In [38]:
```
#insatll all the required libraries
#pip install keras
```

In [39]:
```
#imporing pandas  Library here
import pandas as pd
```

In [40]:
```
#imporing the data set Financial_debt.csv
try:

    df = pd.read_csv("Financial_debt.csv")
    df_visualize = pd.read_csv("Financial_debt.csv")
except FileNotFoundError:
    print("File not found")
```

Task 2 Data Exploration with Python

In [41]:
```
#displaying my  datasets top  5 records
df.head()
```

Out[41]:

	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 [42]:
```
# Summary statistics of the dataset using describe()
df.describe()
```

Out[42]:

	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 [43]:
```
# Displaying datatype of columns and not null values in dataset
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: 43.2+ KB
```

In [44]:

```
#finding number of missing values
df.isnull().sum()
```

Out[44]:

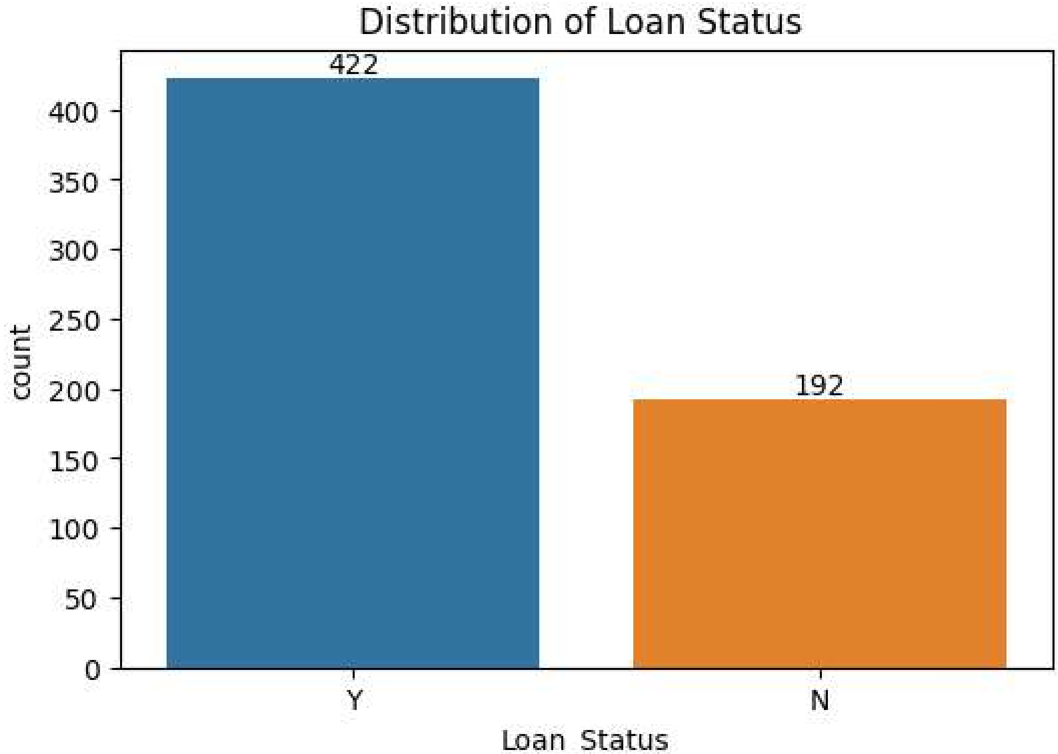
```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

In [45]:

```
#import piplite
#await piplite.install("scikit-learn")
```

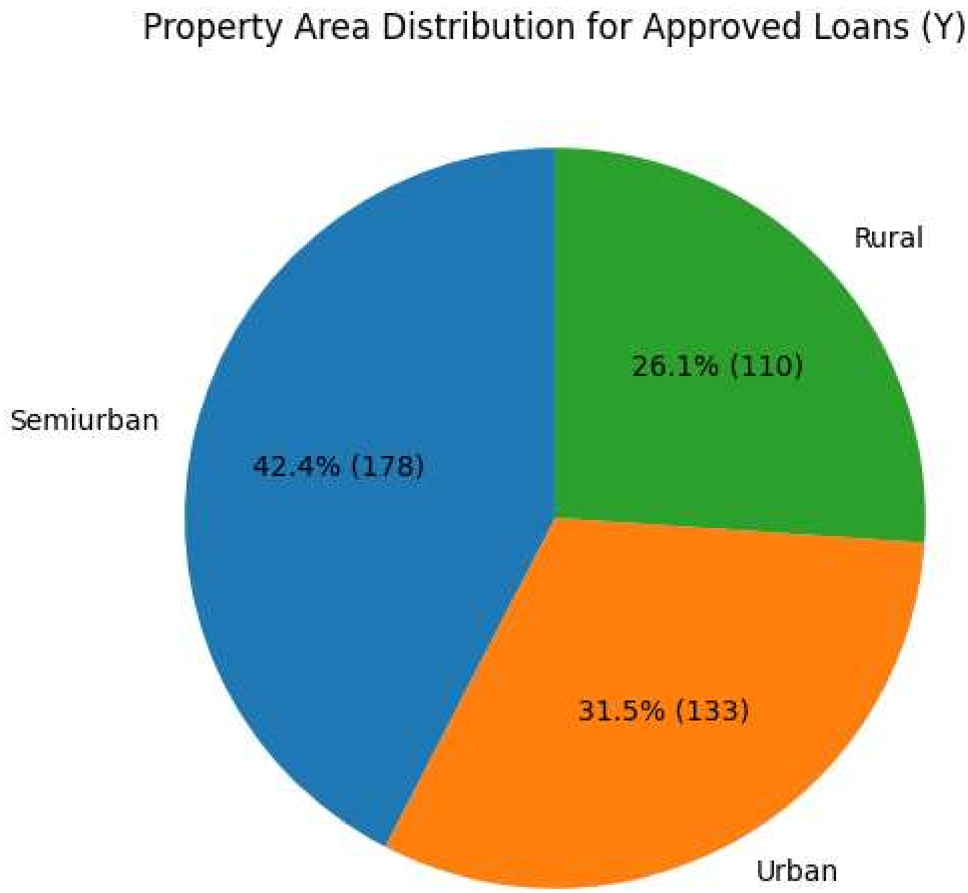
In [46]:

