eda-project-subashis

March 29, 2024

1 *TO UPLOAD THE DATA SET IN JUPYTER NOTE BOOK.*

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
df=pd.read_csv(r"D:\EDA PROJECT\loan_data.csv")
```

[3]:
```
# to read first 10 records
df.head(10)
```

[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0			
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0			
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0			
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0			
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0			
5	LP001011	Male	Yes	2	Graduate	Yes	5417	4196.0	267.0	360.0			
6	LP001013	Male	Yes	0	Not Graduate	No	2333	1516.0	95.0	360.0			
7	LP001014	Male	Yes	3+	Graduate	No	3036	2504.0	158.0	360.0			
8	LP001018	Male	Yes	2	Graduate	No	4006	1526.0	168.0	360.0			
9	LP001020	Male	Yes	1	Graduate	No	12841	10968.0	349.0	360.0			

```
0             1.0         Urban           Y
1             1.0         Rural           N
2             1.0         Urban           Y
3             1.0         Urban           Y
4             1.0         Urban           Y
5             1.0         Urban           Y
6             1.0         Urban           Y
7             0.0     Semiurban           N
8             1.0         Urban           Y
9             1.0     Semiurban           N
```

[4]:
```
# To know all column name
df.columns
```

[4]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

[5]:
```
df.info() #information about the dataset .weather how much column it has or the␣
 ↪null values
```

```
<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
```

2 1. *Display descriptive statistics on the dataset..*

[6]:
```
# basic descriptive stats
descriptive_stats = df.describe()
descriptive_stats
```

[6]:
```
       ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
count       614.000000         614.000000  592.000000         600.00000
mean       5403.459283        1621.245798  146.412162         342.00000
std        6109.041673        2926.248369   85.587325          65.12041
min         150.000000           0.000000    9.000000          12.00000
25%        2877.500000           0.000000  100.000000         360.00000
50%        3812.500000        1188.500000  128.000000         360.00000
75%        5795.000000        2297.250000  168.000000         360.00000
max       81000.000000       41667.000000  700.000000         480.00000

       Credit_History
count      564.000000
mean         0.842199
std          0.364878
min          0.000000
25%          1.000000
50%          1.000000
75%          1.000000
max          1.000000
```

[7]:
```
# Additional statistics on numarical columns like median, Skewness, Kurtosis,␣
 ↪Variance , Coefficient of Variation (CV)
median = df[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',␣
 ↪'Loan_Amount_Term', 'Credit_History']].median()

skewness = df[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',␣
 ↪'Loan_Amount_Term', 'Credit_History']].skew()

kurtosis = df[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',␣
 ↪'Loan_Amount_Term', 'Credit_History']].kurtosis()

variance = df[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',␣
 ↪'Loan_Amount_Term', 'Credit_History']].var()

cv = (df[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',␣
 ↪'Loan_Amount_Term', 'Credit_History']].std() / df[['ApplicantIncome',␣
 ↪'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term', 'Credit_History']].
 ↪mean()) * 100
```

[8]:
```
# print additional statistics
print("Median:")
print(median)

print("Skewness:")
print(skewness)

print("Kurtosis:")
print(kurtosis)

print("\nVariance:")
print(variance)

print("\nCoefficient of Variation (CV):")
print(cv)
```

```
Median:
ApplicantIncome      3812.5
CoapplicantIncome    1188.5
LoanAmount            128.0
Loan_Amount_Term      360.0
Credit_History          1.0
dtype: float64
Skewness:
ApplicantIncome      6.539513
CoapplicantIncome    7.491531
LoanAmount           2.677552
Loan_Amount_Term    -2.362414
Credit_History      -1.882361
dtype: float64
Kurtosis:
ApplicantIncome      60.540676
CoapplicantIncome    84.956384
LoanAmount           10.401533
Loan_Amount_Term      6.673474
Credit_History        1.548763
dtype: float64

Variance:
ApplicantIncome      3.732039e+07
CoapplicantIncome    8.562930e+06
LoanAmount           7.325190e+03
Loan_Amount_Term     4.240668e+03
Credit_History       1.331362e-01
dtype: float64

Coefficient of Variation (CV):
```

```
ApplicantIncome      113.057976
CoapplicantIncome    180.493814
LoanAmount            58.456431
Loan_Amount_Term      19.041056
Credit_History        43.324499
dtype: float64
```

3 *2. Check if any records in the data have any missing values; handle the missing data as appropriate.*

[9]:
```
# Check for missing values in each column
print("Missing values before handelling")
missing_values = df.isnull().sum()
missing_values
```

```
Missing values before handelling
```

[9]:
```
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
```

[10]:
```
# filling the missing values
# in the categorical column we used mode value  and in the numerical column we␣
 ↪used mean value
for column in df.columns:
    if df[column].dtype == 'object':
        df[column] = df[column].fillna(df[column].mode()[0])
    else:
        df[column] = df[column].fillna(df[column].mean())
```

[11]:
```
print("missing values after handelling")
df.isnull().sum()
```

```
missing values after handelling
```

[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
Loan_Status          0
dtype: int64
```

[12]:
```
# To save the clean dataset
df.to_csv("loan_data_clean.csv", index=False)
```

[13]:
```
df=pd.read_csv("loan_data_clean.csv")
df.head()
```

[13]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001002	Male	No	0	Graduate	No	5849	0.0	146.412162	360.0	1.0	Urban	Y
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.000000	360.0	1.0	Rural	N
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.000000	360.0	1.0	Urban	Y
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.000000	360.0	1.0	Urban	Y
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.000000	360.0	1.0	Urban	Y

4 *3. Build a graph visualizing the distribution of one or more individual continuous variables of the dataset.*

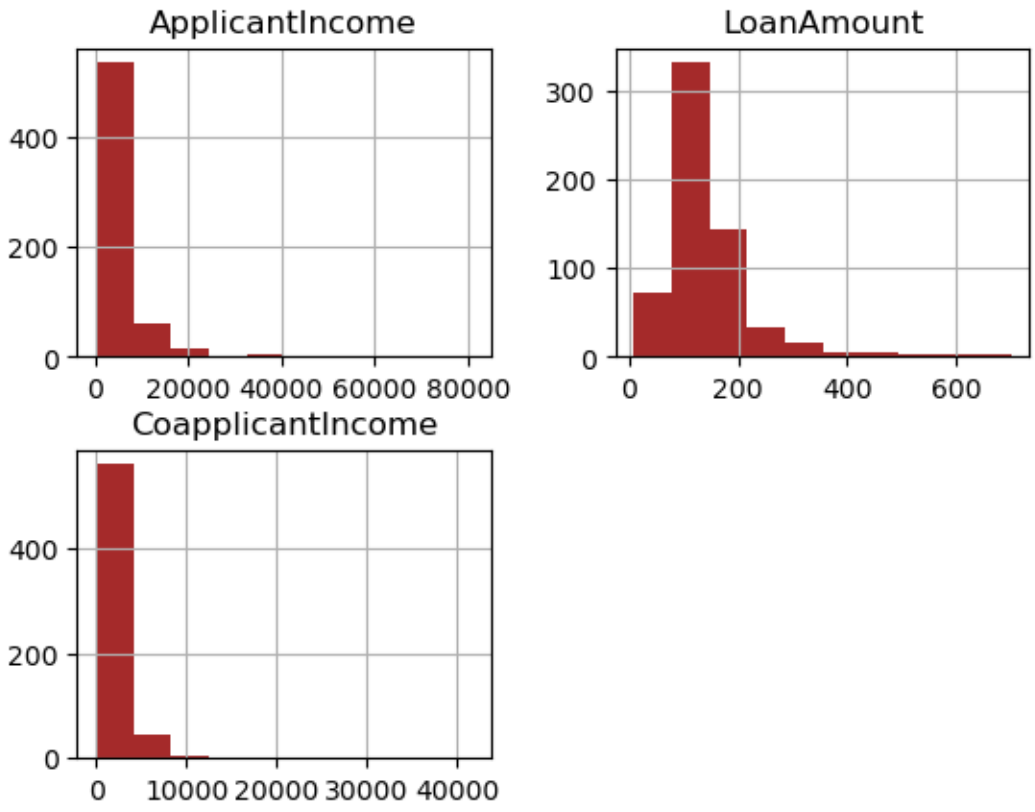
[14]:
```
# Select numerical columns
numerical_columns = df.select_dtypes(include=["float64","int64"]).columns
print(numerical_columns)
```

```
Index(['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History'],
      dtype='object')
```

Univariate Visual Analysis —-> Numerical columns

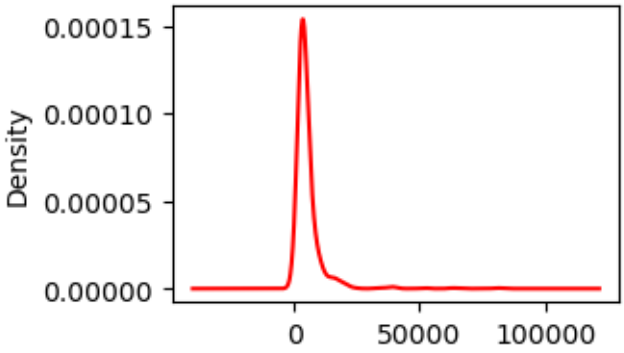
[15]:
```
df.
  ↪hist(['ApplicantIncome','LoanAmount','CoapplicantIncome'],bins=10,color='BROWN')
```

[15]:
```
array([[<Axes: title={'center': 'ApplicantIncome'}>,
        <Axes: title={'center': 'LoanAmount'}>],
       [<Axes: title={'center': 'CoapplicantIncome'}>, <Axes: >]],
      dtype=object)
```



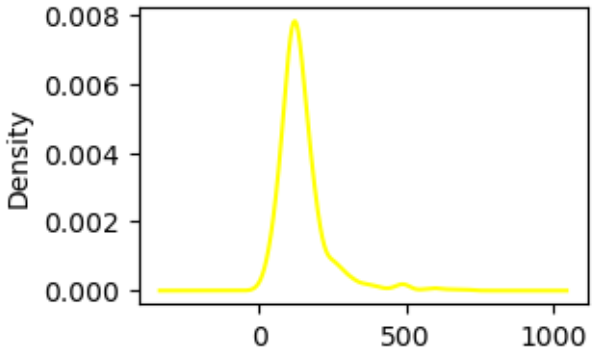
[16]:
```
df['ApplicantIncome'].plot(kind='kde',figsize=(3,2),color='RED')
print("Skewd value",df['ApplicantIncome'].skew())
```

```
Skewd value 6.539513113994625
```



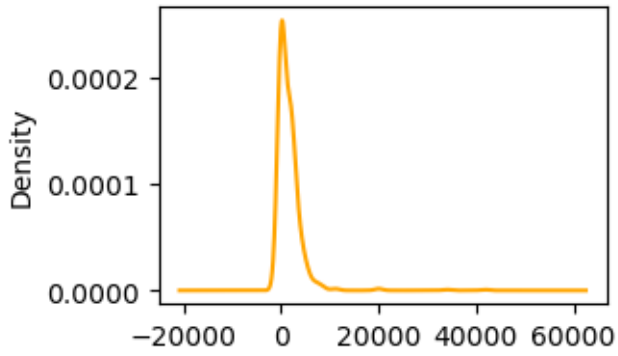
[17]:
```
df['LoanAmount'].plot(kind='kde',figsize=(3,2),color='YELLOW')
print("Skewd value",df['LoanAmount'].skew())
```

```
Skewd value 2.726601144105299
```

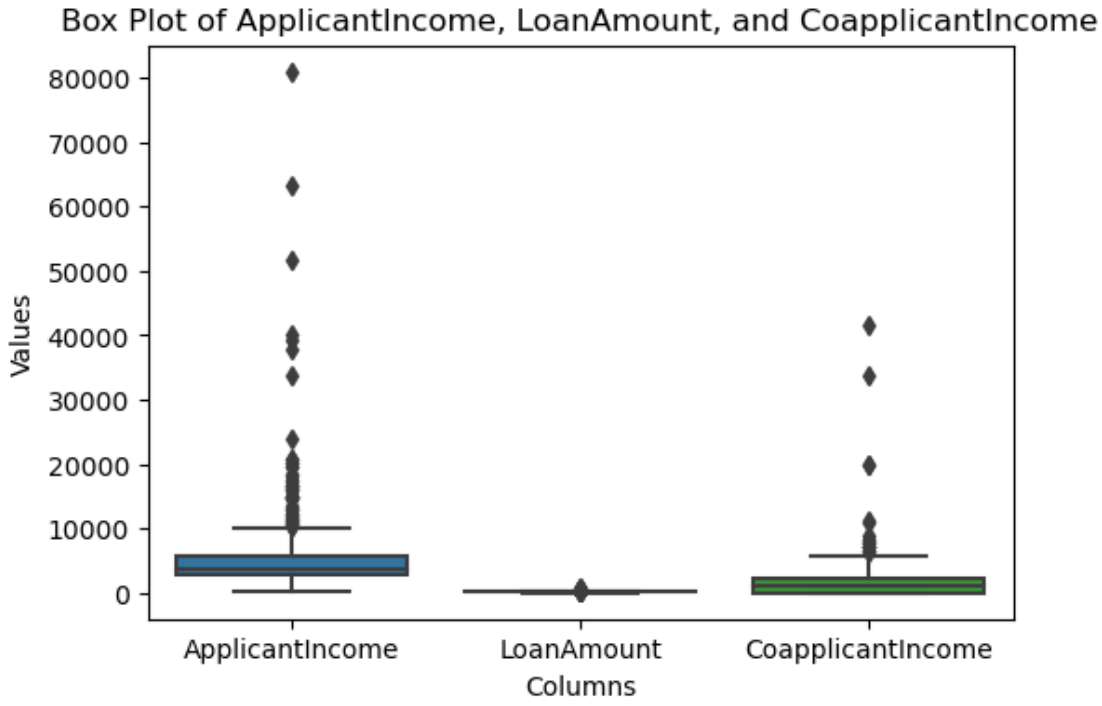


[18]:
```
df['CoapplicantIncome'].plot(kind='kde',figsize=(3,2),color='ORANGE')
print("Skewd value",df['CoapplicantIncome'].skew())
```

```
Skewd value 7.491531216657306
```



[19]:
```
columns_to_plot = ['ApplicantIncome', 'LoanAmount', 'CoapplicantIncome']

plt.figure(figsize=(6, 4))
sns.boxplot(data=df[columns_to_plot])

plt.title('Box Plot of ApplicantIncome, LoanAmount, and CoapplicantIncome')
plt.xlabel('Columns')
plt.ylabel('Values')
plt.show()
```

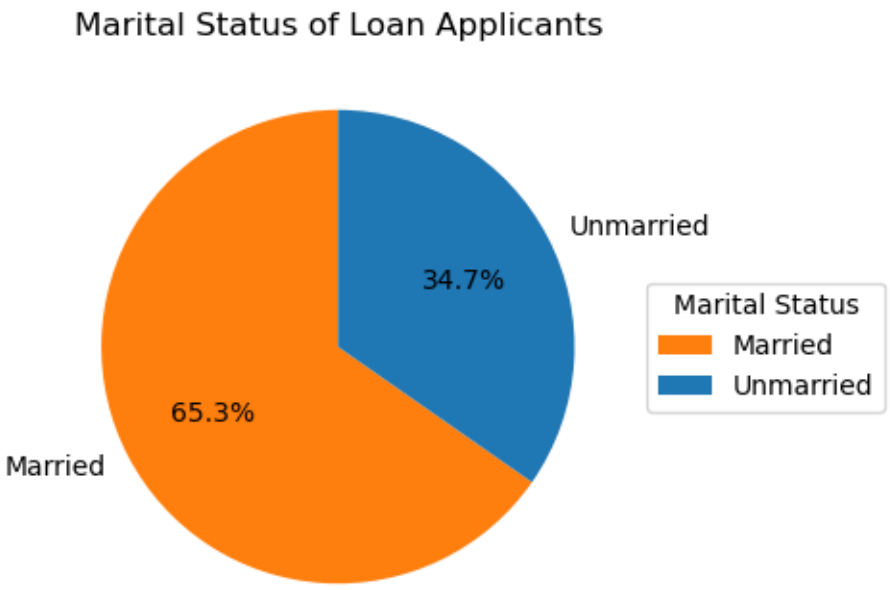


Univariate Visual Analysis —-> Categorical columns

[20]:
```
# Select categorical columns
categorical_columns = df.select_dtypes(include=["object"]).columns
print(categorical_columns)
```

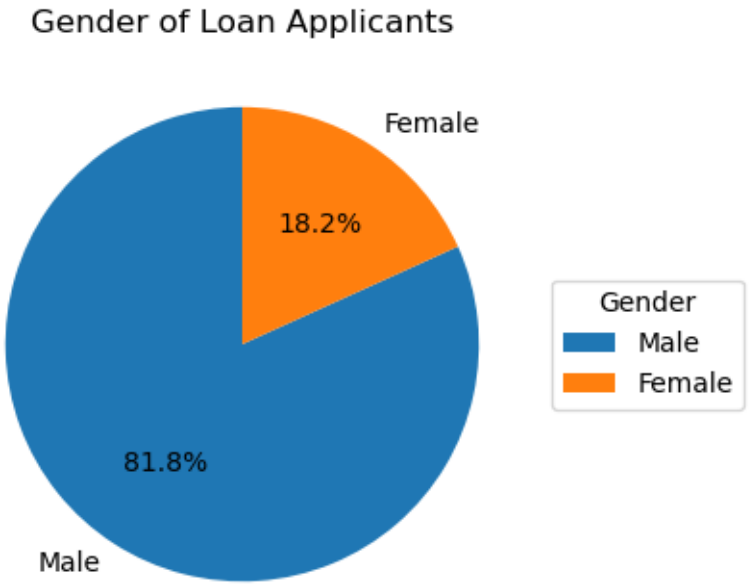
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

[21]:
```
married_count = df[df['Married'] == 'Yes'].shape[0]
unmarried_count = df[df['Married'] == 'No'].shape[0]

marital_status_labels = ['Married', 'Unmarried']
marital_status_counts = [married_count, unmarried_count]
marital_status_colors = ['#ff7f0e', '#1f77b4']

plt.figure(figsize=(4, 4))
plt.pie(marital_status_counts, labels=marital_status_labels, autopct='%1.1f%%',␣
 ↪startangle=90, colors=marital_status_colors)
plt.title('Marital Status of Loan Applicants')
plt.legend(title='Marital Status', loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```



[22]:
```
male_count = df[df['Gender'] == 'Male'].shape[0]
female_count = df[df['Gender'] == 'Female'].shape[0]
```

```
gender_labels = ['Male', 'Female']
gender_counts = [male_count, female_count]
gender_colors = ['#1f77b4', '#ff7f0e']

plt.figure(figsize=(4,4))
plt.pie(gender_counts, labels=gender_labels, autopct='%1.1f%%', startangle=90,␣
 ↪colors=gender_colors)
plt.title('Gender of Loan Applicants')
plt.legend(title='Gender', loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```

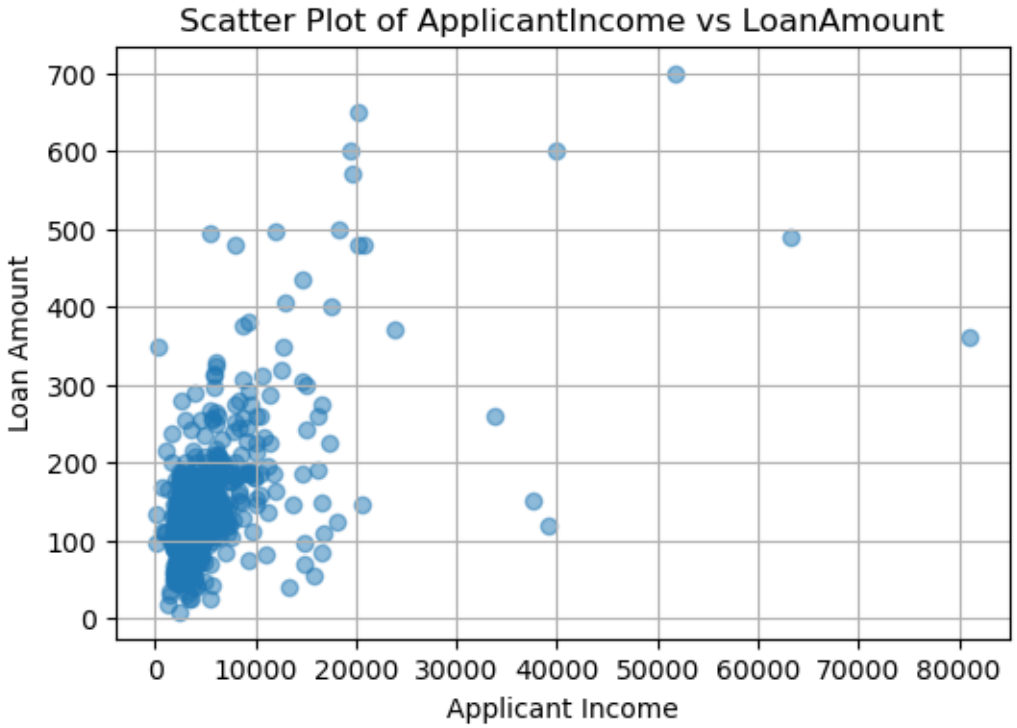


5 *4.Build a graph visualizing the relationship in a pair of continuous variables. Determine the correlation between them.*

Bivariate Visual Analysis ———> Numerical - Numerical

[23]:
```
# from above analysis we can say that applicant income and loan ammount columns␣
 ↪are continuous variables. we can build
# a graph
# Scatter plot of 'ApplicantIncome' vs 'LoanAmount'

plt.figure(figsize=(6, 4))
plt.scatter(df['ApplicantIncome'], df['LoanAmount'], alpha=0.5)
plt.title('Scatter Plot of ApplicantIncome vs LoanAmount')
```

```
plt.xlabel('Applicant Income')
plt.ylabel('Loan Amount')
plt.grid(True)
plt.show()
```



I created a scatter plot to visualize the distribution of 'ApplicantIncome' and 'LoanAmount' variables individually. Through the scatter plot, I observed the frequency distribution of income and loan amounts, identifying central tendencies and variability in each variable.

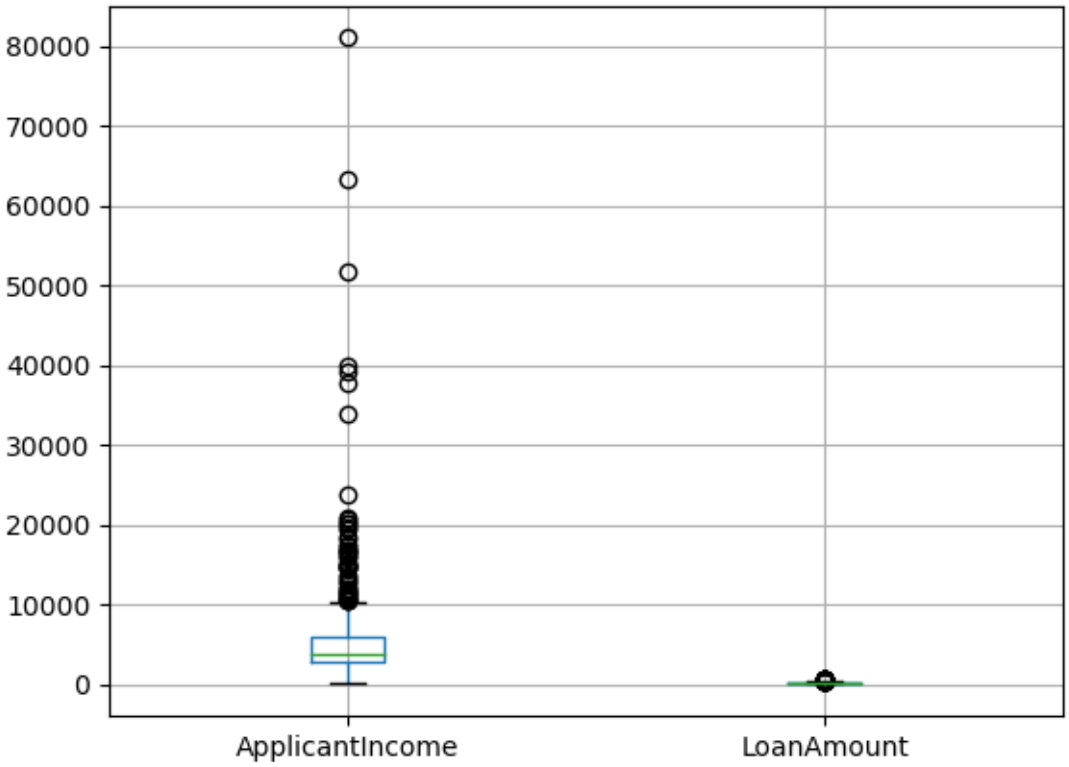
[24]:
```
# Calculate correlation coefficient between 'ApplicantIncome' and 'LoanAmount'

correlation = df.loc[:,'ApplicantIncome'].corr(df.loc[:,'LoanAmount'], method␣
 ↪='pearson',min_periods=1)

print("Correlation coefficient between 'ApplicantIncome' and 'LoanAmount':",␣
 ↪correlation)
```

```
Correlation coefficient between 'ApplicantIncome' and 'LoanAmount':
0.5656204566820273
```

[25]:
```
# from the above cell we can see that the correlation is positive
# Create a box plot between 'ApplicantIncome' and 'LoanAmount'
```

```
boxplot=df.boxplot(column=['ApplicantIncome','LoanAmount'])
plt.figure(figsize=(6,4))
```

[25]: <Figure size 600x400 with 0 Axes>



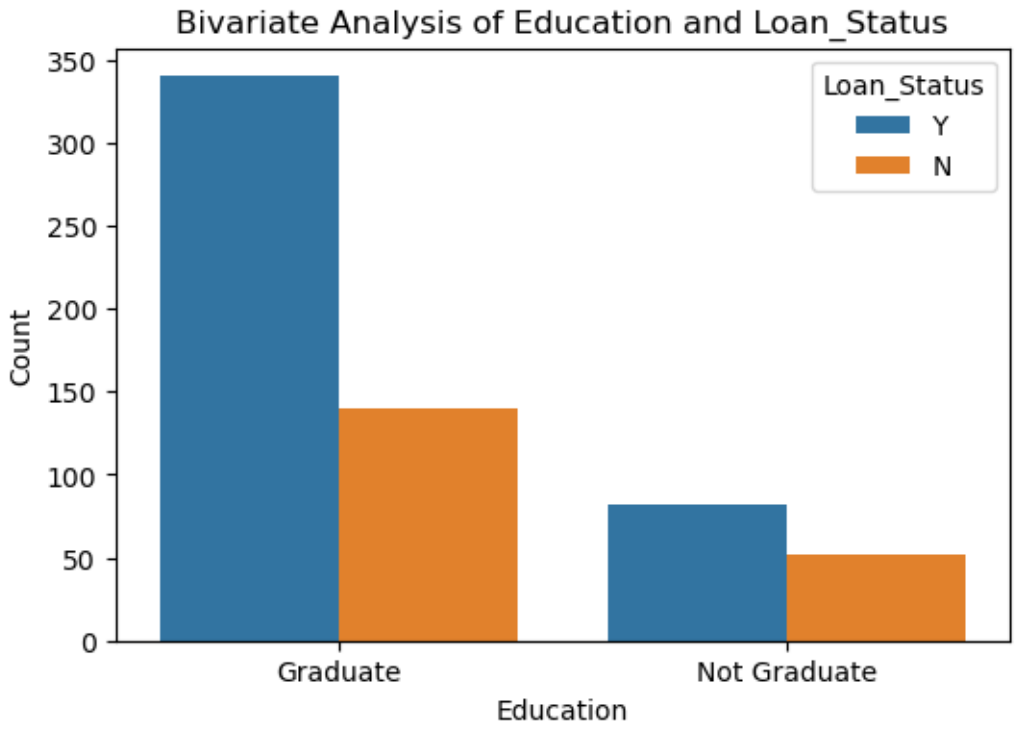
<Figure size 600x400 with 0 Axes>

I created a box plot to visualize the distribution of 'ApplicantIncome' and 'LoanAmount' variables. Through the box plot, I observed the central tendency, spread, and presence of outliers in both variables. Additionally, I determined the correlation between 'ApplicantIncome' and 'LoanAmount' to understand the strength and direction of their relationship.

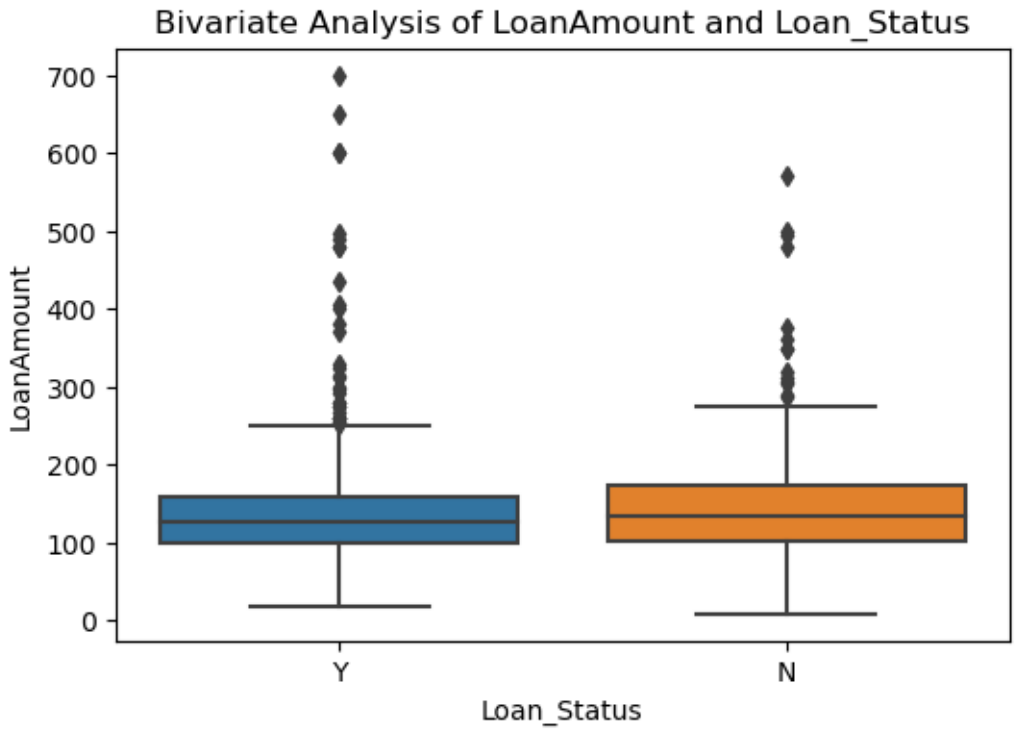
Bivariate Visual Analysis ———> Categorical - Categorical

[26]:
```
categorical_column1 = 'Education'
categorical_column2 = 'Loan_Status'
plt.figure(figsize=(6, 4))
sns.countplot(x=categorical_column1, hue=categorical_column2, data=df)

plt.title('Bivariate Analysis of ' + categorical_column1 + ' and ' +␣
 ↪categorical_column2)
```

```
plt.xlabel(categorical_column1)
plt.ylabel('Count')
plt.show()
```



[27]:
```
numerical_column = 'LoanAmount'
categorical_column = 'Loan_Status'

plt.figure(figsize=(6,4))
sns.boxplot(x=categorical_column, y=numerical_column, data=df)
plt.title('Bivariate Analysis of ' + numerical_column + ' and ' +␣
 ↪categorical_column)
plt.xlabel(categorical_column)
plt.ylabel(numerical_column)
plt.show()
```



6 *5. Display unique values of a categorical variable.*

[28]:
```
#we can see from the above column gender is categorical variable column
# Assuming df is your DataFrame
unique_genders = df['Gender'].unique()
print("Unique values of 'Gender':")
print(unique_genders)
```

```
Unique values of 'Gender':
['Male' 'Female']
```

7 *6. Build a contingency table of two potentially related categorical variables. Conduct a statistical test of the independence between them.*

[29]:
```
categorical_variables = []
for column in df.columns:
    if df[column].dtype == 'object':
        categorical_variables.append(column)
if len(categorical_variables) < 2:
    print("Not enough categorical variables found.")
```

```
else:
    print("Two categorical variables found:", categorical_variables[:2])
```

```
Two categorical variables found: ['Loan_ID', 'Gender']
```

[30]:
```
from scipy.stats import chi2_contingency

categorical_var1 = 'Gender'
categorical_var2 = 'Loan_Status'

# Build a contingency table
contingency_table = pd.crosstab(df[categorical_var1], df[categorical_var2])
print("Contingency Table:")
print(contingency_table)

# Conduct a statistical test of independence
chi2_stat, p_val, dof, expected = chi2_contingency(contingency_table)
print("\nChi-square Statistic:", chi2_stat)
print("P-value:", p_val)
print("Degrees of Freedom:", dof)
print("Expected Frequencies Table:")
print(expected)
```

```
Contingency Table:
Loan_Status    N    Y
Gender
Female        37   75
Male         155  347

Chi-square Statistic: 0.11087854691241235
P-value: 0.7391461310869638
Degrees of Freedom: 1
Expected Frequencies Table:
[[ 35.0228013  76.9771987]
 [156.9771987 345.0228013]]
```

The chi-square test of independence evaluates the null hypothesis that the categorical variables are independent of each other. If the p-value is less than a chosen significance level (e.g., 0.05), we reject the null hypothesis and conclude that there is a significant association between the variables. Otherwise, we fail to reject the null hypothesis, indicating that there is no significant association.

8 *7. Retrieve one or more subset of rows based on two or more criteria and present descriptive statistics on the subset(s).*

[31]:
```
df_row2=df.iloc[1:10,[4,8]]
df_row2.describe()
```

[31]:
```
       LoanAmount
count    9.000000
mean   165.777778
std     88.729614
min     66.000000
25%    120.000000
50%    141.000000
75%    168.000000
max    349.000000
```

9 *8.Conduct a statistical test of the significance of the difference between the means of two subsets of the data.*

10 9.Create one or more tables that group the data by a certain categorical variable and displays summarized information for each group (e.g. the mean or sum within the group).

[32]:
```
# convert categorical column to numerical
df=pd.get_dummies(df,columns=['Gender'],drop_first=True)
```

[33]:
```
boolean_columns = ['Gender_Male']

# Convert selected boolean columns to integers
df[boolean_columns] = df[boolean_columns].astype(int)
```

[34]:
```
df.head()
```

[34]:

	Loan_ID	Married	Dependents	Education	Self_Employed	ApplicantIncome	\
0	LP001002	No	0	Graduate	No	5849	
1	LP001003	Yes	1	Graduate	No	4583	
2	LP001005	Yes	0	Graduate	Yes	3000	
3	LP001006	Yes	0	Not Graduate	No	2583	
4	LP001008	No	0	Graduate	No	6000	

	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	\
0	0.0	146.412162	360.0	1.0	
1	1508.0	128.000000	360.0	1.0	
2	0.0	66.000000	360.0	1.0	

```
3             2358.0  120.000000             360.0             1.0
4                0.0  141.000000             360.0             1.0

  Property_Area Loan_Status  Gender_Male
0         Urban           Y            1
1         Rural           N            1
2         Urban           Y            1
3         Urban           Y            1
4         Urban           Y            1
```

[35]:
```
loan_subset=df[['Gender_Male','ApplicantIncome']]
loan_subset.head()
```

[35]:
```
   Gender_Male  ApplicantIncome
0            1             5849
1            1             4583
2            1             3000
3            1             2583
4            1             6000
```

[36]:
```
loan_subset.describe()
```

[36]:
```
       Gender_Male  ApplicantIncome
count   614.000000       614.000000
mean      0.817590      5403.459283
std       0.386497      6109.041673
min       0.000000       150.000000
25%       1.000000      2877.500000
50%       1.000000      3812.500000
75%       1.000000      5795.000000
max       1.000000     81000.000000
```

[37]:
```
from scipy.stats import ttest_ind

# Define subsets based on gender
subset_male = loan_subset[loan_subset['Gender_Male'] ==1 ]['ApplicantIncome']
subset_female = loan_subset[loan_subset['Gender_Male'] ==0]['ApplicantIncome']

# Perform independent samples t-test
t_statistic, p_value = ttest_ind(subset_male, subset_female, equal_var=False) ␣
 ↪# assuming unequal variances

# Print the results
print("T-statistic:", t_statistic)
print("P-value:", p_value)

# Determine the significance
```

```
alpha = 0.05  # significance level
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant difference␣
 ↪between the means of the two gender groups.")
else:
    print("Fail to reject the null hypothesis: There is no significant␣
 ↪difference between the means of the two gender groups.")
```

```
T-statistic: 2.0798181188614193
P-value: 0.03839165635056259
Reject the null hypothesis: There is a significant difference between the means
of the two gender groups.
```

[38]:
```
print(df.columns)
```

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

[39]:
```
df=pd.get_dummies(df,columns=['Dependents','Education','Loan_Status',
'Married','Property_Area',
'Self_Employed'],drop_first=True)
```

[40]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 16 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Loan_ID                  614 non-null    object
 1   ApplicantIncome          614 non-null    int64
 2   CoapplicantIncome        614 non-null    float64
 3   LoanAmount               614 non-null    float64
 4   Loan_Amount_Term         614 non-null    float64
 5   Credit_History           614 non-null    float64
 6   Gender_Male              614 non-null    int32
 7   Dependents_1             614 non-null    bool
 8   Dependents_2             614 non-null    bool
 9   Dependents_3+            614 non-null    bool
 10  Education_Not Graduate   614 non-null    bool
 11  Loan_Status_Y            614 non-null    bool
 12  Married_Yes              614 non-null    bool
 13  Property_Area_Semiurban  614 non-null    bool
 14  Property_Area_Urban      614 non-null    bool
 15  Self_Employed_Yes        614 non-null    bool
```

```
dtypes: bool(9), float64(4), int32(1), int64(1), object(1)
memory usage: 36.7+ KB
```

[41]:
```
# Select boolean columns to convert
boolean_columns = ['Gender_Male', 'Married_Yes',
       'Dependents_1', 'Dependents_2', 'Dependents_3+',
       'Education_Not Graduate', 'Self_Employed_Yes',
       'Property_Area_Semiurban', 'Property_Area_Urban', 'Loan_Status_Y']

# Convert selected boolean columns to integers
df[boolean_columns] = df[boolean_columns].astype(int)

# Check the updated DataFrame
print(df.head())
```

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

   Credit_History  Gender_Male  Dependents_1  Dependents_2  Dependents_3+  \
0             1.0            1             0             0              0
1             1.0            1             1             0              0
2             1.0            1             0             0              0
3             1.0            1             0             0              0
4             1.0            1             0             0              0

   Education_Not Graduate  Loan_Status_Y  Married_Yes  \
0                       0              1            0
1                       0              0            1
2                       0              1            1
3                       1              1            1
4                       0              1            0

   Property_Area_Semiurban  Property_Area_Urban  Self_Employed_Yes
0                        0                    1                  0
1                        0                    0                  0
2                        0                    1                  1
3                        0                    1                  0
4                        0                    1                  0
```

[42]:
```
df=df.drop('Loan_ID',axis=1)
```

[43]:
```
columns_to_drop = ['Dependents_1', 'Dependents_2', 'Dependents_3+']
df = df.drop(columns_to_drop, axis=1)
```

[44]:
```
df.info() #making sure that all columns are in int type and doesn't have null␣
↪values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 12 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   ApplicantIncome          614 non-null    int64
 1   CoapplicantIncome        614 non-null    float64
 2   LoanAmount               614 non-null    float64
 3   Loan_Amount_Term         614 non-null    float64
 4   Credit_History           614 non-null    float64
 5   Gender_Male              614 non-null    int32
 6   Education_Not Graduate   614 non-null    int32
 7   Loan_Status_Y            614 non-null    int32
 8   Married_Yes              614 non-null    int32
 9   Property_Area_Semiurban  614 non-null    int32
 10  Property_Area_Urban      614 non-null    int32
 11  Self_Employed_Yes        614 non-null    int32
dtypes: float64(4), int32(7), int64(1)
memory usage: 40.9 KB
```

11 *10. Implement a linear regression model and interpret its output.*

[45]:
```
# Import necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
```

[46]:
```
x=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

[47]:
```
#split the data into traing and testing set
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.
↪05,random_state=0)
```

[48]:
```
model=LogisticRegression()
model.fit(x_train,y_train)
```

[48]:
```
LogisticRegression()
```

[49]:
```
y_pred=model.predict(x_test)
```

[51]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score
# Calculate accuracy, precision, and recall scores
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the scores
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
```

```
Accuracy: 0.8064516129032258
Precision: 0.0
Recall: 0.0
```

```
C:\Users\subashis\anaconda3\Lib\site-
packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[ ]:

[ ]: