10_ML - Finding missing value in data.ipynb

In [22]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

10.1 What is missing value

In [2]:
```
dataset = pd.read_csv('loan.csv')
```

In [3]:
```
dataset.head(3)
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	C
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	

In [5]:
```
# To know how many rows and columns are pesent in the data
dataset.shape
```

Out[5]:
```
(614, 13)
```

In [6]:
```
# isnull function returns True where missing data is peresent and returns false in
dataset.isnull()
```

Out[6]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
609	False	False	False	False	False	False	False
610	False	False	False	False	False	False	False
611	False	False	False	False	False	False	False
612	False	False	False	False	False	False	False
613	False	False	False	False	False	False	False

614 rows × 13 columns

In [13]:

```
```

Out[13]: 149

In [7]:

```
dataset.isnull().sum()
```

Out[7]:

```
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
```

In [11]:

```
# To determine the percentage null value of each Head
(dataset.isnull().sum()/dataset.shape[0])
```

Out[11]:
```
Loan_ID              0.000000
Gender               0.021173
Married              0.004886
Dependents           0.024430
Education            0.000000
Self_Employed        0.052117
ApplicantIncome      0.000000
CoapplicantIncome    0.000000
LoanAmount           0.035831
Loan_Amount_Term     0.022801
Credit_History       0.081433
Property_Area        0.000000
Loan_Status          0.000000
dtype: float64
```

In [12]:
```
(dataset.isnull().sum()/dataset.shape[0]) * 100
```

Out[12]:
```
Loan_ID              0.000000
Gender               2.117264
Married              0.488599
Dependents           2.442997
Education            0.000000
Self_Employed        5.211726
ApplicantIncome      0.000000
CoapplicantIncome    0.000000
LoanAmount           3.583062
Loan_Amount_Term     2.280130
Credit_History       8.143322
Property_Area        0.000000
Loan_Status          0.000000
dtype: float64
```

In [14]:
```
# To determine totall null value in the data
dataset.isnull().sum().sum()
```

Out[14]:
```
149
```

In [18]:
```
dataset.shape
```

Out[18]:
```
(614, 13)
```

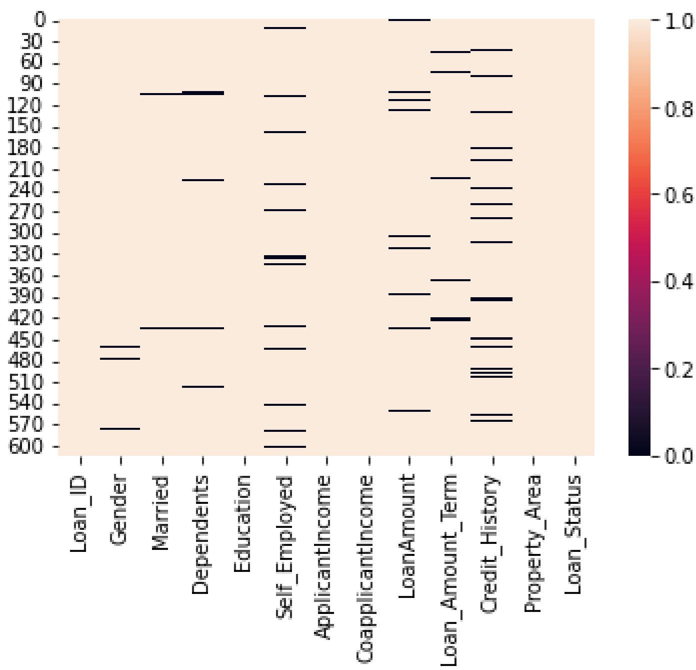
In [20]:
```
# To determine percentage totall null value in the data
# Total number of null data / total number of data * 100
dataset.isnull().sum().sum()/(dataset.shape[0] * dataset.shape[1])*100
```

Out[20]:
```
1.8667000751691305
```

In [21]:
```
# To check not null value in the data
dataset.notnull().sum()
```

```
Out[21]: Loan_ID              614
         Gender               601
         Married              611
         Dependents           599
         Education            614
         Self_Employed        582
         ApplicantIncome      614
         CoapplicantIncome    614
         LoanAmount           592
         Loan_Amount_Term     600
         Credit_History       564
         Property_Area        614
         Loan_Status          614
         dtype: int64
```

```
In [23]: # To graphically plot the null data
         sns.heatmap(dataset.notnull())
         plt.show()
```



10.2 How to handle missing values (Dropping)

Deleting in 2 ways:

1. If a column contains 50% missing value, then delete the whole column
2. Only delete the rows which are having missing values, instead of deleting whole column

Deleting a column from the data

```
In [25]: dataset.isnull().sum()
```

Out[25]:
```
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
```

We will delete Credit_History column as it contains more missing values

In [27]:
```
# Inplace function will enable to make changes in the original datasheet that is da
# It will write changes in the excisting file i.e., load.csv
dataset.drop(columns=['Credit_History'], inplace=True)
```

In [28]:
```
dataset.isnull().sum()
```

Out[28]:
```
Loan_ID              0
Gender              13
Married              3
Dependents          15
Education            0
Self_Employed       32
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount          22
Loan_Amount_Term    14
Property_Area        0
Loan_Status          0
dtype: int64
```

In [29]:
```
dataset.shape
```

Out[29]:
```
(614, 12)
```

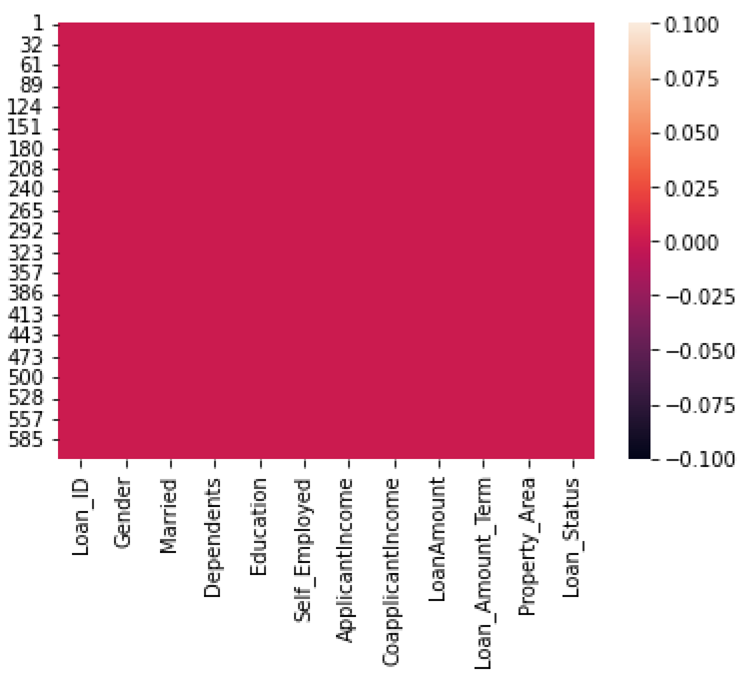
Deleting rows containing null values

In [30]:
```
# Again we use inplace function to write the changes in the same datasheet instead
dataset.dropna(inplace=True)
```

In [31]:
```
dataset.isnull().sum()
```

Out[31]:
```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Property_Area        0
Loan_Status          0
dtype: int64
```

In [32]:
```
sns.heatmap(dataset.isnull())
plt.show()
```



In [34]:
```
dataset.shape
```

Out[34]:
```
(523, 12)
```

To check how much data has been dropped (deleted)

In [37]:
```
((614-523)/614)*100
```

Out[37]:
```
14.82084690553746
```

14% data is lost

10.3 Handling Missing Values (Imputing Category Data)

While dropping the data can be harmful as it may contain essential data, so instead of deleting we will fill the data where the missing values are present

We will import the orginial data of loan.csv, as we have dropped missing data and overwrite the changes in the above data

In [38]:
```
dataset = pd.read_csv('loan.csv')
```

In [39]:
```
dataset.head(3)
```

Out[39]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	C
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	

In [40]:
```
dataset.isnull().sum()
```

Out[40]:
```
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
```

In [42]:
```
# It will fill all the data using the number provided i.e., 10
# this method is not recommended as it will fill dummy data, lead to wrong insight
dataset.fillna(10)
```

Out[42]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
0	LP001002	Male	No	0	Graduate	No	5849
1	LP001003	Male	Yes	1	Graduate	No	4583
2	LP001005	Male	Yes	0	Graduate	Yes	3000
3	LP001006	Male	Yes	0	Not Graduate	No	2583
4	LP001008	Male	No	0	Graduate	No	6000
...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900
610	LP002979	Male	Yes	3+	Graduate	No	4106
611	LP002983	Male	Yes	1	Graduate	No	8072
612	LP002984	Male	Yes	2	Graduate	No	7583
613	LP002990	Female	No	0	Graduate	Yes	4583

614 rows × 13 columns

So we will fill data wisely, so we will first determine the datatype String data type is called object data in ML Data is of two types:

1. Numerical data
2. Categorical data - string data (object type data) Filling in categorical data:
3. Backward filling
4. Forward filling
5. Mod filling

In [43]:

```
dataset.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
```

In [47]:

```python
# Backward filling - Back data will filled , forexample Loan Amount first row is fi
# nichay wala data oper a k fill ho jaye ga
dataset.fillna(method='bfill')
```

Out[47]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
0	LP001002	Male	No	0	Graduate	No	5849
1	LP001003	Male	Yes	1	Graduate	No	4583
2	LP001005	Male	Yes	0	Graduate	Yes	3000
3	LP001006	Male	Yes	0	Not Graduate	No	2583
4	LP001008	Male	No	0	Graduate	No	6000
...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900
610	LP002979	Male	Yes	3+	Graduate	No	4106
611	LP002983	Male	Yes	1	Graduate	No	8072
612	LP002984	Male	Yes	2	Graduate	No	7583
613	LP002990	Female	No	0	Graduate	Yes	4583

614 rows × 13 columns

In [48]:

```python
# Forward filling - oper wala data nicha a k fill ho jaye ga
# by default filling is row wise
dataset.fillna(method='ffill')
```

Out[48]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
0	LP001002	Male	No	0	Graduate	No	5849
1	LP001003	Male	Yes	1	Graduate	No	4583
2	LP001005	Male	Yes	0	Graduate	Yes	3000
3	LP001006	Male	Yes	0	Not Graduate	No	2583
4	LP001008	Male	No	0	Graduate	No	6000
...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900
610	LP002979	Male	Yes	3+	Graduate	No	4106
611	LP002983	Male	Yes	1	Graduate	No	8072
612	LP002984	Male	Yes	2	Graduate	No	7583
613	LP002990	Female	No	0	Graduate	Yes	4583

614 rows × 13 columns

In [50]:

```
# by default filling is row wise - So fill data column wise, we will use axis
dataset.fillna(method='ffill', axis=1)
```

Out[50]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
0	LP001002	Male	No	0	Graduate	No	5849
1	LP001003	Male	Yes	1	Graduate	No	4583
2	LP001005	Male	Yes	0	Graduate	Yes	3000
3	LP001006	Male	Yes	0	Not Graduate	No	2583
4	LP001008	Male	No	0	Graduate	No	6000
...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900
610	LP002979	Male	Yes	3+	Graduate	No	4106
611	LP002983	Male	Yes	1	Graduate	No	8072
612	LP002984	Male	Yes	2	Graduate	No	7583
613	LP002990	Female	No	0	Graduate	Yes	4583

614 rows × 13 columns

- In mode data filling, you will most repeatitive data in missing contents
- fill the missing data in Gender column

Fill particular column containing missing value by mode method

In [51]:
```
dataset['Gender'].mode()
```

Out[51]:
```
0    Male
Name: Gender, dtype: object
```

In [52]:
```
dataset['Gender'].mode()[0]
```

Out[52]:
```
'Male'
```

In [54]:
```
dataset['Gender'].fillna(dataset['Gender'].mode()[0], inplace=True)
```

In [55]:
```
dataset.isnull().sum()
```

Out[55]:
```
Loan_ID               0
Gender                0
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
```

Fill all columns containing missing value by mode method

1. First you will collect all object datatype

In [57]:
```
dataset.select_dtypes(include='object')
```

Out[57]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Property_Area	L
0	LP001002	Male	No	0	Graduate	No	Urban	
1	LP001003	Male	Yes	1	Graduate	No	Rural	
2	LP001005	Male	Yes	0	Graduate	Yes	Urban	
3	LP001006	Male	Yes	0	Not Graduate	No	Urban	
4	LP001008	Male	No	0	Graduate	No	Urban	
...	...	...	...	...	...	...	...	
609	LP002978	Female	No	0	Graduate	No	Rural	
610	LP002979	Male	Yes	3+	Graduate	No	Rural	
611	LP002983	Male	Yes	1	Graduate	No	Urban	
612	LP002984	Male	Yes	2	Graduate	No	Urban	
613	LP002990	Female	No	0	Graduate	Yes	Semiurban	

614 rows × 8 columns

In [58]:

```
dataset.select_dtypes(include='object').isnull()
```

Out[58]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Property_Area	Lo
0	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	
609	False	False	False	False	False	False	False	
610	False	False	False	False	False	False	False	
611	False	False	False	False	False	False	False	
612	False	False	False	False	False	False	False	
613	False	False	False	False	False	False	False	

614 rows × 8 columns

In [59]:

```
dataset.select_dtypes(include='object').isnull().sum()
```

```
Out[59]: Loan_ID           0
         Gender            0
         Married           3
         Dependents       15
         Education         0
         Self_Employed    32
         Property_Area     0
         Loan_Status       0
         dtype: int64
```

```
In [60]: dataset.select_dtypes(include='object').columns
```

```
Out[60]: Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
                'Self_Employed', 'Property_Area', 'Loan_Status'],
               dtype='object')
```

```
In [61]: for i in dataset.select_dtypes(include='object').columns:
             print(i)
```

```
Loan_ID
Gender
Married
Dependents
Education
Self_Employed
Property_Area
Loan_Status
```

```
In [63]: for i in dataset.select_dtypes(include='object').columns:
             #dataset['Gender'].fillna(dataset['Gender'].mode()[0], inplace=True)
             dataset[i].fillna(dataset[i].mode()[0], inplace=True)
```

```
In [64]: dataset.isnull().sum()
```

```
Out[64]: Loan_ID               0
         Gender                0
         Married               0
         Dependents            0
         Education             0
         Self_Employed         0
         ApplicantIncome       0
         CoapplicantIncome     0
         LoanAmount           22
         Loan_Amount_Term     14
         Credit_History       50
         Property_Area         0
         Loan_Status           0
         dtype: int64
```

So all object data has been filled and only numerical data is left to be filled

10.4 Handling Missing Values (Scikit-learn)

Import fresh datasheet

In [65]:
```
dataset = pd.read_csv('loan.csv')
```

In [66]:
```
dataset.head(3)
```

Out[66]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	C
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	

In [67]:
```
dataset.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
```

In [69]:
```
# To show numerical data type (float)
dataset.select_dtypes(include='float64')
```

Out[69]:

	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	0.0	NaN	360.0	1.0
1	1508.0	128.0	360.0	1.0
2	0.0	66.0	360.0	1.0
3	2358.0	120.0	360.0	1.0
4	0.0	141.0	360.0	1.0
...	...	...	...	...
609	0.0	71.0	360.0	1.0
610	0.0	40.0	180.0	1.0
611	240.0	253.0	360.0	1.0
612	0.0	187.0	360.0	1.0
613	0.0	133.0	360.0	0.0

614 rows × 4 columns

In [70]:

```
dataset.select_dtypes(include='float64').columns
```

Out[70]:

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

Find the missing values using scikit learn

In [71]:

```
from sklearn.impute import SimpleImputer
```

- sklearn provide variety of options to fill the data. for example fill the data by mean, most fequency (mode), median
- We are now filling the data by using mean method

In [74]:

```
si = SimpleImputer(strategy='mean')
si.fit_transform(dataset[['CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term',
       'Credit_History']])
```

Out[74]:

```
array([[0.00000000e+00, 1.46412162e+02, 3.60000000e+02, 1.00000000e+00],
       [1.50800000e+03, 1.28000000e+02, 3.60000000e+02, 1.00000000e+00],
       [0.00000000e+00, 6.60000000e+01, 3.60000000e+02, 1.00000000e+00],
       ...,
       [2.40000000e+02, 2.53000000e+02, 3.60000000e+02, 1.00000000e+00],
       [0.00000000e+00, 1.87000000e+02, 3.60000000e+02, 1.00000000e+00],
       [0.00000000e+00, 1.33000000e+02, 3.60000000e+02, 0.00000000e+00]])
```

In [82]:

```
# To convert this data into csv sheet
si = SimpleImputer(strategy='mean')
```

```
arr = si.fit_transform(dataset[['CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Ter
       'Credit_History']])
```

In [89]:
```
new_dataset = pd.DataFrame(arr, columns=dataset.select_dtypes(include='float64').co
```

In [90]:
```
new_dataset.isnull().sum()
```

Out[90]:
```
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
dtype: int64
```

In [91]:
```
new_dataset
```

Out[91]:

	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	0.0	146.412162	360.0	1.0
1	1508.0	128.000000	360.0	1.0
2	0.0	66.000000	360.0	1.0
3	2358.0	120.000000	360.0	1.0
4	0.0	141.000000	360.0	1.0
...	...	...	...	...
609	0.0	71.000000	360.0	1.0
610	0.0	40.000000	180.0	1.0
611	240.0	253.000000	360.0	1.0
612	0.0	187.000000	360.0	1.0
613	0.0	133.000000	360.0	0.0

614 rows × 4 columns

In [92]:
```
dataset['LoanAmount'].mean()
```

Out[92]:
```
146.41216216216216
```

11_ML - One Hot Encoding and Dummy Variables

- To convert categorical data into numerical data, as ML use mathemetical formulas in its model so all string data should be converted into numerical data
- It is normally used when number of data is lees

In [1]:
```
import pandas as pd
```

In [2]:
```
dataset = pd.read_csv('loan.csv')
```

In [3]:
```
dataset.head(3)
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	C
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	

11.1 Find Missing Values and Handle it

In [4]:
```
dataset.isnull().sum()
```

Out[4]:
```
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
```

In [8]:
```
dataset['Gender'].mode()[0]
```

Out[8]:
```
'Male'
```

In [11]:
```
# Gender column contains missing values so we will fill it using mode method
dataset['Gender'].fillna(dataset['Gender'].mode()[0],inplace=True)
```

In [12]:
```
dataset.isnull().sum()
```

Out[12]:
```
Loan_ID              0
Gender               0
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
```

In [13]:
```
# Married column contains missing values so we will fill it using mode method
dataset['Married'].fillna(dataset['Married'].mode()[0],inplace=True)
```

In [14]:
```
dataset.isnull().sum()
```

Out[14]:
```
Loan_ID              0
Gender               0
Married              0
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
```

11.2 Use One Hot Code to handle missing values

First separate Gender and Married data to perform encoding

In [16]:
```
en_data = dataset[['Gender', 'Married']]
en_data
```

Out[16]:

	Gender	Married
0	Male	No
1	Male	Yes
2	Male	Yes
3	Male	Yes
4	Male	No
...	...	...
609	Female	No
610	Male	Yes
611	Male	Yes
612	Male	Yes
613	Female	No

614 rows × 2 columns

In [17]:

```
pd.get_dummies(en_data)
```

Out[17]:

	Gender_Female	Gender_Male	Married_No	Married_Yes
0	0	1	1	0
1	0	1	0	1
2	0	1	0	1
3	0	1	0	1
4	0	1	1	0
...	...	...	...	...
609	1	0	1	0
610	0	1	0	1
611	0	1	0	1
612	0	1	0	1
613	1	0	1	0

614 rows × 4 columns

In [18]:

```
pd.get_dummies(en_data).info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Gender_Female  614 non-null    uint8
 1   Gender_Male    614 non-null    uint8
 2   Married_No     614 non-null    uint8
 3   Married_Yes    614 non-null    uint8
dtypes: uint8(4)
memory usage: 2.5 KB
```

In [19]:
```
from sklearn.preprocessing import OneHotEncoder
```

In [20]:
```
# fit_transfrom() converts categorical data into numerical data
ohe = OneHotEncoder()
ohe.fit_transform(en_data)
```

Out[20]:
```
<614x4 sparse matrix of type '<class 'numpy.float64'>'
	with 1228 stored elements in Compressed Sparse Row format>
```

sparse matrix contains data in 0 and 1 form

In [25]:
```
ohe = OneHotEncoder()
ar = ohe.fit_transform(en_data).toarray()
ar
```

Out[25]:
```
array([[0., 1., 1., 0.],
       [0., 1., 0., 1.],
       [0., 1., 0., 1.],
       ...,
       [0., 1., 0., 1.],
       [0., 1., 0., 1.],
       [1., 0., 1., 0.]])
```

In [27]:
```
# Convert the array data into dataframe

pd.DataFrame(ar, columns=['Gender_Female', 'Gender_Male', 'Married_No', 'Married_Ye
```

Out[27]:

	Gender_Female	Gender_Male	Married_No	Married_Yes
0	0.0	1.0	1.0	0.0
1	0.0	1.0	0.0	1.0
2	0.0	1.0	0.0	1.0
3	0.0	1.0	0.0	1.0
4	0.0	1.0	1.0	0.0
...	...	...	...	...
609	1.0	0.0	1.0	0.0
610	0.0	1.0	0.0	1.0
611	0.0	1.0	0.0	1.0
612	0.0	1.0	0.0	1.0
613	1.0	0.0	1.0	0.0

614 rows × 4 columns

You can see out of 2 column 4 column are produced, so to avoid this, we will use 'drop first' to delete first column after encoding i.e. Gender_Female and Married_No, so use it as follows:

In [28]:

```
ohe = OneHotEncoder(drop='first')
ar = ohe.fit_transform(en_data).toarray()
ar
```

Out[28]:

```
array([[1., 0.],
       [1., 1.],
       [1., 1.],
       ...,
       [1., 1.],
       [1., 1.],
       [0., 0.]])
```

In [29]:

```
# Convert the array data into dataframe

pd.DataFrame(ar, columns=['Gender_Male','Married_Yes'])
```

Out[29]:

	Gender_Male	Married_Yes
0	1.0	0.0
1	1.0	1.0
2	1.0	1.0
3	1.0	1.0
4	1.0	0.0
...	...	...
609	0.0	0.0
610	1.0	1.0
611	1.0	1.0
612	1.0	1.0
613	0.0	0.0

614 rows × 2 columns

In [ ]:

In [ ]:

12_ML - Label Encoder

12.1 Label encoding on Nominal data

In [3]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
```

In [5]:
```
df = pd.DataFrame({'name':['Rashid', 'Lion', 'Computer', 'Gym', 'Plant']})
df
```

Out[5]:

	name
0	Rashid
1	Lion
2	Computer
3	Gym
4	Plant

In [7]:
```
le = LabelEncoder()
df['en_name'] = le.fit_transform(df['name'])
```

In [8]:
```
df
```

Out[8]:

	name	en_name
0	Rashid	4
1	Lion	2
2	Computer	0
3	Gym	1
4	Plant	3

Now work on real time data

In [9]:
```
dataset = pd.read_csv('loan.csv')
```

In [10]:
```
dataset.head(3)
```

Out[10]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	C
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	

In [14]:

```
# To check number of data
dataset['Property_Area'].unique()
```

Out[14]:

```
array(['Urban', 'Rural', 'Semiurban'], dtype=object)
```

In [12]:

```
la = LabelEncoder()
la.fit(dataset['Property_Area'])
```

Out[12]:

```
▾ LabelEncoder
LabelEncoder()
```

In [13]:

```
la.transform(dataset['Property_Area'])
```

Out[13]:

```
array([2, 0, 2, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2,
       1, 0, 1, 1, 1, 2, 2, 1, 2, 2, 0, 1, 0, 2, 2, 1, 2, 1, 2, 2, 2, 1,
       2, 2, 2, 2, 2, 1, 1, 1, 1, 2, 2, 1, 1, 0, 2, 2, 2, 2, 0, 0, 1, 1,
       2, 2, 2, 1, 2, 1, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1,
       2, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 2, 1, 2, 1, 2, 2, 2, 0, 2, 1,
       2, 1, 0, 1, 1, 0, 1, 2, 0, 2, 0, 1, 1, 1, 0, 0, 0, 0, 2, 0, 2, 2,
       1, 1, 1, 1, 0, 2, 1, 0, 0, 2, 1, 1, 2, 1, 2, 2, 0, 1, 0, 0, 2, 0,
       2, 1, 0, 2, 0, 1, 1, 2, 1, 0, 2, 0, 0, 0, 1, 1, 0, 2, 0, 1, 1, 0,
       0, 1, 1, 2, 2, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 2, 1, 0, 1, 0, 2,
       1, 2, 1, 1, 2, 2, 1, 1, 2, 0, 2, 1, 1, 1, 2, 0, 2, 1, 0, 1, 1, 1,
       2, 1, 1, 1, 1, 0, 2, 1, 1, 0, 1, 0, 0, 1, 1, 0, 2, 2, 0, 1, 0, 2,
       2, 0, 1, 2, 2, 2, 1, 2, 1, 2, 0, 1, 2, 0, 0, 2, 0, 1, 2, 1, 1, 0,
       1, 0, 1, 2, 0, 2, 2, 2, 0, 1, 1, 1, 1, 2, 1, 0, 2, 1, 2, 2, 0, 0,
       1, 0, 1, 0, 0, 1, 2, 2, 1, 2, 1, 2, 0, 2, 2, 1, 0, 2, 0, 2, 0, 2,
       0, 0, 1, 1, 0, 0, 0, 2, 1, 2, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 2, 2,
       2, 1, 2, 2, 2, 1, 0, 0, 2, 1, 0, 0, 2, 1, 0, 1, 0, 2, 1, 0, 1, 0,
       0, 0, 1, 2, 0, 2, 2, 1, 1, 1, 2, 2, 0, 0, 1, 0, 1, 0, 1, 1, 0, 2,
       2, 2, 0, 1, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 0, 0, 0, 2, 1, 2, 1,
       2, 2, 0, 1, 2, 0, 1, 1, 0, 1, 2, 0, 1, 0, 1, 2, 0, 0, 1, 2, 2, 2,
       0, 1, 0, 2, 2, 2, 1, 0, 0, 1, 0, 2, 1, 0, 1, 1, 2, 1, 1, 2, 2, 0,
       1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 2, 0, 0, 1, 1, 2, 2, 0, 1, 1, 2,
       0, 1, 1, 0, 2, 1, 1, 2, 1, 0, 1, 2, 0, 0, 1, 1, 1, 2, 0, 0, 1, 1,
       1, 0, 0, 2, 1, 2, 1, 2, 0, 1, 0, 1, 0, 2, 1, 0, 0, 1, 1, 0, 1, 0,
       2, 2, 2, 2, 0, 1, 2, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 2, 0, 1, 2, 0, 2, 1, 0, 0, 1, 1, 1, 2, 1, 0, 1, 0, 1, 0,
       0, 0, 2, 2, 0, 1, 2, 1, 1, 1, 1, 1, 0, 1, 2, 0, 2, 0, 2, 2, 2, 2,
       2, 1, 1, 2, 1, 2, 0, 2, 1, 2, 1, 0, 0, 0, 2, 1, 1, 1, 1, 1, 1, 0,
       2, 0, 0, 1, 0, 2, 2, 0, 2, 0, 1, 2, 1, 0, 0, 0, 0, 2, 2, 1])
```

In [15]:

```
# to replace the property data with encoding data
dataset['Property_Area'] = la.transform(dataset['Property_Area'])
```

In [18]:
```
dataset.head(3)
```

Out[18]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	C
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	

12.1 Label encoding on Ordinal data

12.1.1 Ordincal encoding through Cyclic Line

In [21]:
```
dfo = pd.DataFrame({'size': ['s', 'm', 'l', 'xl', 'xxl', 's', 's', 's', 'xl', 'm',
dfo.head(3)
```

Out[21]:

	size
0	s
1	m
2	l

In [22]:
```
ord_data = [['s', 'm', 'l', 'xl', 'xxl']]
```

In [24]:
```
from sklearn.preprocessing import OrdinalEncoder
```

In [25]:
```
# oe = OrdinalEncoder() : This will encode the data alphabatically
oe = OrdinalEncoder(categories=ord_data)
oe.fit(dfo[['size']])
```

Out[25]:

```
▾                    OrdinalEncoder
OrdinalEncoder(categories=[['s', 'm', 'l', 'xl', 'xxl']])
```

In [27]:
```
oe.transform(dfo[['size']])
```

Out[27]:
```
array([[0.],
       [1.],
       [2.],
       [3.],
       [4.],
       [0.],
       [0.],
       [0.],
       [3.],
       [1.],
       [2.]])
```

In [29]:
```
dfo['size_en'] = oe.transform(dfo[['size']])
dfo
```

Out[29]:

	size	size_en
0	s	0.0
1	m	1.0
2	l	2.0
3	xl	3.0
4	xxl	4.0
5	s	0.0
6	s	0.0
7	s	0.0
8	xl	3.0
9	m	1.0
10	l	2.0

12.1.2 Ordincal encoding through Map function

In [30]:
```
# In map function, you can manually assign numbers to each data type, for example
# You can assign any number
ord_data1 = {'s':0, 'm':1, 'l':2, 'xl':3, 'xxl':4}
```

In [32]:
```
dfo['size'].map(ord_data1)
```

Out[32]:
```
0     0
1     1
2     2
3     3
4     4
5     0
6     0
7     0
8     3
9     1
10    2
Name: size, dtype: int64
```

In [33]:
```
dfo['size_en_map'] = dfo['size'].map(ord_data1)
dfo
```

Out[33]:

	size	size_en	size_en_map
0	s	0.0	0
1	m	1.0	1
2	l	2.0	2
3	xl	3.0	3
4	xxl	4.0	4
5	s	0.0	0
6	s	0.0	0
7	s	0.0	0
8	xl	3.0	3
9	m	1.0	1
10	l	2.0	2

12.2 Perform Ordinal Encoding on big data

In [35]:

```
dataset = pd.read_csv('loan.csv')
```

In [36]:

```
dataset.head(3)
```

Out[36]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	C
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	

In [37]:

```
dataset['Property_Area'].unique()
```

Out[37]:

```
array(['Urban', 'Rural', 'Semiurban'], dtype=object)
```

In [40]:

```
# if there is nan (missing data)m, you can fill it
# if the data is categorical then you should do mode filling
dataset['Property_Area'].fillna(dataset['Property_Area'].mode()[0], inplace=True)
```

In [42]:

```
en_data_loan = [['Urban', 'Rural', 'Semiurban']]
```

In [45]:

```
oen = OrdinalEncoder(categories=en_data_loan)
oen.fit_transform(dataset[['Property_Area']])
```

```
Out[45]: array([[0.],
                [1.],
                [0.],
                [0.],
                [0.],
                [0.],
                [0.],
                [2.],
                [0.],
                [2.],
                [0.],
                [0.],
                [0.],
                [1.],
                [0.],
                [0.],
                [0.],
                [0.],
                [1.],
                [0.],
                [0.],
                [0.],
                [2.],
                [1.],
                [2.],
                [2.],
                [2.],
                [0.],
                [0.],
                [2.],
                [0.],
                [0.],
                [1.],
                [2.],
                [1.],
                [0.],
                [0.],
                [2.],
                [0.],
                [2.],
                [0.],
                [0.],
                [0.],
                [2.],
                [0.],
                [0.],
                [0.],
                [0.],
                [0.],
                [2.],
                [2.],
                [2.],
                [2.],
                [0.],
                [0.],
                [2.],
```

```
[2.],
[1.],
[0.],
[0.],
[0.],
[0.],
[1.],
[1.],
[2.],
[2.],
[0.],
[0.],
[0.],
[2.],
[0.],
[2.],
[2.],
[2.],
[2.],
[0.],
[0.],
[0.],
[2.],
[2.],
[2.],
[2.],
[0.],
[2.],
[0.],
[2.],
[2.],
[2.],
[0.],
[2.],
[2.],
[2.],
[0.],
[2.],
[2.],
[0.],
[2.],
[2.],
[2.],
[2.],
[0.],
[2.],
[0.],
[2.],
[0.],
[0.],
[0.],
[1.],
[0.],
[2.],
[0.],
[2.],
```

```
[1.],
[2.],
[2.],
[1.],
[2.],
[0.],
[1.],
[0.],
[1.],
[2.],
[2.],
[2.],
[1.],
[1.],
[1.],
[1.],
[0.],
[1.],
[0.],
[0.],
[2.],
[2.],
[2.],
[2.],
[1.],
[0.],
[2.],
[1.],
[1.],
[0.],
[2.],
[2.],
[0.],
[2.],
[0.],
[0.],
[1.],
[2.],
[1.],
[1.],
[0.],
[1.],
[0.],
[2.],
[1.],
[0.],
[1.],
[2.],
[2.],
[0.],
[2.],
[1.],
[0.],
[1.],
[1.],
[1.],
```

```
[2.],
[2.],
[1.],
[0.],
[1.],
[2.],
[2.],
[1.],
[1.],
[2.],
[2.],
[0.],
[0.],
[1.],
[2.],
[2.],
[2.],
[2.],
[1.],
[1.],
[1.],
[1.],
[1.],
[2.],
[0.],
[2.],
[1.],
[2.],
[1.],
[0.],
[2.],
[0.],
[2.],
[2.],
[0.],
[0.],
[2.],
[2.],
[0.],
[1.],
[0.],
[2.],
[2.],
[2.],
[0.],
[1.],
[0.],
[2.],
[1.],
[2.],
[2.],
[2.],
[0.],
[2.],
[2.],
[2.],
```

```
[2.],
[1.],
[0.],
[2.],
[2.],
[1.],
[2.],
[1.],
[1.],
[2.],
[2.],
[1.],
[0.],
[0.],
[1.],
[2.],
[1.],
[0.],
[0.],
[1.],
[2.],
[0.],
[0.],
[0.],
[2.],
[0.],
[2.],
[0.],
[1.],
[2.],
[0.],
[1.],
[1.],
[0.],
[1.],
[2.],
[0.],
[2.],
[2.],
[1.],
[2.],
[1.],
[2.],
[0.],
[1.],
[0.],
[0.],
[0.],
[1.],
[2.],
[2.],
[2.],
[2.],
[0.],
[2.],
[1.],
```

```
[0.],
[2.],
[0.],
[0.],
[1.],
[1.],
[2.],
[1.],
[2.],
[1.],
[1.],
[2.],
[0.],
[0.],
[2.],
[0.],
[2.],
[0.],
[1.],
[0.],
[0.],
[2.],
[1.],
[0.],
[1.],
[0.],
[1.],
[0.],
[1.],
[1.],
[2.],
[2.],
[1.],
[1.],
[1.],
[0.],
[2.],
[0.],
[2.],
[1.],
[2.],
[2.],
[1.],
[1.],
[1.],
[1.],
[1.],
[2.],
[0.],
[0.],
[0.],
[2.],
[0.],
[0.],
[0.],
[2.],
```

```
[1.],
[1.],
[0.],
[2.],
[1.],
[1.],
[0.],
[2.],
[1.],
[2.],
[1.],
[0.],
[2.],
[1.],
[2.],
[1.],
[1.],
[1.],
[2.],
[0.],
[1.],
[0.],
[0.],
[2.],
[2.],
[2.],
[0.],
[0.],
[1.],
[1.],
[2.],
[1.],
[2.],
[1.],
[2.],
[2.],
[1.],
[0.],
[0.],
[0.],
[1.],
[2.],
[0.],
[0.],
[2.],
[2.],
[0.],
[0.],
[0.],
[0.],
[2.],
[0.],
[0.],
[1.],
[1.],
[1.],
```

```
[0.],
[2.],
[0.],
[2.],
[0.],
[0.],
[1.],
[2.],
[0.],
[1.],
[2.],
[2.],
[1.],
[2.],
[0.],
[1.],
[2.],
[1.],
[2.],
[0.],
[1.],
[1.],
[2.],
[0.],
[0.],
[0.],
[1.],
[2.],
[1.],
[0.],
[0.],
[0.],
[2.],
[1.],
[1.],
[2.],
[1.],
[0.],
[2.],
[1.],
[2.],
[2.],
[0.],
[2.],
[2.],
[0.],
[0.],
[1.],
[2.],
[1.],
[2.],
[2.],
[1.],
[1.],
[1.],
[1.],
```

```
[1.],
[2.],
[1.],
[0.],
[1.],
[1.],
[2.],
[2.],
[0.],
[0.],
[1.],
[2.],
[2.],
[0.],
[1.],
[2.],
[2.],
[1.],
[0.],
[2.],
[2.],
[0.],
[2.],
[1.],
[2.],
[0.],
[1.],
[1.],
[2.],
[2.],
[2.],
[0.],
[1.],
[1.],
[2.],
[2.],
[2.],
[1.],
[1.],
[0.],
[2.],
[0.],
[2.],
[0.],
[1.],
[2.],
[1.],
[2.],
[1.],
[0.],
[2.],
[1.],
[1.],
[2.],
[2.],
[1.],
```

```
       [2.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [2.],
       [0.],
       [2.],
       [1.],
       [1.],
       [2.],
       [2.],
       [2.],
       [1.],
       [2.],
       [2.],
       [1.],
       [1.],
       [2.],
       [1.],
       [2.],
       [2.],
       [2.],
       [2.],
       [1.],
       [0.],
       [1.],
       [2.],
       [0.],
       [1.],
       [0.],
       [2.],
       [1.],
       [1.],
       [2.],
       [2.],
       [2.],
       [0.],
       [2.],
       [1.],
       [2.],
       [1.],
       [2.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
       [2.],
       [0.],
       [2.],
       [2.],
       [2.],
```

```
[2.],
[2.],
[1.],
[2.],
[0.],
[1.],
[0.],
[1.],
[0.],
[0.],
[0.],
[0.],
[0.],
[2.],
[2.],
[0.],
[2.],
[0.],
[1.],
[0.],
[2.],
[0.],
[2.],
[1.],
[1.],
[1.],
[0.],
[2.],
[2.],
[2.],
[2.],
[2.],
[2.],
[1.],
[0.],
[1.],
[1.],
[2.],
[1.],
[0.],
[0.],
[1.],
[0.],
[1.],
[2.],
[0.],
[2.],
[1.],
[1.],
[1.],
[1.],
[0.],
[0.],
[2.]])
```

In [47]:
```
dataset['Property_Area'] = oen.fit_transform(dataset[['Property_Area']])
```

In [49]:
```
dataset.head(3)
```

Out[49]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
0	LP001002	Male	No	0	Graduate	No	5849
1	LP001003	Male	Yes	1	Graduate	No	4583
2	LP001005	Male	Yes	0	Graduate	Yes	3000

13_Outlier

13.1 Detecting Outlier

In [3]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
dataset = pd.read_csv('loan.csv')
dataset.head(3)
```

Out[2]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	C
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	

In [4]:
```
dataset.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
```

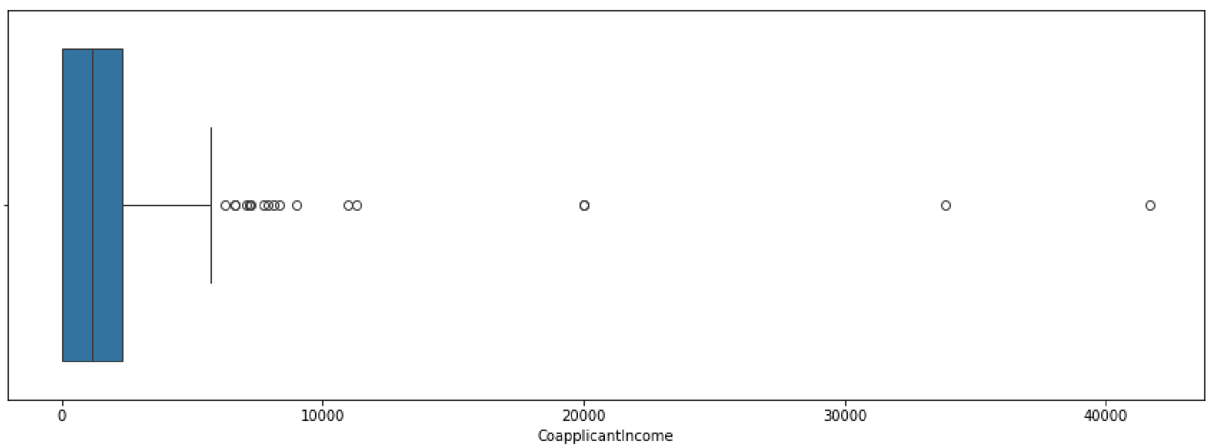
In [6]:
```
dataset.describe()
```

Out[6]:

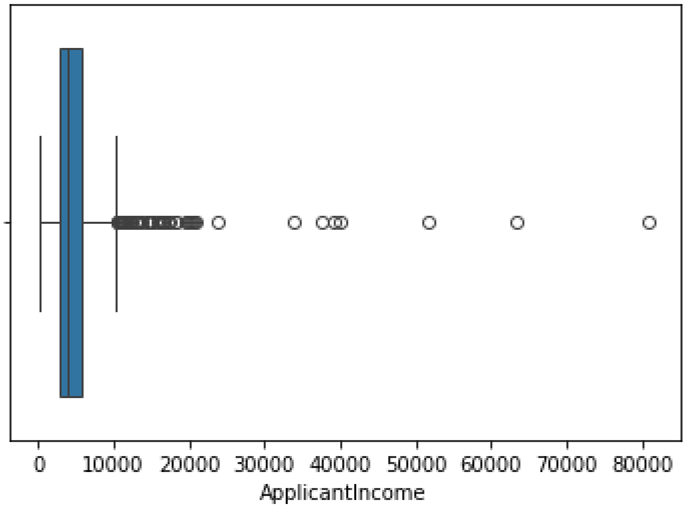
	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_Histo
count	614.000000	614.000000	592.000000	600.00000	564.0000
mean	5403.459283	1621.245798	146.412162	342.00000	0.8421
std	6109.041673	2926.248369	85.587325	65.12041	0.3648
min	150.000000	0.000000	9.000000	12.00000	0.0000
25%	2877.500000	0.000000	100.000000	360.00000	1.0000
50%	3812.500000	1188.500000	128.000000	360.00000	1.0000
75%	5795.000000	2297.250000	168.000000	360.00000	1.0000
max	81000.000000	41667.000000	700.000000	480.00000	1.0000

