Loan Eligibility Analysis and Prediction

1. Loading required libraries and dataset

(a) Importing libraries such as pandas and matplotlib to manipulate and visualize dataset and its features.

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

(b) Reading dataset from directory

In [3]:
```
loan_train = pd.read_csv('dataset/loan-train.csv')
loan_test = pd.read_csv('dataset/loan-test.csv')
```

Dataset Description

In [4]:
```
loan_train
```

Out[4]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	360.0	1.0
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	180.0	1.0
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	1.0
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	1.0
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	360.0	0.0

614 rows × 13 columns

Attributes and their description:

Index	Attribute / Column	Description
1.	Loan_ID	Unique Loan ID
2	Gender	Male/ Female
3.	Married	Applicant married (Y/N)
4.	Dependents	Number of dependents the applicant has
5.	Education	Applicant Education (Graduate/ Under Graduate)
6.	Self_Employed	Is the applicant self-employed (Y/N)
7.	ApplicantIncome	Applicant income
8.	CoapplicantIncome	Co-applicant income
9.	LoanAmount	Loan amount in thousands
10.	Loan_Amount_Term	Term of a loan in months
11.	Credit_History	credit history meets guidelines (0/1)
12.	Property_Area	Urban/ Semi-Urban/ Rural
13.	Loan_Status	Loan approved (Y/N) (Target variable)

In [5]:
```
print("Rows: ", len(loan_train))
print("Columns: ", len(loan_train.columns))
print("Shape : ", loan_train.shape)
```

```
Rows:  614
Columns:  13
Shape :  (614, 13)
```

In [6]:
```
loan_train.describe()
```

Out[6]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

2. Data Analysis and Cleaning

(a) Analyzing each column for unique values

In [8]:
```
def dataset_value_counts():
    for column in loan_train.columns:
        if loan_train[column].dtype == 'object':
            print('Unique values in {column} and their counts are: \n'.format(column = column), loan_train[column].value_counts()
            print('\n')
```

This function will iterate through each categorical feature and print all the unique values in that column along with its value count.

In [9]:
```
dataset_value_counts()
```

```
Unique values in Loan_ID and their counts are:
 Loan_ID
LP001002    1
LP002328    1
LP002305    1
LP002308    1
LP002314    1
           ..
LP001692    1
LP001693    1
LP001698    1
LP001699    1
LP002990    1
Name: count, Length: 614, dtype: int64


Unique values in Gender and their counts are:
 Gender
Male      489
Female    112
Name: count, dtype: int64


Unique values in Married and their counts are:
 Married
Yes    398
No     213
Name: count, dtype: int64
```

(b) Checking for Null values

In [13]:
```
loan_train[loan_train.isna().any(axis = 1)]
```

Out[13]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0
11	LP001027	Male	Yes	2	Graduate	NaN	2500	1840.0	109.0	360.0	1.0
16	LP001034	Male	No	1	Not Graduate	No	3596	0.0	100.0	240.0	NaN
19	LP001041	Male	Yes	0	Graduate	NaN	2600	3500.0	115.0	NaN	1.0
23	LP001050	NaN	Yes	2	Not Graduate	No	3365	1917.0	112.0	360.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...
592	LP002933	NaN	No	3+	Graduate	Yes	9357	0.0	292.0	360.0	1.0
597	LP002943	Male	No	NaN	Graduate	No	2987	0.0	88.0	360.0	0.0
600	LP002949	Female	No	3+	Graduate	NaN	416	41667.0	350.0	180.0	NaN
601	LP002950	Male	Yes	0	Not Graduate	NaN	2894	2792.0	155.0	360.0	1.0
605	LP002960	Male	Yes	0	Not Graduate	No	2400	3800.0	NaN	180.0	1.0

134 rows × 13 columns

There are 134 rows with at least one nan (null) value.

In [14]:
```
loan_train.isna().sum()
```

Out[14]:
```
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
```

13 rows in Gender column have nan and so on.

(c) Dealing with null columns

Categorical Columns with null:

1. Gender
2. Married
3. Dependents
4. Education
5. Self Employed
6. Credit History

Null values in categorical (non – quantitative) columns can be replaced with the most frequently occurring value of that particular column.

In [15]:
```
loan_train['Gender'] = loan_train['Gender'].fillna(loan_train['Gender'].mode()[0])
loan_test['Gender'] = loan_test['Gender'].fillna(loan_test['Gender'].mode()[0])

loan_train['Dependents'] = loan_train['Dependents'].fillna(loan_train['Dependents'].mode()[0])
loan_test['Dependents'] = loan_test['Dependents'].fillna(loan_test['Dependents'].mode()[0])

loan_train['Married'] = loan_train['Married'].fillna(loan_train['Married'].mode()[0])
loan_test['Married'] = loan_test['Married'].fillna(loan_test['Married'].mode()[0])

loan_train['Self_Employed'] = loan_train['Self_Employed'].fillna(loan_train['Self_Employed'].mode()[0])
loan_test['Self_Employed'] = loan_test['Self_Employed'].fillna(loan_test['Self_Employed'].mode()[0])

loan_train['Credit_History'] = loan_train['Credit_History'].fillna(loan_train['Credit_History'].mode()[0])
loan_test['Credit_History'] = loan_test['Credit_History'].fillna(loan_test['Credit_History'].mode()[0])
```

In [20]:
```
loan_train[loan_train.isna().any(axis = 1)].shape
```

Out[20]:
```
(36, 13)
```

This has decreased the rows with nan to 36.

Quantitative Columns with null:

1. Loan Amount
2. Loan Amount Term

Null values in quantitative columns can be replaced with the most mean occurring value of that particular column.

In [6]:
```
loan_train['LoanAmount'] = loan_train['LoanAmount'].fillna(loan_train['LoanAmount'].mean())
loan_test['LoanAmount'] = loan_test['LoanAmount'].fillna(loan_train['LoanAmount'].mean())

loan_train['Loan_Amount_Term'] = loan_train['Loan_Amount_Term'].fillna(loan_train['Loan_Amount_Term'].mean())
loan_test['Loan_Amount_Term'] = loan_test['Loan_Amount_Term'].fillna(loan_train['Loan_Amount_Term'].mean())
```

In [7]:
```
loan_train[loan_train.isna().any(axis = 1)]
```

Out[7]:

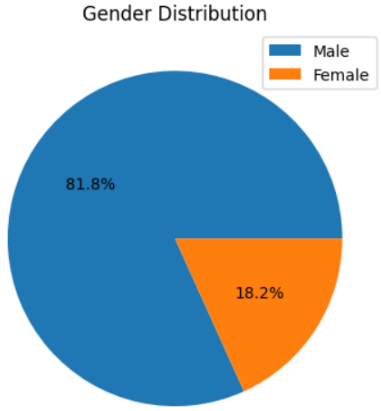
Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Pr

All nan rows has been filled with relevant values.

3. Data Visualization

(a) Distribution of gender in dataset.

```
In [9]: gender_data = loan_train.Gender.value_counts()
        fig, ax = plt.subplots()
        ax.pie(gender_data  , autopct='%1.1f%%')
        plt.title('Gender Distribution')
        plt.legend(['Male', 'Female'])
        plt.show()
        plt.close()
```

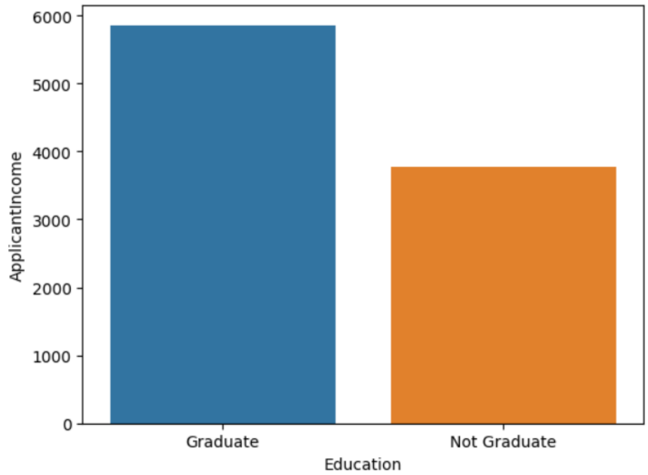


Over 80% of loan submissions were from males.

(b) Does a higher education correlate with higher mean income?

```
In [10]: Educationincome = loan_train.ApplicantIncome.groupby(loan_train['Education']).mean().reset_index()
```

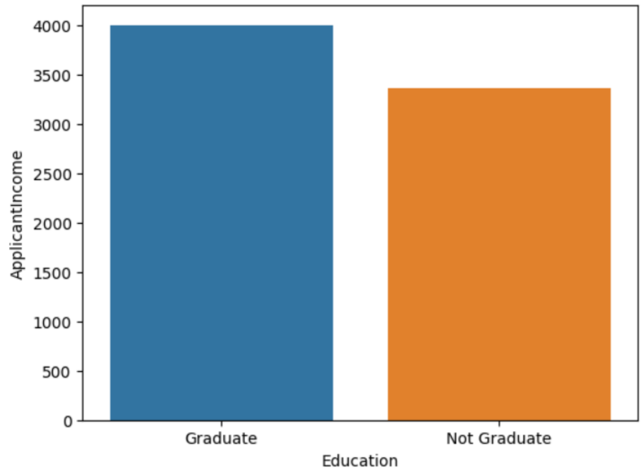
```
In [12]: sns.barplot(data = Educationincome, x = 'Education', y = 'ApplicantIncome', hue="Education")
         plt.show()
         plt.close()
```



A graduate on average has higher income than a non-graduate.

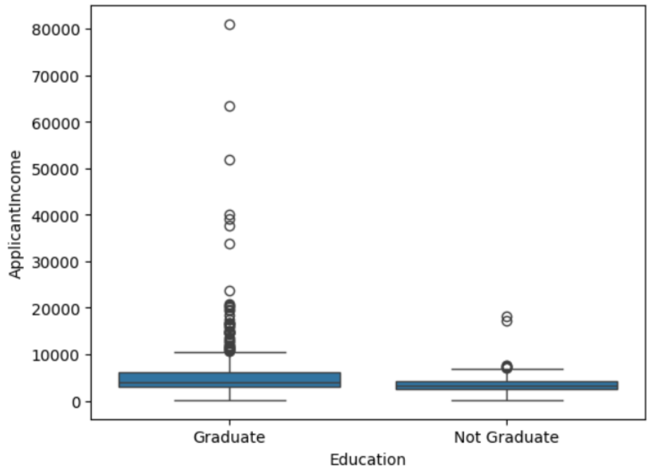
Lets check median.

```
In [24]: sns.barplot(data = Educationincomemedian, x = 'Education', y = 'ApplicantIncome', hue="Education")
plt.show()
plt.close()
```



The median is much closer suggesting some outliers increasing the graduate-income mean.

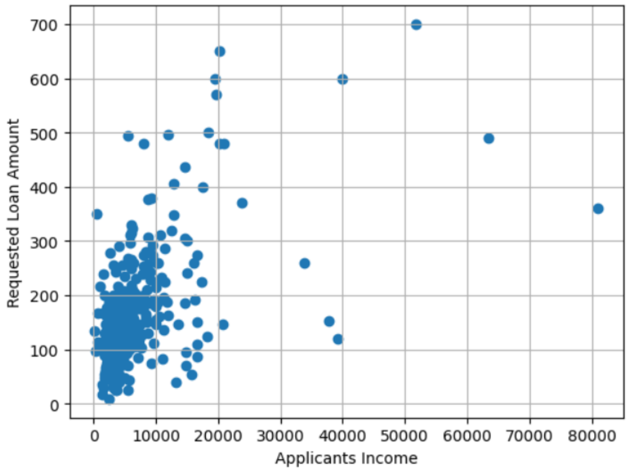
```
In [13]: sns.boxplot(data = loan_train, x = 'Education', y = 'ApplicantIncome')
plt.show()
plt.close()
```



(c) Plotting applicant income and loan amount

These are two quantitative variables and can be plotted through a scatter plot.

```
In [16]: plt.scatter(x = loan_train.ApplicantIncome, y = loan_train.LoanAmount)
         plt.xlabel('Applicants Income')
         plt.ylabel('Requested Loan Amount')
         plt.grid()
         plt.show()
         plt.close()
```



The plot shows a slight positive relation.

```
In [17]: from scipy.stats import pearsonr
```

```
In [18]: corr_income_loanamount, p = pearsonr(loan_train.ApplicantIncome,loan_train.LoanAmount)
         print(corr_income_loanamount)

         0.5656204566820268
```

Pearson relation can be used to show correlation between two quantitative variables.

4. Preprocessing

```
In [34]: loan_train.Loan_Status = loan_train.Loan_Status.replace({"Y": 1, "N" : 0})

         loan_train.Gender = loan_train.Gender.replace({"Male": 1, "Female" : 0})
         loan_test.Gender = loan_test.Gender.replace({"Male": 1, "Female" : 0})

         loan_train.Married = loan_train.Married.replace({"Yes": 1, "No" : 0})
         loan_test.Married = loan_test.Married.replace({"Yes": 1, "No" : 0})

         loan_train.Self_Employed = loan_train.Self_Employed.replace({"Yes": 1, "No" : 0})
         loan_test.Self_Employed = loan_test.Self_Employed.replace({"Yes": 1, "No" : 0})

         loan_train.Education = loan_train.Education.replace({'Graduate': 1, 'Not Graduate' : 0})
         loan_test.Education = loan_test.Education.replace({'Graduate': 1, 'Not Graduate' : 0})

         loan_train.Dependents = loan_train.Dependents.replace({'3+' : 4})
         loan_test.Dependents = loan_test.Dependents.replace({'3+' : 4})
```

Replacing categorical values with relevant numerical values.

One Hot Encoding

One-hot encoding is a technique used in machine learning and data processing to convert categorical variables into a numerical format that can be more easily processed by algorithms.

```
In [24]: train_dataset = pd.get_dummies(train_dataset, columns = ['Property_Area'])
         train_dataset = train_dataset.replace({True: 1, False: 0})
```

```
In [26]: test_dataset = pd.get_dummies(test_dataset, columns = ['Property_Area'])
         test_dataset = test_dataset.replace({True: 1, False: 0})
```

5. Creating Model

Since the target variable is a binary classification, logistic regression is suitable.

```
In [28]: from sklearn.linear_model import LogisticRegression

         from sklearn.metrics import accuracy_score
```

```
In [29]: logistic_model = LogisticRegression()
```

6. Training the Model

Features are chosen that will be used to train the model.

Features are stored as a numpy array x_train. The target variable is stored as another numpy array y_train.

```
In [45]: train_features = ['Gender', 'Married', 'Dependents', 'Education','Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAm

         x_train = train_dataset[train_features].values
         y_train = train_dataset['Loan_Status'].values.astype('int')

         x_test = test_dataset[train_features].values
```

Scaling the data to reduce the effect of outliers.

```
In [37]: x_train_scaled = MinMaxScaler().fit_transform(x_train)
         x_test_scaled = MinMaxScaler().fit_transform(x_test)
```

Fitting the logistic regression model.

```
In [30]: logistic_model.fit(x_train_scaled, y_train)

Out[30]: LogisticRegression
         LogisticRegression()
```

7. Model Testing and Evaluation

```
In [31]: predicted = logistic_model.predict(x_test_scaled)
```

```
In [32]: score = logistic_model.score(x_train_scaled, y_train)
         print('accuracy_score overall :', score)
         print('accuracy_score percent :', round(score*100,2))

         accuracy_score overall : 0.8094462540716613
         accuracy_score percent : 80.94
```