```
#visualizong loan status
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(6, 4))
loan_status = sns.countplot(x="Loan_Status", data=df)

approved_status = df[df['Loan_Status'] == 'Y'].shape[0]
not_approved_status = df[df['Loan_Status'] == 'N'].shape[0]

# Annotate the approved and not-approved counts on top of each bar
loan_status.annotate(f'{approved_status}', xy=(0,approved_status), ha='center', va='bo
loan_status.annotate(f'{not_approved_status}', xy=(1,not_approved_status), ha='center'

plt.title("Distribution of Loan Status ")
plt.show()
```

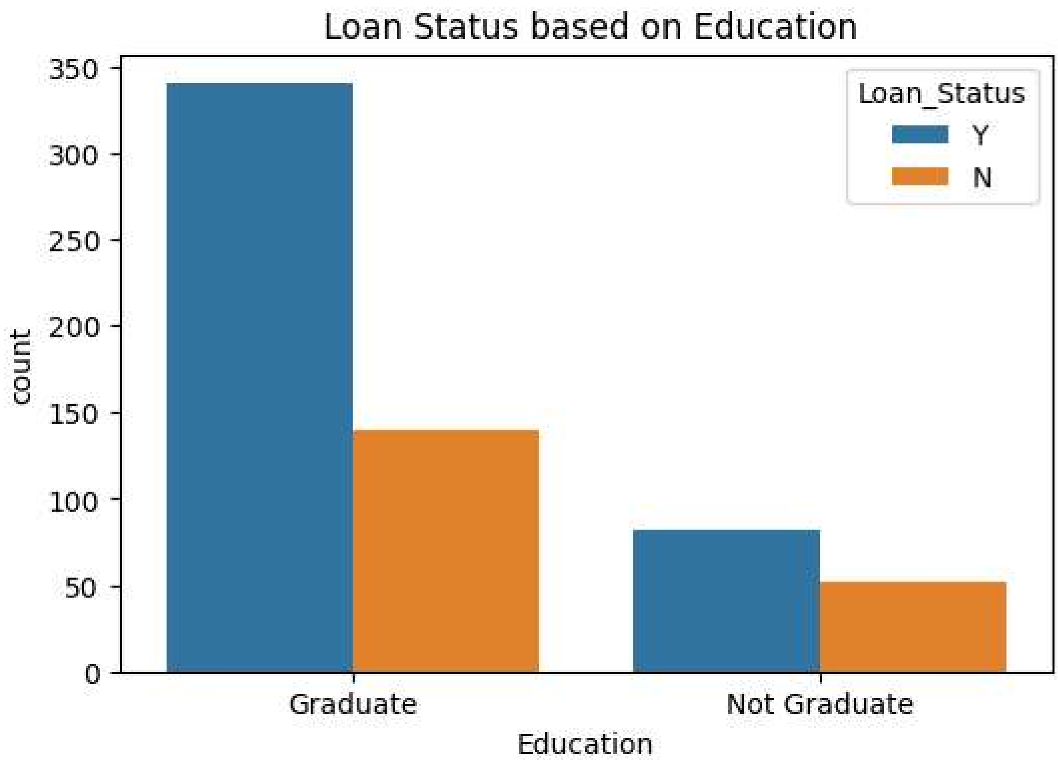


In [47]:

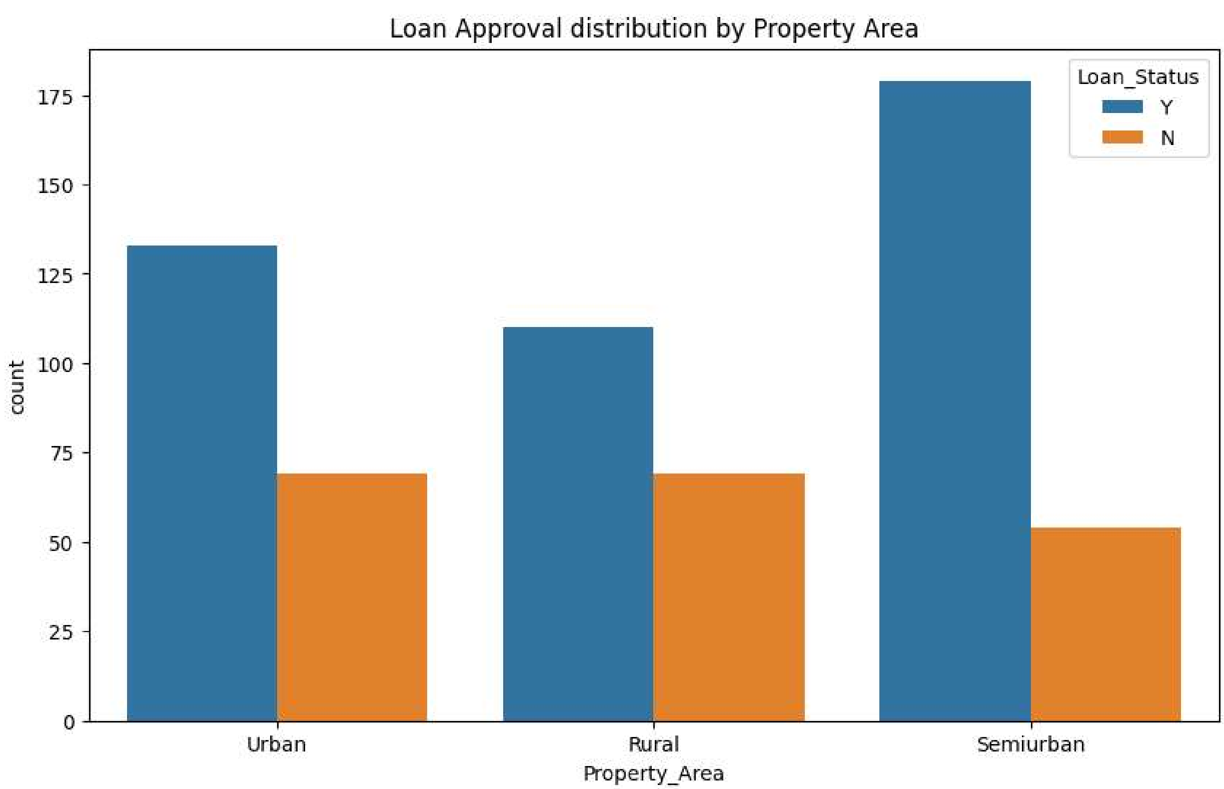
```
#Distribution of financial debt approval by property
import matplotlib.pyplot as plt
import seaborn as sns

approved_status_df = df[df['Loan_Status'] == 'Y']

# Calculate the count
total_property_area= approved_status_df['Property_Area'].value_counts()
total_approved_loans = approved_status_df.shape[0]
percentages = (total_property_area / total_approved_loans) * 100

#pie chart plot
plt.figure(figsize=(6, 6))
plt.pie(total_property_area, labels=total_property_area.index, autopct=lambda p: f'{p:
plt.title("Property Area Distribution for Approved Loans (Y)")

plt.show()
```



In [48]:
```
# Loan status based on graduation status (Graduate or Not graduate)
plt.figure(figsize=(6, 4))
sns.countplot(x="Education", hue="Loan_Status", data=df)
plt.title(" Loan Status based on Education")
plt.show()
```

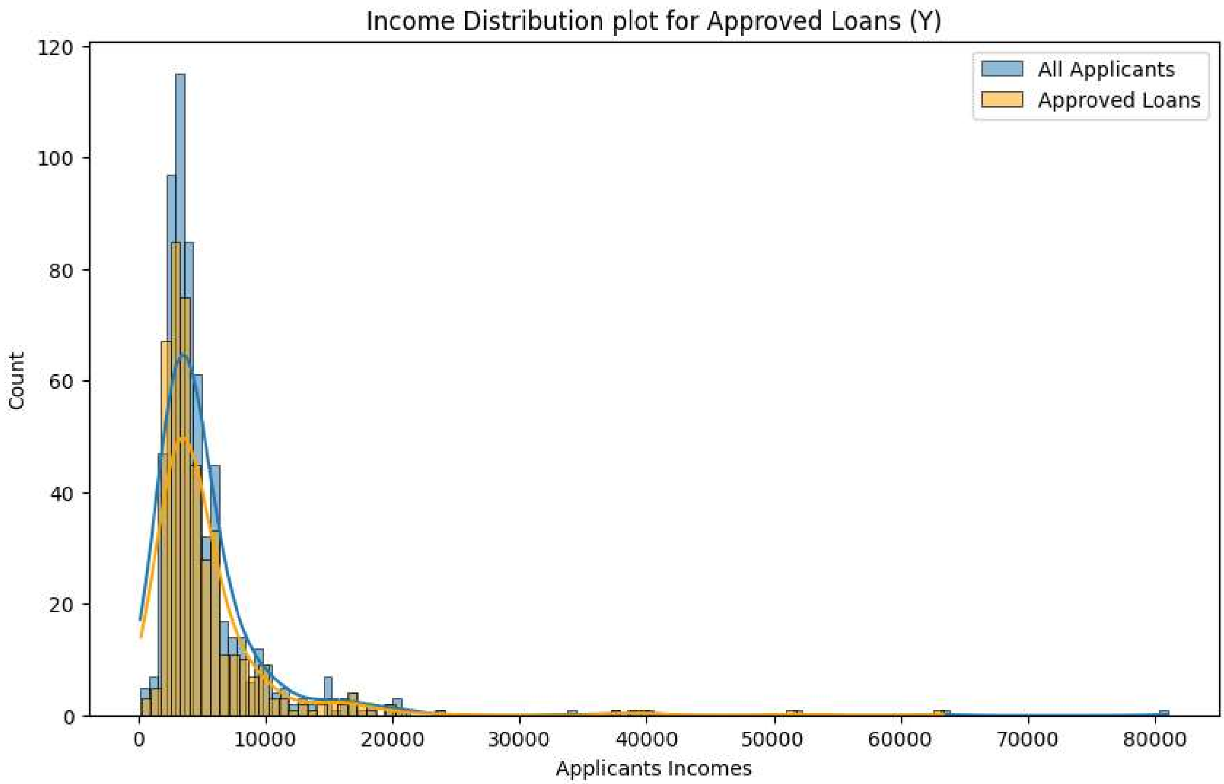


In [49]:
```
#Loan Approval distribution by Property Area(Urban , rural , semi urban)
plt.figure(figsize=(10, 6))
sns.countplot(x="Property_Area", hue="Loan_Status", data=df)
plt.title("Loan Approval distribution by Property Area")
plt.show()
```



In [50]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# include only rows with 'Loan_Status' == 'Y'
approved_status_df = df[df['Loan_Status'] == 'Y']

plt.figure(figsize=(10, 6))
ts= plt.gca()

sns.histplot(df['ApplicantIncome'], kde=True, ax=ts, label='All Applicants')

sns.histplot(approved_status_df['ApplicantIncome'], kde=True, ax=ts, color='orange', l

plt.xlabel('Applicants Incomes')
plt.ylabel('Count')
plt.title('Income Distribution plot for Approved Loans (Y)')

plt.legend()

plt.show()
```



In [51]:
```
#Preprocessing the data
#data conversion
df['Dependents'] = pd.to_numeric(df['Dependents'], errors='coerce')

# Converting  'Loan_Amount_Term' and 'LoanAmount' named columns from the dataset to nu
df['Loan_Amount_Term'] = pd.to_numeric(df['Loan_Amount_Term'], errors='coerce')
df['LoanAmount'] = pd.to_numeric(df['LoanAmount'], errors='coerce')


#  Handling of missing values
# Here, we are filling missing values in 'Loan_Amount_Term' and 'LoanAmount' with thei
loan_amount_mean = df['LoanAmount'].mean()
loan_term_mean = df['Loan_Amount_Term'].mean()

df['LoanAmount'].fillna(loan_amount_mean, inplace=True)
df['Loan_Amount_Term'].fillna(loan_term_mean, inplace=True)
```

```
# Here we are using one-hot encoding to Convert categorical variables to numerical
df = pd.get_dummies(df, columns=['Gender', 'Married', 'Education', 'Self_Employed', 'P

# Here we are converting 'Loan_Status' to binary values
df['Loan_Status'] = df['Loan_Status'].map({'Y': 1, 'N': 0})
```

In [52]:

```
df.isnull().sum()
```

Out[52]:

```
Loan_ID                     0
Dependents                 66
ApplicantIncome             0
CoapplicantIncome           0
LoanAmount                  0
Loan_Amount_Term            0
Credit_History             50
Loan_Status                 0
Gender_Male                 0
Married_Yes                 0
Education_Not Graduate      0
Self_Employed_Yes           0
Property_Area_Semiurban     0
Property_Area_Urban         0
dtype: int64
```

In [53]:

```
# displaying values
df.head()
```

Out[53]:

