

```
In [76]: #rahil Serasiya  
#8899516
```

Task 1.b

Loan Dataset: The loan dataset holds excellent potential for comprehensive analysis and is highly suitable for machine learning applications. The dataset contains diverse and relevant features, such as gender, marital status, education, income details, credit history, and property area, which play critical roles in loan approval and repayment behavior. These attributes offer insight into applicant demographics, financial capacity, and creditworthiness. Leveraging machine learning algorithms on this dataset can enable predictive modeling for loan approval, risk assessment, and identifying factors influencing loan outcomes. Classification tasks can be employed to predict whether loan applications will be approved or denied based on the provided features. Additionally, the dataset's suitability for analysis makes it an ideal resource for automating the loan approval process, improving decision-making, and enhancing efficiency in the lending industry. The wealth of information available in the loan dataset, combined with machine learning techniques, empowers financial institutions to make more informed, fair, and accurate loan decisions while uncovering hidden patterns and trends that might be difficult to identify using traditional analysis methods.

```
In [77]: #import Required Libraries here  
import pandas as pd  
import numpy as np
```

```
In [78]: #importing the data set here  
df = pd.read_csv("C:/Users/Rahil/Downloads/Assignment 10/Loan_Data.csv")  
df_1 = pd.read_csv("C:/Users/Rahil/Downloads/Assignment 10/Loan_Data.csv")
```

Task 2 Data Exploration with Python

```
In [79]: # Display the first few rows to get an overview of the data  
df.head()
```

```
Out[79]:
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
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 [80]: # Summary statistics of dataset  
df.describe()
```

Out[80]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.000000	564.000000
mean	5403.459283	1621.245798	146.412162	342.000000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.000000	0.000000
25%	2877.500000	0.000000	100.000000	360.000000	1.000000
50%	3812.500000	1188.500000	128.000000	360.000000	1.000000
75%	5795.000000	2297.250000	168.000000	360.000000	1.000000
max	81000.000000	41667.000000	700.000000	480.000000	1.000000

In [81]:

```
#check datatypes and missing values
print(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
None
```

In [82]:

```
#finding all the missing values
df.isnull().sum()
```

Out[82]:

```
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
```

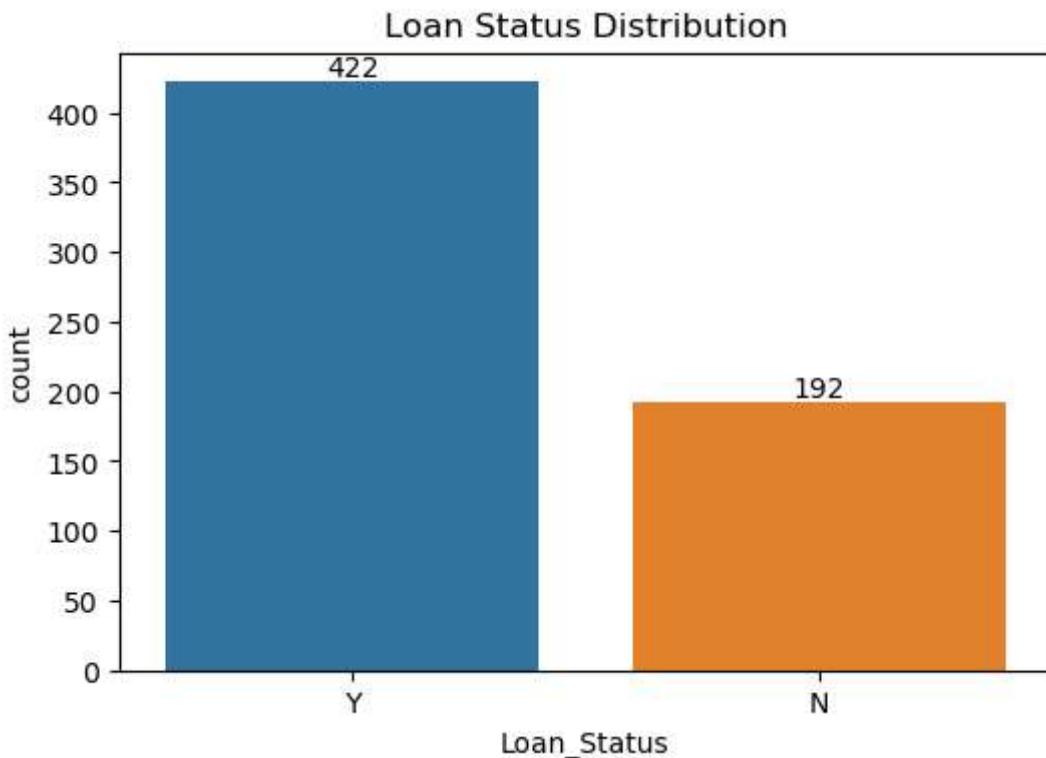
```
In [83]: #identifying Loan status Distribution of approved Y and not approved N
import matplotlib.pyplot as plt
import seaborn as sns

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

approved_count = df[df['Loan_Status'] == 'Y'].shape[0]
not_approved_count = df[df['Loan_Status'] == 'N'].shape[0]

# Annotate the approved and not-approved counts on top of each bar
ax.annotate(f'{approved_count}', xy=(0, approved_count), ha='center', va='bottom')
ax.annotate(f'{not_approved_count}', xy=(1, not_approved_count), ha='center', va='bottom')

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



