dspl-assignment-3

February 5, 2025

[51]:
```
import pandas as pd
data = pd.read_csv('loan.csv')
columns = data.dtypes
print(columns)
```

```
Loan_ID               object
Gender                object
Married               object
Dependents            object
Education             object
Self_Employed         object
ApplicantIncome        int64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area         object
Loan_Status           object
dtype: object
```

1. Identify the most freuent values for all categorical features

[52]:
```
categorical_columns = data.select_dtypes(include=['object', 'category']).columns
print(categorical_columns)
```

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

[53]:
```
most_frequent_values = data[categorical_columns].mode().iloc[0]
print(most_frequent_values)
```

```
Loan_ID           LP001002
Gender                Male
Married                Yes
Dependents               0
Education         Graduate
Self_Employed           No
Property_Area    Semiurban
```

```
Loan_Status                 Y
Name: 0, dtype: object
```

2. Give descriptive statistics of numerical features in the dataset. Comment about the distribution of data from it.

[54]:
```
data.describe()
```

[54]:
```
       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
```

3. Replace the missing values in categorial features using appropriate techniques.

[55]:
```
data.describe(include='O')
```

[55]:
```
         Loan_ID Gender Married Dependents Education Self_Employed  \
count        614    601     611        599       614           582
unique       614      2       2          4         2             2
top     LP001002   Male     Yes          0  Graduate            No
freq           1    489     398        345       480           500

       Property_Area Loan_Status
count            614         614
unique             3           2
top        Semiurban           Y
freq             233         422
```

[56]:
```
data['Dependents'].unique()
```

[56]:
```
array(['0', '1', '2', '3+', nan], dtype=object)
```

[57]:
```
data['Dependents'] = data['Dependents'].fillna(data['Dependents'].mode()[0])
data['Dependents'].unique()
```

[57]:
```
array(['0', '1', '2', '3+'], dtype=object)
```

4. Demonstrate various encoding techniques for categorical features

Label Encoding

[58]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()

# Apply label encoding
data['LoanID_labelEncoder'] = label_encoder.fit_transform(data['Loan_ID'])
print(data['LoanID_labelEncoder'])
```

```
0        0
1        1
2        2
3        3
4        4
      …
609    609
610    610
611    611
612    612
613    613
Name: LoanID_labelEncoder, Length: 614, dtype: int32
```

[59]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()

# Apply label encoding
data['LoanID_labelEncoder'] = label_encoder.fit_transform(data['Loan_ID'])
print(data['LoanID_labelEncoder'])
```

```
0        0
1        1
2        2
3        3
4        4
      …
609    609
610    610
611    611
612    612
```

```
613    613
Name: LoanID_labelEncoder, Length: 614, dtype: int32
```

One-Hot Encoding

[60]:
```python
one_hot_encoded_gender = pd.get_dummies(data['Gender'], prefix='Gender')
print(one_hot_encoded_gender)
```

```
     Gender_Female  Gender_Male
0            False         True
1            False         True
2            False         True
3            False         True
4            False         True
..             …            …
609           True        False
610          False         True
611          False         True
612          False         True
613           True        False

[614 rows x 2 columns]
```

5. for numerical features, replace missing values using

a. using simple imputer (mean, median)

[61]:
```python
data.describe()
```

[61]:
```
       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  LoanID_labelEncoder
count      564.000000           614.000000
mean         0.842199           306.500000
std          0.364878           177.390811
min          0.000000             0.000000
25%          1.000000           153.250000
50%          1.000000           306.500000
75%          1.000000           459.750000
max          1.000000           613.000000
```

[62]:
```
data['LoanAmount_mean'] = data['LoanAmount'].fillna(data['LoanAmount'].mean())
print(data[['LoanAmount_mean','LoanAmount']])
```

```
     LoanAmount_mean  LoanAmount
0         146.412162         NaN
1         128.000000       128.0
2          66.000000        66.0
3         120.000000       120.0
4         141.000000       141.0
..               …           …
609        71.000000        71.0
610        40.000000        40.0
611       253.000000       253.0
612       187.000000       187.0
613       133.000000       133.0

[614 rows x 2 columns]
```