Detect Outlier through Boxplot

In [16]:
```
plt.figure(figsize=(15,5))
sns.boxplot(x='CoapplicantIncome', data=dataset)
plt.show()
```

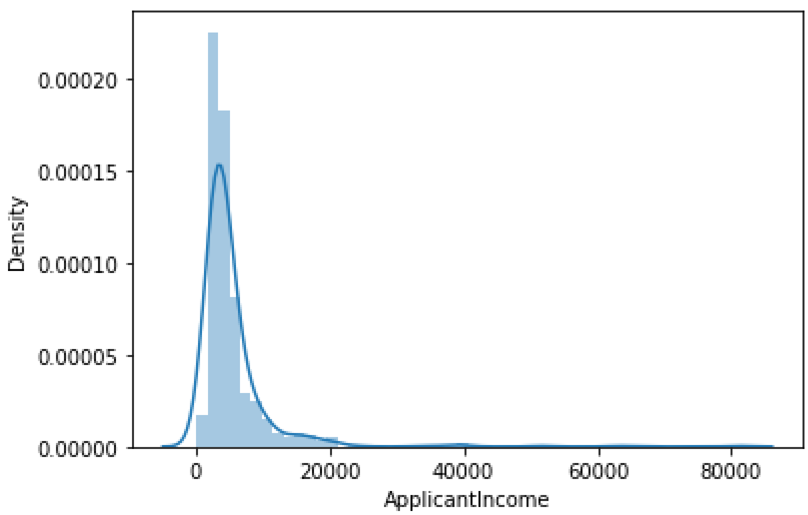


In [8]:
```
sns.boxplot(x='ApplicantIncome', data=dataset)
plt.show()
```



In [9]:
```
sns.distplot(dataset['ApplicantIncome'])
plt.show()
```

```
C:\Users\rashi\AppData\Local\Temp/ipykernel_8588/1976060950.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataset['ApplicantIncome'])
```



You can see that tail is too long, so definitely outlier is persent in this

13.2 Removing Outlier

There are two methods for removing outlier:

1. IQR (Inter Quartile Range) method

2. Z-Score method

13.2.1 Removing Outlier through IQR Method

In [11]:
```
dataset.shape
```

Out[11]: (614, 13)

In [13]:
```
q1 = dataset['CoapplicantIncome'].quantile(0.25)
q3 = dataset['CoapplicantIncome'].quantile(0.75)
q1, q3
```

Out[13]: (0.0, 2297.25)

In [14]:
```
IQR = q3 - q1
IQR
```

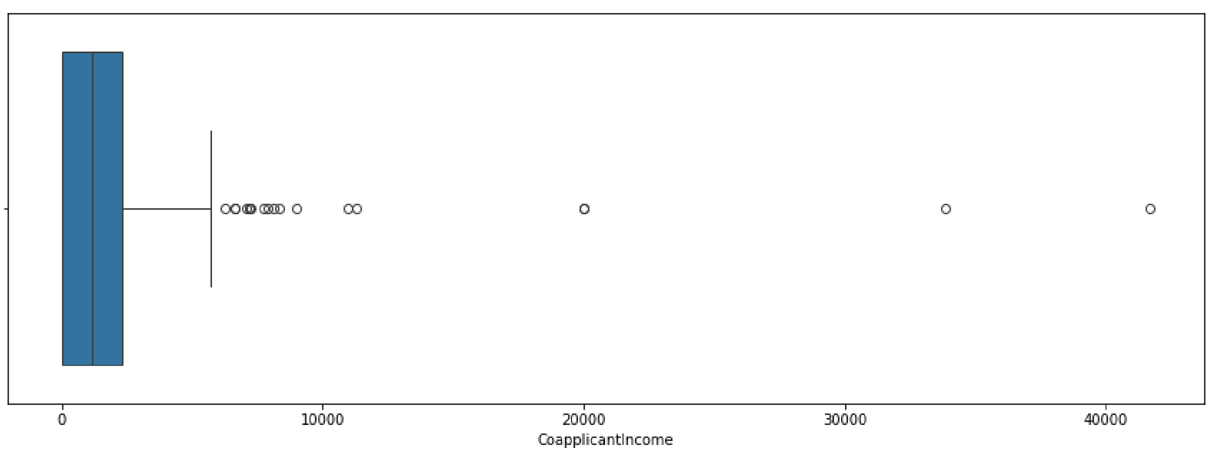
Out[14]: 2297.25

In [15]:
```
min_range = q1 - (1.5*IQR)
max_range = q3 + (1.5*IQR)
min_range, max_range
```

Out[15]: (-3445.875, 5743.125)

We will discard min_range as it is in negative while our data does not contain negative value. max_range is about 5000 as evident in graph below

In [17]:
```
plt.figure(figsize=(15,5))
sns.boxplot(x='CoapplicantIncome', data=dataset)
plt.show()
```



So now we will remove the outlier from the data

In [18]:
```
dataset.head(3)
```

Out[18]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	C
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	

In [19]:
```
dataset['CoapplicantIncome'] < max_range
```

Out[19]:
```
0       True
1       True
2       True
3       True
4       True
        ...
609     True
610     True
611     True
612     True
613     True
Name: CoapplicantIncome, Length: 614, dtype: bool
```

In [23]:
```
dataset.shape
```

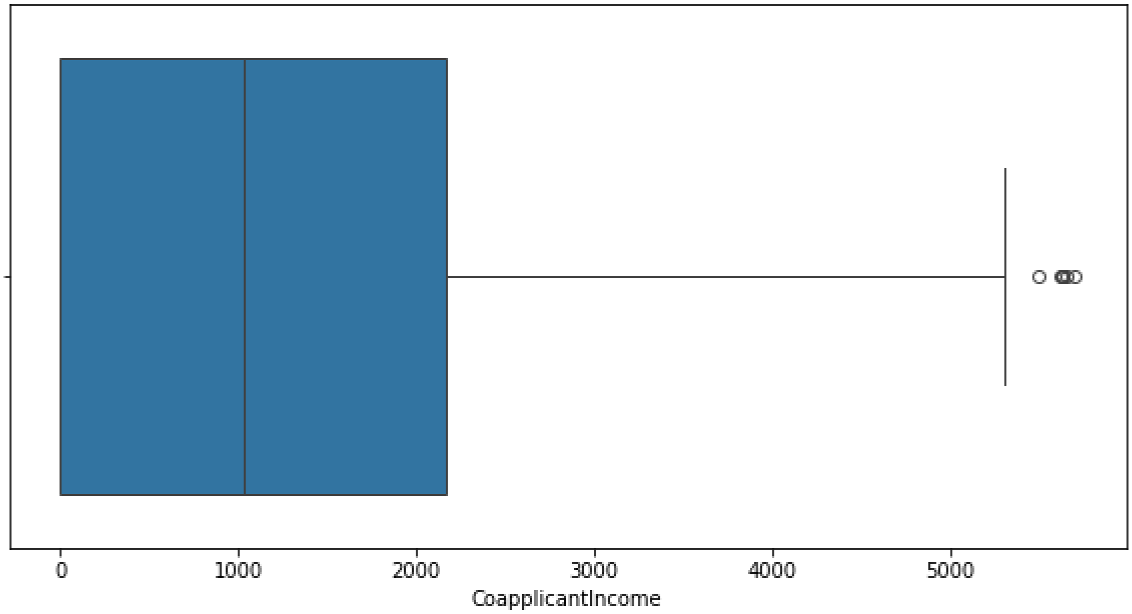
Out[23]:
```
(614, 13)
```

In [22]:
```
new_dataset = dataset[dataset['CoapplicantIncome'] < max_range]
new_dataset.shape
```

Out[22]:
```
(596, 13)
```

It means that 18 rows are removed which were containing outlier in new_dataset

In [26]:
```
plt.figure(figsize=(10,5))
sns.boxplot(x='CoapplicantIncome', data=new_dataset)
plt.show()
```



So number of outliers have been decreased significantly

Outliers may contain essential data so be careful in removing outliter. ML methods like decision tree is not affected by outlier, so you may keep outlier when using decision tree. Linear regression is very affected by outlier so you should remove outlier when using linear regression, but be careful you must not lose essential data

13.2.2 Removing Outlier through Z-Score Method 1

In [28]:
```
dataset.isnull().sum()
```

Out[28]:
```
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
```

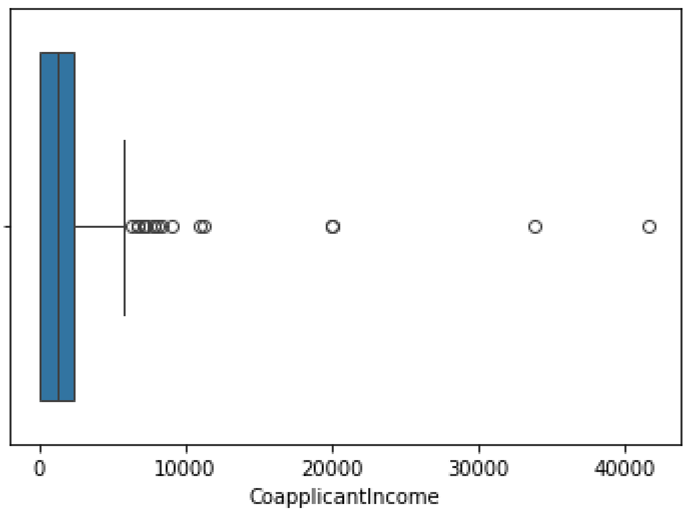
In [29]:
```
dataset.describe()
```

Out[29]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_Histo
count	614.000000	614.000000	592.000000	600.00000	564.0000
mean	5403.459283	1621.245798	146.412162	342.00000	0.8421
std	6109.041673	2926.248369	85.587325	65.12041	0.3648
min	150.000000	0.000000	9.000000	12.00000	0.0000
25%	2877.500000	0.000000	100.000000	360.00000	1.0000
50%	3812.500000	1188.500000	128.000000	360.00000	1.0000
75%	5795.000000	2297.250000	168.000000	360.00000	1.0000
max	81000.000000	41667.000000	700.000000	480.00000	1.0000

In [30]:

```
sns.boxplot(x='CoapplicantIncome', data=dataset)
plt.show()
```

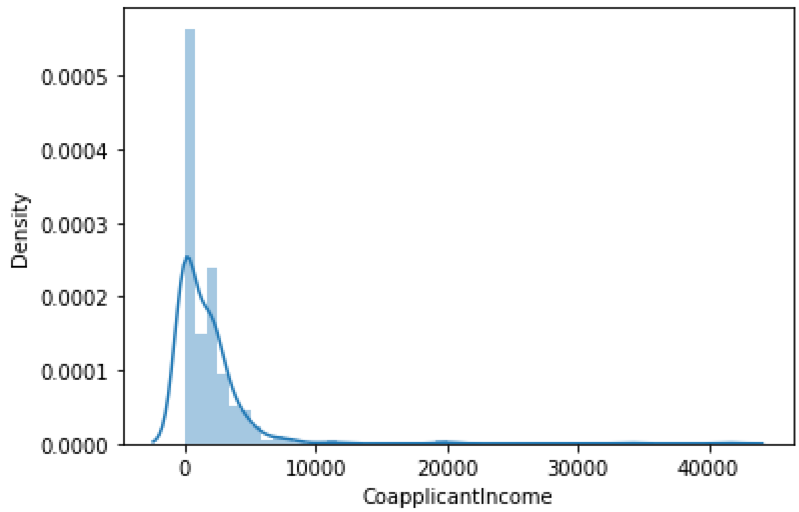


In [31]:

```
sns.distplot(dataset['CoapplicantIncome'])
```

```
C:\Users\rashi\AppData\Local\Temp/ipykernel_8588/4274022579.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataset['CoapplicantIncome'])
```

Out[31]: <Axes: xlabel='CoapplicantIncome', ylabel='Density'>



In [36]:
```
min_range = dataset['CoapplicantIncome'].mean() - (3*dataset['CoapplicantIncome'].s
max_range = dataset['CoapplicantIncome'].mean() + (3*dataset['CoapplicantIncome'].s
min_range, max_range
```

Out[36]: (-7157.4993096454655, 10399.990905699668)

- So will ignore min_range b/c its value is negative and our data doesn't contain any -ve value, so will ignore it
- We will take max_range and remove the data greater than this

In [43]:
```
new_dataset_z = dataset[dataset['CoapplicantIncome'] <= max_range]
```

In [44]:
```
dataset.shape, new_dataset_z.shape
```

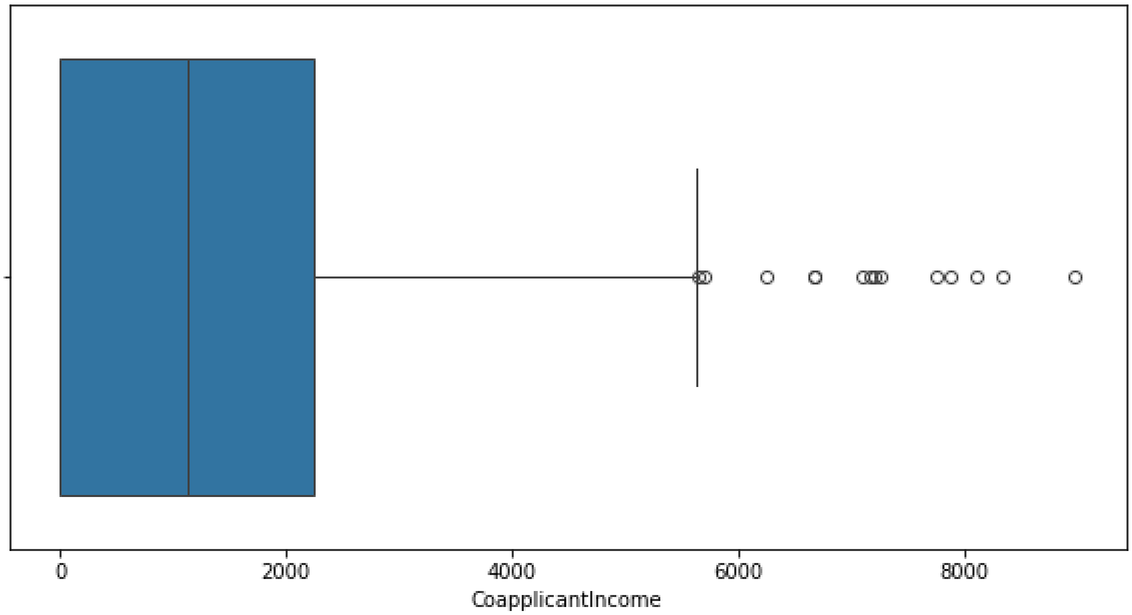
Out[44]: ((614, 13), (608, 13))

In [45]:
```
614-608
```

Out[45]: 6

So You can see 6 rows are deleted containing outliers

In [46]:
```
plt.figure(figsize=(10,5))
sns.boxplot(x='CoapplicantIncome', data=new_dataset_z)
plt.show()
```



13.2.3 Removing Outlier through Z-Score Method 2

In [48]:
```
# Formula of z_score
z_score = (dataset['CoapplicantIncome'] - dataset['CoapplicantIncome'].mean())/data
z_score
```

Out[48]:
```
0     -0.554036
1     -0.038700
2     -0.554036
3      0.251774
4     -0.554036
         ...
609   -0.554036
610   -0.554036
611   -0.472019
612   -0.554036
613   -0.554036
Name: CoapplicantIncome, Length: 614, dtype: float64
```

In [52]:
```
dataset['Z_score'] = z_score
dataset.head(3)
```

Out[52]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	C
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	

In [59]:
```
dataset['Z_score']
```

Out[59]:
```
0     -0.554036
1     -0.038700
2     -0.554036
3      0.251774
4     -0.554036
         ...
609   -0.554036
610   -0.554036
611   -0.472019
612   -0.554036
613   -0.554036
Name: Z_score, Length: 614, dtype: float64
```

In [60]:
```
# new_dataset_z = dataset[dataset['CoapplicantIncome'] <= max_range]
new_dataset_z_2 = dataset[dataset['Z_score'] < 3]
new_dataset_z_2.shape
```

Out[60]:
```
(608, 14)
```

So both method 1 and method 2 for removing outlier by z-score are equal

In [ ]:

14. Feature Scaling Technique

Problem:

- Some data are too large in thousands and some data are too less in zeros, so ML algo will dominate the large data and neglect the small data
- To address this problem we introduce Feature scaling technique to bring both the both the datas in the same pitch
- The big data will reduce to bring equal to small data
- You should do feature scaling before training your data

Types of Feature Scaling:

1. Standardization
2. Normalization

Standardization (Z-score normalization)

- It is a very effective technique which re-scales a feature value so that it has distribution with 0 mean value and variance equals to 1

The formula for standardization is:

$$X_{new} = \frac{X_i - \mu}{\sigma}$$

where:

- (X_{new}) is the standardized value,
- (X_i) is the original value,
- (\mu) is the mean of the data,
- (\sigma) is the standard deviation of the data.