```
In [84]: #Loan approval by property area distribution
import matplotlib.pyplot as plt
import seaborn as sns

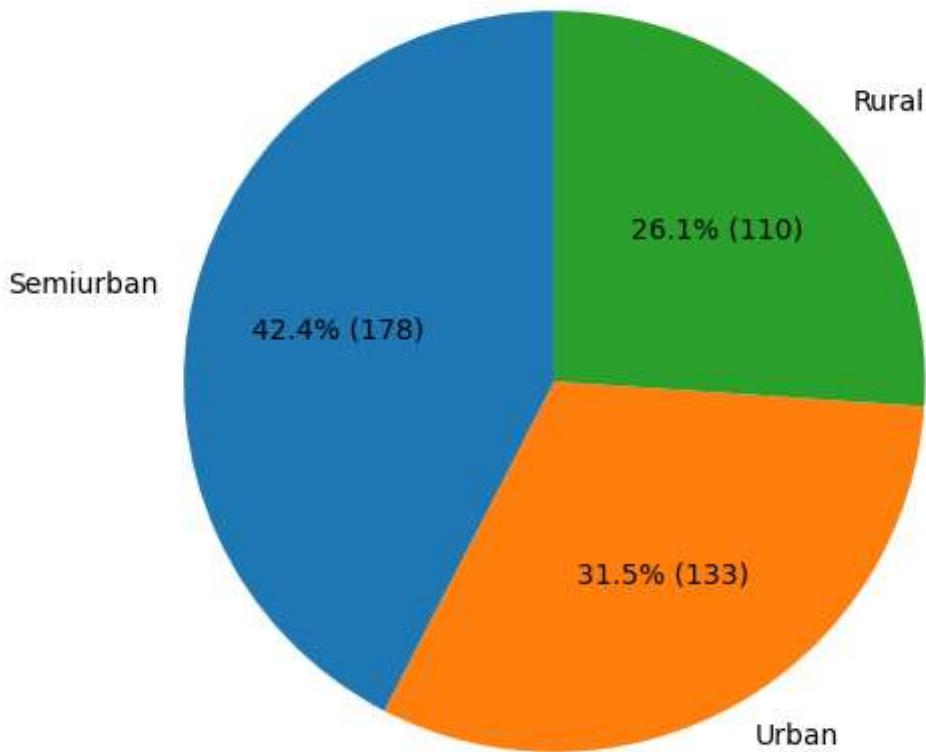
approved_df = df[df['Loan_Status'] == 'Y']

# Calculate the count
property_area_counts = approved_df['Property_Area'].value_counts()
total_approved_loans = approved_df.shape[0]
property_area_percentages = (property_area_counts / total_approved_loans) * 100

#plot the pie chart
plt.figure(figsize=(6, 6))
plt.pie(property_area_counts, labels=property_area_counts.index, autopct=lambda p: f'{p:.1f}%')
plt.title("Property Area Distribution for Approved Loans (Y)")

plt.show()
```

Property Area Distribution for Approved Loans (Y)



```
In [85]: #finding gender wise Loan status approval
plt.figure(figsize=(6, 4))
sns.countplot(x="Gender", hue="Loan_Status", data=df)
plt.title("Gender and Loan Status")
plt.show()
```