[63]:
```
data['Loan_Amount_Term_median'] = data[        'Loan_Amount_Term'].
 ↪fillna(data['Loan_Amount_Term'].median())
print(data[['Loan_Amount_Term_median','Loan_Amount_Term']])
```

```
     Loan_Amount_Term_median  Loan_Amount_Term
0                      360.0             360.0
1                      360.0             360.0
2                      360.0             360.0
3                      360.0             360.0
4                      360.0             360.0
..                       …                 …
609                    360.0             360.0
610                    180.0             180.0
611                    360.0             360.0
612                    360.0             360.0
613                    360.0             360.0

[614 rows x 2 columns]
```

b. using random sample imputation

[64]:
```
import numpy as np

def random_sample_imputation(data, column_name):

    before_imputation = data[column_name].isna().sum()

    non_missing_values = data[column_name].dropna()
```

```
    random_sample = np.random.choice(non_missing_values, size=data[column_name].
↪isna().sum())


    data[column_name].loc[data[column_name].isna()] = random_sample


    after_imputation = data[column_name].isna().sum()

    print(f"Before imputation, null values: {before_imputation}")
    print(f"After imputation, null values: {after_imputation}")

    return data

data = random_sample_imputation(data, 'Credit_History')


print(data[data['Credit_History'].isna()])
```

```
Before imputation, null values: 50
After imputation, null values: 0
Empty DataFrame
Columns: [Loan_ID, Gender, Married, Dependents, Education, Self_Employed,
ApplicantIncome, CoapplicantIncome, LoanAmount, Loan_Amount_Term,
Credit_History, Property_Area, Loan_Status, LoanID_labelEncoder,
LoanAmount_mean, Loan_Amount_Term_median]
Index: []
```

```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_9748\862060778.py:13:
FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in
certain cases, but when using Copy-on-Write (which will become the default
behaviour in pandas 3.0) this will never work to update the original DataFrame
or Series, because the intermediate object on which we are setting values will
behave as a copy.
A typical example is when you are setting values in a column of a DataFrame,
like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in
a single step and ensure this keeps updating the original `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

  data[column_name].loc[data[column_name].isna()] = random_sample
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_9748\862060778.py:13:
```

```
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data[column_name].loc[data[column_name].isna()] = random_sample
```

6. Give descriptive statistics of numerical features in the dataset afterhandling missing values.Comment about the destribution od data from it.

[65]:
```
data.describe()
```

[65]:

	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	LoanID_labelEncoder	LoanAmount_mean
count	614.00000	614.000000	614.000000
mean	0.84202	306.500000	146.412162
std	0.36502	177.390811	84.037468
min	0.00000	0.000000	9.000000
25%	1.00000	153.250000	100.250000
50%	1.00000	306.500000	129.000000
75%	1.00000	459.750000	164.750000
max	1.00000	613.000000	700.000000

	Loan_Amount_Term_median
count	614.000000
mean	342.410423
std	64.428629
min	12.000000
25%	360.000000
50%	360.000000
75%	360.000000
max	480.000000

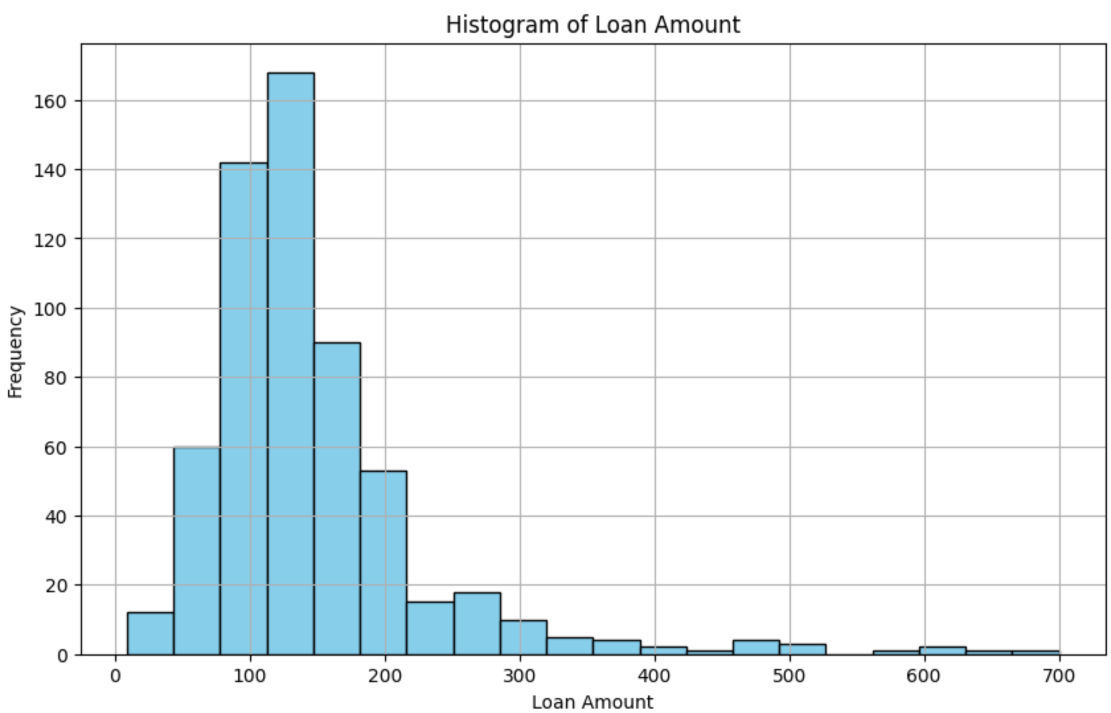
After replacing the null values in Credit_History using random sample imputation, the mean value became closer to the median value compared to the previous mean and median values.

7. Plot following graphs. Label X and Y axis, give appropriate title to the graph.

a. Plot histogram for Loan Amount and mention ur observations

[66]:
```
import matplotlib.pyplot as plt


plt.figure(figsize=(10, 6))
plt.hist(data['LoanAmount'].dropna(), bins=20, color='skyblue',␣
 ↪edgecolor='black')
plt.title('Histogram of Loan Amount')
plt.xlabel('Loan Amount')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```



This graph shows a negative skew because the mean is less than the median.

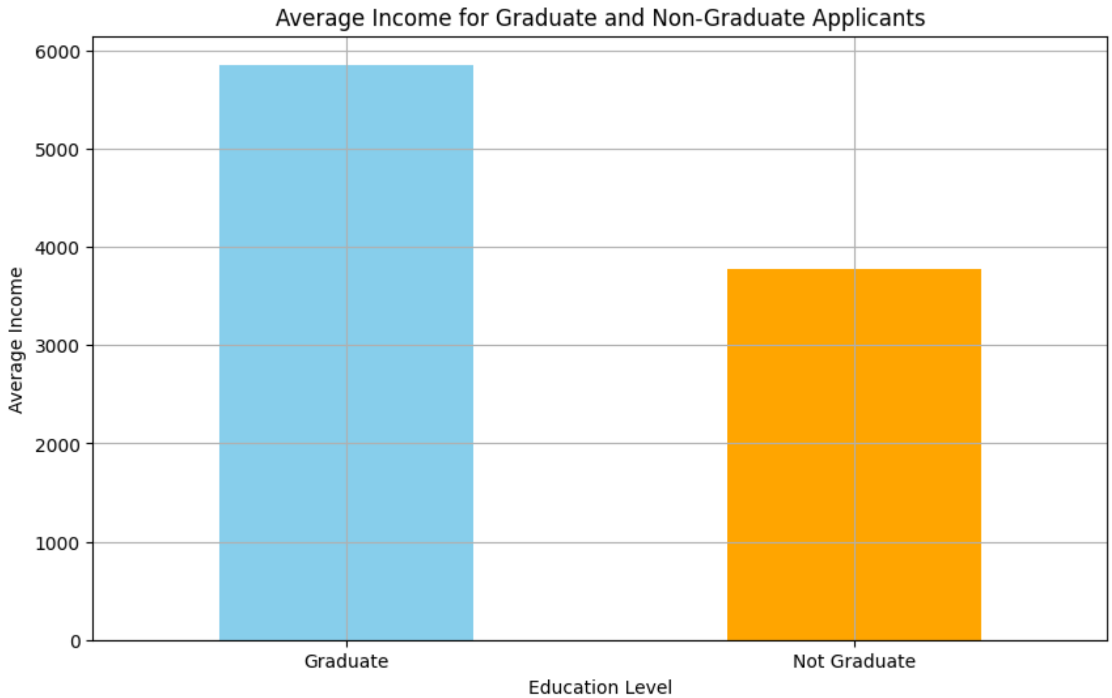
b. plot bar graph showing income for graduate and non-graduate applicant and mention ur observations

[67]:
```
import matplotlib.pyplot as plt

income_by_education = data.groupby('Education')['ApplicantIncome'].mean()


plt.figure(figsize=(10, 6))
income_by_education.plot(kind='bar', color=['skyblue', 'orange'])
plt.title('Average Income for Graduate and Non-Graduate Applicants')
```

```
plt.xlabel('Education Level')
plt.ylabel('Average Income')
plt.xticks(rotation=0)
plt.grid(True)
plt.show()
```



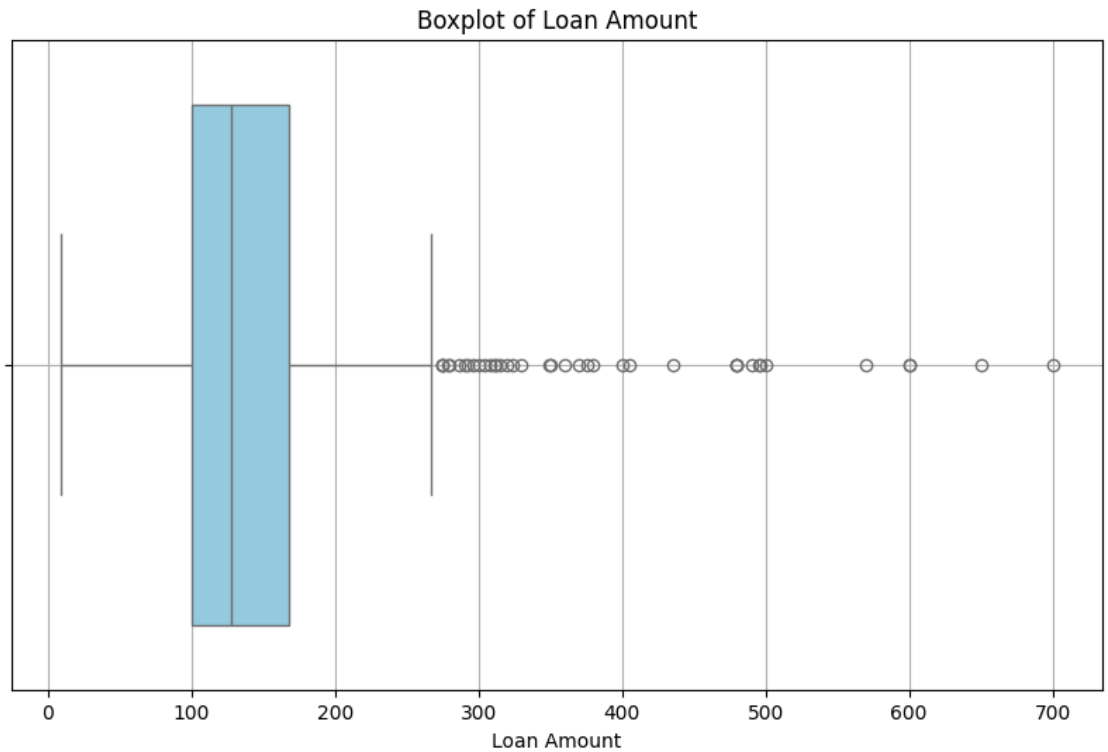
The bar heights will show whether graduates tend to earn more than non-graduates, which is a typical observation in many datasets.

c. Plot the boxplot for Loan amount. Give the five value summary from it.

[68]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Plotting the boxplot for LoanAmount column
plt.figure(figsize=(10, 6))
sns.boxplot(x=data['LoanAmount'].dropna(), color='skyblue')
plt.title('Boxplot of Loan Amount')
plt.xlabel('Loan Amount')
plt.grid(True)
plt.show()

# Five-number summary
five_number_summary = data['LoanAmount'].describe(percentiles=[.25, .5, .75])
print(five_number_summary)
```




```
count    592.000000
mean     146.412162
std       85.587325
min        9.000000
25%      100.000000
50%      128.000000
75%      168.000000
max      700.000000
Name: LoanAmount, dtype: float64
```

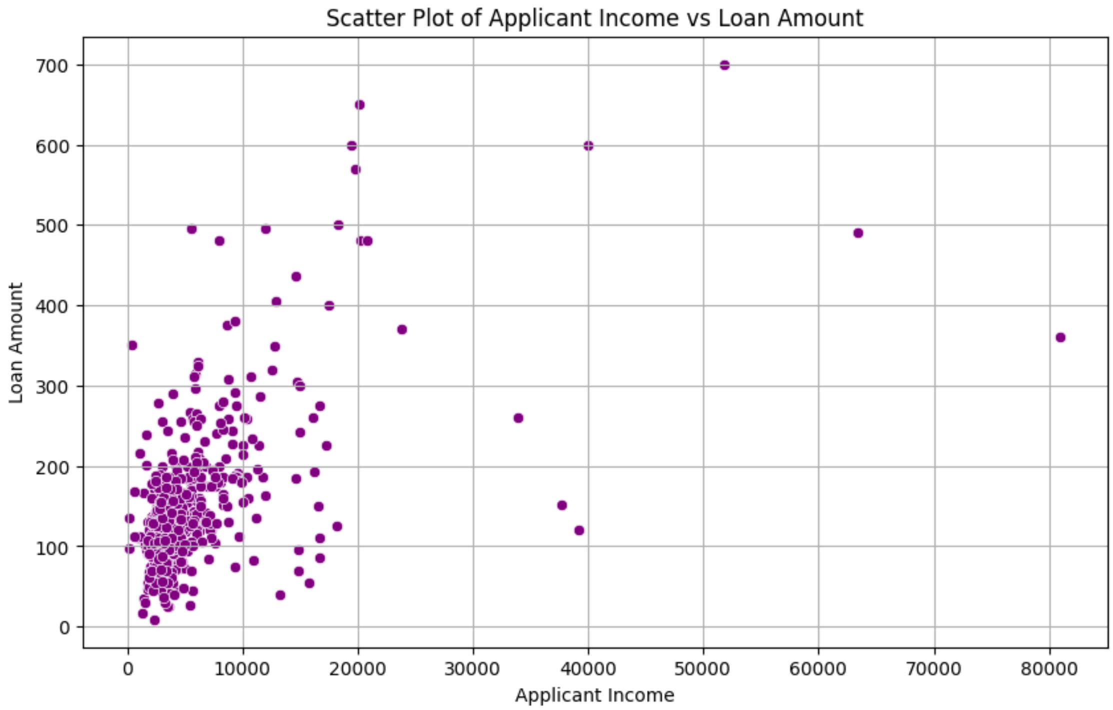
d. Comment on the correlation between Applicant's income and Loan amount using appropriate graph.

[69]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Plotting a scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x=data['ApplicantIncome'], y=data['LoanAmount'], color='purple')
plt.title('Scatter Plot of Applicant Income vs Loan Amount')
plt.xlabel('Applicant Income')
plt.ylabel('Loan Amount')
plt.grid(True)
```

```
plt.show()

# Calculate the correlation coefficient
correlation = data[['ApplicantIncome', 'LoanAmount']].corr().iloc[0, 1]
print(f"Correlation between Applicant Income and Loan Amount: {correlation}")
```




```
Correlation between Applicant Income and Loan Amount: 0.5709090389885663
```

e. Give descriptive statistics of numerical features in the dataset. Comment about the distribution of data from it.

[70]:
```
data.describe()
```

[70]:

	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	LoanID_labelEncoder	LoanAmount_mean	\
count	614.00000	614.000000	614.000000	

```
mean            0.84202          306.500000       146.412162
std             0.36502          177.390811        84.037468
min             0.00000            0.000000         9.000000
25%             1.00000          153.250000       100.250000
50%             1.00000          306.500000       129.000000
75%             1.00000          459.750000       164.750000
max             1.00000          613.000000       700.000000

       Loan_Amount_Term_median
count               614.000000
mean                342.410423
std                  64.428629
min                  12.000000
25%                 360.000000
50%                 360.000000
75%                 360.000000
max                 480.000000
```