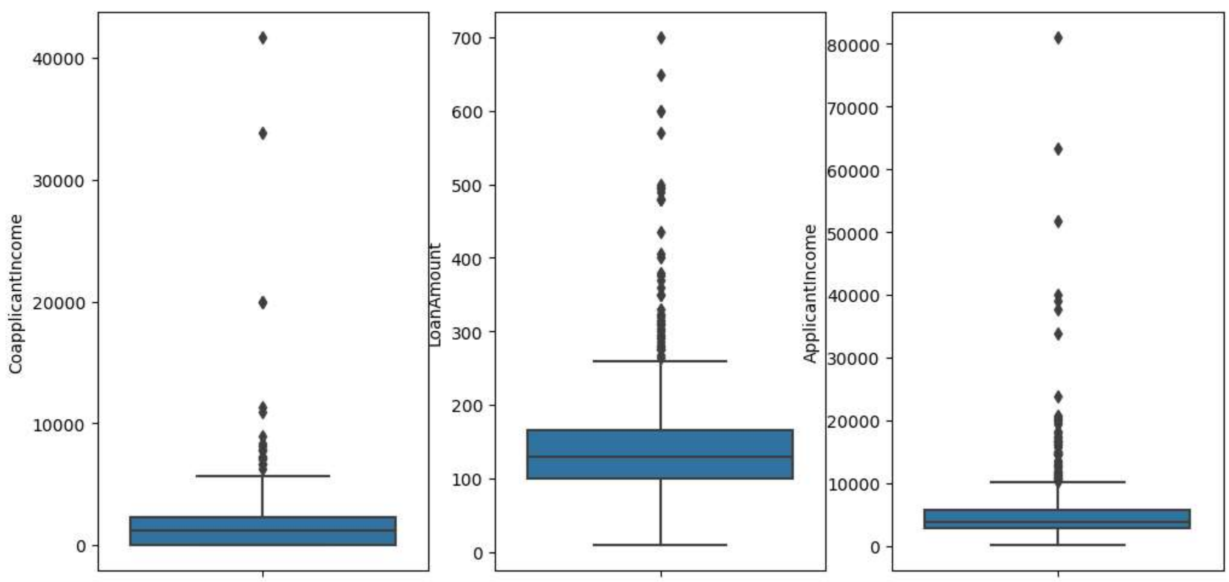
	Loan_ID	Dependents	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Cr
0	LP001002	0.0	5849	0.0	146.412162	360.0	
1	LP001003	1.0	4583	1508.0	128.000000	360.0	
2	LP001005	0.0	3000	0.0	66.000000	360.0	
3	LP001006	0.0	2583	2358.0	120.000000	360.0	
4	LP001008	0.0	6000	0.0	141.000000	360.0	

In [54]:

```
# we found that column Dependents and Credit_History contains missing values of 66 and
# Now we handle missing values in  'Credit_History' and 'Dependents'columns
# Here, we fill missing categorical values with the mode i.e most frequent value
df['Dependents'].fillna(df['Dependents'].mode()[0], inplace=True)
df['Credit_History'].fillna(df['Credit_History'].mode()[0], inplace=True)

# Print the updated count of missing values in each column
print(df.isnull().sum())
```

```
Loan_ID                    0
Dependents                 0
ApplicantIncome            0
CoapplicantIncome          0
LoanAmount                 0
Loan_Amount_Term           0
Credit_History             0
Loan_Status                0
Gender_Male                0
Married_Yes                0
Education_Not Graduate     0
Self_Employed_Yes          0
Property_Area_Semiurban    0
Property_Area_Urban        0
dtype: int64
```

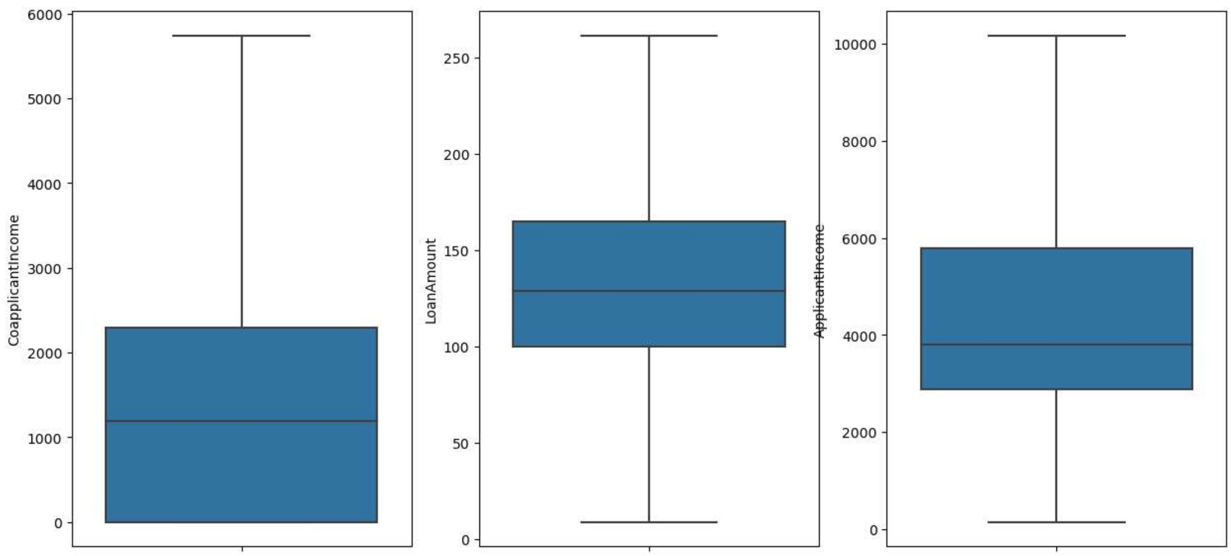
In [55]:
```
#checking for outliers by ploting box plot
'LoanAmount'
'CoapplicantIncome'
plt.figure(figsize=(12, 6))
plt.subplot(1, 3, 1)
sns.boxplot(y='CoapplicantIncome', data=df)
plt.subplot(1, 3, 2)
sns.boxplot(y='LoanAmount', data=df)
plt.subplot(1, 3, 3)
sns.boxplot(y='ApplicantIncome', data=df)
plt.show()
```