```
In [86]: import matplotlib.pyplot as plt
import seaborn as sns

# Filter the DataFrame to include only rows with 'Loan_Status' == 'Y'
approved_df = df[df['Loan_Status'] == 'Y']

# Create a figure and axes
plt.figure(figsize=(10, 6))
ax = plt.gca()

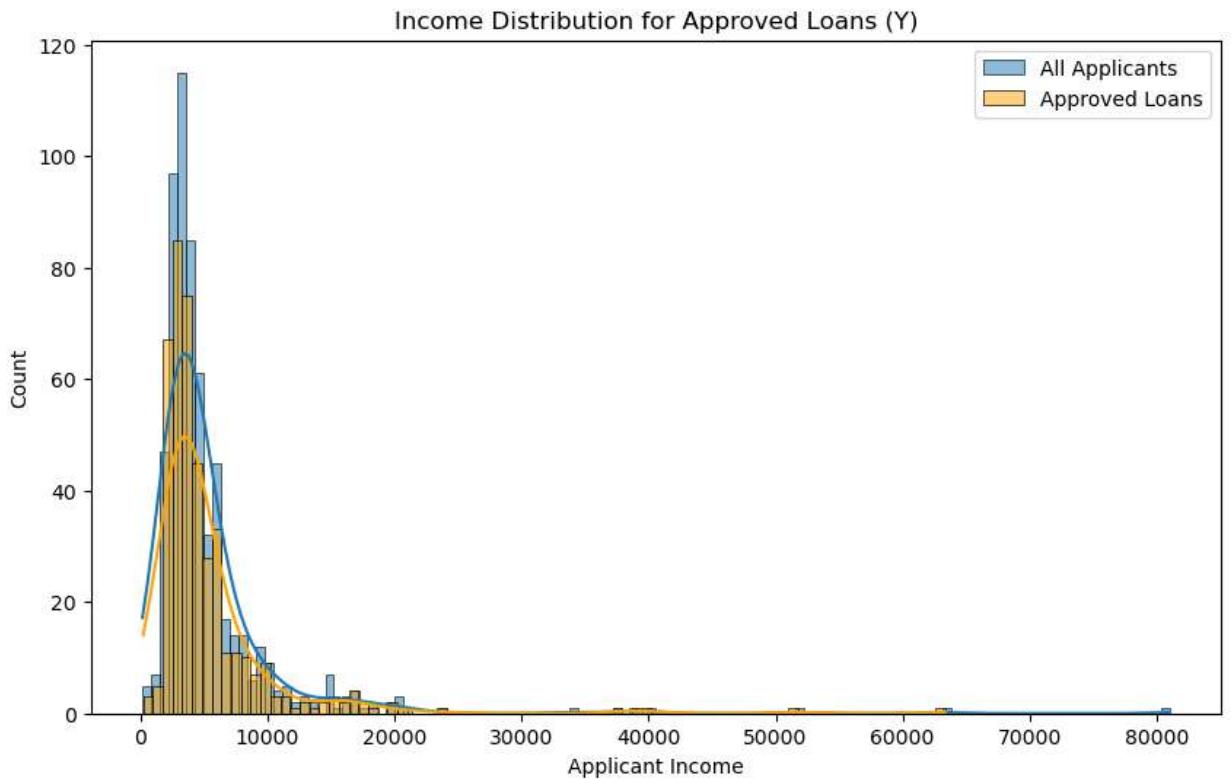
# Plot the histogram for Applicant Income
sns.histplot(df['ApplicantIncome'], kde=True, ax=ax, label='All Applicants')

# Plot the histogram for Applicant Income for approved Loans ('Y')
sns.histplot(approved_df['ApplicantIncome'], kde=True, ax=ax, color='orange', label='Approved')

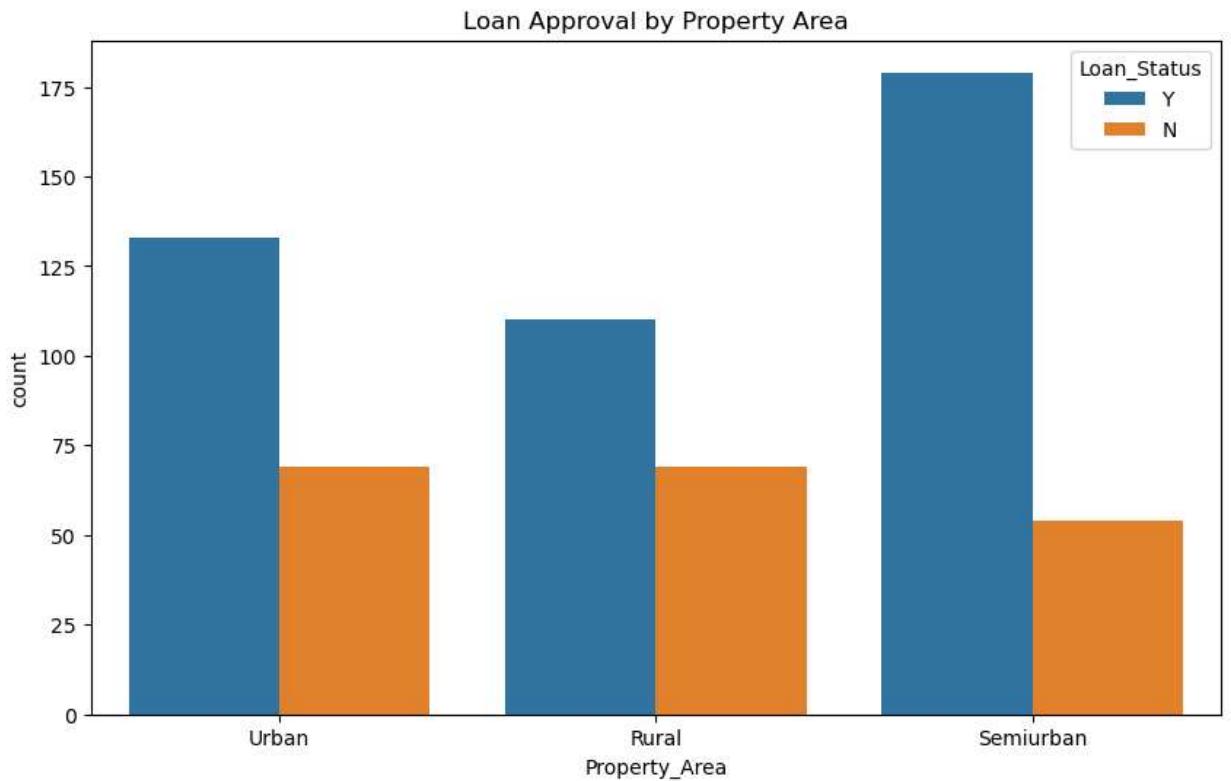
# Set Labels and title
plt.xlabel('Applicant Income')
plt.ylabel('Count')
plt.title('Income Distribution for Approved Loans (Y)')

