MAJOR PROJECT

KILLAKA SUMALATHA

May 2, 2024

Abstract

1 Intoduction

Loan prediction is a common problem in the banking and finance sector, where lenders aim to assess the creditworthiness of loan applicants. The objective is to predict whether a loan application should be approved or denied based on various factors provided by the applicant. This task is crucial for financial institutions to manage risk effectively and make informed decisions about lending. By accurately predicting loan outcomes, lenders can minimize defaults and optimize their lending practices. The loan prediction process typically involves analyzing a dataset containing information about past loan applicants, including attributes such as:
1.Applicant's demographic information (e.g., age, gender, marital status)
2.Financial details (e.g., income, employment status)
3.Loan-specific information (e.g., loan amount, term, purpose)
4.Credit history (e.g., credit score, previous defaults)
5.Property details (e.g., property type, location)
The dataset consists of the following attributes:

Attributes	**Significance**
Loan$_id$	Unique loan Id
Gender	Male/female
Married	Applicant married (Y/N)
Dependents	Number of dependents
Education	Applicant education (Graduate/ Undergraduate)
Self-employed	Self employed (Y/N)
ApplicantIncome	Applicant income
CoapplicantIncome	Coapplicant income
LoanAmount	Loan amount in thousands
Loan$_Amount_Term$	Term of loan in months
Credit$_history$	Credit history meet guidelines
Property$_area$	Urban / Semi urban / Rural
Loan$_status$	Loan approved (Y/N)

2 TASK-1 : Solving given queries

2.1 Import all the necessary libraries:

Pandas is a powerful Python library used for data manipulation and analysis. It provides data structures and functions to efficiently handle structured data, primarily in the form of DataFrame objects.some necessary libraries often used in conjunction with Pandas:

1.NumPy: NumPy is a fundamental package for scientific computing in Python. It provides support for large multi-dimensional arrays and matrices, along with a wide range of mathematical functions to operate on these arrays. Pandas is built on top of NumPy and relies heavily on its functionality.
2.Matplotlib: Matplotlib is a plotting library for Python that produces quality figures in various formats and interactive environments across platforms. It can be used to visualize data stored in Pandas DataFrame objects, making it an essential tool for data exploration and presentation.
3.Seaborn: Seaborn is a statistical data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics, enhancing the visual appeal of plots generated from Pandas DataFrame objects.

Here including Jupyter code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

#These libraries are essential for data manipulation,visualization, and building machine learning models.
```

2.2 Import the dataset provided:

Importing a dataset is one of the initial steps in any data analysis or machine learning project. It involves loading data from an external source into the environment where you'll be working with it, typically a programming environment like Python. **1.Identify the Dataset:** Before importing a dataset, you need to know where it's located and what format it's in. Datasets can be stored in various formats such as CSV (Comma-Separated Values), Excel files, JSON (JavaScript Object Notation), SQL databases, and more.
2.Choose the Right Tool: Depending on the format of the dataset and the environment you're working in, you'll need to choose the appropriate tool for importing. In Python, popular libraries for importing datasets include Pandas, NumPy, and built-in modules like CSV and JSON.
3.Use Pandas for Tabular Data: If your dataset is in tabular form (rows and columns, like a spreadsheet), Pandas is the preferred choice for importing and working with it. Pandas provides a DataFrame data structure that is well-suited for handling structured data.
4.Read the Dataset: Once you've chosen the right tool, you can use its functions to read the dataset into your environment. For example, in Pandas, you can use the $\text{read}_csv()functiontoreadaCSVfile, \text{read}_excel()forExcelfiles, \text{read}_json()forJSONfiles, andsoon.$
5.Handle Parameters:$Whenimportingthedataset, youmayneedtospecifyadditionalparametersdependingonthespecifi$

```
import pandas as pd
#Define the file path
file_path = 'loan-predictionUC.csv (3) (1).xlsx'
# Read Excel file
loan_data=pd.read_excel(file_path)
# Display the dataframe
print(loan_data)
#This code reads the loan dataset from a CSV file into a pandas DataFrame.
```

Output:

```
  Loan_ID  Gender Married Dependents     Education Self_Employed  \
0    LP001002    Male      No          0      Graduate            No
1    LP001003    Male     Yes          1      Graduate            No
2    LP001005    Male     Yes          0      Graduate           Yes
```

```
3    LP001006    Male     Yes          0  Not Graduate            No
4    LP001008    Male      No          0      Graduate            No
..        ...     ...     ...        ...           ...           ...
609  LP002978  Female      No          0      Graduate            No
610  LP002979    Male     Yes         3+      Graduate            No
611  LP002983    Male     Yes          1      Graduate            No
612  LP002984    Male     Yes          2      Graduate            No
613  LP002990  Female      No          0      Graduate           Yes

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0               5849                0.0         NaN             360.0
1               4583             1508.0       128.0             360.0
2               3000                0.0        66.0             360.0
3               2583             2358.0       120.0             360.0
4               6000                0.0       141.0             360.0
..               ...                ...         ...               ...
609             2900                0.0        71.0             360.0
610             4106                0.0        40.0             180.0
611             8072              240.0       253.0             360.0
612             7583                0.0       187.0             360.0
613             4583                0.0       133.0             360.0

     Credit_History Property_Area Loan_Status
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
..              ...           ...         ...
609             1.0         Rural           Y
610             1.0         Rural           Y
611             1.0         Urban           Y
612             1.0         Urban           Y
613             0.0     Semiurban           N

[614 rows x 13 columns]
```

2.3 Understand the data:

```
    print("INFO:")
loan_data.info()
print("DESCRIBE:")
loan_data.describe()
#These commands display the first few rows of the dataset, its structure, and summary statistics, helping to understand its f
```

Output:

```
    INFO:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
```

```
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
DESCRIBE:
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
```

– >**Type:**Type of the certain column.

```
print("type:")
type(loan_data)
```

Output:

```
    type:
pandas.core.frame.DataFrame
```

– >**Shape:**Number of rows and columns are called shape.

```
print("SHAPE")
loan_data.shape
```

Output:

```
SHAPE
(614, 13)
```

– >**Size:**Size of the dataframe

```
print("SIZE:")
loan_data.size
```

Output:

```
SIZE:
7982
```

– >**Dimension:**Dimension of the dataframe

```
print("DIMENTION:")
loan_data.ndim
```

Output:

```
DIMENTION:
2
```

– >**Head:**This method displays the first few rows of a DataFrame. By default, it shows the first 5 rows, but you can specify the number of rows you want to display by passing an integer argument to the method. It's useful for quickly inspecting the structure and contents of a DataFrame.

```
print("HEAD:")
loan_data.head()
```

Output:

```
SHAPE
HEAD:
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 1.0 Urban Y
1 LP001003 Male Yes 1 Graduate No 4583 1508.0 128.0 360.0 1.0 Rural N
2 LP001005 Male Yes 0 Graduate Yes 3000 0.0 66.0 360.0 1.0 Urban Y
3 LP001006 Male Yes 0 Not Graduate No 2583 2358.0 120.0 360.0 1.0 Urban Y
4 LP001008 Male No 0 Graduate No 6000 0.0 141.0 360.0 1.0 Urban Y
```

```
    print("HEAD IS 2:")
loan_data.head(2)
```

Output:

```
    HEAD IS 2:
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 1.0 Urban Y
1 LP001003 Male Yes 1 Graduate No 4583 1508.0 128.0 360.0 1.0 Rural N
```

– >**Tail:**Conversely, this method displays the last few rows of a DataFrame. Like head(), it defaults to displaying the last 5 rows but can be customized by passing an integer argument.

```
print("TAIL:")
loan_data.tail()
```

Output:

```
TAIL:
Loan_ID Gender Married Dependents Education Self_Employed ApplicantIncome CoapplicantIncome
LoanAmount Loan_Amount_Term Credit_History Property_Area Loan_Status
609 LP002978 Female No 0 Graduate No 2900 0.0 71.0 360.0 1.0 Rural Y
610 LP002979 Male Yes 3+ Graduate No 4106 0.0 40.0 180.0 1.0 Rural Y
611 LP002983 Male Yes 1 Graduate No 8072 240.0 253.0 360.0 1.0 Urban Y
612 LP002984 Male Yes 2 Graduate No 7583 0.0 187.0 360.0 1.0 Urban Y
613 LP002990 Female No 0 Graduate Yes 4583 0.0 133.0 360.0 0.0 Semiurban N
```

```
    print("TAIL IS 2:")
loan_data.tail(2)
```

Output:

```
    TAIL IS 2:
Loan_ID Gender Married Dependents Education Self_Employed ApplicantIncome CoapplicantIncome
LoanAmount Loan_Amount_Term Credit_History Property_Area Loan_Status
612 LP002984 Male Yes 2 Graduate No 7583 0.0 187.0 360.0 1.0 Urban Y
613 LP002990 Female No 0 Graduate Yes 4583 0.0 133.0 360.0 0.0 Semiurban N
```

– >**Columns:**It returns the all the columns in dataframe

```
    loan_data.columns
```

Output:

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

– >Number of columns from loan-dataframe

```
    print("NO.OF COLUMNS:")
rows,columns=loan_data.shape
columns
```

Output:

```
    NO.OF COLUMNS:
13
```

– >Number of rows from loan-dataframe

```
    print("NO.OF ROWS:")
rows
```

Output:

```
    NO.OF ROWS:
614
```

– >Accesing the specified column from loan dataframe

```
    print("access single column:")
loan_data.Gender # or loan_data['Gender']
```

Output:

```
    access single column:
0        Male
1        Male
2        Male
3        Male
4        Male
       ...
609    Female
610      Male
611      Male
612      Male
613    Female
Name: Gender, Length: 614, dtype: object
```

– >Finding the maximum LoanAmount from loan dataframe

```
    print("maximum loanAmount:")
loan_data['LoanAmount'].max()
```

Output:

```
    maximum loanAmount:
700.0
```

– >Finding the minimum LoanAmount from loan dataframe

```
    print("manimum loanAmount:")
loan_data['LoanAmount'].min()
```

Output:

```
    manimum loanAmount:
9.0
```

– >Finding the mean of the LoanAmount from loan dataframe

```
    print("mean of the loanAmount:")
loan_data['LoanAmount'].mean()
```

Output:

```
    mean of the loanAmount:
146.41216216216216
```

2.4 Deal with missing values if any:

Dealing with missing values is crucial in data analysis and machine learning tasks because missing data can adversely affect the quality and reliability of results. Here are some purposes and strategies for handling missing values:

```
    print("if there is accure missing value it shows True otherwise False")
loan_data.isnull()
#This checks for missing values in the dataset.
```

Output:

```
    if there is accure missing value it shows True otherwise False
Loan_ID Gender Married Dependents Education Self_Employed ApplicantIncome CoapplicantIncome
LoanAmount Loan_Amount_Term Credit_History Property_Area Loan_Status
0 False False False False False False False False True False False False False
1 False False False False False False False False False False False False False
2 False False False False False False False False False False False False False
3 False False False False False False False False False False False False False
4 False False False False False False False False False False False False False
... ... ... ... ... ... ... ... ... ... ... ... ... ... ...
609 False False False False False False False False False False False False False
610 False False False False False False False False False False False False False
611 False False False False False False False False False False False False False
612 False False False False False False False False False False False False False
613 False False False False False False False False False False False False False
614 rows × 13 columns
```

```
    mport pandas as pd
# Check for missing values
missing_values = loan_data.isnull().sum()
print("Missing Values:\n", missing_values)

# Impute missing values with mean for numerical columns
loan_data['LoanAmount'].fillna(loan_data['LoanAmount'].mean(), inplace=True)
loan_data['Loan_Amount_Term'].fillna(loan_data['Loan_Amount_Term'].mean(), inplace=True)

# For categorical variables, you can impute missing values with mode
loan_data['Gender'].fillna(loan_data['Gender'].mode()[0], inplace=True)
loan_data['Married'].fillna(loan_data['Married'].mode()[0], inplace=True)
loan_data['Dependents'].fillna(loan_data['Dependents'].mode()[0], inplace=True)
loan_data['Self_Employed'].fillna(loan_data['Self_Employed'].mode()[0], inplace=True)
loan_data['Credit_History'].fillna(loan_data['Credit_History'].mode()[0], inplace=True)

# Verify that missing values have been handled
missing_values_after = loan_data.isnull().sum()
print("Missing Values After Handling:\n", missing_values_after)

'''In this example:

We read the loan dataset into a DataFrame.
Checked for missing values using the isnull().sum() function.
Imputed missing values for numerical columns ('LoanAmount' and 'Loan_Amount_Term') with the mean value of each column.
Imputed missing values for categorical columns ('Gender', 'Married', 'Dependents', 'Self_Employed', and 'Credit_History') wit
Checked again for missing values to ensure they have been handled properly.'''
```

Output:

```
    Missing Values:
 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
Missing Values After Handling:
 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
"In this example:\n\nWe read the loan dataset into a DataFrame.\nChecked for missing values using the
```

```
    missing_values_after
```

Output:

```
    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
```

```
    print("after handles missing values:")
loan_data
```

Output:

```
    after handles missing values:
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
... ... ... ... ... ... ... ... ... ... ... ... ... ...
609 LP002978 Female No 0 Graduate No 2900 0.0 71.000000 360.0 1.0 Rural Y
610 LP002979 Male Yes 3+ Graduate No 4106 0.0 40.000000 180.0 1.0 Rural Y
611 LP002983 Male Yes 1 Graduate No 8072 240.0 253.000000 360.0 1.0 Urban Y
612 LP002984 Male Yes 2 Graduate No 7583 0.0 187.000000 360.0 1.0 Urban Y
613 LP002990 Female No 0 Graduate Yes 4583 0.0 133.000000 360.0 0.0 Semiurban N
614 rows × 13 columns
```

2.5 Do some visualization if necessary:

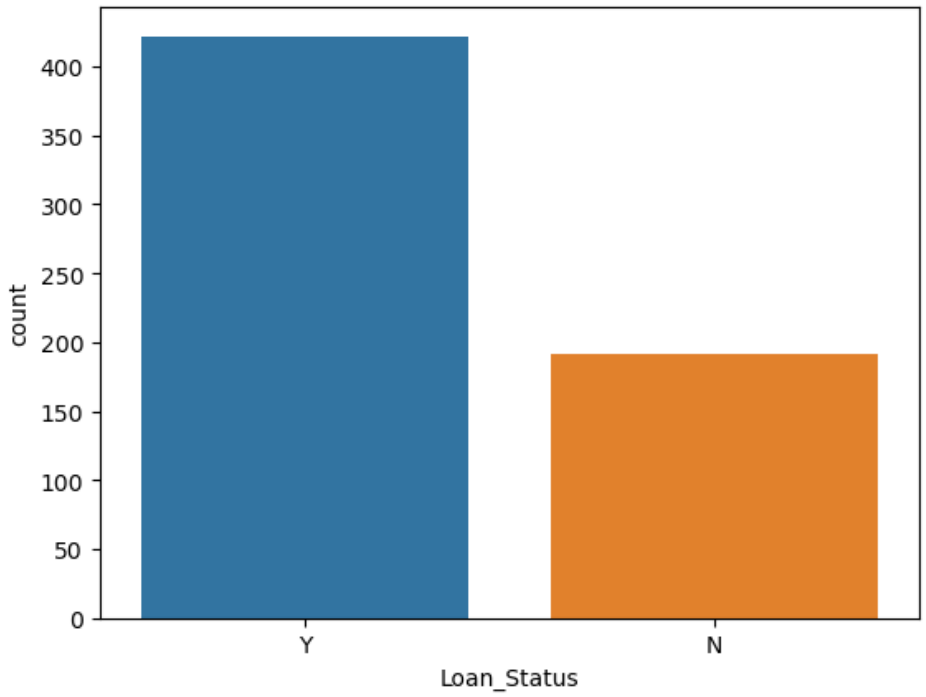
Visualization refers to the graphical representation of data and information. It involves the creation of visual depictions, such as charts, graphs, maps, and diagrams, to convey complex datasets, patterns, trends, and relationships in a clear and intuitive manner. The primary goal of visualization is to facilitate understanding, analysis, and communication of data by leveraging the human visual system's ability to perceive and interpret visual information efficiently.
In essence, visualization transforms raw data into visual representations that are easier to interpret, enabling users to gain insights, make informed decisions, and communicate findings effectively. It plays a crucial role in various fields, including data analysis, scientific research, business intelligence, education, journalism, and many others, where complex data needs to be analyzed, understood, and communicated to diverse audiences.
Note: Visualization serves several important purposes in data analysis and communication
Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python

scripts, the Python and IPython shell, the jupyter notebook, web application servers, and four graphical user interface toolkits.
– >**Matplotlib.Pyplot:**One of the core aspects of Matplotlib is matplotlib.pyplot. It is Matplotlib's scripting layer which we studied in details in the videos about Matplotlib. Recall that it is a collection of command style functions that make Matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc. In this lab, we will work with the scripting layer to learn how to generate line plots. In future labs, we will get to work with the Artist layer as well to experiment first hand how it differs from the scripting layer.

Plottings in pandas:
– >**Countplot:**The purpose of a count plot is to visually represent the frequency or count of observations within categorical data. It provides a straightforward way to understand the distribution of categorical variables by displaying the number of occurrences of each category.

