PYTHON CASE STUDY

HARSHINI V

Data Processing with Pandas Case Study

Problem Statement:-

Automate the loan eligibility process (real-time) based on customer detail provided while filling the online application form. These details are Gender, Marital Status, Education, Number of Dependents, Income, Loan Amount, Credit History, and others.

The major aim of this notebook is to predict which of the customers will have their loan approved.

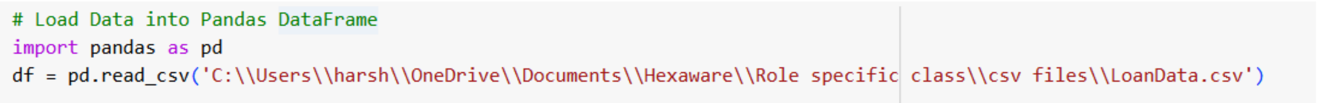
- Loading Data in Pandas DataFrame
- Printing rows of the Data
- Printing the column names of the DataFrame
- Summary of Data Frame
- Descriptive Statistical Measures of a DataFrame
- Missing Data Handing
- Sorting DataFrame values
- Merge Data Frames
- Apply Function
- By using the lambda operator
- Visualizing DataFrame

LOADING DATA IN PANDAS DATA FRAME:

```
import pandas as pd

df = pd.read_csv('C:\\Users\\harsh\\OneDrive\\Documents\\Hexaware\\Role specific class\\csv files\\LoanData.csv')
```

EXPLANATION:

The code imports the pandas library, a powerful Python library used for data manipulation and analysis. It then uses the read_csv method from pandas to load a CSV (Comma-Separated Values) file named LoanData.csv into a pandas DataFrame, which is a two-dimensional, tabular data structure similar to a spreadsheet or database table.



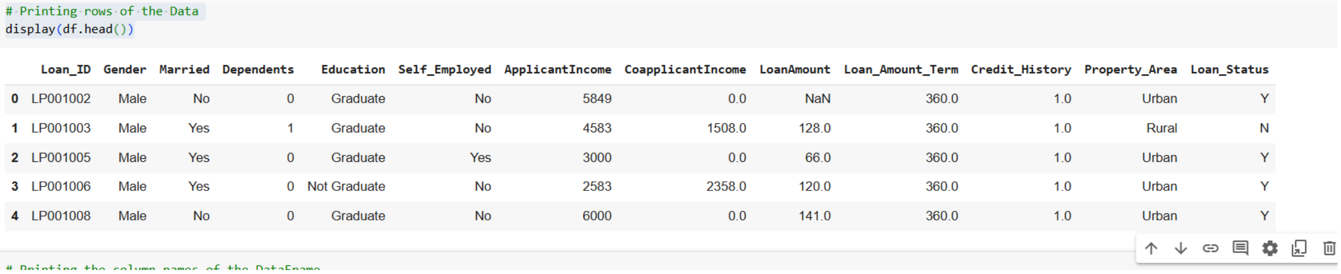
```
# Load Data into Pandas DataFrame
import pandas as pd
df = pd.read_csv('C:\\Users\\harsh\\OneDrive\\Documents\\Hexaware\\Role specific class\\csv files\\LoanData.csv')
```

PRINTING ROWS OF THE DATA:

display(df.head())

EXPLANATION:

The display(df.head()) command is used to view the first few rows of the DataFrame df in a clean and visually formatted manner, particularly in Jupyter Notebooks or similar environments that support enhanced outputs. The head() method retrieves the first 5 rows of the DataFrame by default, providing a quick preview of the data's structure, including the column names, data types, and some sample values.



```
# Printing rows of the Data
display(df.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

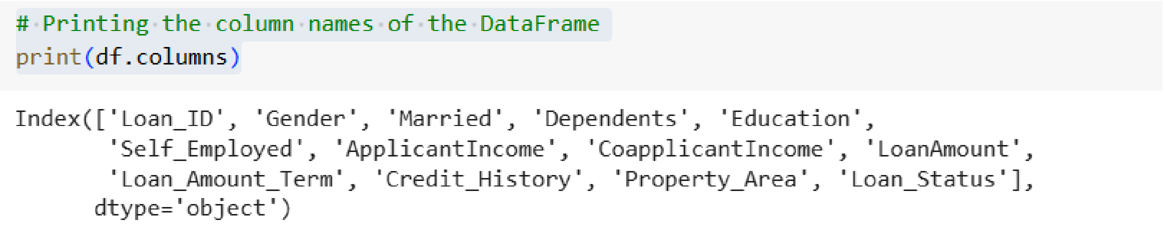
```
# Printing the column names of the DataFrame
```

PRINTING THE COLUMN NAMES OF THE DATAFRAME:

print(df.columns)

EXPLANATION:

The print(df.columns) command displays the column names of the DataFrame df. Each column in a DataFrame represents a specific feature or attribute of the dataset, and this command helps in understanding the structure of the data by listing all the available columns.



```
# Printing the column names of the DataFrame
print(df.columns)
```

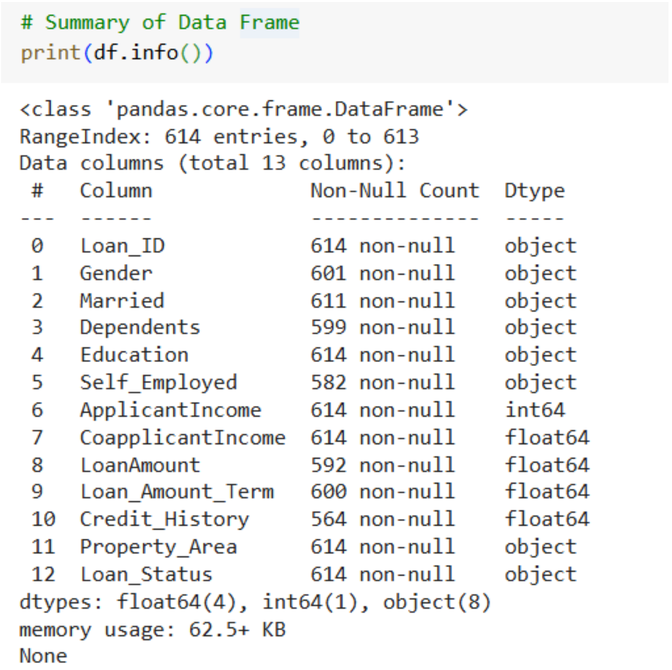
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

SUMMARY OF DATA FRAME:

print(df.info())

EXPLANATION:

The print(df.info()) command provides a concise summary of the DataFrame df, including details about its structure and content. The output lists the total number of rows and columns, the column names, their respective data types (int64, float64, object, etc.), and the non-null count for each column, which helps identify missing data.



```
# Summary of Data Frame
print(df.info())
```

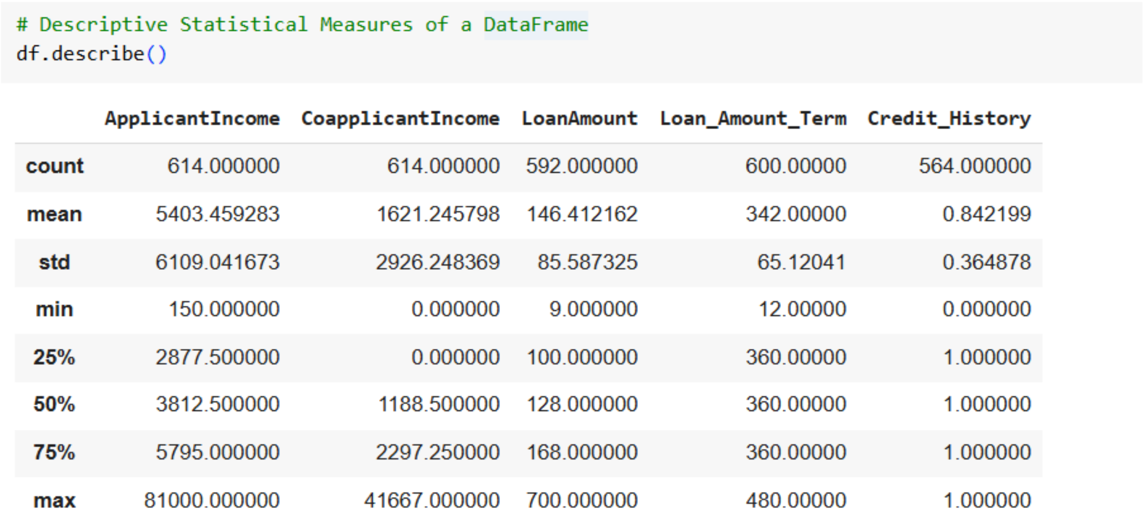
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            614 non-null    object 
 1   Gender             601 non-null    object 
 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
None
```

DESCRIPTIVE STATISTICAL MEASURES OF A DATAFRAME:

df.describe()

EXPLANATION:

The df.describe() command generates a summary of descriptive statistical measures for the numerical columns in the DataFrame df. The output includes key statistics such as the count (number of non-missing values), mean (average), standard deviation (spread of data), minimum and maximum values, and the 25th, 50th (median), and 75th percentiles. This summary helps in understanding the distribution and variability of the numerical data, identifying potential outliers, and gaining insights into the central tendency and range of the dataset.



```
# Descriptive Statistical Measures of a DataFrame
df.describe()
```

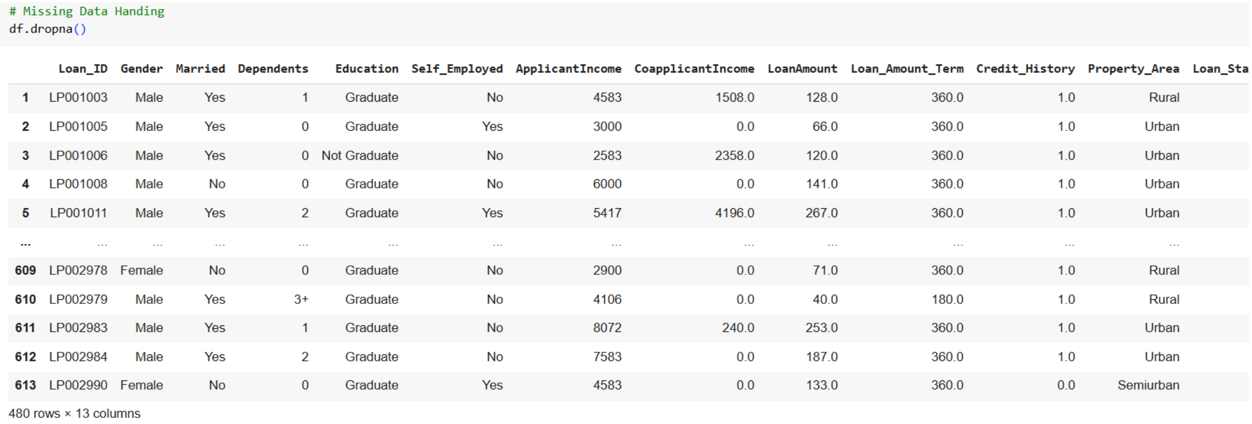
	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
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

MISSING DATA HANDING:

df.dropna()

EXPLANATION:

The df.dropna() command is used to handle missing data in the DataFrame df by removing any rows that contain NaN (Not a Number) or missing values. When this method is applied, it returns a new DataFrame with the rows containing missing values excluded, leaving only the complete cases. This is a simple and effective way to clean data, especially when the proportion of missing values is small and removing them will not significantly affect the dataset.



```
# Missing Data Handing
df.dropna()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Sta
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	
5	LP001011	Male	Yes	2	Graduate	Yes	5417	4196.0	267.0	360.0	1.0	Urban	
...	...	...	...	...	...	...	...	...	...	...	...	...	
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	360.0	1.0	Rural	
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	180.0	1.0	Rural	
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	1.0	Urban	
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	1.0	Urban	
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	360.0	0.0	Semiurban	

480 rows × 13 columns

SORTING DATAFRAME VALUES:

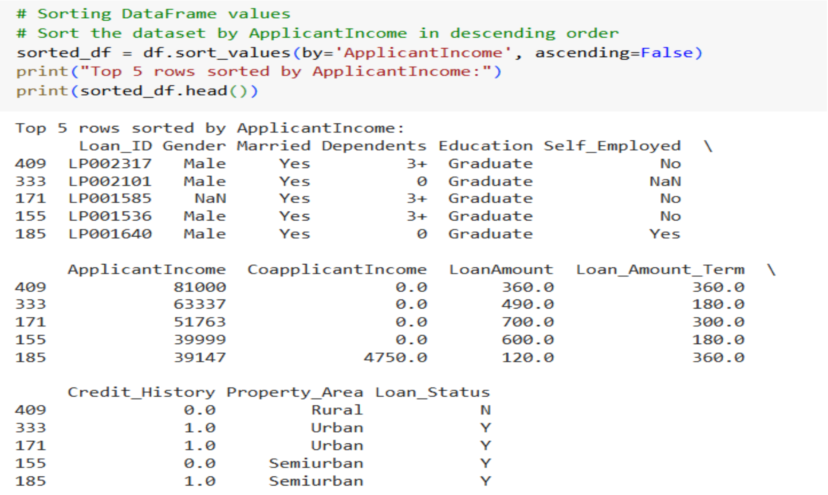
sorted_df = df.sort_values(by='ApplicantIncome', ascending=False)

print("Top 5 rows sorted by ApplicantIncome:")

print(sorted_df.head())

EXPLANATION:

The code sorts the DataFrame df based on the values in the ApplicantIncome column in descending order using the sort_values method. The by='ApplicantIncome' parameter specifies the column to sort by, and ascending=False ensures the sorting is done in descending order, meaning rows with the highest income values appear first. The sorted DataFrame is stored in the variable sorted_df. The print(sorted_df.head()) statement then displays the first five rows of this sorted DataFrame, showing the applicants with the highest incomes.



```
# Sorting DataFrame values
# Sort the dataset by ApplicantIncome in descending order
sorted_df = df.sort_values(by='ApplicantIncome', ascending=False)
print("Top 5 rows sorted by ApplicantIncome:")
print(sorted_df.head())
```

```
Top 5 rows sorted by ApplicantIncome:
      Loan_ID Gender Married Dependents Education Self_Employed  \
409  LP002317   Male     Yes         3+  Graduate            No
333  LP002101   Male     Yes          0  Graduate           NaN
171  LP001585    NaN     Yes         3+  Graduate            No
155  LP001536   Male     Yes         3+  Graduate            No
185  LP001640   Male     Yes          0  Graduate           Yes

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
409            81000                0.0       360.0             360.0
333            63337                0.0       490.0             180.0
171            51763                0.0       700.0             300.0
155            39999                0.0       600.0             180.0
185            39147             4750.0       120.0             360.0

     Credit_History Property_Area Loan_Status
409             0.0         Rural           N
333             1.0         Urban           Y
171             1.0         Urban           Y
155             0.0     Semiurban           Y
185             1.0     Semiurban           Y
```

MERGE DATA FRAMES:

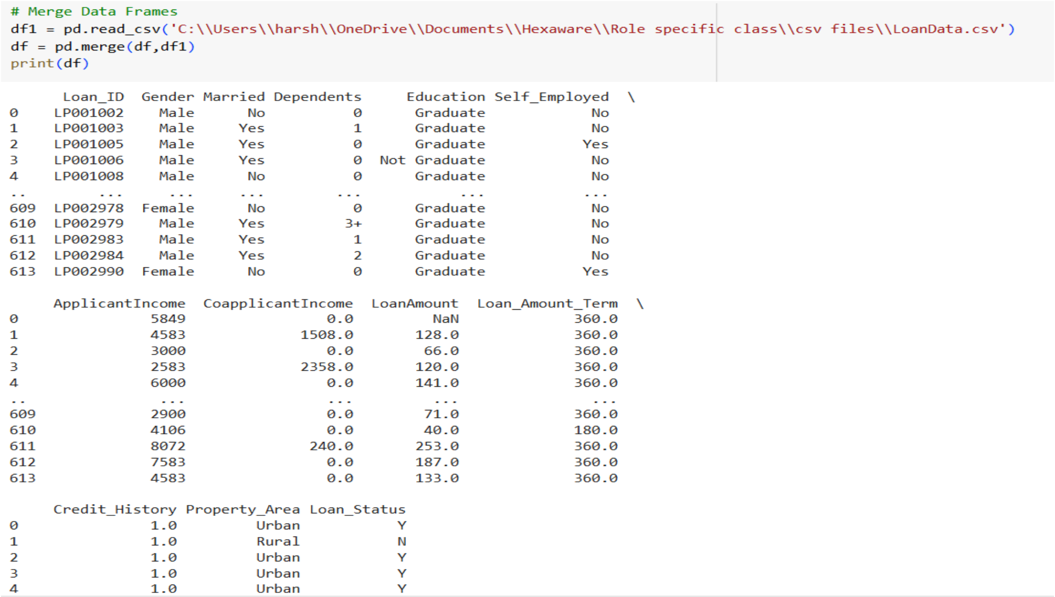
df1 = pd.read_csv('C:\\Users\\harsh\\OneDrive\\Documents\\Hexaware\\Role specific class\\csv files\\LoanData.csv')

df = pd.merge(df,df1)

print(df)

EXPLANATION:

The code demonstrates how to merge two DataFrames, df and df1, using the pd.merge() function from pandas. First, the df1 DataFrame is loaded from a CSV file, just like df. The pd.merge(df, df1) function combines the two DataFrames based on a common column or index. By default, merge() will perform an inner join, meaning only rows with matching values in the common columns from both DataFrames will be retained.



```
# Merge Data Frames
df1 = pd.read_csv('C:\\Users\\harsh\\OneDrive\\Documents\\Hexaware\\Role specific class\\csv files\\LoanData.csv')
df = pd.merge(df,df1)
print(df)
```

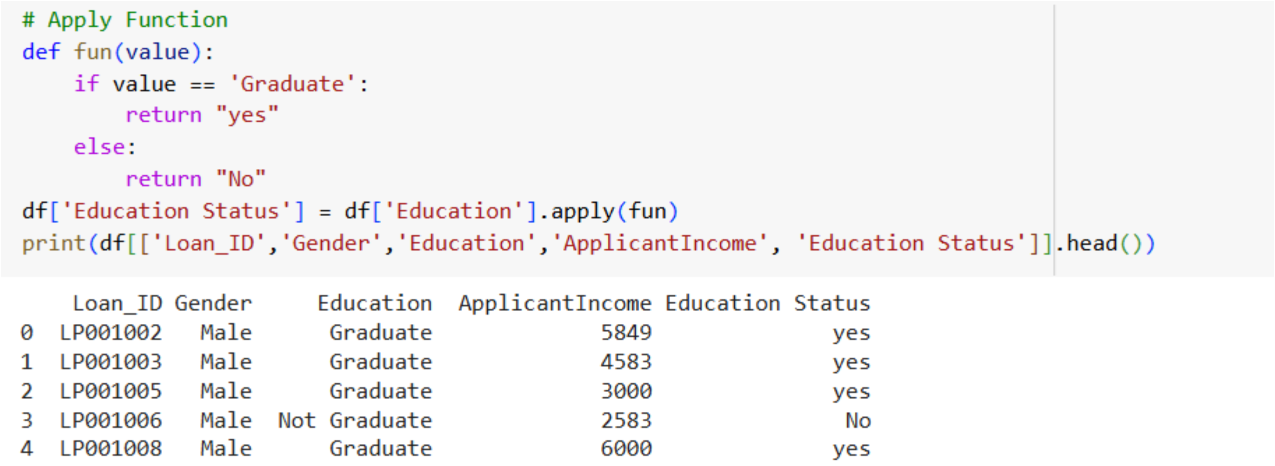
```
      Loan_ID  Gender Married Dependents     Education Self_Employed  \
0    LP001002    Male      No          0      Graduate            No   
1    LP001003    Male     Yes          1      Graduate            No   
2    LP001005    Male     Yes          0      Graduate           Yes   
3    LP001006    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  
```

APPLY FUNCTION:

```
def fun(value):
    if value == 'Graduate':
        return "yes"
    else:
        return "No"
df['Education Status'] = df['Education'].apply(fun)
print(df[['Loan_ID','Gender','Education','ApplicantIncome', 'Education Status']].head())
```

EXPLANATION:

The code defines a custom function fun(value) that takes an input value (in this case, a value from the Education column of the DataFrame). If the value is 'Graduate', the function returns "yes", otherwise, it returns "No". This function is then applied to the Education column of the DataFrame using the apply() method, which applies the function to each element in the column. Finally, the print(df[['Loan_ID','Gender','Education','ApplicantIncome', 'Education Status']].head()) statement displays the first five rows of the DataFrame, showing selected columns including the newly created Education Status column, providing a quick overview of the transformed data.



```
# Apply Function
def fun(value):
    if value == 'Graduate':
        return "yes"
    else:
        return "No"
df['Education Status'] = df['Education'].apply(fun)
print(df[['Loan_ID','Gender','Education','ApplicantIncome', 'Education Status']].head())
```

```
    Loan_ID Gender     Education  ApplicantIncome Education Status
0  LP001002   Male      Graduate             5849              yes
1  LP001003   Male      Graduate             4583              yes
2  LP001005   Male      Graduate             3000              yes
3  LP001006   Male  Not Graduate             2583               No
4  LP001008   Male      Graduate             6000              yes
```

BY USING THE LAMBDA OPERATOR:

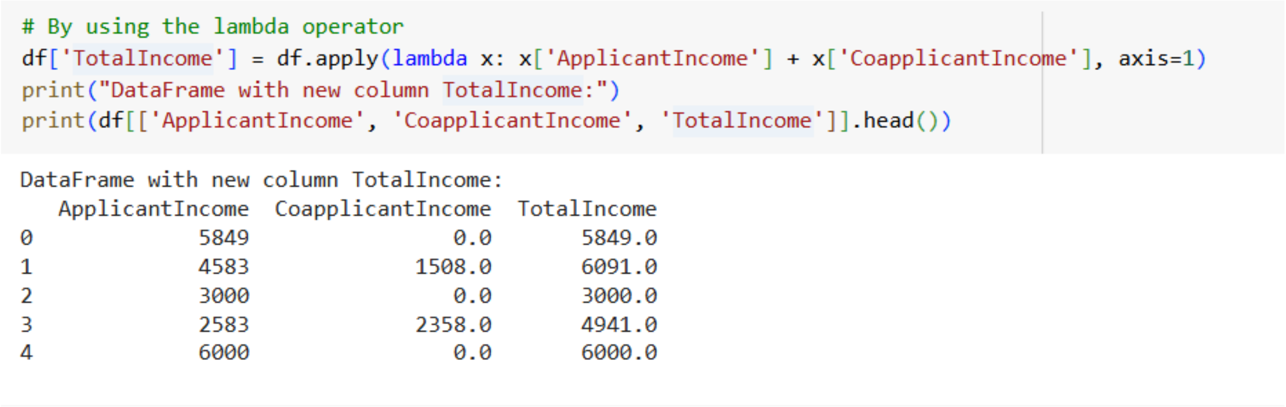
df['TotalIncome'] = df.apply(lambda x: x['ApplicantIncome'] + x['CoapplicantIncome'], axis=1)

print("DataFrame with new column TotalIncome:")

print(df[['ApplicantIncome', 'CoapplicantIncome', 'TotalIncome']].head())

EXPLANATION:

The code uses the lambda function to create a new column, TotalIncome, in the DataFrame df. The lambda x: x['ApplicantIncome'] + x['CoapplicantIncome'] expression defines an anonymous function that adds the values of ApplicantIncome and CoapplicantIncome for each row. The apply() method applies this function to each row of the DataFrame (specified by axis=1 for row-wise operation). The result is stored in a new column TotalIncome, which holds the sum of the applicant's and coapplicant's incomes. After adding the new column, the print(df[['ApplicantIncome', 'CoapplicantIncome', 'TotalIncome']].head()) statement displays the first five rows of the DataFrame, showing the original ApplicantIncome and CoapplicantIncome columns alongside the newly created TotalIncome column, which reflects the combined income for each applicant and coapplicant.



```
# By using the lambda operator
df['TotalIncome'] = df.apply(lambda x: x['ApplicantIncome'] + x['CoapplicantIncome'], axis=1)
print("DataFrame with new column TotalIncome:")
print(df[['ApplicantIncome', 'CoapplicantIncome', 'TotalIncome']].head())
```

```
DataFrame with new column TotalIncome:
   ApplicantIncome  CoapplicantIncome  TotalIncome
0             5849                0.0       5849.0
1             4583             1508.0       6091.0
2             3000                0.0       3000.0
3             2583             2358.0       4941.0
4             6000                0.0       6000.0
```

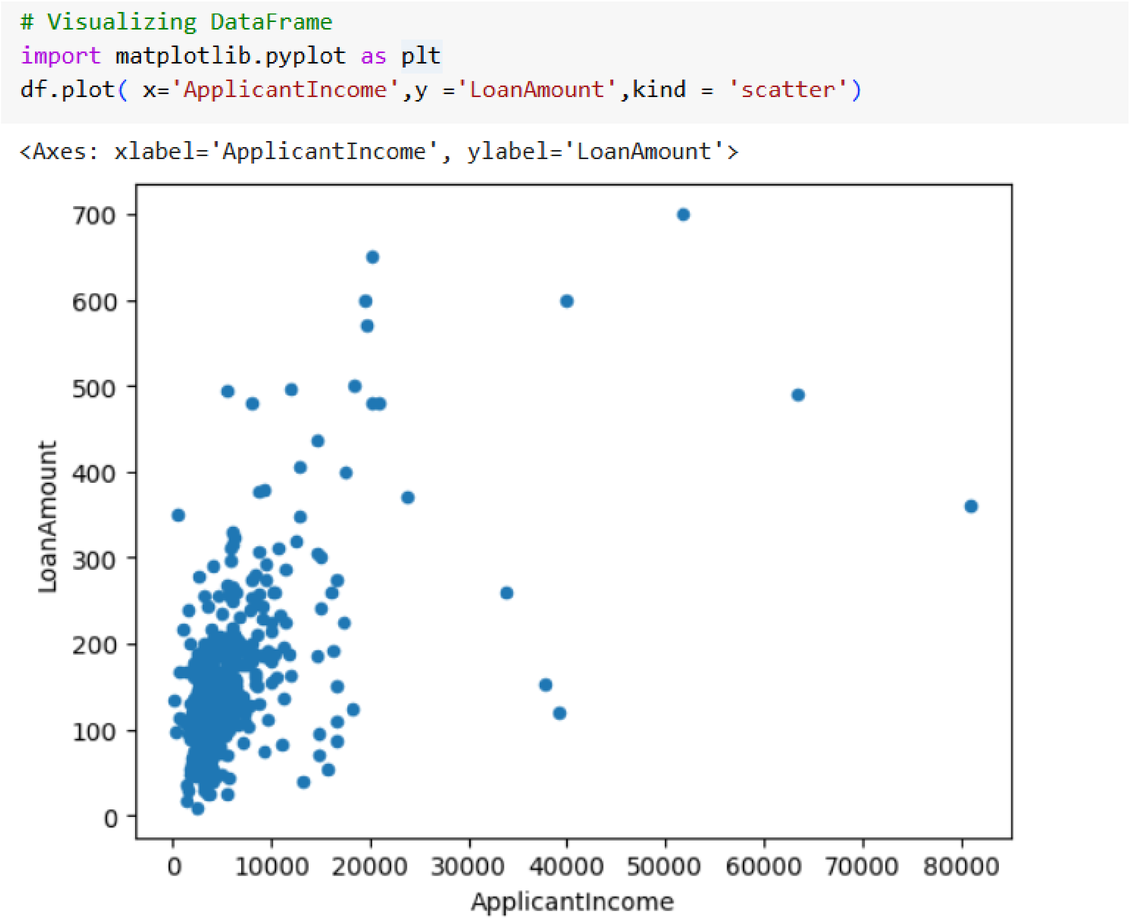
VISUALIZING DATAFRAME:

import matplotlib.pyplot as plt

df.plot(x='ApplicantIncome',y ='LoanAmount',kind = 'scatter')

EXPLANATION:

The code uses the matplotlib.pyplot library to create a scatter plot that visualizes the relationship between two columns, ApplicantIncome and LoanAmount, from the DataFrame df. The df.plot() method is called with parameters x='ApplicantIncome' and y='LoanAmount' to specify that the ApplicantIncome values should be plotted on the x-axis and the LoanAmount values on the y-axis. The kind='scatter' argument specifies that a scatter plot should be generated, which is useful for visualizing the correlation between two continuous variables. The scatter plot will show individual data points as dots, providing a clear view of how ApplicantIncome relates to the LoanAmount. By visualizing this relationship, you can identify any trends, clusters, or potential outliers in the data.



```python
# Visualizing DataFrame
import matplotlib.pyplot as plt
df.plot( x='ApplicantIncome',y ='LoanAmount',kind = 'scatter')
```

```
<Axes: xlabel='ApplicantIncome', ylabel='LoanAmount'>
```