# Add a Legend
plt.legend()

# Show the plot
plt.show()
```



```
In [87]: #Loan Approval by Property Area
plt.figure(figsize=(10, 6))
sns.countplot(x="Property_Area", hue="Loan_Status", data=df)
plt.title("Loan Approval by Property Area")
plt.show()
```



Task 3 Data Preprocessing with Python

```
In [88]: df['Dependents'] = pd.to_numeric(df['Dependents'], errors='coerce')

# Convert 'LoanAmount' and 'Loan_Amount_Term' columns to numeric
df['LoanAmount'] = pd.to_numeric(df['LoanAmount'], errors='coerce')
df['Loan_Amount_Term'] = pd.to_numeric(df['Loan_Amount_Term'], errors='coerce')

# Handling missing values
# In this example, we will fill missing values in 'LoanAmount' and 'Loan_Amount_Term'
mean_loan_amount = df['LoanAmount'].mean()
mean_loan_term = df['Loan_Amount_Term'].mean()

df['LoanAmount'].fillna(mean_loan_amount, inplace=True)
df['Loan_Amount_Term'].fillna(mean_loan_term, inplace=True)

# Convert categorical variables to numerical using one-hot encoding
df = pd.get_dummies(df, columns=['Gender', 'Married', 'Education', 'Self_Employed', 'Property_Area'])

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

```
In [89]: #checking if the data types of the columns are changed or not
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
 ---  --  
 0   Loan_ID          614 non-null    object 
 1   Dependents       548 non-null    float64
 2   ApplicantIncome  614 non-null    int64  
 3   CoapplicantIncome 614 non-null    float64
 4   LoanAmount        614 non-null    float64
 5   Loan_Amount_Term  614 non-null    float64
 6   Credit_History   564 non-null    float64
 7   Loan_Status       614 non-null    int64  
 8   Gender_Male      614 non-null    uint8  
 9   Married_Yes       614 non-null    uint8  
 10  Education_Not_Graduate 614 non-null  uint8  
 11  Self_Employed_Yes 614 non-null    uint8  
 12  Property_Area_Semiurban 614 non-null  uint8  
 13  Property_Area_Urban   614 non-null    uint8  
dtypes: float64(5), int64(2), object(1), uint8(6)
memory usage: 42.1+ KB
```

```
In [90]: df.head()
```

```
Out[90]:
```

	Loan_ID	Dependents	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
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 [91]: df.isnull().sum()
```

```
Out[91]:
```

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 [92]: #from the above code we found that column Dependents and Credit_History contains missing values  
# Handle missing values in 'Dependents' and 'Credit_History' columns  
# Here, we fill missing categorical values with the most frequent value (mode)  
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

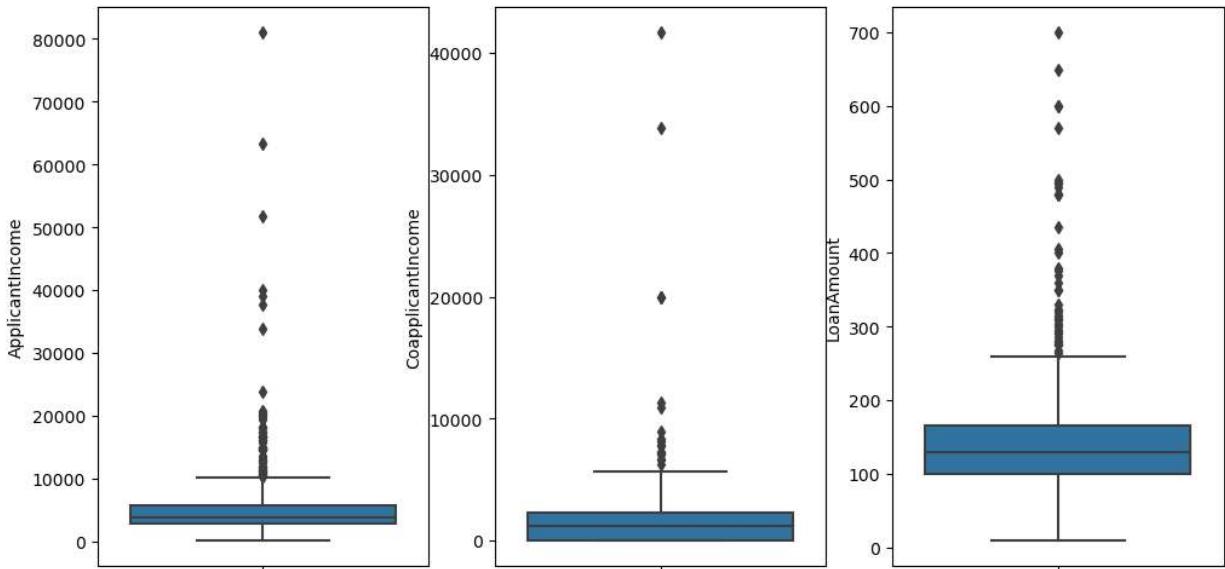
dtype: int64

```
In [93]: #check for outliers by box plot  
plt.figure(figsize=(12, 6))  
plt.subplot(1, 3, 1)
```

```

sns.boxplot(y='ApplicantIncome', data=df)
plt.subplot(1, 3, 2)
sns.boxplot(y='CoapplicantIncome', data=df)
plt.subplot(1, 3, 3)
sns.boxplot(y='LoanAmount', data=df)
plt.show()

```



```

In [94]: def handle_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

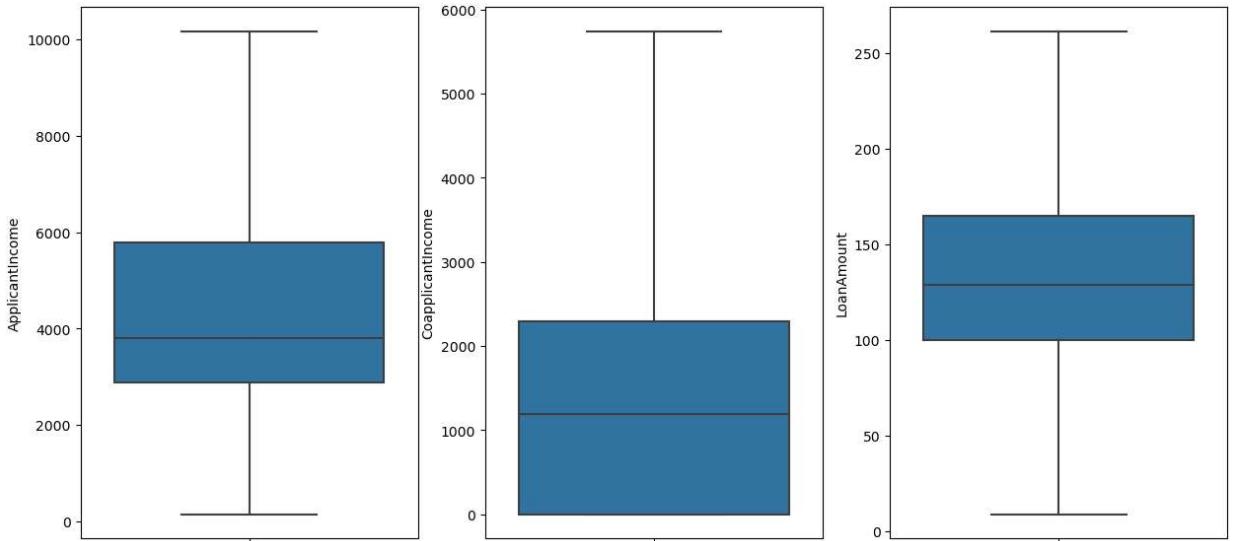
# Applying IQR method to 'ApplicantIncome' column
handle_outliers_iqr(df, 'ApplicantIncome')
handle_outliers_iqr(df, 'CoapplicantIncome')
handle_outliers_iqr(df, 'LoanAmount')

```

```

In [95]: #check for outliers after processing outliers
#check for outliers by box plot
plt.figure(figsize=(15, 7))
plt.subplot(1, 3, 1)
sns.boxplot(y='ApplicantIncome', data=df)
plt.subplot(1, 3, 2)
sns.boxplot(y='CoapplicantIncome', data=df)
plt.subplot(1, 3, 3)
sns.boxplot(y='LoanAmount', data=df)
plt.show()

```



In [96]: `df.head()`

Out[96]:

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

Task 4: Implement Machine Learning Models with Python

In [97]:

```

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 your DataFrame with the dataset

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

# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features for better model performance
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train Logistic Regression model

```

```
logistic_model = LogisticRegression(random_state=42)
logistic_model.fit(X_train_scaled, y_train)

# Make predictions on the test data
y_pred_logistic = logistic_model.predict(X_test_scaled)

# Evaluate Logistic Regression model
accuracy_logistic = accuracy_score(y_test, y_pred_logistic)
confusion_matrix_logistic = confusion_matrix(y_test, y_pred_logistic)
classification_report_logistic = classification_report(y_test, y_pred_logistic)

print("Logistic Regression Model Accuracy:", accuracy_logistic)
print("Confusion Matrix (Logistic Regression):")
print(confusion_matrix_logistic)
print("Classification Report (Logistic Regression):")
print(classification_report_logistic)

# Train Random Forest Classifier model
random_forest_model = RandomForestClassifier(random_state=42)
random_forest_model.fit(X_train_scaled, y_train)

# Make predictions on the test data
y_pred_rf = random_forest_model.predict(X_test_scaled)

# Evaluate Random Forest Classifier model
accuracy_rf = accuracy_score(y_test, y_pred_rf)
confusion_matrix_rf = confusion_matrix(y_test, y_pred_rf)
classification_report_rf = classification_report(y_test, y_pred_rf)

print("Random Forest Classifier Model Accuracy:", accuracy_rf)
print("Confusion Matrix (Random Forest Classifier):")
print(confusion_matrix_rf)
print("Classification Report (Random Forest Classifier):")
print(classification_report_rf)
```

```
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

```
Random Forest Classifier Model Accuracy: 0.7723577235772358
```

```
Confusion Matrix (Random Forest Classifier):
```

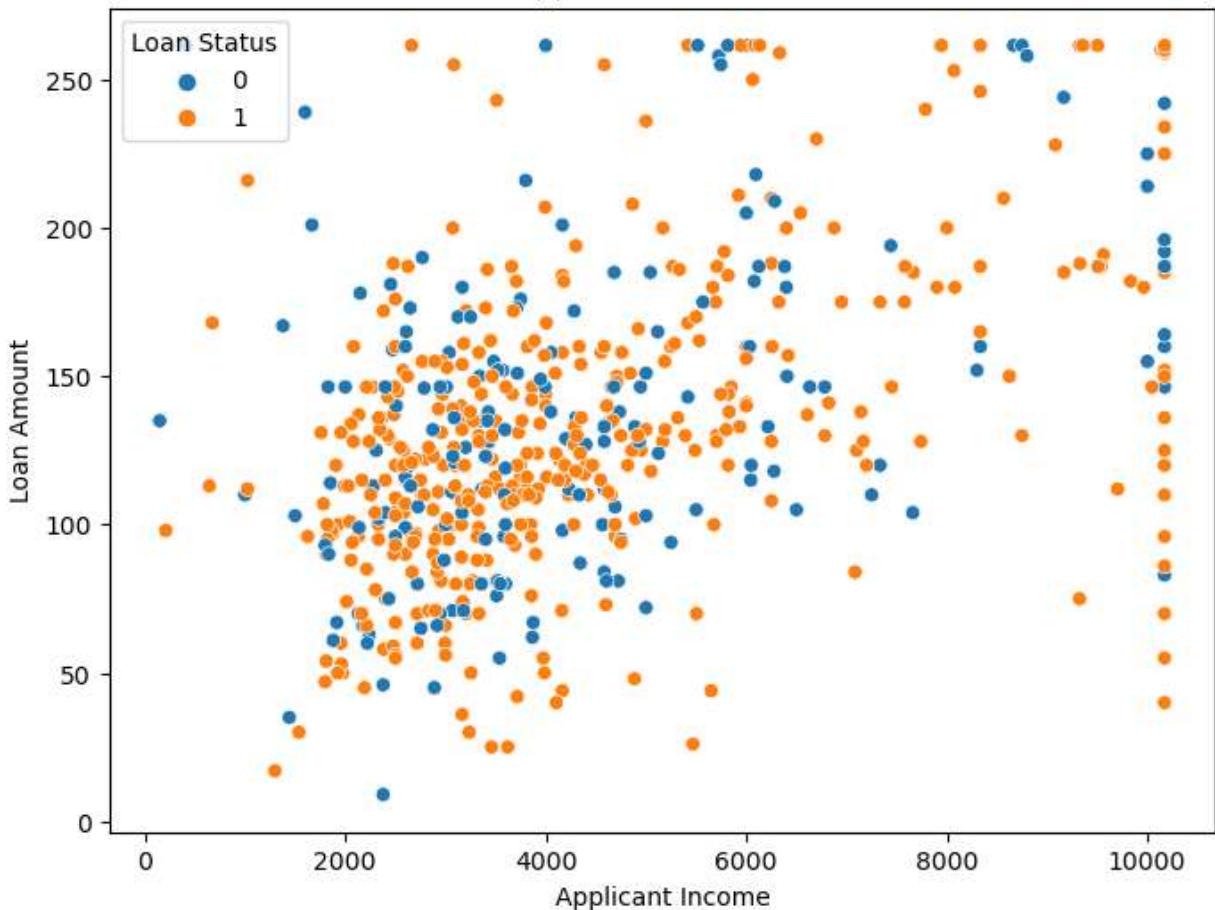
```
[[19 24]
 [ 4 76]]
```

```
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

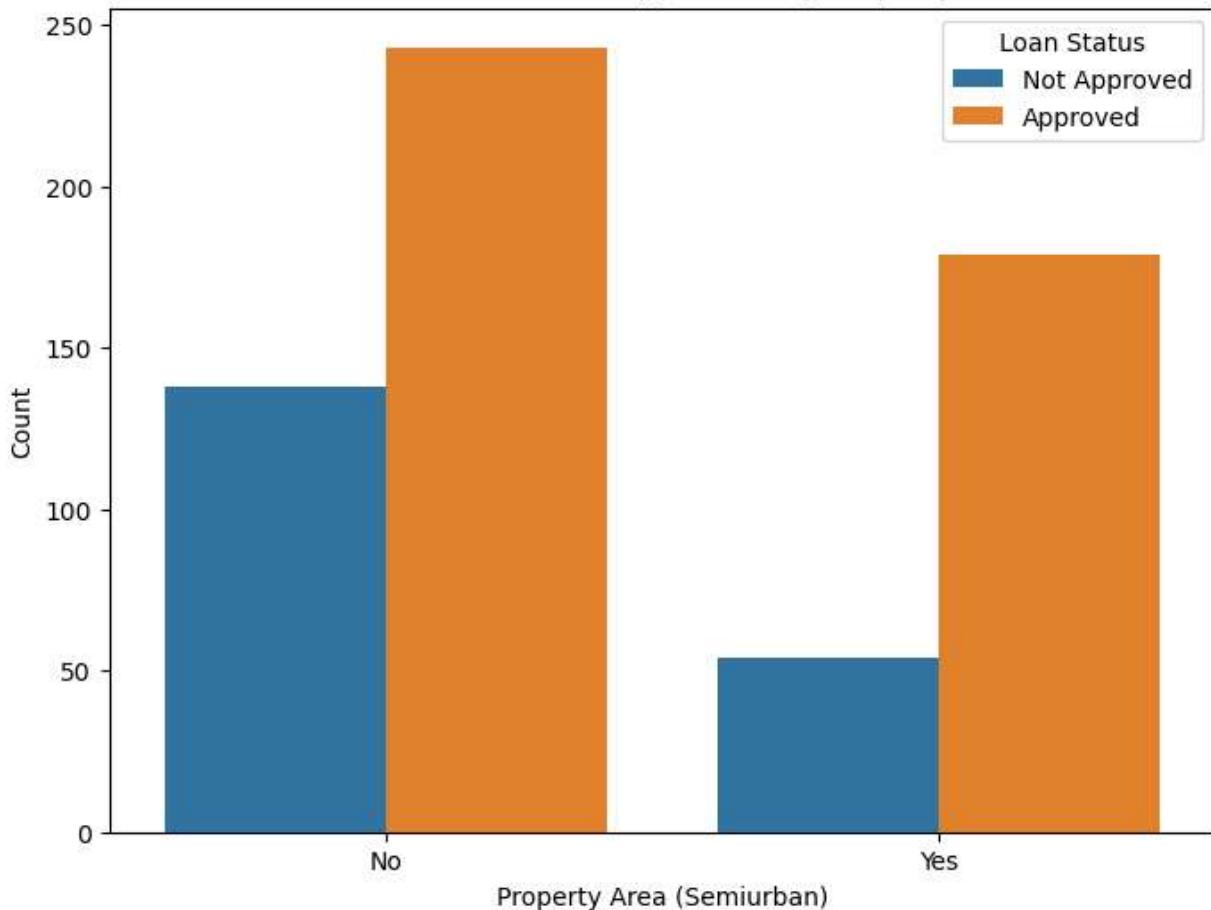
```
In [98]: plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='ApplicantIncome', y='LoanAmount', hue='Loan_Status')
plt.title('Scatter Plot: ApplicantIncome vs LoanAmount')
plt.xlabel('Applicant Income')
plt.ylabel('Loan Amount')
plt.legend(title='Loan Status')
plt.show()
```

Scatter Plot: ApplicantIncome vs LoanAmount



```
In [99]: plt.figure(figsize=(8, 6))
sns.countplot(data=df, x='Property_Area_Semiurban', hue='Loan_Status')
plt.title('Bar Chart: 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()
```

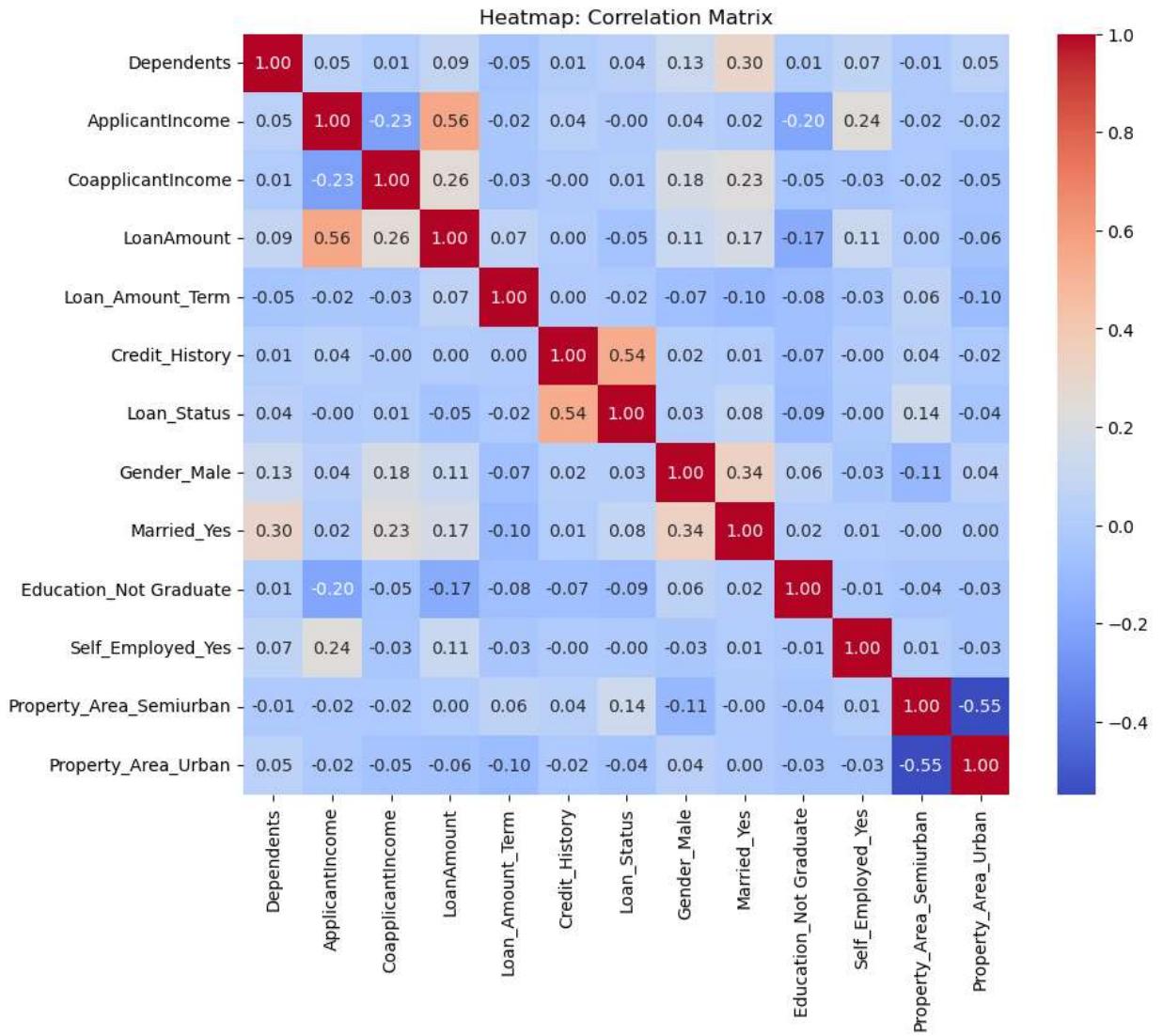
Bar Chart: Number of Applicants by Property Area



```
In [100]: correlation_matrix = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Heatmap: Correlation Matrix')
plt.show()
```

C:\Users\Rahil\AppData\Local\Temp\ipykernel_19688\3022833810.py:1: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.



In [101]:

```

fig1 = px.bar(df_1, x='Loan_Status', title='Loan Status by Gender', color='Gender')

# Interactive bar chart for Loan_Status by Property_Area
fig2 = px.bar(df_1, x='Loan_Status', title='Loan Status by Property Area', color='Prop

# Interactive scatter plot for ApplicantIncome and LoanAmount
fig3 = px.scatter(df_1, x='ApplicantIncome', y='LoanAmount', title='Applicant Income \

# Show the interactive plots
fig1.show()
fig2.show()
fig3.show()

```



```
In [102... df.head()
```

```
Out[102]:
```

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

```
In [ ]:
```