In [56]:
```
def outliers_IQR(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    df.loc[df[column] < lower_bound, column] = lower_bound
    df.loc[df[column] > upper_bound, column] = upper_bound

# using Applying IQR method on column 'ApplicantIncome'

outliers_IQR(df, 'CoapplicantIncome')
```

```
outliers_IQR(df, 'LoanAmount')
outliers_IQR(df, 'ApplicantIncome')
```

In [57]:
```
#checking for outliers after processing outliers
#checking for outliers by ploting box plot
plt.figure(figsize=(15, 7))
plt.subplot(1, 3, 1)
sns.boxplot(y='CoapplicantIncome', data=df)
plt.subplot(1, 3, 2)
sns.boxplot(y='LoanAmount', data=df)
plt.subplot(1, 3, 3)
sns.boxplot(y='ApplicantIncome', data=df)
plt.show()
```



In [58]:
```
df.head()
```

Out[58]:

	Loan_ID	Dependents	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Cr
0	LP001002	0.0	5849.0	0.0	146.412162	360.0	
1	LP001003	1.0	4583.0	1508.0	128.000000	360.0	
2	LP001005	0.0	3000.0	0.0	66.000000	360.0	
3	LP001006	0.0	2583.0	2358.0	120.000000	360.0	
4	LP001008	0.0	6000.0	0.0	141.000000	360.0	

Task 4: Implement Machine Learning Algorithms

In [59]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Assuming df is my DataFrame with the dataset
```

```
# Separating the target variable (Loan_Status) from the features
x = df.drop(columns=['Loan_ID', 'Loan_Status','Gender_Male'])
y = df['Loan_Status']

# (80% training, 20% testing) Spliting the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=

# for better model performance standardize the features
scaler = StandardScaler()
x_train_scale = scaler.fit_transform(x_train)
x_test_scale= scaler.transform(x_test)

# Training the logistic regression model
logistic_model = LogisticRegression(random_state=42)
logistic_model.fit(x_train_scale, y_train)

# Making predictions on the test data
y_pred_logistic = logistic_model.predict(x_test_scale)

# Evaluating the Logistic Regression model
accuracy_logistic_ts = accuracy_score(y_test, y_pred_logistic)
confusion_matrix_logistic_ts= confusion_matrix(y_test, y_pred_logistic)
classification_report_logistic_ts= classification_report(y_test, y_pred_logistic)

print("Logistic Regression Model Accuracy:", accuracy_logistic_ts)
print("Confusion Matrix (Logistic Regression):")
print(confusion_matrix_logistic_ts)
print("Classification Report (Logistic Regression):")
print(classification_report_logistic_ts)

# Training the Random Forest Classifier model
random_forest_model_ts = RandomForestClassifier(random_state=42)
random_forest_model_ts.fit(x_train_scale, y_train)

# Making predictions on the test data
y_pred_ts = random_forest_model_ts.predict(x_test_scale)

# Evaluating the Random Forest Classifier model
accuracy_ts = accuracy_score(y_test, y_pred_ts)
confusion_matrix_ts = confusion_matrix(y_test, y_pred_ts)
classification_report_ts = classification_report(y_test, y_pred_ts)

print("Model Accuracy Random Forest Classifier:", accuracy_ts)
print("Confusion Matrix using (Random Forest Classifier):")
print(confusion_matrix_ts)
print("Classification Report (Random Forest Classifier):")
print(classification_report_ts)
```

```
Logistic Regression Model Accuracy: 0.7886178861788617
Confusion Matrix (Logistic Regression):
[[18 25]
 [ 1 79]]
Classification Report (Logistic Regression):
              precision    recall  f1-score   support

           0       0.95      0.42      0.58        43
           1       0.76      0.99      0.86        80

    accuracy                           0.79       123
   macro avg       0.85      0.70      0.72       123
weighted avg       0.83      0.79      0.76       123

Model Accuracy Random Forest Classifier: 0.7723577235772358
Confusion Matrix using (Random Forest Classifier):
[[19 24]
 [ 4 76]]
Classification Report (Random Forest Classifier):
              precision    recall  f1-score   support

           0       0.83      0.44      0.58        43
           1       0.76      0.95      0.84        80

    accuracy                           0.77       123
   macro avg       0.79      0.70      0.71       123
weighted avg       0.78      0.77      0.75       123
```

In [60]:
```
#The accuracy of the Logistic Regression model (0.7886) is slightly higher than that c
#However, the Random Forest model has a higher recall for class 0 (0.44) compared to t
#which means it is better at identifying true negatives. On the other hand, the Logist
#for class 1 (0.99) compared to the Random Forest model (0.95), indicating that it is
```

In [61]:
```
df.head()
```

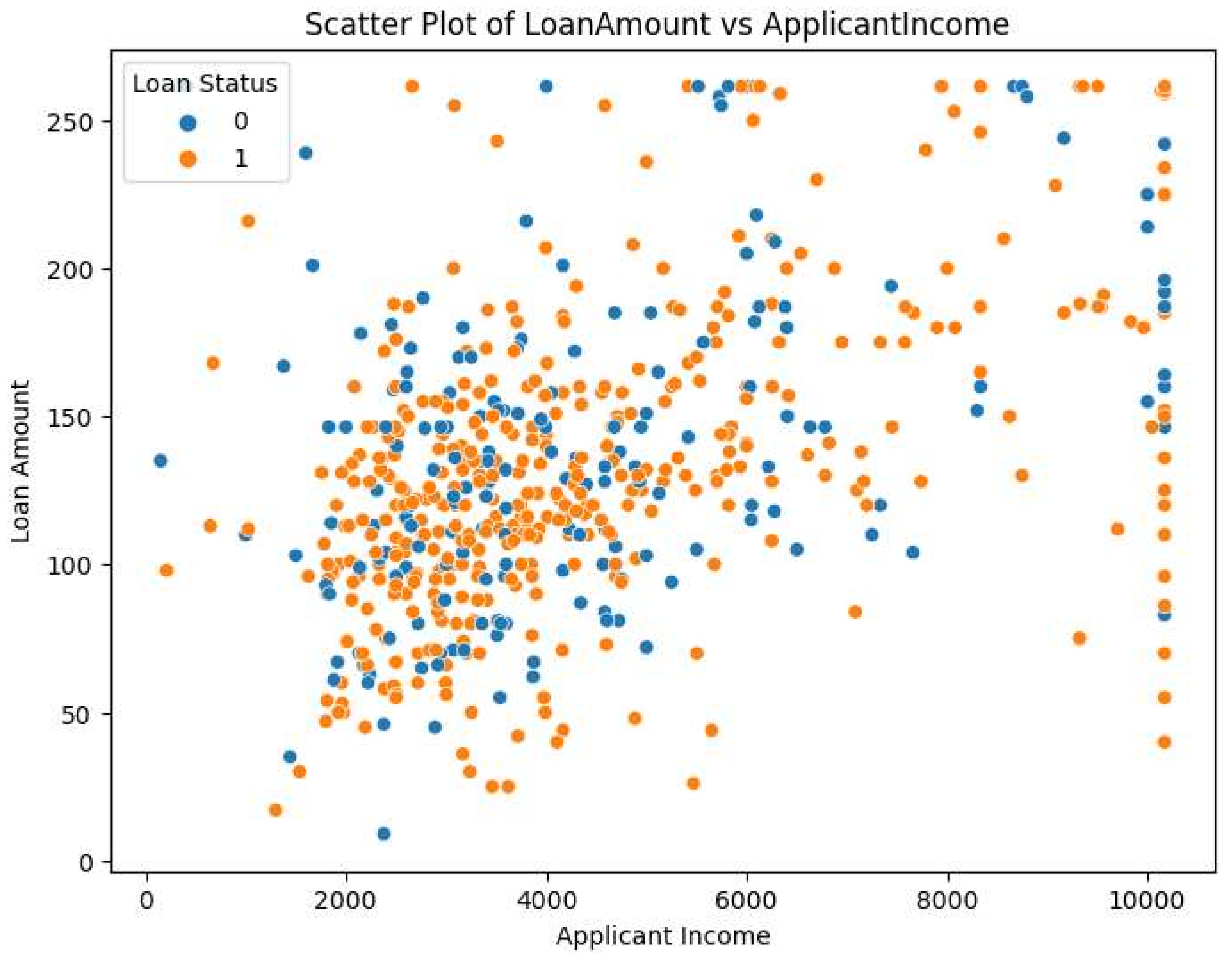
Out[61]:

	Loan_ID	Dependents	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Cr
0	LP001002	0.0	5849.0	0.0	146.412162	360.0	
1	LP001003	1.0	4583.0	1508.0	128.000000	360.0	
2	LP001005	0.0	3000.0	0.0	66.000000	360.0	
3	LP001006	0.0	2583.0	2358.0	120.000000	360.0	
4	LP001008	0.0	6000.0	0.0	141.000000	360.0	

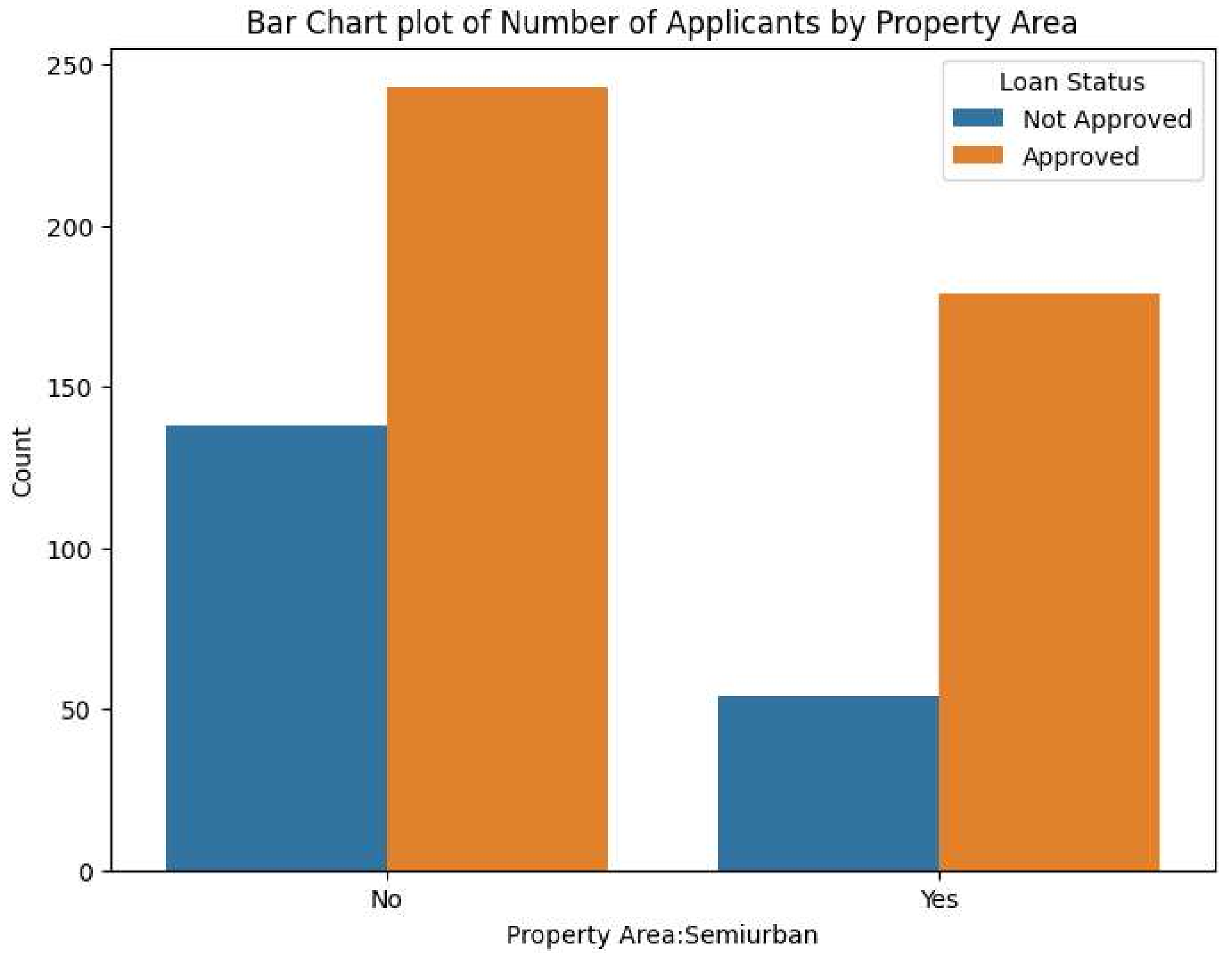
Task 5:

In [62]:
```
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='ApplicantIncome', y='LoanAmount', hue='Loan_Status')
plt.title('Scatter Plot of LoanAmount vs ApplicantIncome ')
plt.xlabel('Applicant Income')
plt.ylabel('Loan Amount')
```

```
plt.legend(title='Loan Status')
plt.show()
```



In [63]:
```
plt.figure(figsize=(8, 6))
sns.countplot(data=df, x='Property_Area_Semiurban', hue='Loan_Status')
plt.title('Bar Chart plot of Number of Applicants by Property Area')
plt.xlabel('Property Area:Semiurban')
plt.ylabel('Count')
plt.legend(title='Loan Status', labels=['Not Approved', 'Approved'])
plt.xticks(ticks=[0, 1], labels=['No', 'Yes'])
plt.show()
```


Bar Chart plot of Number of Applicants by Property Area
Loan Status
Not Approved
Approved
Count
250
200
150
100
50
0
No
Yes
Property Area:Semiurban