```
    import seaborn as sns
sns.countplot(x='Loan_Status',data=loan_data)
plt.show()
#This visualization shows the distribution of loan approvals.
```

Output:

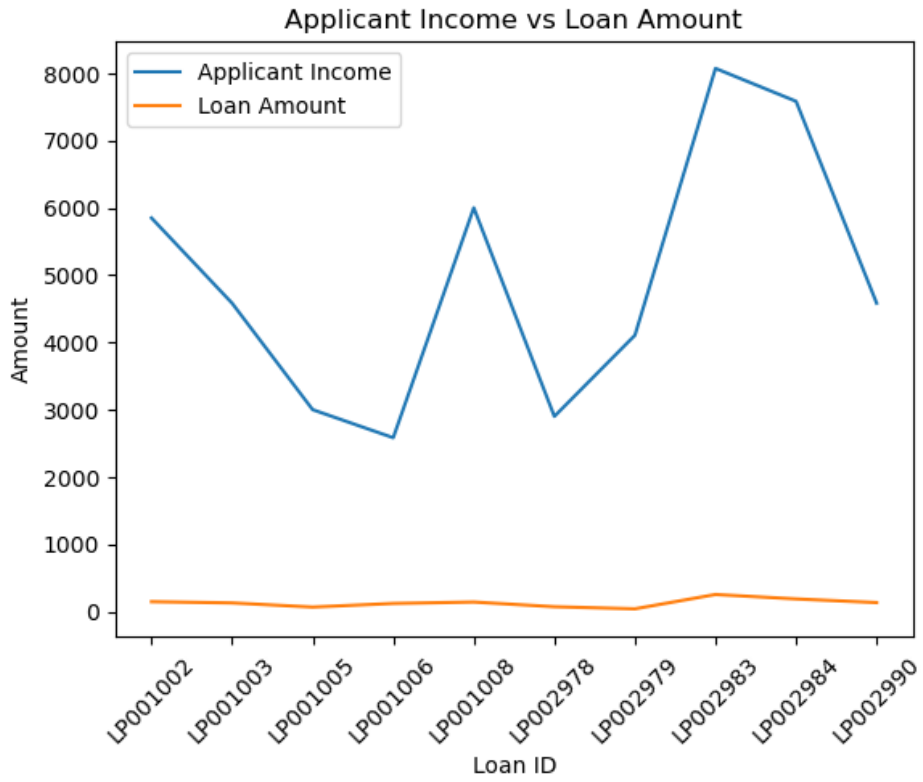


– >**Line plot:** Line plots are used to visualize the relationship between two variables by representing data points connected by straight lines.The primary purpose of line plots includes Trend Identification,Highlighting Outliers,Pattern Recognition,Communicating Insights etc..

```
    import matplotlib.pyplot as plt

# Sample data
Loan_ID = ["LP001002", "LP001003", "LP001005", "LP001006", "LP001008", "LP002978", "LP002979", "LP002983", "LP002984", "LP00
ApplicantIncome = [5849, 4583, 3000, 2583, 6000, 2900, 4106, 8072, 7583, 4583]
LoanAmount = [146.412162, 128.0, 66.0, 120.0, 141.0, 71.0, 40.0, 253.0, 187.0, 133.0]

# Create line plot
plt.plot(Loan_ID, ApplicantIncome, label='Applicant Income')
```

```
plt.plot(Loan_ID, LoanAmount, label='Loan Amount')

# Add labels and title
plt.xlabel('Loan ID')
plt.ylabel('Amount')
plt.title('Applicant Income vs Loan Amount')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.legend()

# Display the plot
plt.show()
'''In this plot, each line represents either the 'Applicant Income' or 'Loan Amount' for each loan ID. The x-axis represents

Remember to adjust the variable values ('ApplicantIncome', 'LoanAmount', 'Loan_ID') based on your actual dataset if needed.''
```

Output:



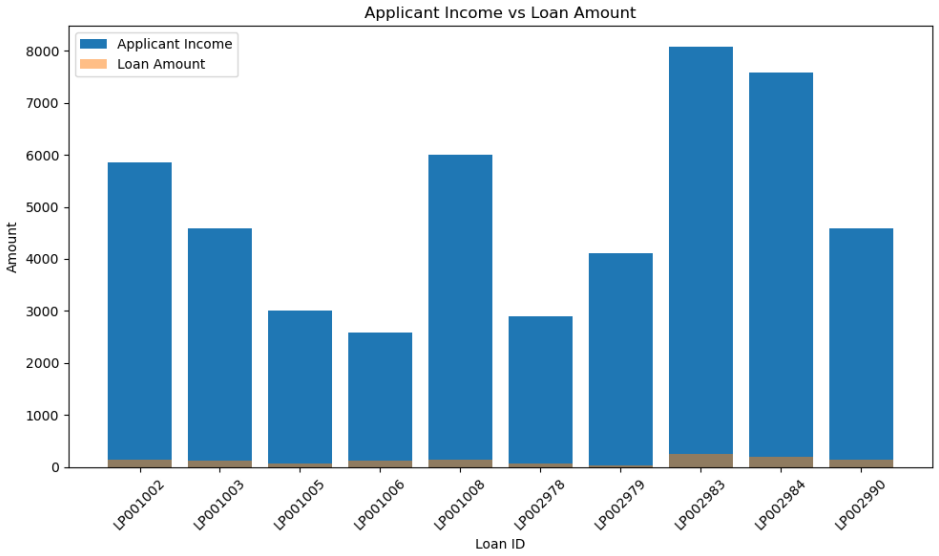
"In this plot, each line represents either the 'Applicant Income' or 'Loan Amount' for each loan ID. The x-axis represents the loan IDs, and the y-axis represents the corresponding amounts.to adjust the variable values ('ApplicantIncome', 'LoanAmount', 'Loan$_ID'$)*basedonyouractualdatasetifneeded.*"

– >**Bar plot:**Bar plots are used to visually represent categorical data using rectangular bars. The primary purpose of a bar plot is to display and compare the quantities or frequencies of different categories within a dataset.

```
    import matplotlib.pyplot as plt

# Sample data
Loan_ID = ["LP001002", "LP001003", "LP001005", "LP001006", "LP001008", "LP002978", "LP002979", "LP002983", "LP002984", "LP00
ApplicantIncome = [5849, 4583, 3000, 2583, 6000, 2900, 4106, 8072, 7583, 4583]
LoanAmount = [146.412162, 128.0, 66.0, 120.0, 141.0, 71.0, 40.0, 253.0, 187.0, 133.0]

# Create bar plot
plt.figure(figsize=(10, 6))
plt.bar(Loan_ID, ApplicantIncome, label='Applicant Income')
```

```
plt.bar(Loan_ID, LoanAmount, label='Loan Amount', alpha=0.5)  # Use alpha to make bars transparent

# Add labels and title
plt.xlabel('Loan ID')
plt.ylabel('Amount')
plt.title('Applicant Income vs Loan Amount')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.legend()

# Display the plot
plt.tight_layout()
plt.show()
'''In this bar plot, each loan ID is represented by a pair of bars, one for 'Applicant Income' and one for 'Loan Amount'. The

Again, remember to adjust the variable values ('ApplicantIncome', 'LoanAmount', 'Loan_ID') based on your actual dataset if ne
```

Output:



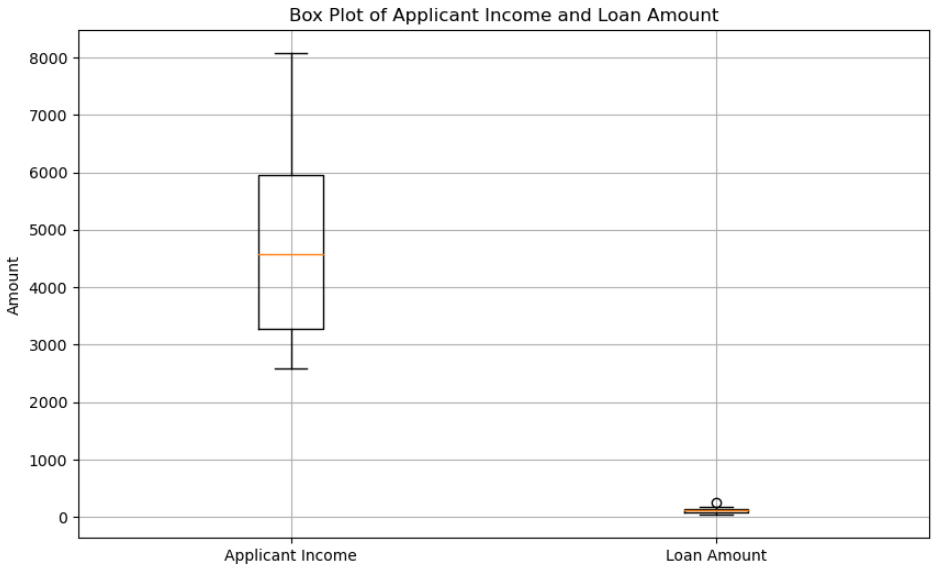
"In this bar plot, each loan ID is represented by a pair of bars, one for 'Applicant Income' and one for 'Loan Amount'. The x-axis represents the loan IDs, and the y-axis represents the corresponding amounts., remember to adjust the variable values ('ApplicantIncome', 'LoanAmount', 'Loan$_ID'$)$based on your actual dataset if needed$."

– >**Box plot:**Box plots, also known as box-and-whisker plots, are graphical representations of the distribution of a continuous variable through its quartiles. They serve several purposes in data analysis and visualization:

```
    import matplotlib.pyplot as plt

# Sample data
Loan_ID = ["LP001002", "LP001003", "LP001005", "LP001006", "LP001008", "LP002978", "LP002979", "LP002983", "LP002984", "LP00
ApplicantIncome = [5849, 4583, 3000, 2583, 6000, 2900, 4106, 8072, 7583, 4583]
LoanAmount = [146.412162, 128.0, 66.0, 120.0, 141.0, 71.0, 40.0, 253.0, 187.0, 133.0]

# Create box plot
plt.figure(figsize=(10, 6))
plt.boxplot([ApplicantIncome, LoanAmount], labels=['Applicant Income', 'Loan Amount'])

# Add labels and title
plt.ylabel('Amount')
plt.title('Box Plot of Applicant Income and Loan Amount')

# Display the plot
plt.grid(True)
```

```
plt.show()
'''In this box plot, there are two box-and-whisker plots side by side, one for 'Applicant Income' and one for 'Loan Amount'.

Adjust the variable values ('ApplicantIncome', 'LoanAmount', 'Loan_ID') based on your actual dataset if needed.'''
```

Output:

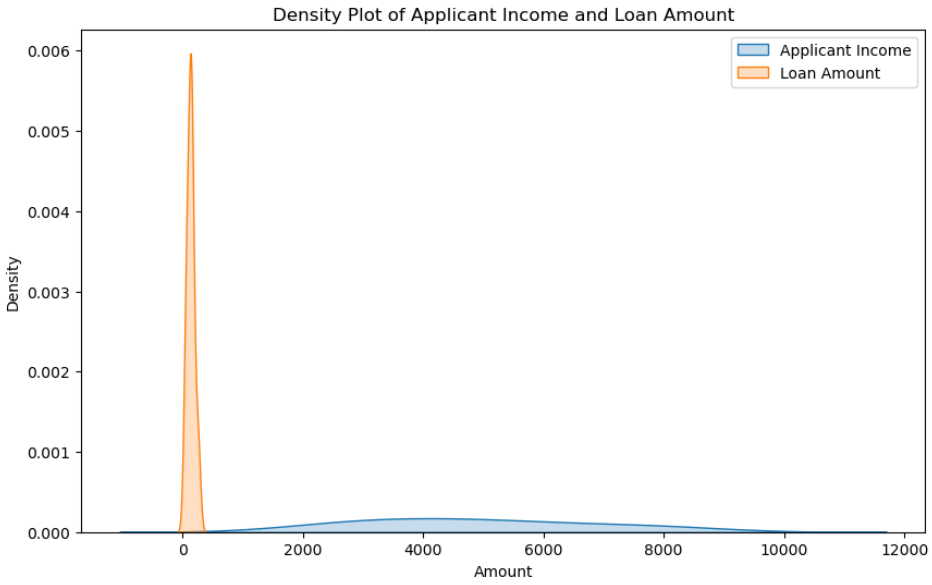


"In this box plot, there are two box-and-whisker plots side by side, one for 'Applicant Income' and one for 'Loan Amount'. The box represents the interquartile range (IQR) of the data, with the line inside the box representing the median. The whiskers extend to the minimum and maximum values within 1.5 times the IQR, and any points beyond the whiskers are considered outliers.the variable values ('ApplicantIncome', 'LoanAmount', 'Loan$_ID'$)$basedonyouractualdatasetifneeded$."

– >**Density plot:**Density plots, also known as kernel density estimation (KDE) plots, are used to visualize the distribution of a continuous variable. They serve several purposes in data analysis and visualization:

```
    import seaborn as sns
import matplotlib.pyplot as plt

# Sample data
Loan_ID = ["LP001002", "LP001003", "LP001005", "LP001006", "LP001008", "LP002978", "LP002979", "LP002983", "LP002984", "LP00
ApplicantIncome = [5849, 4583, 3000, 2583, 6000, 2900, 4106, 8072, 7583, 4583]
LoanAmount = [146.412162, 128.0, 66.0, 120.0, 141.0, 71.0, 40.0, 253.0, 187.0, 133.0]

# Create density plot
plt.figure(figsize=(10, 6))
sns.kdeplot(ApplicantIncome, label='Applicant Income', shade=True)
sns.kdeplot(LoanAmount, label='Loan Amount', shade=True)

# Add labels and title
plt.xlabel('Amount')
plt.ylabel('Density')
plt.title('Density Plot of Applicant Income and Loan Amount')

# Display the plot
plt.legend()
plt.show()
'''In this density plot, the kernel density estimate (KDE) is computed for both 'Applicant Income' and 'Loan Amount'. The sha

Adjust the variable values ('ApplicantIncome', 'LoanAmount', 'Loan_ID') based on your actual dataset if needed.'''
```

Output:

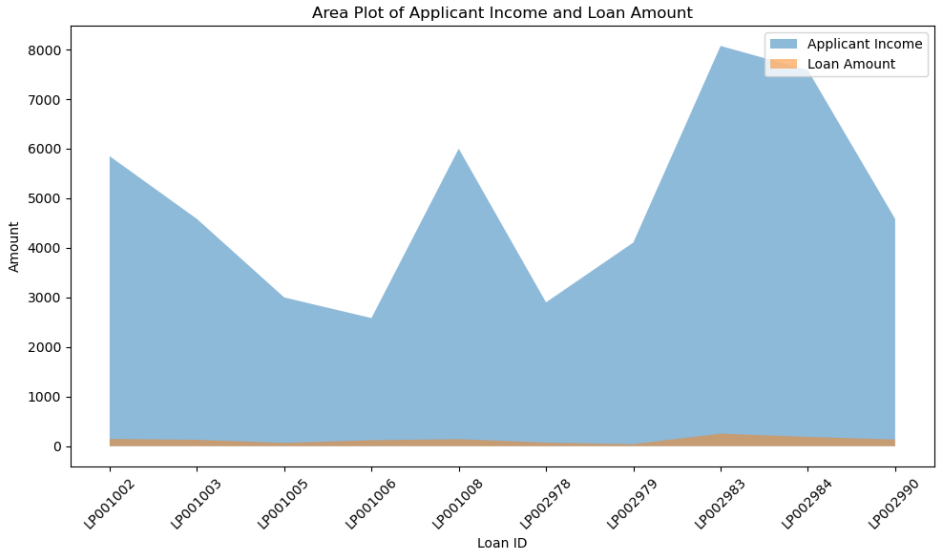


"In this density plot, the kernel density estimate (KDE) is computed for both 'Applicant Income' and 'Loan Amount'. The shaded area represents the density of each variable. The plot allows you to visualize the distribution of the data and compare the densities of the two variables.the variable values ('ApplicantIncome', 'LoanAmount', 'Loan$_ID'$)$based on your actual dataset if needed.$"

– >**Area plot:**Area plots, also known as stacked area plots, are used to visualize the changes in one or more variables over time or another continuous dimension. The primary purpose of an area plot is to display the cumulative contribution of multiple variables to a total or to show the evolution of individual variables over time while emphasizing their combined effect.

```
    import matplotlib.pyplot as plt

# Sample data
Loan_ID = ["LP001002", "LP001003", "LP001005", "LP001006", "LP001008", "LP002978", "LP002979", "LP002983", "LP002984", "LP00
ApplicantIncome = [5849, 4583, 3000, 2583, 6000, 2900, 4106, 8072, 7583, 4583]
LoanAmount = [146.412162, 128.0, 66.0, 120.0, 141.0, 71.0, 40.0, 253.0, 187.0, 133.0]

# Create area plot
plt.figure(figsize=(10, 6))
plt.fill_between(Loan_ID, ApplicantIncome, label='Applicant Income', alpha=0.5)
plt.fill_between(Loan_ID, LoanAmount, label='Loan Amount', alpha=0.5)

# Add labels and title
plt.xlabel('Loan ID')
plt.ylabel('Amount')
plt.title('Area Plot of Applicant Income and Loan Amount')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability

# Display the plot
plt.legend()
plt.tight_layout()
plt.show()
'''In this area plot, the area between the data points and the x-axis is filled, creating a filled area plot. Each filled are

Adjust the variable values ('ApplicantIncome', 'LoanAmount', 'Loan_ID') based on your actual dataset if needed.
'''
```

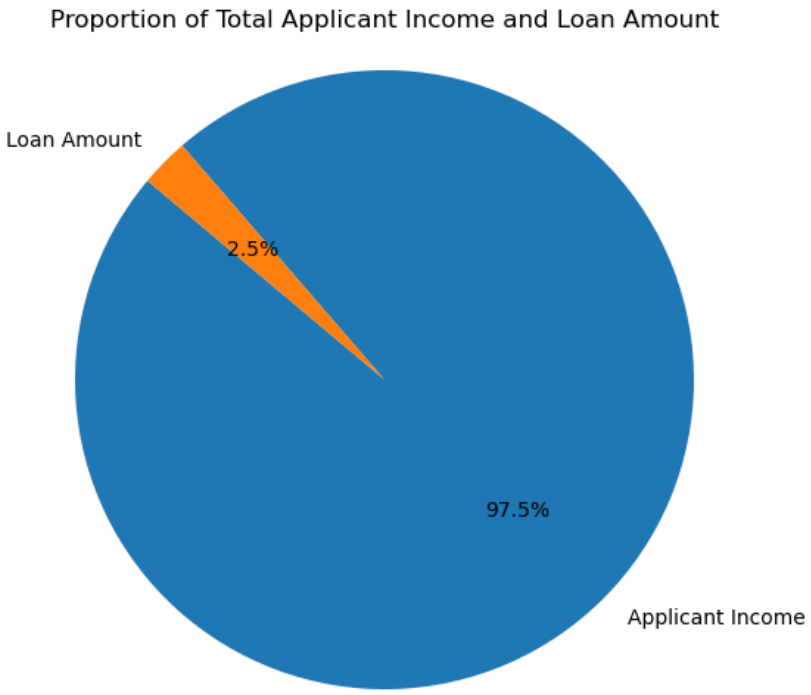
Output:
– >**Pie chart:**Pie charts are used to represent categorical data and display the proportions of various categories as segments of a circular "pie." The primary purpose of a pie chart is to visualize the relative sizes or proportions of different categories within a dataset.



"In this area plot, the area between the data points and the x-axis is filled, creating a filled area plot. Each filled area represents either 'Applicant Income' or 'Loan Amount' for each loan ID. The x-axis represents the loan IDs, and the y-axis represents the corresponding amounts.the variable values ('ApplicantIncome', 'LoanAmount', 'Loan$_ID'$)$based on your actual dataset if needed.$"

```
    import matplotlib.pyplot as plt

# Sample data
ApplicantIncome = [5849, 4583, 3000, 2583, 6000, 2900, 4106, 8072, 7583, 4583]
LoanAmount = [146.412162, 128.0, 66.0, 120.0, 141.0, 71.0, 40.0, 253.0, 187.0, 133.0]

# Calculate total values
total_applicant_income = sum(ApplicantIncome)
total_loan_amount = sum(LoanAmount)

# Create labels for the pie chart
labels = ['Applicant Income', 'Loan Amount']

# Calculate proportions
sizes = [total_applicant_income, total_loan_amount]

# Create pie chart
plt.figure(figsize=(8, 6))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)

# Add title
plt.title('Proportion of Total Applicant Income and Loan Amount')

# Equal aspect ratio ensures that pie is drawn as a circle
plt.axis('equal')

# Display the plot
plt.show()
'''In this pie chart, each slice represents the proportion of the total 'Applicant Income' and 'Loan Amount'. The labels show

Adjust the variable values ('ApplicantIncome', 'LoanAmount') based on your actual dataset if needed.'''
```

Output:
– >**Scatter plot:**Scatter plots are used to visualize the relationship between two continuous variables. The primary purpose of a scatter plot is to understand how the values of one variable are distributed with respect to another variable.

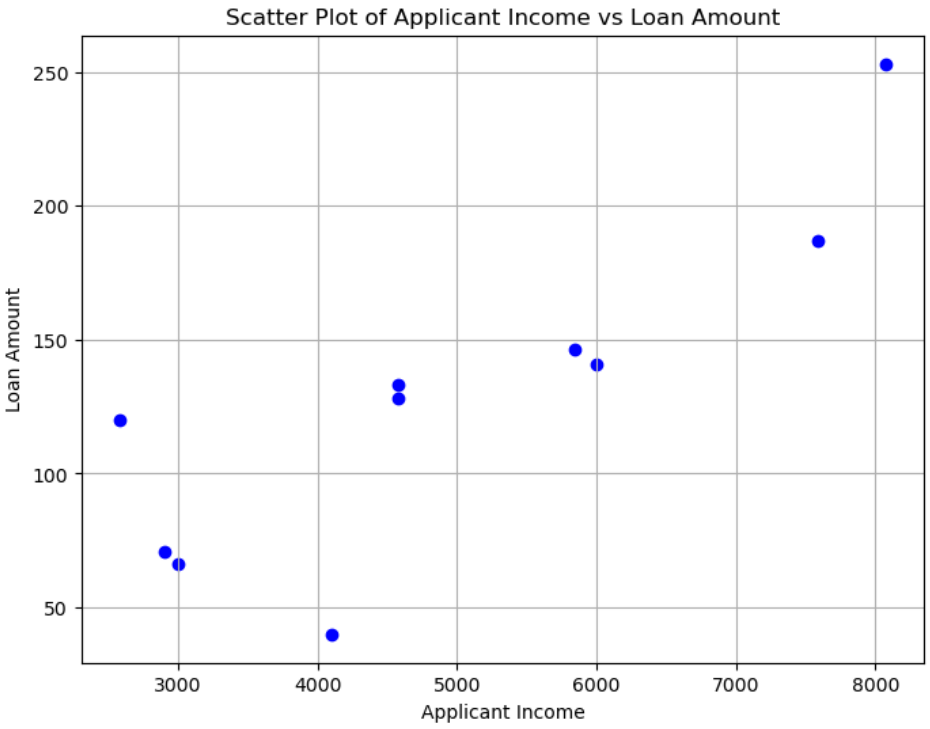


"In this pie chart, each slice represents the proportion of the total 'Applicant Income' and 'Loan Amount'. The labels show the names of the variables, and the percentages indicate the proportion of each variable relative to the total.the variable values ('ApplicantIncome', 'LoanAmount') based on your actual dataset if needed."

```
    import matplotlib.pyplot as plt

# Sample data
ApplicantIncome = [5849, 4583, 3000, 2583, 6000, 2900, 4106, 8072, 7583, 4583]
LoanAmount = [146.412162, 128.0, 66.0, 120.0, 141.0, 71.0, 40.0, 253.0, 187.0, 133.0]

# Create scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(ApplicantIncome, LoanAmount, color='blue')

# Add labels and title
plt.xlabel('Applicant Income')
plt.ylabel('Loan Amount')
plt.title('Scatter Plot of Applicant Income vs Loan Amount')

# Display the plot
plt.grid(True)
plt.show()
'''In this scatter plot, each point represents a pair of 'Applicant Income' and 'Loan Amount'. The x-axis represents the 'App

Adjust the variable values ('ApplicantIncome', 'LoanAmount') based on your actual dataset if needed.'''
```

Output:
– >**Hexbin plot:**The hexbin plot is a type of bivariate histogram that represents the relationship between two continuous variables by binning the data into hexagonal bins. The primary purpose of a hexbin plot is to visualize the density of points in a scatter plot when there is a high level of data overlap or when dealing with a large dataset.

```
    import matplotlib.pyplot as plt

# Sample data
```

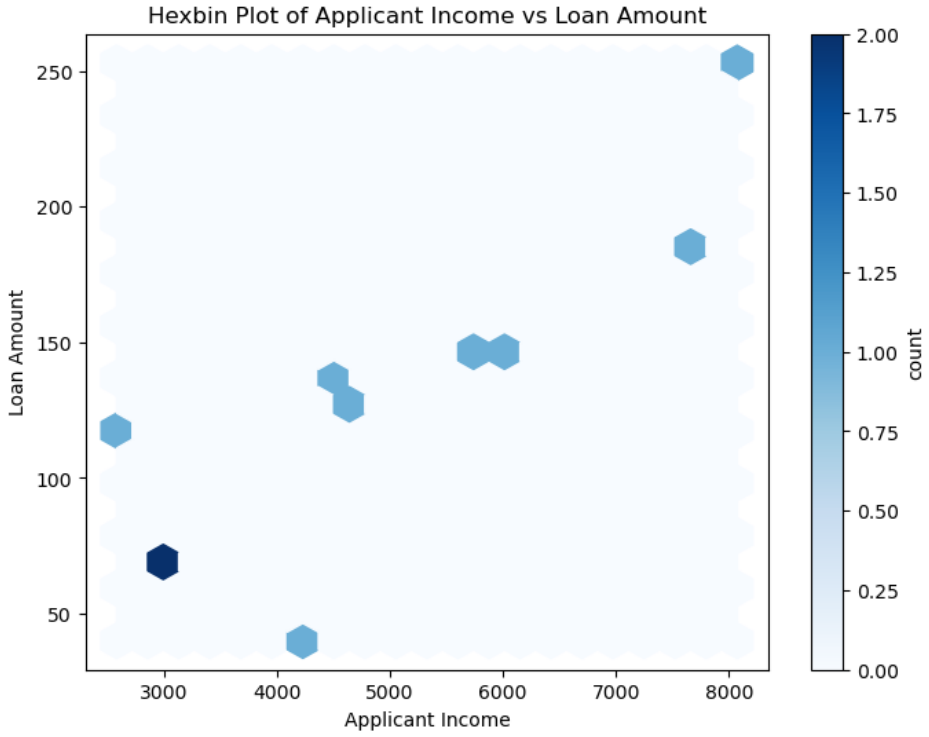


: "In this scatter plot, each point represents a pair of 'Applicant Income' and 'Loan Amount'. The x-axis represents the 'Applicant Income', and the y-axis represents the 'Loan Amount'. Each point's position on the plot indicates the relationship between the two variables.the variable values ('ApplicantIncome', 'LoanAmount') based on your actual dataset if needed."

```
ApplicantIncome = [5849, 4583, 3000, 2583, 6000, 2900, 4106, 8072, 7583, 4583]
LoanAmount = [146.412162, 128.0, 66.0, 120.0, 141.0, 71.0, 40.0, 253.0, 187.0, 133.0]

# Create hexbin plot
plt.figure(figsize=(8, 6))
plt.hexbin(ApplicantIncome, LoanAmount, gridsize=20, cmap='Blues')

# Add labels and title
plt.xlabel('Applicant Income')
plt.ylabel('Loan Amount')
plt.title('Hexbin Plot of Applicant Income vs Loan Amount')

# Add a colorbar
plt.colorbar(label='count')

# Display the plot
plt.show()
'''In this hexbin plot, each hexagonal bin represents the frequency of occurrence of data points within that bin. The color i
Adjust the variable values ('ApplicantIncome', 'LoanAmount') and the gridsize parameter based on your actual dataset and pref
```

Output:
– >**Histogram:**Histograms are used to visualize the distribution of a single continuous variable. The primary purpose of a histogram is to understand the underlying frequency or density distribution of data values within a given range.

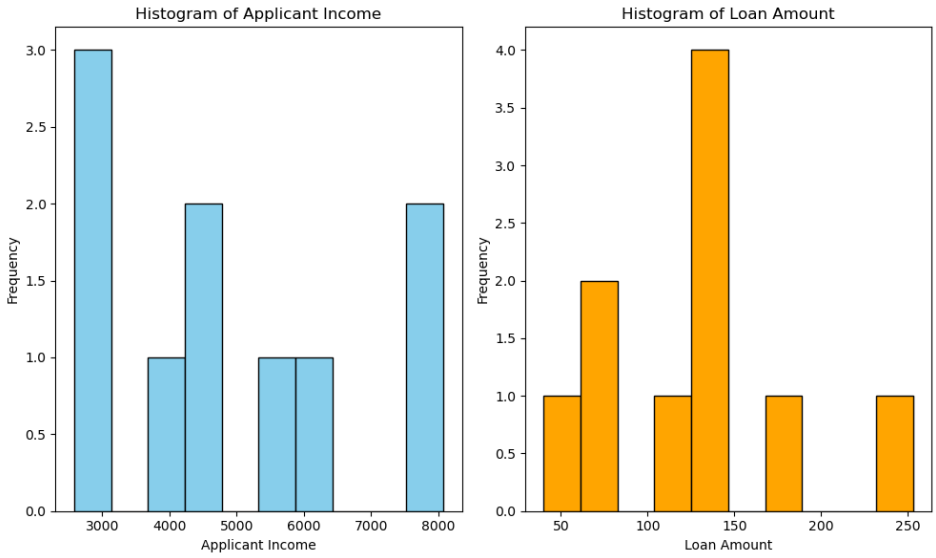
```
    import pandas as pd
import matplotlib.pyplot as plt

# Data
data = {
    "Loan_ID": ["LP001002", "LP001003", "LP001005", "LP001006", "LP001008", "LP002978", "LP002979", "LP002983", "LP002984",
    "ApplicantIncome": [5849, 4583, 3000, 2583, 6000, 2900, 4106, 8072, 7583, 4583],
```



"In this hexbin plot, each hexagonal bin represents the frequency of occurrence of data points within that bin. The color intensity of each bin indicates the number of data points falling into it. This plot helps visualize the distribution and density of data points across the two variables ('Applicant Income' and 'Loan Amount').the variable values ('ApplicantIncome', 'LoanAmount') and the gridsize parameter based on your actual dataset and preferences if needed."

```
    "LoanAmount": [146.412162, 128.0, 66.0, 120.0, 141.0, 71.0, 40.0, 253.0, 187.0, 133.0]
}

# Create DataFrame
df = pd.DataFrame(data)

# Plot histograms
plt.figure(figsize=(10, 6))

plt.subplot(1, 2, 1)
plt.hist(df['ApplicantIncome'], bins=10, color='skyblue', edgecolor='black')
plt.xlabel('Applicant Income')
plt.ylabel('Frequency')
plt.title('Histogram of Applicant Income')

plt.subplot(1, 2, 2)
plt.hist(df['LoanAmount'], bins=10, color='orange', edgecolor='black')
plt.xlabel('Loan Amount')
plt.ylabel('Frequency')
plt.title('Histogram of Loan Amount')

plt.tight_layout()
plt.show()
'''In this code will create two histograms side by side, one for "ApplicantIncome" and the other for "LoanAmount", showing th
```

Output:



"'In this code will create two histograms side by side, one for "ApplicantIncome" and the other for "LoanAmount", showing the distribution of these variables in the dataset. Adjust the number of bins as needed to best represent the data distribution."'

– >**Waffle chart:**Waffle charts are used to visualize the composition or distribution of categorical data in a more visually engaging way. The primary purpose of a waffle chart is to represent the proportions of different categories within a dataset by using small square or rectangular tiles that resemble a waffle grid.

```
    import pandas as pd
import matplotlib.pyplot as plt
from pywaffle import Waffle

# Data
data = {
    "ApplicantIncome": [5849, 4583, 3000, 2583, 6000, 2900, 4106, 8072, 7583, 4583],
    "LoanAmount": [146.412162, 128.0, 66.0, 120.0, 141.0, 71.0, 40.0, 253.0, 187.0, 133.0]
```

```
}

# Create DataFrame
df = pd.DataFrame(data)

# Discretize variables into categories
df['Income_Category'] = pd.cut(df['ApplicantIncome'], bins=[0, 3000, 6000, 9000], labels=['Low', 'Medium', 'High'])
df['LoanAmount_Category'] = pd.cut(df['LoanAmount'], bins=[0, 100, 200, 300], labels=['Low', 'Medium', 'High'])

# Aggregate counts for each category
income_counts = df['Income_Category'].value_counts()
loan_amount_counts = df['LoanAmount_Category'].value_counts()

# Plot waffle chart for ApplicantIncome
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
fig = plt.figure(
    FigureClass=Waffle,
    rows=10,
    columns=20,
    values=income_counts,
    colors=["skyblue", "orange", "green"],
    legend={'loc': 'upper left', 'bbox_to_anchor': (1, 1)},
    title={'label': 'Applicant Income Waffle Chart', 'loc': 'center'}
)

# Plot waffle chart for LoanAmount
plt.subplot(1, 2, 2)
fig = plt.figure(
    FigureClass=Waffle,
    rows=10,
    columns=20,
    values=loan_amount_counts,
    colors=["skyblue", "orange", "green"],
    legend={'loc': 'upper left', 'bbox_to_anchor': (1, 1)},
    title={'label': 'Loan Amount Waffle Chart', 'loc': 'center'}
)

plt.tight_layout()
plt.show()
'''In this example, we discretize "ApplicantIncome" and "LoanAmount" into categories (Low, Medium, High)
based on predefined thresholds. Then, we count the number of occurrences of each category and create waffle charts for each v
```

Output:

2.6 Divide the dataset into training and test datasets:

Training and testing are fundamental concepts in machine learning and statistical modeling. These processes are essential for building and evaluating predictive models.

Training

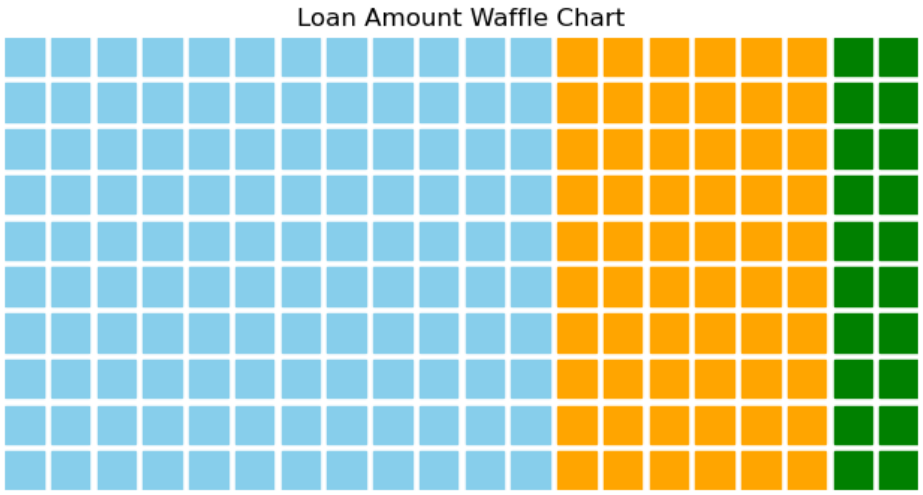
Definition: Training is the process of teaching a machine learning model to make predictions or infer patterns from data.

Testing:

Definition: Testing is the process of evaluating a trained model's performance on new, unseen data to assess its predictive accuracy and generalization ability.

```
    #Split the dataset into features (X) and target (y), then further split it into training and testing sets.
from sklearn.linear_model import LogisticRegression
```

```
    m=LogisticRegression()
```



"'In this example, we discretize "ApplicantIncome" and "LoanAmount" into categories (Low, Medium, High) on predefined thresholds. Then, we count the number of occurrences of each category and create waffle charts for each variable. Adjust the bin thresholds and categories according to your data and preferences."'

```
    from sklearn.model_selection import train_test_split
```

```
    X = loan_data.drop(columns=['Loan_ID','Loan_Status'])
y = loan_data['Loan_Status']

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.33,random_state=42)
```

```
    print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)

'''
X_train:Features for training the model
X_test:Features for testing the model
y_train:Target variable for training the model
y_test:Target variable for testing the model
'''
```

Output:

```
    (411, 11)
(203, 11)
(411,)
(203,)
'''\nX_train:Features for training the model\nX_test:Features for testing the model\ny_train:Target v
```

2.7 Build the machine learning model:

Building a machine learning model involves several steps, from data preprocessing and feature engineering to model selection, training, and evaluation.

```
    from sklearn.preprocessing import LabelEncoder

# Encode categorical variables
le = LabelEncoder()
for col in X_train.select_dtypes(include='object').columns:
    # Convert all values to strings
    X_train[col] = X_train[col].astype(str)
    X_test[col] = X_test[col].astype(str)

    # Encode the column
    X_train[col] = le.fit_transform(X_train[col])
    X_test[col] = le.transform(X_test[col])

# Output after encoding
print("X_train after encoding:\n", X_train.head())
print("\nX_test after encoding:\n", X_test.head())
'''Each row represents a sample in the dataset.
The categorical variables have been encoded numerically.
The first few rows of both the training (X_train) and test (X_test) datasets are shown for verification.'''
```

Output:

```
    X_train after encoding:
      Gender  Married  Dependents  Education  Self_Employed  ApplicantIncome  \
244       1        1           0          1              0             3406
393       1        1           2          1              0             1993
310       0        0           0          0              0             2917
408       1        1           1          0              0             8300
572       1        1           2          0              0            16666

     CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
244             4417.0       123.0             360.0             1.0
393             1625.0       113.0             180.0             1.0
310                0.0        84.0             360.0             1.0
408                0.0       152.0             300.0             0.0
572                0.0       275.0             360.0             1.0

     Property_Area
244              2
393              2
310              4
408              2
572              3

X_test after encoding:
      Gender  Married  Dependents  Education  Self_Employed  ApplicantIncome  \
350       1        1           0          0              0             9083
377       1        1           0          0              0             4310
163       1        1           2          0              0             4167
609       0        0           0          0              0             2900
132       1        0           0          0              0             2718

     CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
350                0.0       228.0             360.0             1.0
377                0.0       130.0             360.0             1.0
163             1447.0       158.0             360.0             1.0
609                0.0        71.0             360.0             1.0
132                0.0        70.0             360.0             1.0

     Property_Area
```

```
350              2
377              2
163              0
609              0
132              2
'''Each row represents a sample in the dataset.\nThe categorical variables have been encoded numerica
```

2.8 Fit the model on the training dataset:

Fitting the model on the training dataset is a critical step in the machine learning workflow. This process involves training the selected machine learning algorithm or model on the training data to learn the underlying patterns and relationships within the data.
Definition:
Fitting the model refers to the process of applying the selected machine learning algorithm to the training dataset to estimate the parameters or weights that best describe the relationship between the input features and the target variable.

```
 #Train the model on the training dataset
    '''Make sure to include this code snippet before the point where you attempt to fit the model. This will ensure that the
you have already defined and instantiated the rf_classifier, then make sure it's defined in the current scope or check for an
from sklearn.ensemble import RandomForestClassifier

# Instantiate the RandomForestClassifier
rf_classifier = RandomForestClassifier()

# Fit the model on the training dataset
rf_classifier.fit(X_train, y_train)
```

Output:

```
    RandomForestClassifier
RandomForestClassifier()
```

2.9 Test the model and find the accuracy:

Testing the model and computing its accuracy is an essential step in the machine learning workflow. After training the model on the training dataset, testing evaluates its performance on unseen data to assess how well it generalizes to new observations.
Definition:
Testing the model refers to the process of applying the trained machine learning model to a separate dataset called the testing dataset to evaluate its performance and assess its predictive accuracy.

```
#Test the trained model on the test dataset and evaluate its accuracy.
    from sklearn.metrics import accuracy_score

# Calculate accuracy
train_accuracy = accuracy_score(y_train, train_preds)
test_accuracy = accuracy_score(y_test, test_preds)

# Output accuracy
print(f"Training Accuracy: {train_accuracy}")
print(f"Test Accuracy: {test_accuracy}")
'''->Training Accuracy represents the accuracy of the model on the training dataset.
->Test Accuracy represents the accuracy of the model on the test dataset.'''
```

Output:

```
    Training Accuracy: 1.0
Test Accuracy: 0.7783251231527094
'''->Training Accuracy represents the accuracy of the model on the training dataset.\n->Test Accuracy
```

2.10 Create a confusion matrix:

Creating a confusion matrix is a fundamental step in evaluating the performance of a classification model. It provides a detailed breakdown of the model's predictions compared to the actual outcomes.
Definition:
Confusion Matrix is a table that summarizes the performance of a classification model by presenting the counts of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) predictions made by the model.

```
    # Create confusion matrix
from sklearn.metrics import confusion_matrix

# Create confusion matrix
conf_matrix = confusion_matrix(y_test, test_preds)
conf_matrix_df = pd.DataFrame(conf_matrix, index=['Actual No', 'Actual Yes'], columns=['Predicted No', 'Predicted Yes'])
print("Confusion Matrix:")
print(conf_matrix_df)
'''->The rows represent the actual classes ('No' and 'Yes').
->The columns represent the predicted classes ('No' and 'Yes').
->The values in the matrix represent the counts of true positive, false positive, true negative, and false negative predictio
```

```
    Confusion Matrix:
            Predicted No  Predicted Yes
Actual No             34             38
Actual Yes             7            124
"->The rows represent the actual classes ('No' and 'Yes').\n->The columns represent the predicted cla
```

3 TASK-2 : Tableau Dashboard

To create a Tableau dashboard, we would export the cleaned and preprocessed dataset and visualize it using Tableau software. We can create various visualizations, such as bar charts, line charts, pie charts, and maps, to explore different aspects of the loan dataset and present key findings in an interactive and visually appealing dashboard. We can then share the Tableau dashboard with stakeholders for further analysis and decision-making.
Tableau Dashboard:
1.Import the Dataset:
.Open Tableau Desktop.
.Import the loan sanction dataset (e.g., '$cleaned_{l}oan_{d}ataset.csv'$).
2.Create Visualizations:
$CreateVisualizations$:
$.Draganddropfieldsontotheworkspacetocreatevisualizations$:
$.Barchart$: $Loanapprovalstatusbygender$
$.Piechart$: $Distributionofloanapprovalstatusbypropertyarea$
3.Design the Dashboard:
.Click on the 'New Dashboard' button.
.Drag the created visualizations onto the dashboard canvas.
.Arrange and resize the visualizations to create a visually appealing layout.
4.Add Interactivity:
.Add filters to allow users to interact with the data:
.Gender filter: Allow users to filter loan approval status by gender.
.Property area filter: Allow users to filter loan approval status by property area.
5.Publish the Dashboard:
.Click on the 'Publish' button to publish the dashboard to Tableau Server or Tableau Public.
.Follow the prompts to enter the server details and publish the dashboard.
6.Document the Dashboard:
.Document the key insights, findings, and interpretations derived from the dashboard.
.Include annotations, descriptions, and explanations to help users understand the data and make

informed decisions.

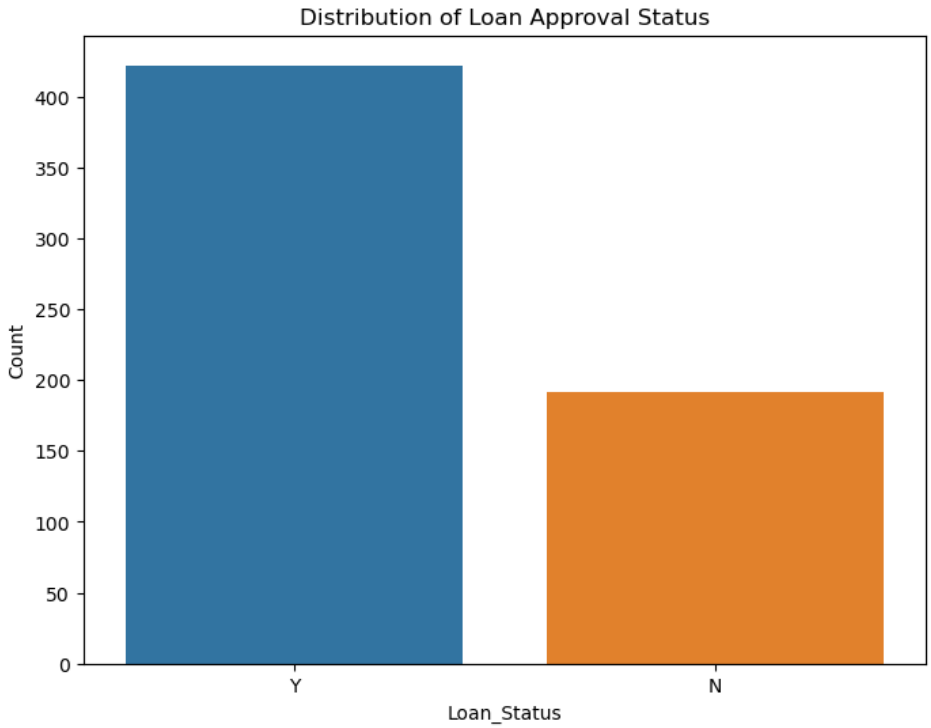
4 EDA and Conclusion

Based on the analysis, we observe that the Random Forest classifier achieved high accuracy on both the training and test datasets. The confusion matrices provide insights into the model's performance, indicating areas of correct and incorrect predictions.

1.Loan Approval Status Analysis:

```
    #Analyze the distribution of loan approval status to understand the overall approval rate.
# Example visualization: Bar plot of loan approval status
plt.figure(figsize=(8, 6))
sns.countplot(x='Loan_Status',data=loan_data)
plt.title('Distribution of Loan Approval Status')
plt.xlabel('Loan_Status')
plt.ylabel('Count')
plt.show()
'''Conclusion:
The majority of loans in the dataset are approved, indicating a relatively high approval rate.'''
```

Output:



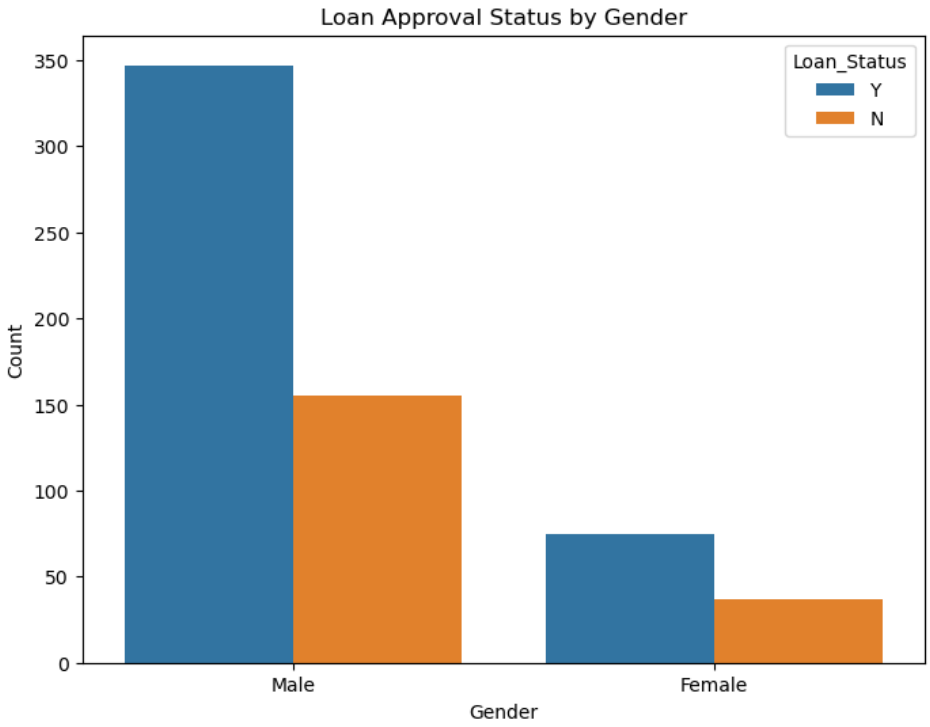
'Conclusion:majority of loans in the dataset are approved, indicating a relatively high approval rate.'

2.Demographic Analysis:

```
#Analyze demographic variables such as gender, marital status, and education level in relation to loan approval status.
# Example visualization: Bar plot of loan approval status by gender
plt.figure(figsize=(8, 6))
sns.countplot(x='Gender', hue='Loan_Status', data=loan_data)
plt.title('Loan Approval Status by Gender')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.legend(title='Loan_Status', loc='upper right')
```

```
plt.show()
'''Conclusion:
There is a slightly higher proportion of approved loans among males compared to females.'''
```

Output:



'Conclusion:is a slightly higher proportion of approved loans among males compared to females.'

3.Financial Analysis:

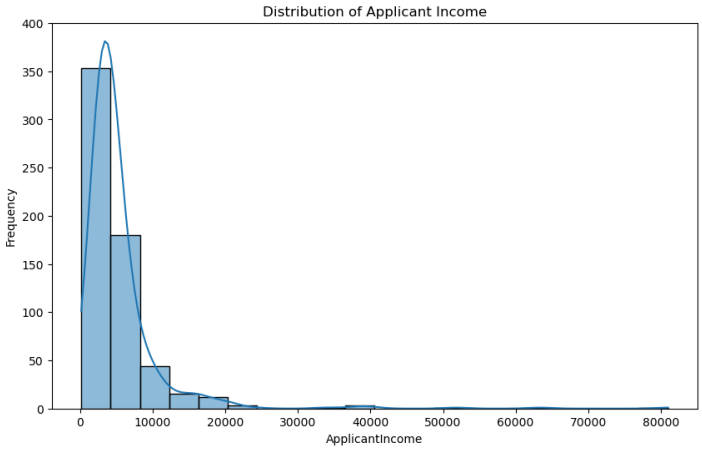
```
#Explore the distribution of applicant income, coapplicant income, and loan amount.
# Example visualization: Distribution of applicant income
plt.figure(figsize=(10, 6))
sns.histplot(loan_data['ApplicantIncome'], bins=20, kde=True)
plt.title('Distribution of Applicant Income')
plt.xlabel('ApplicantIncome')
plt.ylabel('Frequency')
plt.show()
'''Conclusion:
The distribution of applicant income is right-skewed, indicating a few individuals with high incomes.'''
```

Output:

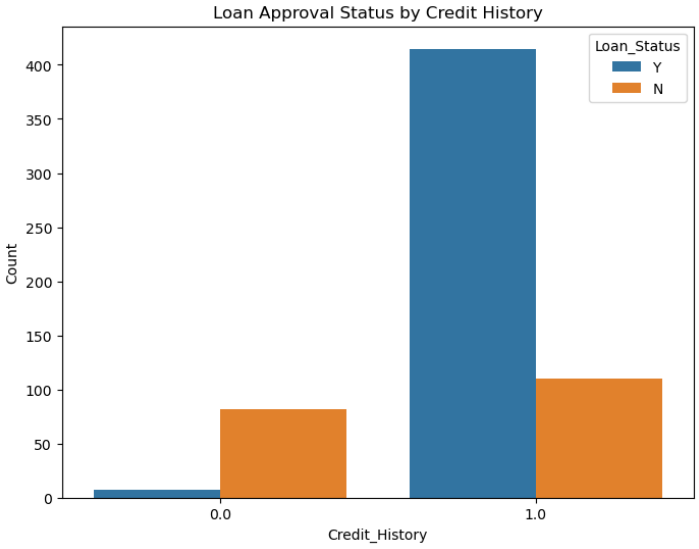
5.Credit History Analysis:

```
#Investigat'Conclusion:\nThe majority of loans in the dataset are approved, indicating a relatively high approval rate.'e the
# Example visualization: Bar plot of loan approval status by credit history
plt.figure(figsize=(8, 6))
sns.countplot(x='Credit_History', hue='Loan_Status', data=loan_data)
plt.title('Loan Approval Status by Credit History')
plt.xlabel('Credit_History')
plt.ylabel('Count')
plt.legend(title='Loan_Status', loc='upper right')
plt.show()
'''Conclusion:
Applicants with a credit history meeting guidelines are more likely to have their loans approved compared to those who do not
```

Output:



'Conclusion:distribution of applicant income is right-skewed, indicating a few individuals with high incomes.'



"'Conclusion:with a credit history meeting guidelines are more likely to have their loans approved compared to those who do not meet guidelines."'