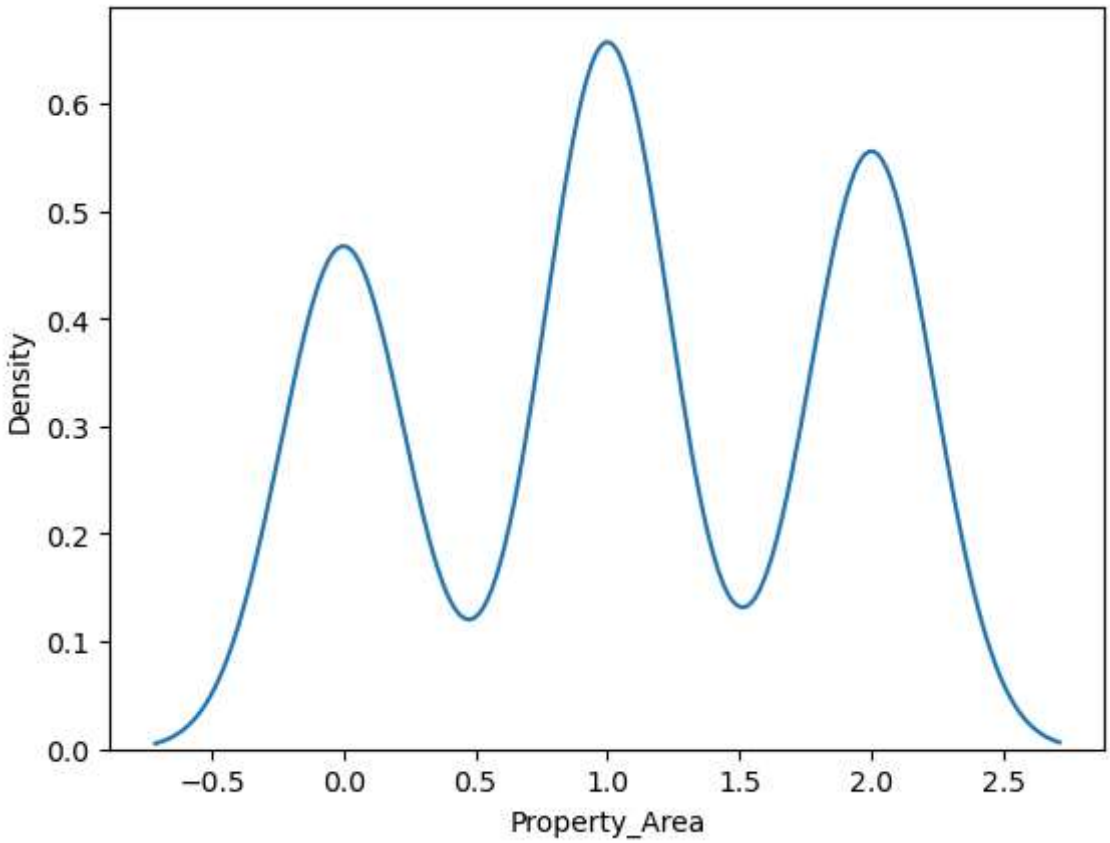


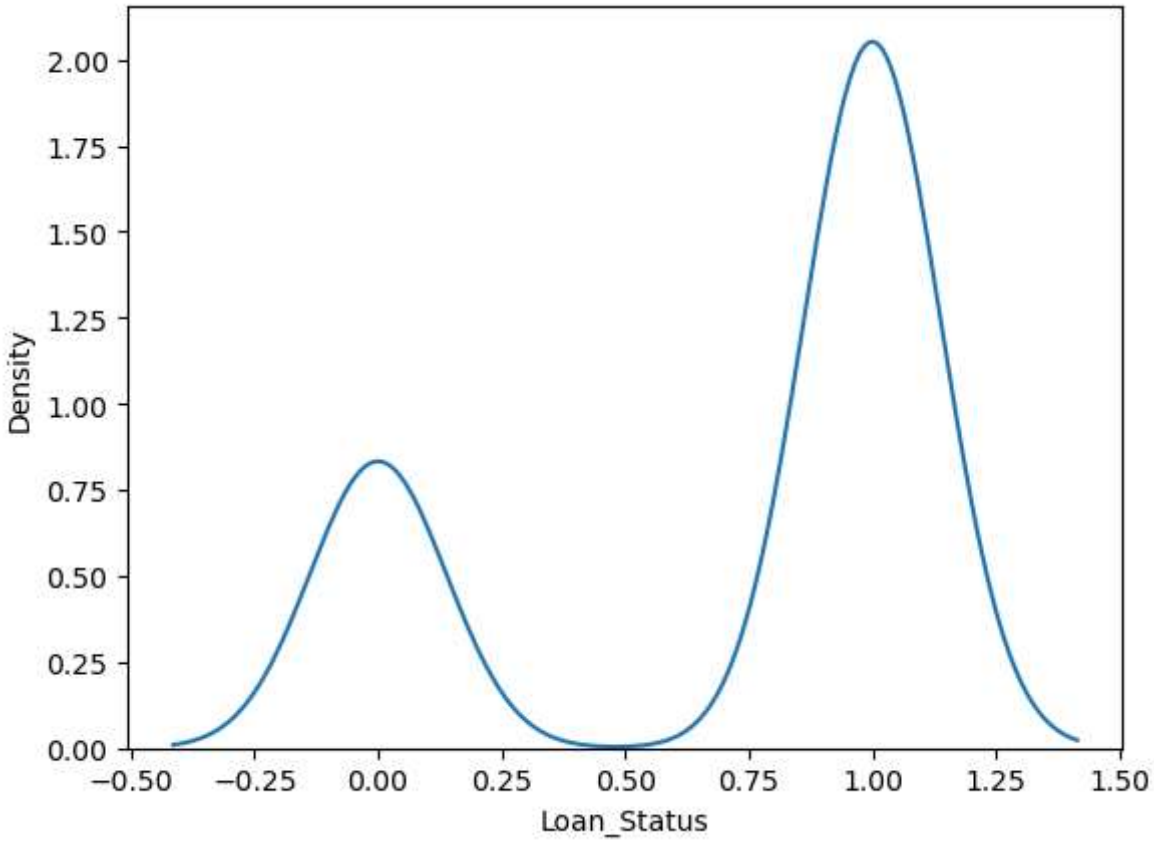

```
C:\Users\ACER\AppData\Local\Temp\ipykernel_3556\2570921382.py:2: UserWarning: Dataset has 0 variance; skipping density estimate. Pass `warn_singular=False` to disable this warning.
  sns.kdeplot(data=data,x=i)
```




```
C:\Users\ACER\AppData\Local\Temp\ipykernel_3556\2570921382.py:2: UserWarning: Dataset has 0 variance; skipping density estimate. Pass `warn_singular=False` to disable this warning.
  sns.kdeplot(data=data,x=i)
```







In [ ]:

Separating the Input Feature & Target Variable

In [49]:
```
data.columns
```

Out[49]:
```
Index(['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
       'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [50]:
```
X= data[['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
       'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area']]

y = data.Loan_Status
```

In [ ]:

Splitting the data into Training data & Testing Data

In [51]:
```
x_train,x_test,y_train,y_test= train_test_split(X,y,test_size=0.25,random_state=0)
```

In [ ]:

Importing the Model

In [52]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
```

In [53]:
```
LR=LogisticRegression()
```

In [ ]:

Training the Model

In [54]:
```
LR.fit(x_train,y_train)
```

Out[54]:
```
LogisticRegression()
```

In [ ]:

Testing the Model

In [55]:
```
y_pred= LR.predict(x_test)
```

In [ ]:

Checking the Accuracy of the Model

In [56]:
```
accuracy_score(y_test,y_pred)
```

Out[56]:
```
0.6979166666666666
```

In [ ]:

Imporing another Model

In [57]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [ ]:

Training the Model

In [79]:
```
DTC = DecisionTreeClassifier(criterion='gini',random_state=0,max_depth=2)
```

In [80]:
```
DTC.fit(x_train,y_train)
```

Out[80]:
```
DecisionTreeClassifier(max_depth=2, random_state=0)
```

In [ ]:

Testing the Model

In [81]:
```
y_pred = DTC.predict(x_test)
```

In [ ]:

Checking the Accuracy of the Model

In [82]:
```
accuracy_score(y_test,y_pred)
```

Out[82]:
```
0.7083333333333334
```

In [ ]:

Importing the Confusion Matrix to check in how many values the model gets confused

In [83]:
```
from sklearn.metrics import confusion_matrix
```

In [84]:
```
confusion_matrix(y_test,y_pred)
```

Out[84]:
```
array([[ 0, 28],
       [ 0, 68]], dtype=int64)
```

In [ ]: