```
[2]: # Importing the necessary libraries for data analysis and visualization
import pandas as pd  # Pandas is used for handling and manipulating the dataset
import matplotlib.pyplot as plt  # Matplotlib is used for creating static visualizations like plots and charts
import seaborn as sns  # Seaborn builds on Matplotlib to provide more aesthetic and informative visualizations
```

```
# Task 1: Data Exploration
# Load the dataset into a Python environment (e.g., JupyterNotebook).
# Display the first few rows of the dataset to understand its structure.
# Check for missing values and handle them if necessary.
# Summarize basic statistics (mean, median, standard deviation, etc.)for the numeric columns.
```

```
[3]: # Loading the loan data from a CSV file into a Pandas DataFrame
loanfile = "C:/Users/subro/Downloads/loan_sanction_test.csv"  # Specifying the file path of the dataset
loandata = pd.read_csv(loanfile)  # Reading the CSV file into a DataFrame for further analysis and manipulation
```

```
[4]: # Displaying the first few rows of the dataset to get an initial understanding of the structure
print("                  First Few Rows Of The Dataset:- ")
print(loandata.head())  # This prints the top 5 rows, helping to visualize the dataset's columns and sample values

# Providing detailed information about the dataset such as column names, data types, and non-null counts
print()  # Adding a blank line for better readability in the output
print(loandata.info())  # This command gives a summary of the dataset, showing column names, data types, and missing data
```

```
                  First Few Rows Of The Dataset:-
    Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001015   Male     Yes          0      Graduate            No
1  LP001022   Male     Yes          1      Graduate            No
2  LP001031   Male     Yes          2      Graduate            No
3  LP001035   Male     Yes          2      Graduate            No
4  LP001051   Male      No          0  Not Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5720                  0       110.0             360.0
1             3076               1500       126.0             360.0
2             5000               1800       208.0             360.0
3             2340               2546       100.0             360.0
4             3276                  0        78.0             360.0

   Credit_History Property_Area
0             1.0         Urban
1             1.0         Urban
2             1.0         Urban
3             NaN         Urban
4             1.0         Urban

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 367 entries, 0 to 366
```

```
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            367 non-null    object
 1   Gender             356 non-null    object
 2   Married            367 non-null    object
 3   Dependents         357 non-null    object
 4   Education          367 non-null    object
 5   Self_Employed      344 non-null    object
 6   ApplicantIncome    367 non-null    int64
 7   CoapplicantIncome  367 non-null    int64
 8   LoanAmount         362 non-null    float64
 9   Loan_Amount_Term   361 non-null    float64
 10  Credit_History     338 non-null    float64
 11  Property_Area      367 non-null    object
dtypes: float64(3), int64(2), object(7)
memory usage: 34.5+ KB
None
```

[5]:
```python
# Checking for missing values in each column of the dataset
print("\nMissing values in each column:-")
print(loandata.isnull().sum())  # Summing up the number of missing values in each column to identify incomplete data
```

```
Missing values in each column:-
Loan_ID               0
Gender               11
Married               0
Dependents           10
Education             0
Self_Employed        23
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            5
Loan_Amount_Term      6
Credit_History       29
Property_Area         0
dtype: int64
```

[8]:
```python
# Handling missing values in the dataset by imputing them with appropriate strategies
# Filling missing 'Gender' values with the most frequent value (mode)
loandata['Gender'].fillna(loandata['Gender'].mode()[0], inplace=True)
# Filling missing 'Dependents' values with the most frequent value (mode)
loandata['Dependents'].fillna(loandata['Dependents'].mode()[0], inplace=True)
# Filling missing 'Self_Employed' values with the most frequent value (mode)
loandata['Self_Employed'].fillna(loandata['Self_Employed'].mode()[0], inplace=True)
# Filling missing 'LoanAmount' values with the mean of the column
loandata['LoanAmount'].fillna(loandata['LoanAmount'].mean(), inplace=True)
# Filling missing 'Loan_Amount_Term' values with the mode
loandata['Loan_Amount_Term'].fillna(loandata['Loan_Amount_Term'].mode()[0], inplace=True)
 # Filling missing 'Credit_History' values with the most frequent value (mode)
loandata['Credit_History'].fillna(loandata['Credit_History'].mode()[0], inplace=True)
```

•[10]:
```python
# Explanation:
# Purpose: This section addresses missing values through imputation, which is vital for maintaining data integrity and ensuring accurate analysis.
# Imputation Strategy:
# For categorical variables like Gender and Dependents, using the mode (most common value) is appropriate,
# as it preserves the distribution of the data.
# For numerical variables like LoanAmount, using the mean prevents bias in central tendency measures and
# ensures no drastic changes to the data distribution.
# Importance: Handling missing values is crucial in preparing the dataset for analysis, as many algorithms require complete datasets.
```

[11]:
```python
# Verifying if all missing values have beenhandled after imputation
check_missing_values_after_imputation = loandata.isnull().sum()  # Checking if there are any remaining missing values in the dataset
check_missing_values_after_imputation  # Display the number of missing values for each column to confirm no data is missing
```

[11]:
```
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
dtype: int64
```

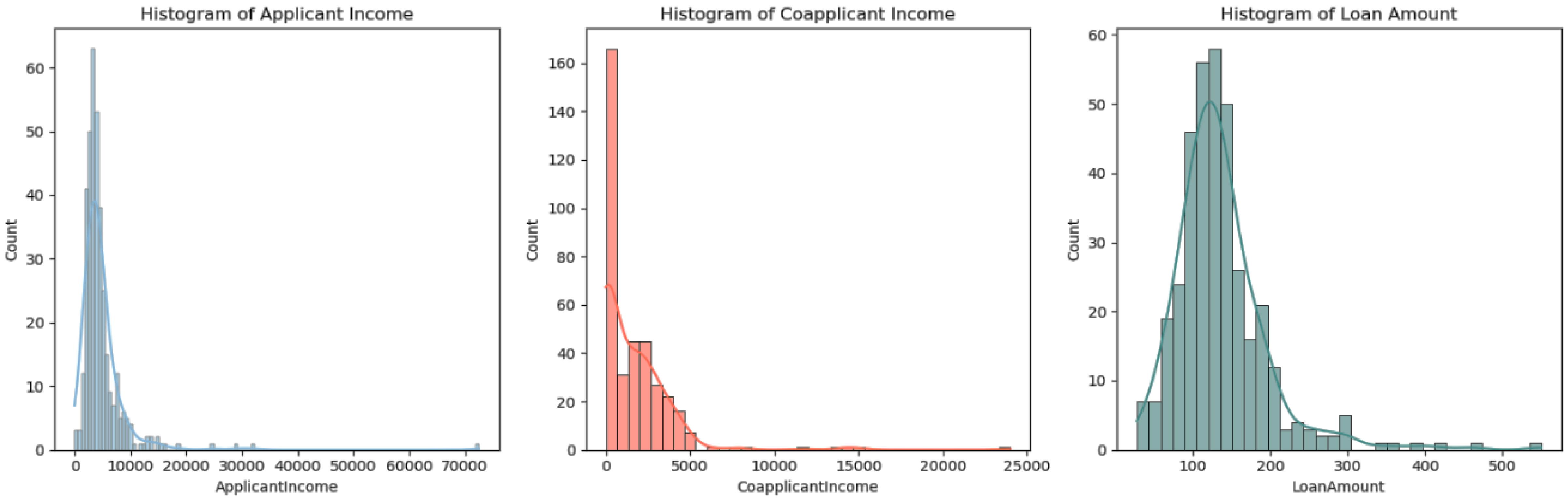
[13]:
```python
# Generating descriptive statistics for the numerical columns in the dataset
print(loandata.describe())
# This command provides a summary of statistical measures such as mean, standard deviation, min, max, and quartiles for numerical columns
```

```
       ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
count       367.000000         367.000000  367.000000        367.000000
mean       4805.599455        1569.577657  136.132597        342.822888
std        4910.685399        2334.232099   60.946040         64.658402
min           0.000000           0.000000   28.000000          6.000000
25%        2864.000000           0.000000  101.000000        360.000000
50%        3786.000000        1025.000000  126.000000        360.000000
75%        5060.000000        2430.500000  157.500000        360.000000
max       72529.000000       24000.000000  550.000000        480.000000

       Credit_History
count      367.000000
mean         0.839237
std          0.367814
min          0.000000
25%          1.000000
50%          1.000000
75%          1.000000
max          1.000000
```

```
# Task 2: Data Visualization
# 2.1 Univariate Analysis
# Explore the distribution of numeric columns using the following visualizations:
# Histograms: Plot the frequency distribution of key numeric variables.
# Box Plots: Identify potential outliers and visualize the spread of data.
# Analyze categorical variables by creating the following plots:
# Bar Charts: Visualize the frequency distribution of categoricalvariables.
# Pie Charts: Represent the composition of categorical variables
```

•[14]:

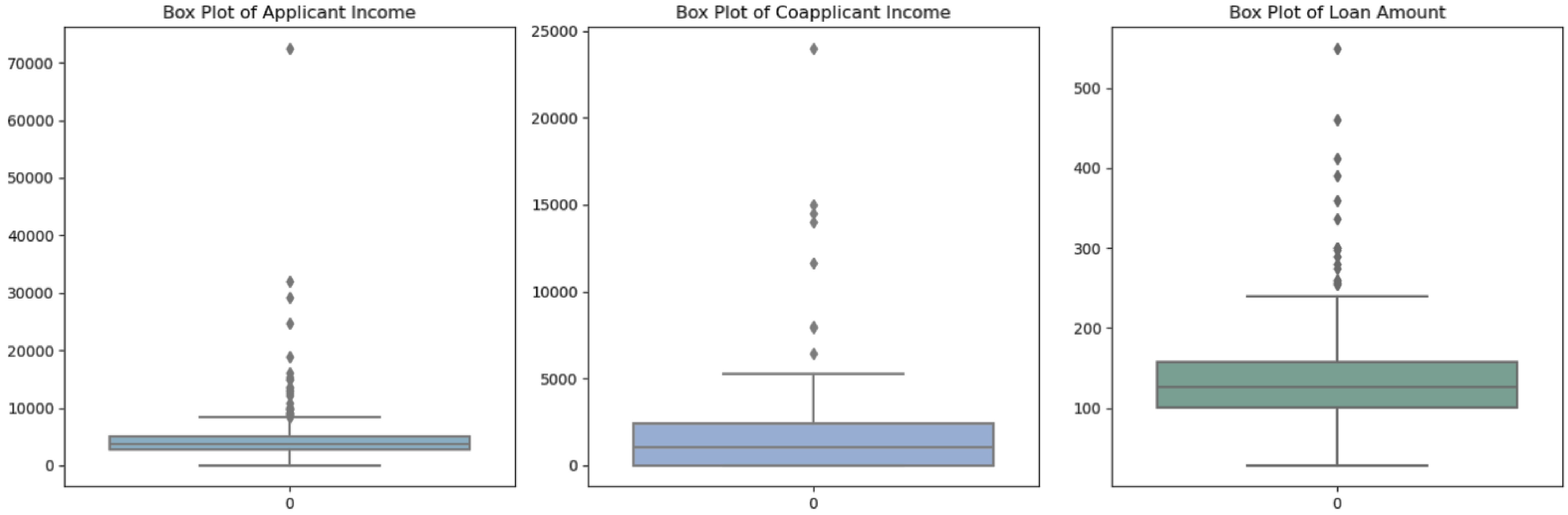
```
# Univariate Analysis: Generating Histograms for Numeric Columns
plt.figure(figsize=(15, 5))  # Setting the figure size for better visualization

# Histogram for Applicant Income
plt.subplot(1, 3, 1)  # Creating the first subplot
# Plotting the histogram with a kernel density estimate (KDE)
sns.histplot(loandata['ApplicantIncome'], kde=True, color='skyblue', edgecolor='black')
plt.title(f'Histogram of Applicant Income')  # Setting the title for the first histogram

# Histogram for Coapplicant Income
plt.subplot(1, 3, 2)  # Creating the second subplot
sns.histplot(loandata['CoapplicantIncome'], kde=True, color='#FF5733', edgecolor='black')  # Plotting the histogram for coapplicant income
plt.title(f'Histogram of Coapplicant Income')  # Setting the title for the second histogram

# Histogram for Loan Amount
plt.subplot(1, 3, 3)  # Creating the third subplot
sns.histplot(loandata['LoanAmount'], kde=True, color=plt.cm.viridis(0.5), edgecolor='black')  # Plotting the histogram for loan amounts
plt.title(f'Histogram of Loan Amount')  # Setting the title for the third histogram

plt.tight_layout()  # Adjusting the layout for better spacing between plots
plt.show()  # Displaying the plots
```



.6]:
```
"""
# Explanation:
"In this section, I conducted a univariate analysis of the numeric columns by generating histograms for three key variables:
'ApplicantIncome', 'CoapplicantIncome', and 'LoanAmount'.
# Purpose:
Histograms are used to visualize the distribution of numeric variables.
KDE (Kernel Density Estimation): The KDE curve overlay provides a smoother representation of the distribution, enhancing interpretability.
# Insights:
The first histogram illustrates the distribution of 'ApplicantIncome', showing how many applicants fall into various income ranges.

The second histogram presents 'CoapplicantIncome', allowing us to assess the financial contribution of co-applicants.

The third histogram focuses on 'LoanAmount', providing insights into the range of loan amounts applicants are requesting.
"""
```

[17]:

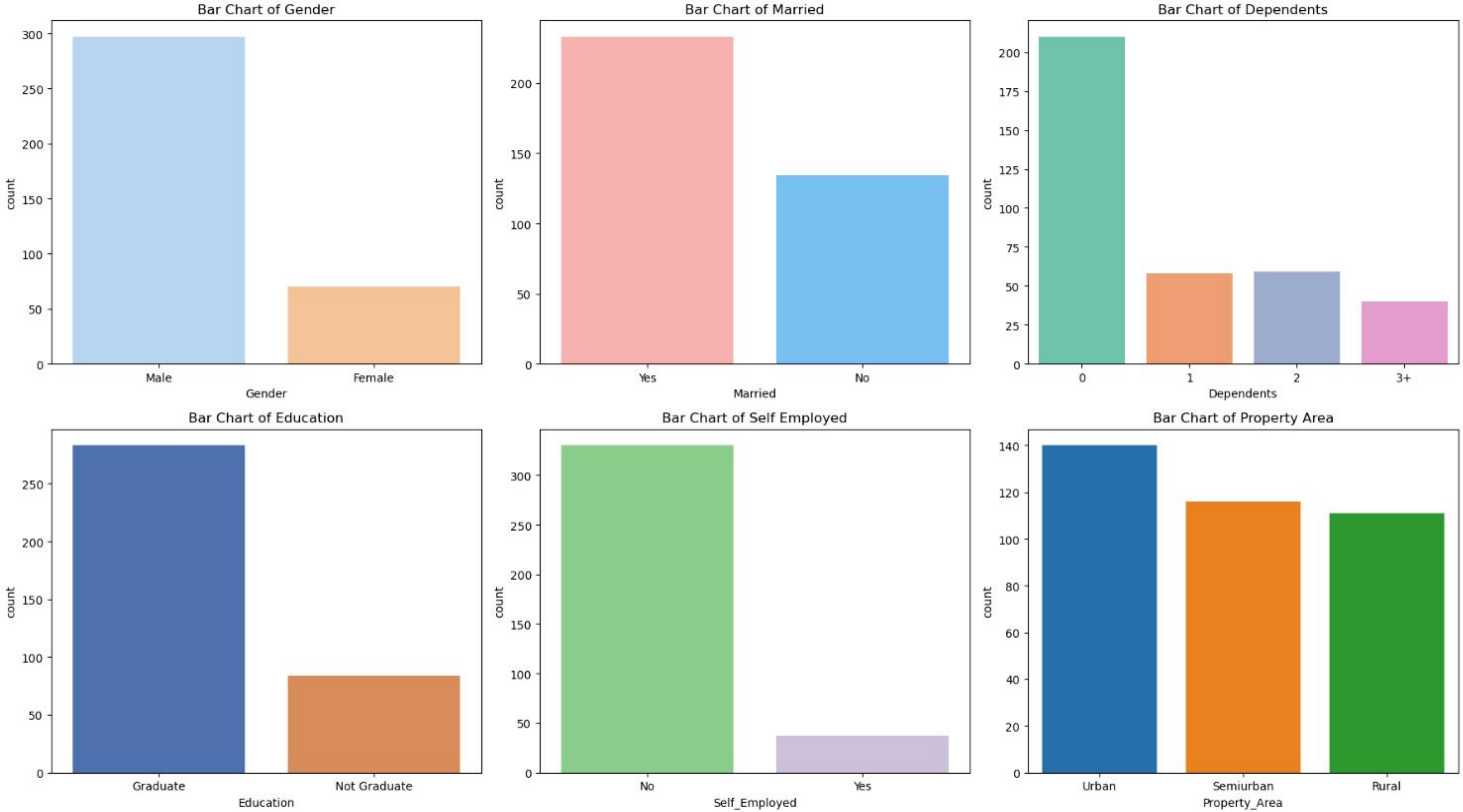
```python
# Univariate Analysis: Generating Box Plots for Numeric Columns
plt.figure(figsize=(15, 5))  # Setting the figure size for better visualization

# Box Plot for Applicant Income
plt.subplot(1, 3, 1)  # Creating the first subplot
sns.boxplot(loandata['ApplicantIncome'], color='skyblue')  # Plotting the box plot for applicant income
plt.title(f'Box Plot of Applicant Income')  # Setting the title for the first box plot

# Box Plot for Coapplicant Income
plt.subplot(1, 3, 2)  # Creating the second subplot
sns.boxplot(loandata['CoapplicantIncome'], palette='pastel')  # Plotting the box plot for coapplicant income
plt.title(f'Box Plot of Coapplicant Income')  # Setting the title for the second box plot

# Box Plot for Loan Amount
plt.subplot(1, 3, 3)  # Creating the third subplot
sns.boxplot(loandata['LoanAmount'], palette='Set2')  # Plotting the box plot for loan amounts
plt.title(f'Box Plot of Loan Amount')  # Setting the title for the third box plot

plt.tight_layout()  # Adjusting the layout for better spacing between plots
plt.show()  # Displaying the plots
```



[ ]:
```
"""
# Explanation:
In this section, I performed a univariate analysis of the numeric columns by generating box plots for
'ApplicantIncome', 'CoapplicantIncome', and 'LoanAmount'.
# Purpose:
Box plots summarize the distribution of numeric data and highlight outliers.
# Insights:
The first box plot illustrates 'ApplicantIncome', providing insights into the income distribution and
highlighting any outliers in the applicant population.

The second box plot focuses on 'CoapplicantIncome', allowing us to assess the range and variability of incomes contributed by co-applicants.

The third box plot displays 'LoanAmount', revealing the distribution of loan amounts requested and identifying potential outliers.
"""
```

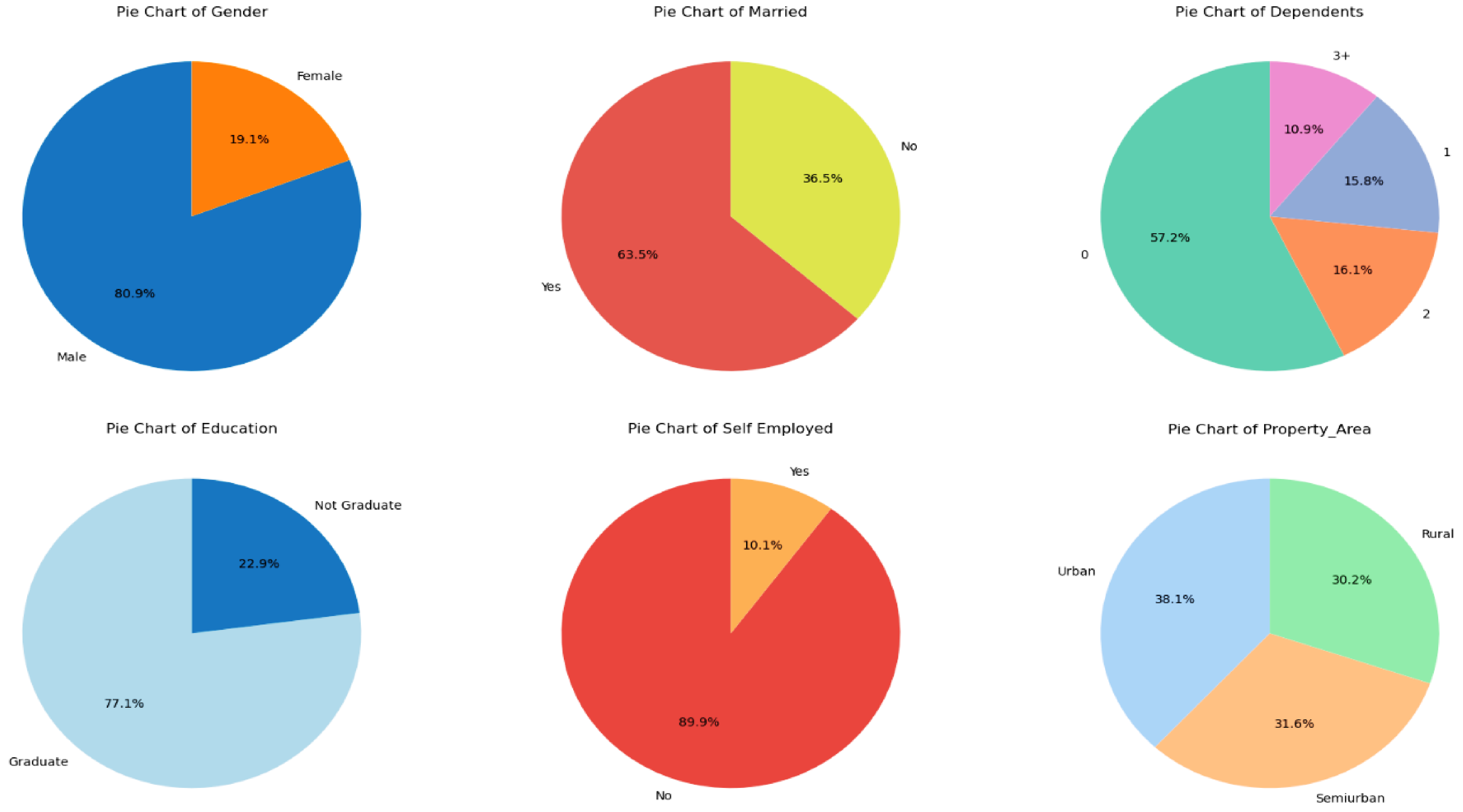
[18]:

```python
# Univariate Analysis: Generating Bar Charts for Categorical Columns
plt.figure(figsize=(18, 10))  # Setting the figure size for better visualization

# Bar Chart for Gender
plt.subplot(2, 3, 1)  # Creating the first subplot
sns.countplot(x='Gender', data=loandata, palette='pastel')  # Plotting the count of applicants by gender
plt.title(f'Bar Chart of Gender')  # Setting the title for the first bar chart

# Bar Chart for Married Status
plt.subplot(2, 3, 2)  # Creating the second subplot
sns.countplot(x='Married', data=loandata, palette=['#FF9999', '#66B3FF'])  # Plotting the count of married vs. unmarried applicants
plt.title(f'Bar Chart of Married')  # Setting the title for the second bar chart

# Bar Chart for Dependents
plt.subplot(2, 3, 3)  # Creating the third subplot
sns.countplot(x='Dependents', data=loandata, palette='Set2')  # Plotting the count of applicants based on the number of dependents
plt.title(f'Bar Chart of Dependents')  # Setting the title for the third bar chart

# Bar Chart for Education
plt.subplot(2, 3, 4)  # Creating the fourth subplot
sns.countplot(x='Education', data=loandata, palette='deep')  # Plotting the count of applicants based on education level
plt.title(f'Bar Chart of Education')  # Setting the title for the fourth bar chart

# Bar Chart for Self Employed
plt.subplot(2, 3, 5)  # Creating the fifth subplot
sns.countplot(x='Self_Employed', data=loandata, palette='Accent')  # Plotting the count of self-employed applicants
plt.title(f'Bar Chart of Self Employed')  # Setting the title for the fifth bar chart

# Bar Chart for Property Area
plt.subplot(2, 3, 6)  # Creating the sixth subplot
sns.countplot(x='Property_Area', data=loandata)  # Plotting the count of applicants based on property area
plt.title(f'Bar Chart of Property Area')  # Setting the title for the sixth bar chart

plt.tight_layout()  # Adjusting the layout for better spacing between plots
plt.show()  # Displaying the plots
```


Bar Chart of Gender
Bar Chart of Married
Bar Chart of Dependents
Bar Chart of Education
Bar Chart of Self Employed
Bar Chart of Property Area
count
Male
Female
Gender
Yes
No
Married
0
1
2
3+
Dependents
Graduate
Not Graduate
Education
No
Yes
Self_Employed
Urban
Semiurban
Rural
Property_Area

```
[ ]: """
# Explanation:
In this section, I performed a univariate analysis of categorical columns by generating bar charts for several key variables:
'Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', and 'Property_Area'.
# Purpose:
Bar charts are effective for visualizing the frequency of categorical data. Each chart illustrates the count of applicants in each category,
allowing for a clear comparison of different groups.
# Insights:
# The first bar chart shows the distribution of applicants by gender, providing insights into the gender representation within the dataset.

# The second chart displays the marital status of applicants, highlighting the proportion of married versus unmarried individuals.

# The third chart illustrates the number of dependents for applicants, which may impact loan approval and amounts.

# The fourth chart presents the educational background of the applicants, revealing the proportion of graduates and non-graduates.

# The fifth chart focuses on the self-employment status of applicants, providing a glimpse into the employment landscape.

# The final chart represents the distribution of applicants across different property areas,
  giving insights into the geographical diversity of the applicants.
"""
```

[20]:
```python
# Univariate Analysis: Generating Pie Charts for Categorical Columns
plt.figure(figsize=(18, 10))  # Setting the figure size for better visualization

# Pie Chart for Gender
plt.subplot(2, 3, 1)  # Creating the first subplot
# Plotting pie chart for gender distribution
loandata['Gender'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90, colors=sns.color_palette('tab10'))
plt.title(f'Pie Chart of Gender')  # Setting the title for the first pie chart
plt.ylabel('')  # Hiding the y-label for better aesthetics

# Pie Chart for Married Status
plt.subplot(2, 3, 2)  # Creating the second subplot
# Plotting pie chart for marital status
loandata['Married'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90, colors=sns.color_palette('hls'))
plt.title(f'Pie Chart of Married')  # Setting the title for the second pie chart
plt.ylabel('')  # Hiding the y-label

# Pie Chart for Dependents
plt.subplot(2, 3, 3)  # Creating the third subplot
# Plotting pie chart for dependents
loandata['Dependents'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90, colors=sns.color_palette('Set2'))
plt.title(f'Pie Chart of Dependents')  # Setting the title for the third pie chart
plt.ylabel('')  # Hiding the y-label

# Pie Chart for Education
plt.subplot(2, 3, 4)  # Creating the fourth subplot
# Plotting pie chart for education levels
loandata['Education'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90, colors=sns.color_palette('Paired'))
plt.title(f'Pie Chart of Education')  # Setting the title for the fourth pie chart
plt.ylabel('')  # Hiding the y-label

# Pie Chart for Self Employed
plt.subplot(2, 3, 5)  # Creating the fifth subplot
# Plotting pie chart for self-employment status
loandata['Self_Employed'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90, colors=sns.color_palette('Spectral'))
plt.title(f'Pie Chart of Self Employed')  # Setting the title for the fifth pie chart
plt.ylabel('')  # Hiding the y-label

# Pie Chart for Property Area
plt.subplot(2, 3, 6)  # Creating the sixth subplot
# Plotting pie chart for property area
loandata['Property_Area'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90, colors=sns.color_palette('pastel'))
plt.title(f'Pie Chart of Property_Area')  # Setting the title for the sixth pie chart
plt.ylabel('')  # Hiding the y-label

plt.tight_layout()  # Adjusting the layout for better spacing between plots
plt.show()  # Displaying the plots
```

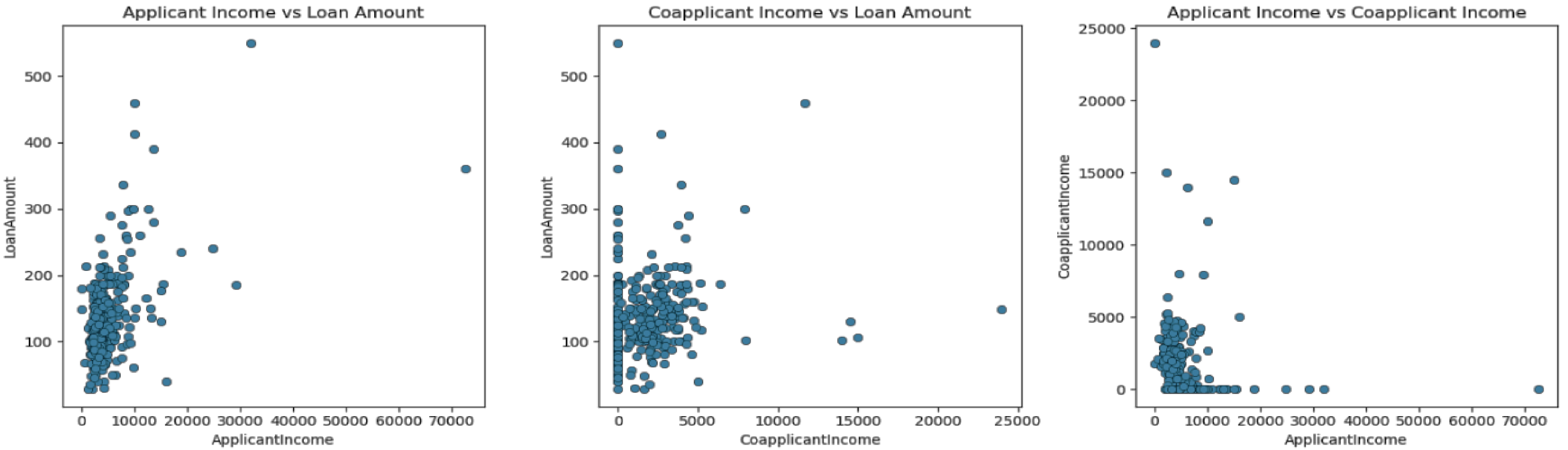


Pie Chart of Gender
Female
19.1%
80.9%
Male
Pie Chart of Married
No
36.5%
63.5%
Yes
Pie Chart of Dependents
3+
10.9%
1
15.8%
0
57.2%
16.1%
2
Pie Chart of Education
Not Graduate
22.9%
77.1%
Graduate
Pie Chart of Self Employed
Yes
10.1%
89.9%
No
Pie Chart of Property_Area
Rural
Urban
30.2%
38.1%
31.6%
Semiurban

[ ]:
```
"""
# Explanation:
 In this section, I performed a univariate analysis of categorical columns by generating pie charts for several key variables:
 'Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', and 'Property_Area'.
# Purpose:
 Pie charts are useful for visualizing the proportions of categories within a whole.
 Each pie chart displays the percentage distribution of each category, allowing for an intuitive understanding of the composition of each variable.
# Insights:
    The first pie chart represents the gender distribution of applicants, providing insights into the representation of male and female applicants.

    The second pie chart shows the marital status of applicants, highlighting the proportion of married versus unmarried individuals.

    The third pie chart illustrates the number of dependents, revealing how many applicants have dependents and their distribution.

    The fourth pie chart focuses on the educational qualifications of applicants, showcasing the proportion of graduates and non-graduates.

    The fifth pie chart displays the self-employment status of applicants, indicating the number of self-employed individuals in the dataset.

    The final pie chart represents the distribution of applicants across different property areas,
    giving insights into the geographical diversity of the applicants.
"""
```

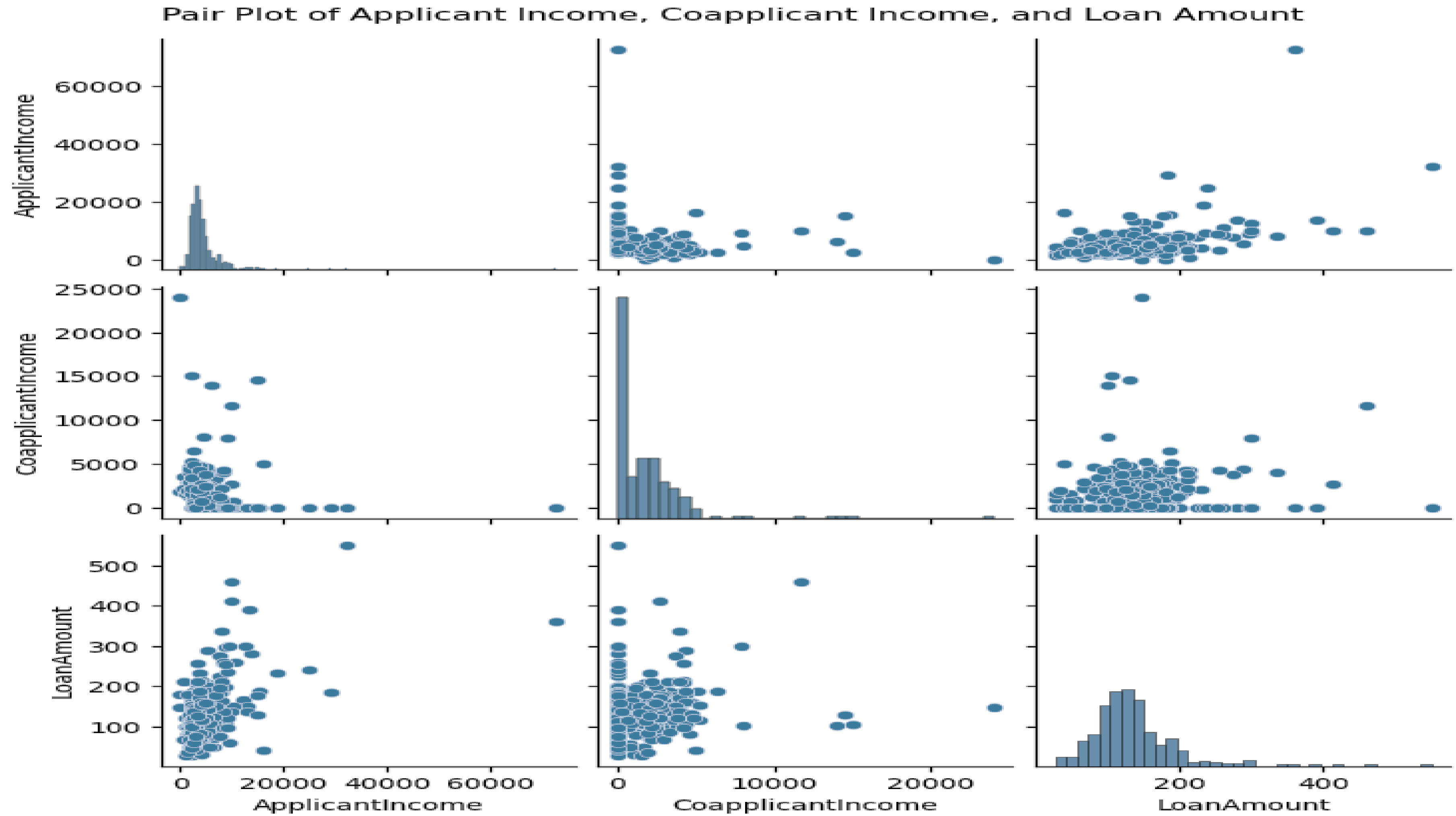
```
# 2.2 Bivariate Analysis
# Create scatter plots to explore relationships between pairs of numeric variables.
# Use pair plots (scatter matrix) to visualize interactions betweenmultiple numeric variables simultaneously.
# Investigate the relationship between categorical and numeric variables using box plots or violin plots.
```

```
21]: # Bivariate Analysis: Generating Scatter Plots for Numeric Columns
plt.figure(figsize=(15, 5))  # Setting the figure size for better visualization

# Scatter Plot for Applicant Income vs Loan Amount
plt.subplot(1, 3, 1)  # Creating the first subplot
# Plotting the scatter plot for applicant income vs loan amount
sns.scatterplot(x='ApplicantIncome', y='LoanAmount', data=loandata, palette='Set2', edgecolor='black')
plt.title('Applicant Income vs Loan Amount')  # Setting the title for the first scatter plot

# Scatter Plot for Coapplicant Income vs Loan Amount
plt.subplot(1, 3, 2)  # Creating the second subplot
# Plotting the scatter plot for coapplicant income vs loan amount
sns.scatterplot(x='CoapplicantIncome', y='LoanAmount', data=loandata, palette=['#FF9999', '#66B3FF', '#99FF99'], edgecolor='black')
plt.title('Coapplicant Income vs Loan Amount')  # Setting the title for the second scatter plot

# Scatter Plot for Applicant Income vs Coapplicant Income
plt.subplot(1, 3, 3)  # Creating the third subplot
# Plotting the scatter plot for applicant income vs coapplicant income
sns.scatterplot(x='ApplicantIncome', y='CoapplicantIncome', data=loandata, palette='dark', edgecolor='black')
plt.title('Applicant Income vs Coapplicant Income')  # Setting the title for the third scatter plot

plt.tight_layout()  # Adjusting the layout for better spacing between plots
plt.show()  # Displaying the plots
```



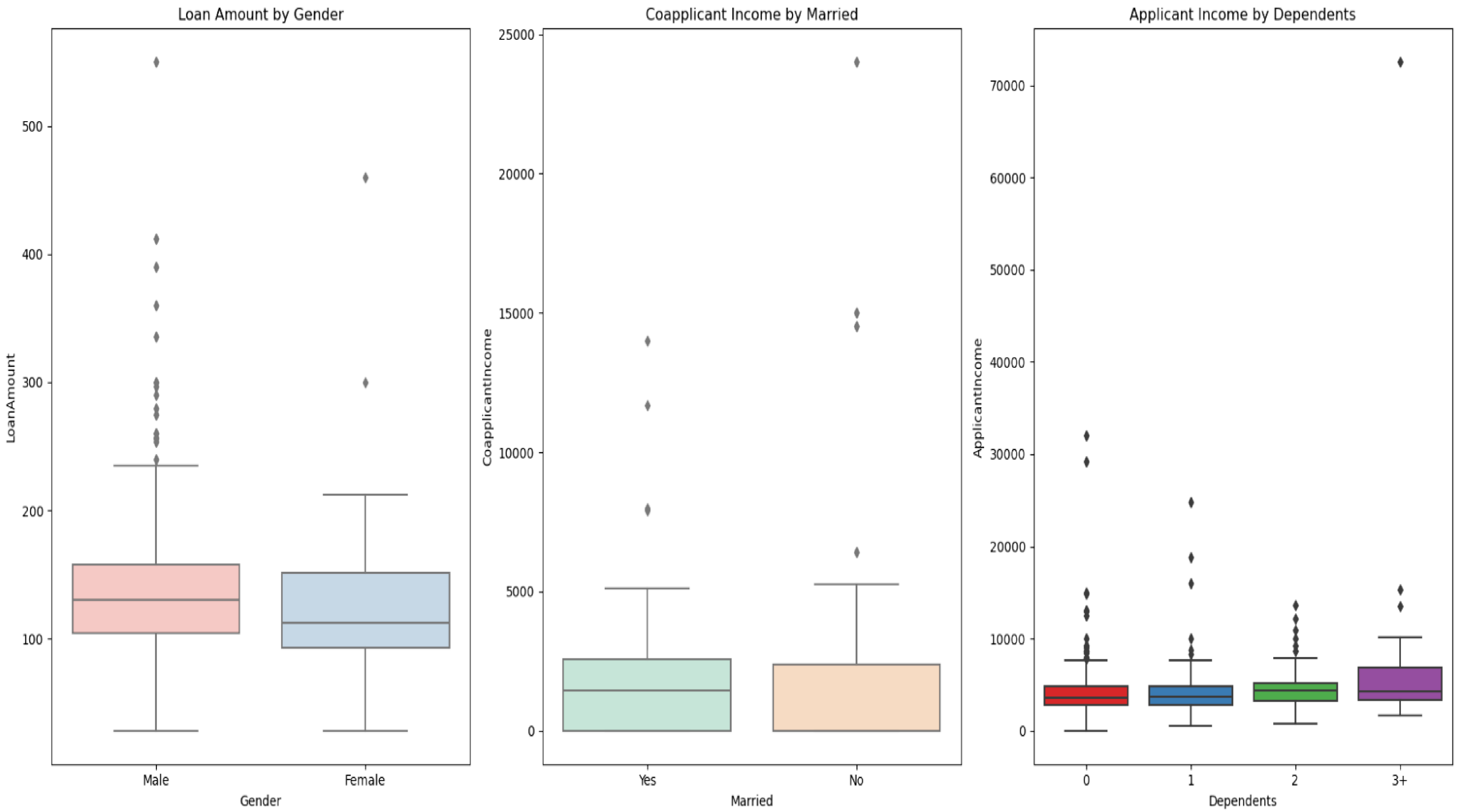
[ ]:
```
"""
# Explanation:
 In this section, I performed a bivariate analysis of the numeric columns by generating scatter plots to explore the relationships between
 pairs of variables: 'ApplicantIncome' and 'LoanAmount','CoapplicantIncome' and 'LoanAmount', as well as 'ApplicantIncome' and 'CoapplicantIncome'
# Purpose:
  Scatter plots are effective for visualizing the relationship between two quantitative variables.
  Each point on the plot represents an individual loan application, allowing us to identify any potential correlations or trends in the data.
# Insights:
  The first scatter plot illustrates the relationship between 'ApplicantIncome' and 'LoanAmount'.
  This plot helps to determine if higher incomes correlate with larger loan amounts.

  The second scatter plot focuses on 'CoapplicantIncome' and 'LoanAmount', providing insights into
  how the income of co-applicants influences the loan amount.

  The third scatter plot explores the relationship between 'ApplicantIncome' and 'CoapplicantIncome',
  enabling us to understand how these two income streams interact.
"""
```

```
# Pair Plot: Visualizing Interactions Between Multiple Numeric Variables
sns.pairplot(loandata[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']])  # Creating pair plots for selected numeric variables
plt.suptitle('Pair Plot of Applicant Income, Coapplicant Income, and Loan Amount', y=1.02)  # Setting the overall title for the pair plot
plt.show()  # Displaying the pair plot
```

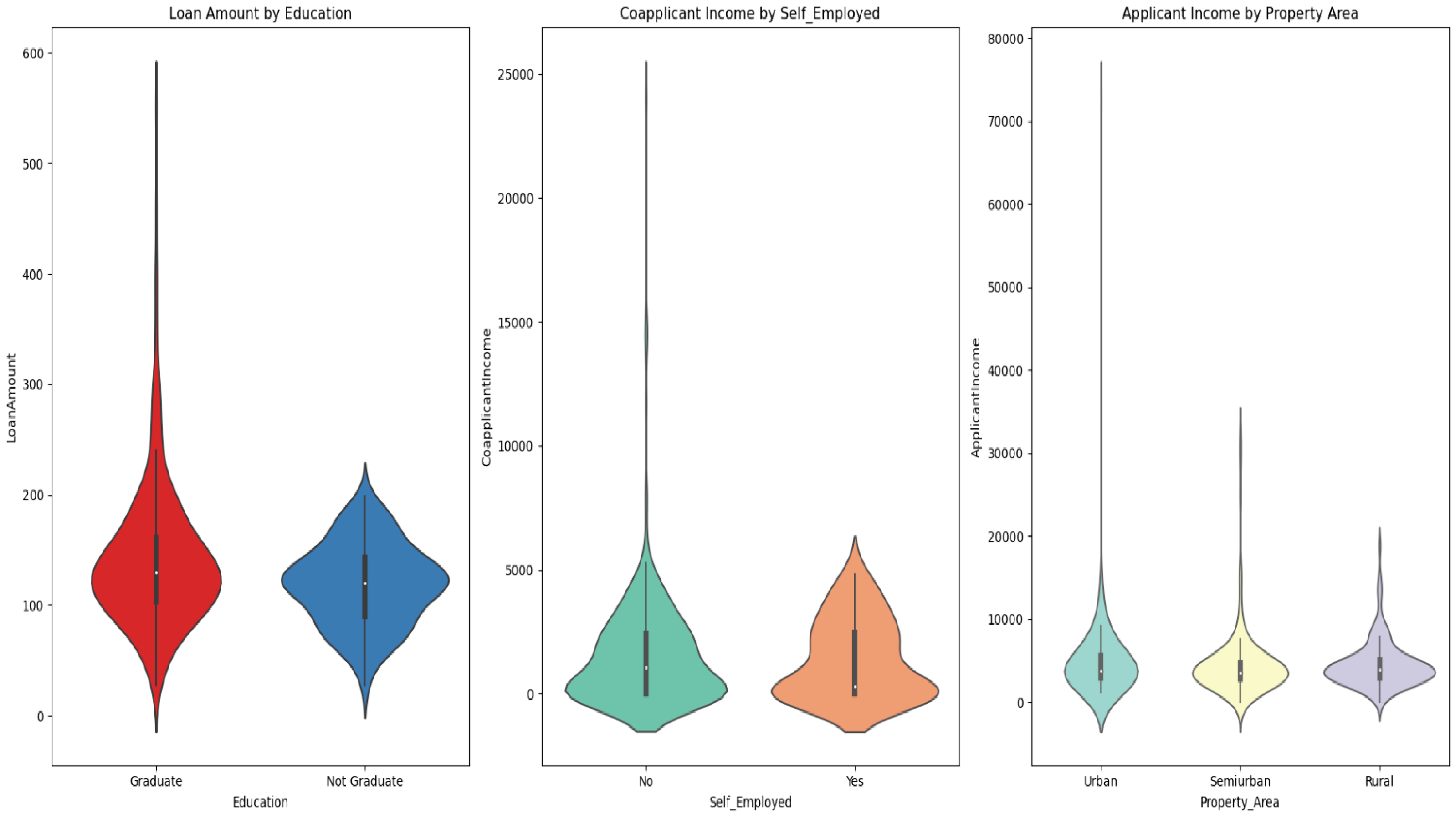


```
"""

# Explanation:
  In this section, I utilized a pair plot to visualize the interactions between multiple numeric variables:
  'ApplicantIncome', 'CoapplicantIncome', and 'LoanAmount'.
# Purpose:
  Pair plots are a powerful tool for exploratory data analysis, allowing us to see the relationships between several variables simultaneously.
  Each subplot within the pair plot provides a scatter plot for each pair of variables,
  along with histograms on the diagonal to show the distribution of each variable.
# Insights:
  This specific pair plot reveals potential correlations between 'ApplicantIncome', 'CoapplicantIncome', and 'LoanAmount'.
  By examining these scatter plots, we can identify trends, clusters, or any possible outliers that may exist in the data.
  The overall title, adjusted with y=1.02, ensures it does not overlap with the plots and enhances readability.
"""
```

```python
# Bivariate Analysis: Box Plots for Categorical vs Numeric Variables
plt.figure(figsize=(18, 8))  # Setting the figure size for better visualization

# Box Plot for Loan Amount by Gender
plt.subplot(1, 3, 1)  # Creating the first subplot
sns.boxplot(x='Gender', y='LoanAmount', data=loandata, palette='Pastel1')  # Plotting the box plot for loan amount by gender
plt.title('Loan Amount by Gender')  # Setting the title for the first box plot

# Box Plot for Coapplicant Income by Married Status
plt.subplot(1, 3, 2)  # Creating the second subplot
sns.boxplot(x='Married', y='CoapplicantIncome', data=loandata, palette='Pastel2')  # Plotting the box plot for coapplicant income by marital status
plt.title('Coapplicant Income by Married')  # Setting the title for the second box plot

# Box Plot for Applicant Income by Dependents
plt.subplot(1, 3, 3)  # Creating the third subplot
sns.boxplot(x='Dependents', y='ApplicantIncome', data=loandata, palette='Set1')  # Plotting the box plot for applicant income by dependents
plt.title('Applicant Income by Dependents')  # Setting the title for the third box plot

plt.tight_layout()  # Adjusting the layout for better spacing between plots
plt.show()  # Displaying the box plots

# Violin Plots for Categorical vs Numeric Variables
plt.figure(figsize=(18, 8))  # Setting the figure size for better visualization

# Violin Plot for Loan Amount by Education
plt.subplot(1, 3, 1)  # Creating the first subplot
sns.violinplot(x='Education', y='LoanAmount', data=loandata, palette='Set1')  # Plotting the violin plot for loan amount by education
plt.title('Loan Amount by Education')  # Setting the title for the first violin plot

# Violin Plot for Coapplicant Income by Self Employed Status
plt.subplot(1, 3, 2)  # Creating the second subplot
# Plotting the violin plot for coapplicant income by self-employment status
sns.violinplot(x='Self_Employed', y='CoapplicantIncome', data=loandata, palette='Set2')
plt.title('Coapplicant Income by Self_Employed')  # Setting the title for the second violin plot

# Violin Plot for Applicant Income by Property Area
plt.subplot(1, 3, 3)  # Creating the third subplot
# Plotting the violin plot for applicant income by property area
sns.violinplot(x='Property_Area', y='ApplicantIncome',data=loandata, palette='Set3')
plt.title('Applicant Income by Property Area')  # Setting the title for the third violin plot

plt.tight_layout()  # Adjusting the layout for better spacing between plots
plt.show()  # Displaying the violin plots
```



Loan Amount by Gender
LoanAmount
100
200
300
400
500
Male
Female
Gender
Coapplicant Income by Married
CoapplicantIncome
0
5000
10000
15000
20000
25000
Yes
No
Married
Applicant Income by Dependents
ApplicantIncome
0
10000
20000
30000
40000
50000
60000
70000
0
1
2
3+
Dependents



Loan Amount by Education
LoanAmount
600
500
400
300
200
100
0
Graduate
Not Graduate
Education
Coapplicant Income by Self_Employed
CoapplicantIncome
25000
20000
15000
10000
5000
0
No
Yes
Self_Employed
Applicant Income by Property Area
ApplicantIncome
80000
70000
60000
50000
40000
30000
20000
10000
0
Urban
Semiurban
Rural
Property_Area

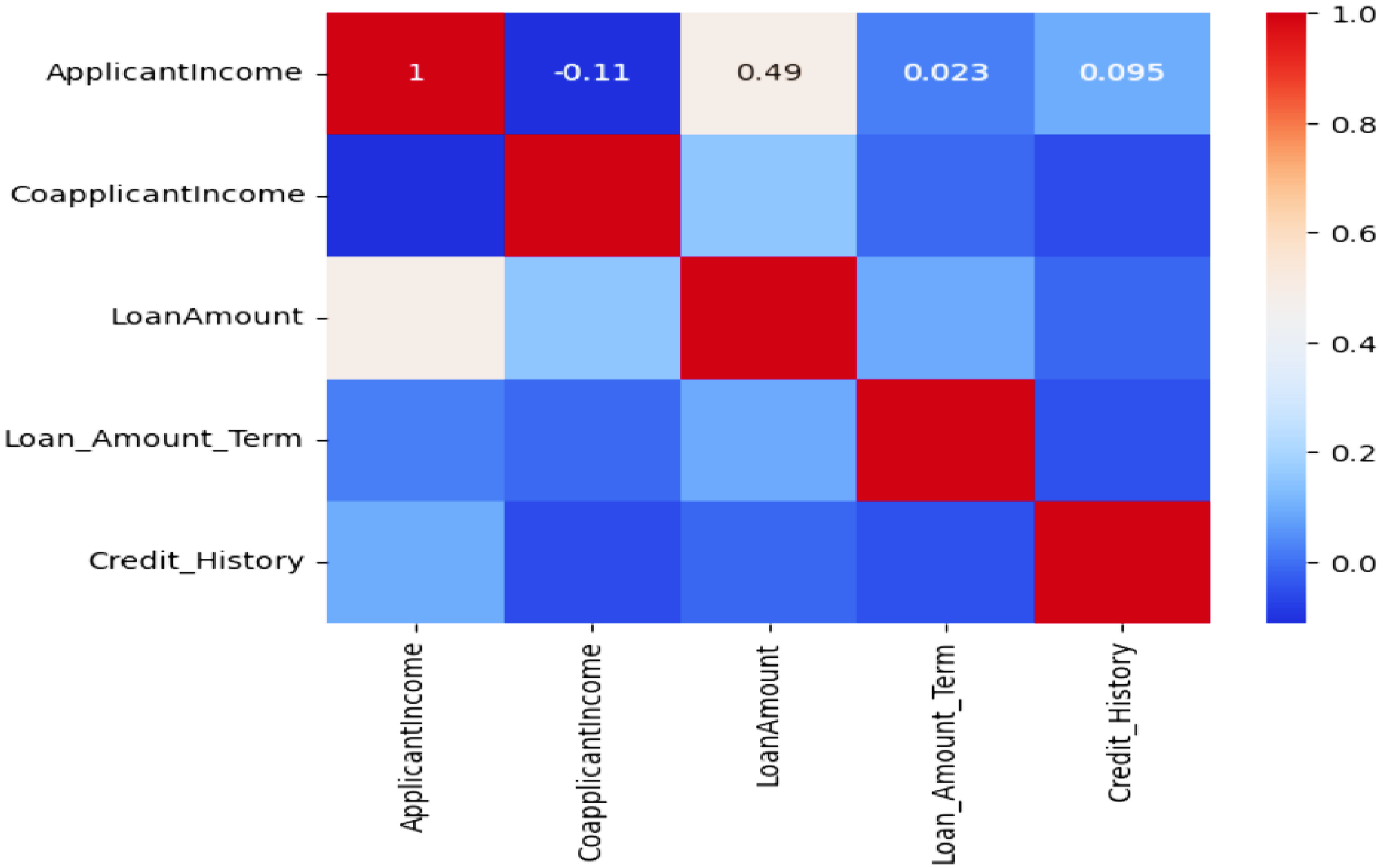
```
"""
# Explanation:
  In this section, I conducted a bivariate analysis using box plots and violin plots to explore the relationship
  between categorical and numeric variables.
# Purpose:
  The box plots provide a summary of the distribution of a numeric variable
  (Loan Amount, Coapplicant Income, Applicant Income) for each category of a categorical variable (Gender, Married, Dependents).
# Insights:
  The first box plot illustrates how 'Loan Amount' varies by 'Gender', highlighting differences
  in loan amounts granted to male and female applicants.

  The second box plot shows 'Coapplicant Income' based on 'Married' status,
  which can help determine if marital status influences income levels.

  The third box plot presents 'Applicant Income' against 'Dependents',
  revealing how the number of dependents may affect applicant income distributions.
# Purpose:
  Violin plots extend the information provided by box plots by displaying the density of the data at different values,
  thus providing more insight into the distribution shape.
# Insights:
  The first violin plot visualizes 'Loan Amount' by 'Education', allowing us to understand how education levels impact loan amounts.

  The second violin plot examines 'Coapplicant Income' by 'Self_Employed' status,
  offering insights into income disparities between self-employed individuals and others.

  The third violin plot illustrates 'Applicant Income' based on 'Property Area',
  providing a visual representation of income distribution across different geographical locations.
"""
```

2.3 Multivariate Analysis

Perform a correlation analysis to identify relationships between numeric variables. Visualize correlations using a heatmap.

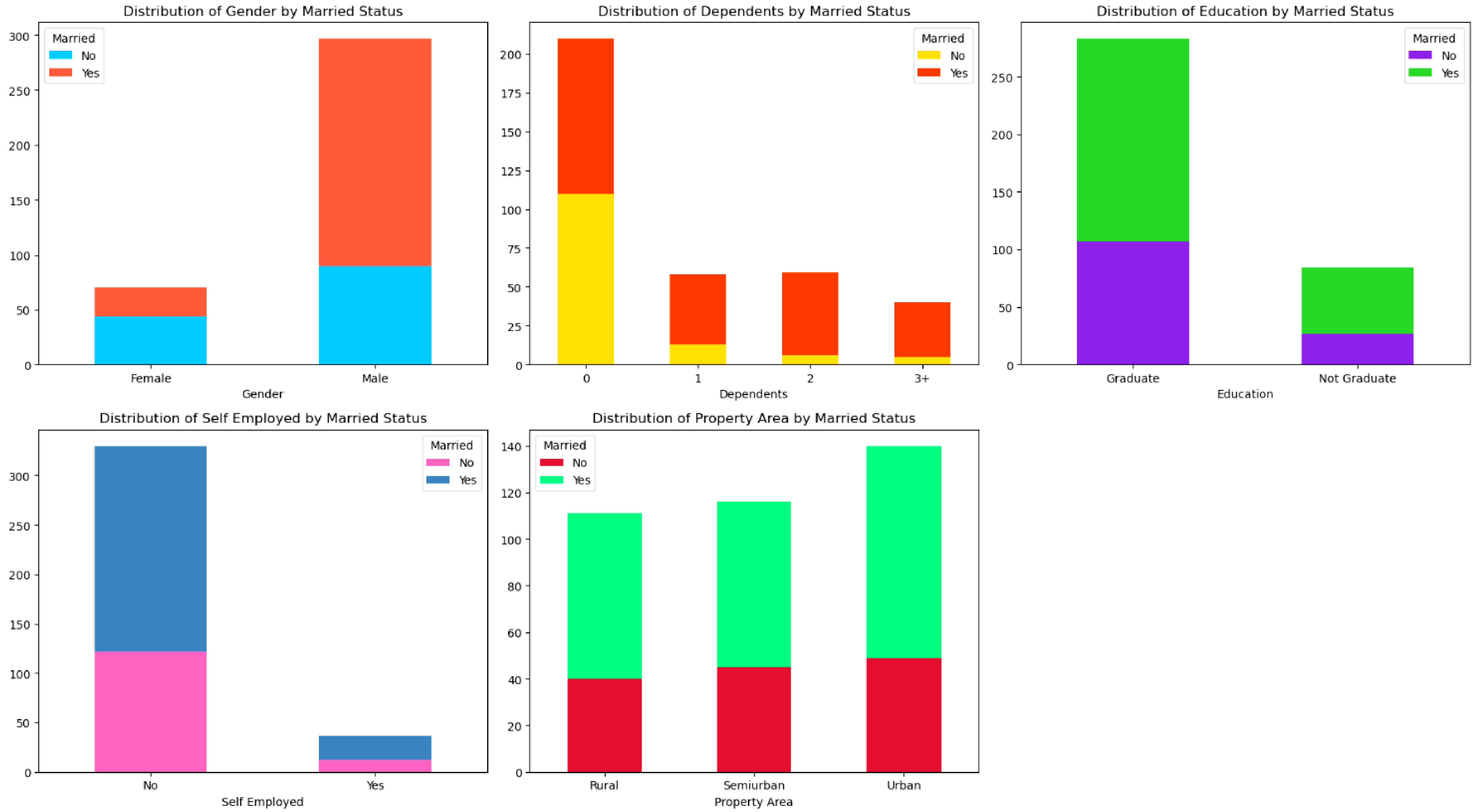
Create a stacked bar chart to show the distribution of categorical variables across multiple categories.

```python
# Selecting Only Numeric Columns for Correlation Analysis
numeric_data = loandata.select_dtypes(include=['int64', 'float64'])  # Selecting columns with numeric data types

# Calculating the Correlation Matrix
correlation = numeric_data.corr()  # Computing the correlation matrix for numeric data

# Plotting the Heatmap of the Correlation Matrix
sns.heatmap(correlation, annot=True, cmap='coolwarm')  # Creating a heatmap to visualize the correlation matrix
plt.show()  # Displaying the heatmap
```


ApplicantIncome
CoapplicantIncome
LoanAmount
Loan_Amount_Term
Credit_History
1
-0.11
0.49
0.023
0.095
1.0
0.8
0.6
0.4
0.2
0.0
ApplicantIncome
CoapplicantIncome
LoanAmount
Loan_Amount_Term
Credit_History

]:
```
"""
# Explanation:
    In this section, I performed a correlation analysis on the numeric columns of our dataset to assess
    the relationships between different financial variables.
# Purpose:
    A correlation heatmap visualizes the strength and direction of relationships between numeric variables.
    This analysis is essential for identifying which variables are correlated.
# Insights:
    Numeric Column Selection: I first selected only the numeric columns from the dataset using select_dtypes(include=['int64', 'float64']).
    This allows us to focus on variables suitable for correlation analysis, such as 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', etc.

    Correlation Matrix Calculation: I calculated the correlation matrix using the corr() function. This matrix provides a measure of the
    strength and direction of relationships between pairs of numeric variables, with values ranging
    from -1 (perfect negative correlation) to +1 (perfect positive correlation). A value of 0 indicates no correlation.

    Heatmap Visualization: The correlation matrix was visualized using a heatmap with sns.heatmap().
    This graphical representation helps to easily identify strong and weak correlations among variables.

    The annot=True parameter adds the correlation coefficient values directly onto the heatmap for clarity.
    The cmap='coolwarm' parameter specifies the color palette used, with cooler colors representing negative
    correlations and warmer colors representing positive correlations.
"""
```

```python
# Creating Subplots for Categorical Variables
fig, axes = plt.subplots(2, 3, figsize=(18, 10))  # Setting up a 2x3 grid of subplots

# Plot for Gender
# Creating a stacked bar plot for Gender and Married status with new colors
pd.crosstab(loandata['Gender'], loandata['Married']).plot(kind='bar', stacked=True, ax=axes[0, 0], color=['#00BFFF', '#FF6347'])
axes[0, 0].set_title('Distribution of Gender by Married Status')  # Title for the first subplot
axes[0, 0].set_xlabel('Gender')  # X-axis label
axes[0, 0].set_xticklabels(axes[0, 0].get_xticklabels(), rotation=0)  # Adjusting x-axis tick labels

# Plot for Dependents
# Stacked bar plot for Dependents and Married status with new colors
pd.crosstab(loandata['Dependents'], loandata['Married']).plot(kind='bar', stacked=True, ax=axes[0, 1], color=['#FFD700', '#FF4500'])
axes[0, 1].set_title('Distribution of Dependents by Married Status')  # Title for the second subplot
axes[0, 1].set_xlabel('Dependents')  # X-axis label
axes[0, 1].set_xticklabels(axes[0, 1].get_xticklabels(), rotation=0)  # Adjusting x-axis tick labels

# Plot for Education
# Stacked bar plot for Education and Married status with new colors
pd.crosstab(loandata['Education'], loandata['Married']).plot(kind='bar', stacked=True, ax=axes[0, 2], color=['#8A2BE2', '#32CD32'])
axes[0, 2].set_title('Distribution of Education by Married Status')  # Title for the third subplot
axes[0, 2].set_xlabel('Education')  # X-axis label
axes[0, 2].set_xticklabels(axes[0, 2].get_xticklabels(), rotation=0)  # Adjusting x-axis tick labels

# Plot for Self Employed
# Stacked bar plot for Self Employed and Married status with new colors
pd.crosstab(loandata['Self_Employed'], loandata['Married']).plot(kind='bar', stacked=True, ax=axes[1, 0], color=['#FF69B4', '#4682B4'])
axes[1, 0].set_title('Distribution of Self Employed by Married Status')  # Title for the fourth subplot
axes[1, 0].set_xlabel('Self Employed')  # X-axis label
axes[1, 0].set_xticklabels(axes[1, 0].get_xticklabels(), rotation=0)  # Adjusting x-axis tick labels

# Plot for Property Area
# Stacked bar plot for Property Area and Married status with new colors
pd.crosstab(loandata['Property_Area'], loandata['Married']).plot(kind='bar', stacked=True, ax=axes[1, 1], color=['#DC143C', '#00FF7F'])
axes[1, 1].set_title('Distribution of Property Area by Married Status')  # Title for the fifth subplot
axes[1, 1].set_xlabel('Property Area')  # X-axis label
axes[1, 1].set_xticklabels(axes[1, 1].get_xticklabels(), rotation=0)  # Adjusting x-axis tick labels

# Hide the last subplot (empty)
axes[1, 2].axis('off')  # Hiding the empty subplot

plt.tight_layout()  # Adjusting the layout for better spacing
plt.show()  # Displaying the plots
```


Distribution of Gender by Married Status
Married
No
Yes
0
50
100
150
200
250
300
Female
Male
Gender
Distribution of Dependents by Married Status
Married
No
Yes
0
25
50
75
100
125
150
175
200
0
1
2
3+
Dependents
Distribution of Education by Married Status
Married
No
Yes
0
50
100
150
200
250
Graduate
Not Graduate
Education
Distribution of Self Employed by Married Status
Married
No
Yes
0
50
100
150
200
250
300
No
Yes
Self Employed
Distribution of Property Area by Married Status
Married
No
Yes
0
20
40
60
80
100
120
140
Rural
Semiurban
Urban
Property Area

[29]:

```
"""
# Explanation:
    In this section, I created stacked bar plots to analyze the distribution of various categorical variables against the 'Married' status.
# Purpose:
    Stacked bar charts show the relationship between two categorical variables,
    revealing the distribution of categories within another category.
# Insights:
        The first plot illustrates the distribution of gender among married and unmarried individuals.
        The second plot shows how the number of dependents varies with marital status.
        The third plot examines the education levels among married and unmarried individuals.
        The fourth plot analyzes the self-employment status in relation to marital status.
        The fifth plot looks at how property area influences marital status.
# Visualization Details:
        I customized the titles and x-axis labels for clarity, and adjusted the tick labels for improved readability.
        The last subplot was left empty and hidden to maintain a clean layout.
        Color Combinations: Each plot has been customized with different color combinations to make each subplot visually distinct
        and easier to interpret.
        This allows for quick comparison between different categorical variables.
        Subplots Structure: The fig, axes structure allows you to create a 2x3 grid of subplots,
        where each plot represents the relationship between a categorical variable and the "Married" status.
"""
```

```python
# Task 4:
# Geospatial Analysis (Optional)
# If the dataset contains geographical information, visualize data on amap to identify regional trends.
# Use scatter plots or heatmaps to display data patterns acrossgeographic locations.
```

[ ]:

```python
"""
Upon reviewing the dataset for the geospatial analysis task, it has been determined that the dataset lacks relevant geographical
information necessary for visualizing data trends across geographic locations.
"""
```