- By Scaling - outliers are not removed, though magnitude of outlier will be reduced, but will not affect it significantly

In [2]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [3]:
```
dataset = pd.read_csv('loan.csv')
dataset.head(3)
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
0	LP001002	Male	No	0	Graduate	No	5849
1	LP001003	Male	Yes	1	Graduate	No	4583
2	LP001005	Male	Yes	0	Graduate	Yes	3000

Data Cleaning (Identifying and Removing Null Value)

In [4]:
```
dataset.isnull().sum()
```

Out[4]:
```
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
```

In [5]:
```
dataset['ApplicantIncome'].fillna(dataset['ApplicantIncome'].mode()[0],inplace=True
```

In [6]:
```
dataset.isnull().sum()
```

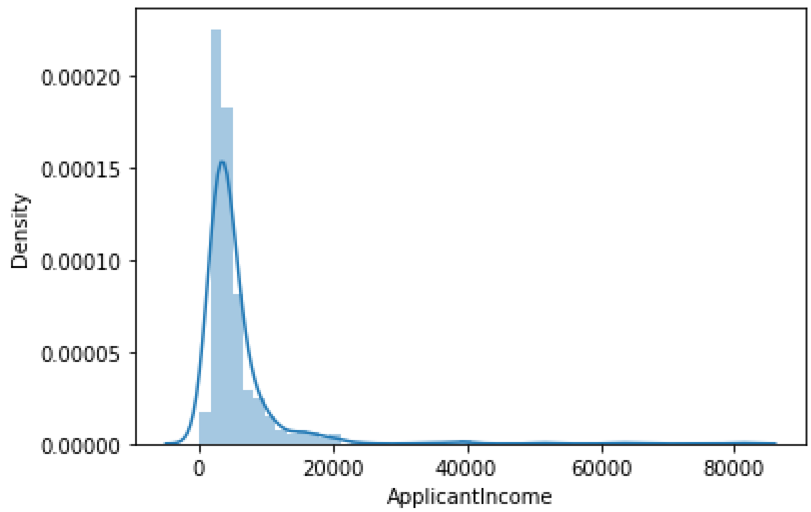
Out[6]:
```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

Check the nature of Data (Outlier Detection)

data: dataset['ApplicantIncome']

In [16]:
```
sns.distplot(dataset['ApplicantIncome'])
plt.show()
```

```
C:\Users\rashi\AppData\Local\Temp/ipykernel_4156/1976060950.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataset['ApplicantIncome'])
```



Outlier is present as long tail is evident from the graph showing number of outliers present in the data

In [8]:
```
dataset.describe()
```

Out[8]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_Histo
count	614.000000	614.000000	592.000000	600.00000	564.0000
mean	5403.459283	1621.245798	146.412162	342.00000	0.8421
std	6109.041673	2926.248369	85.587325	65.12041	0.3648
min	150.000000	0.000000	9.000000	12.00000	0.0000
25%	2877.500000	0.000000	100.000000	360.00000	1.0000
50%	3812.500000	1188.500000	128.000000	360.00000	1.0000
75%	5795.000000	2297.250000	168.000000	360.00000	1.0000
max	81000.000000	41667.000000	700.000000	480.00000	1.0000

14.1 Feature Scaling of Data by Standardization

In [9]:
```
from sklearn.preprocessing import StandardScaler
```

In [10]:
```
ss = StandardScaler()
ss.fit(dataset[['ApplicantIncome']])
```

Out[10]:
```
▾ StandardScaler
StandardScaler()
```

In [11]:
```
# Transform the data and stored in csv file in another column called ApplicantIncom
dataset['ApplicantIncome_ss'] = pd.DataFrame(ss.transform(dataset[['ApplicantIncome
dataset.head(3)
```

Out[11]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	C
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	

In [12]:
```
dataset.describe()
```

Out[12]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_Histo
count	614.000000	614.000000	592.000000	600.00000	564.0000
mean	5403.459283	1621.245798	146.412162	342.00000	0.8421
std	6109.041673	2926.248369	85.587325	65.12041	0.3648
min	150.000000	0.000000	9.000000	12.00000	0.0000
25%	2877.500000	0.000000	100.000000	360.00000	1.0000
50%	3812.500000	1188.500000	128.000000	360.00000	1.0000
75%	5795.000000	2297.250000	168.000000	360.00000	1.0000
max	81000.000000	41667.000000	700.000000	480.00000	1.0000

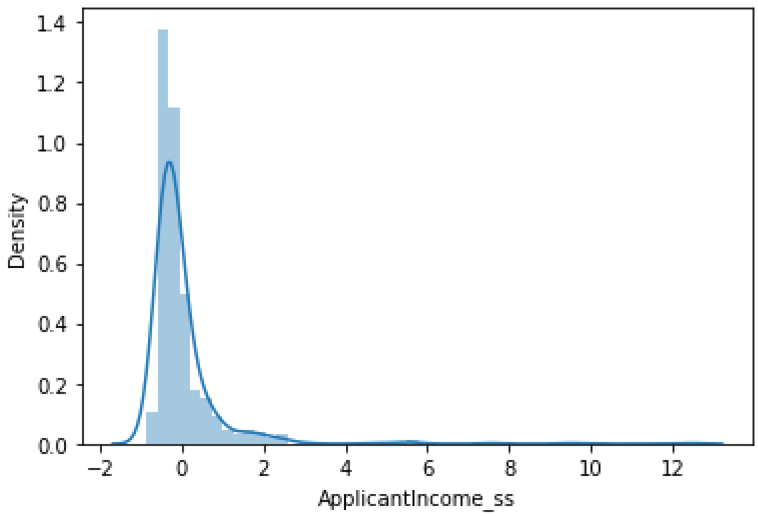
- varaiance is square of standard deviation

Check the nature of transformed Data (Outlier Detection)

data: dataset['ApplicantIncome_ss']

In [13]:
```
sns.distplot(dataset['ApplicantIncome_ss'])
plt.show()
```

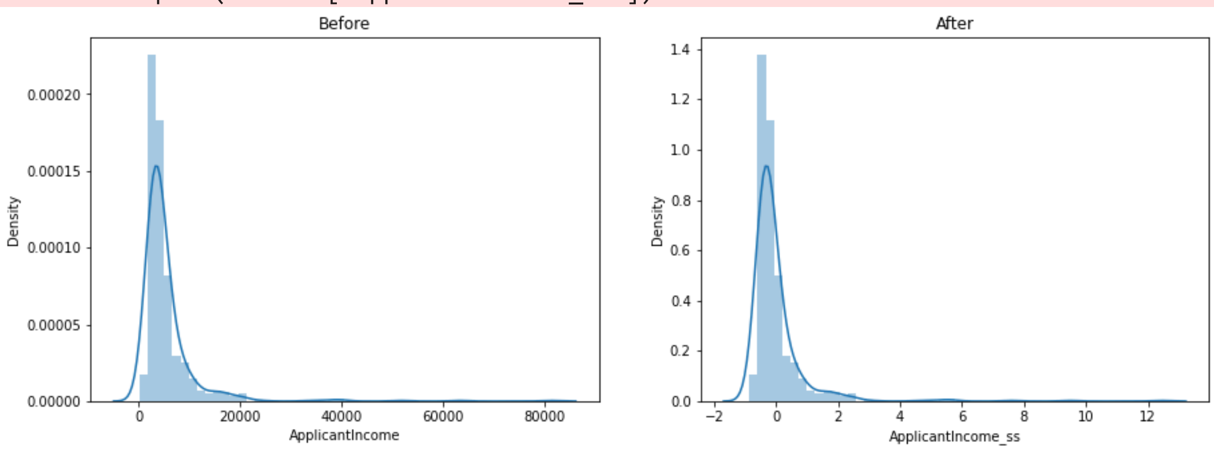
```
C:\Users\rashi\AppData\Local\Temp/ipykernel_4156/3877852283.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataset['ApplicantIncome_ss'])
```



- In order to compare both graphs, we will use subplot function to draw both graphs side by side
- subplot(number_of_rows,number_of_col, position)

In [15]:
```
plt.figure(figsize=(15,5))
# plot#1 plt.subplot: row=1, col=2, position=1
plt.subplot(1,2,1)
plt.title('Before')
sns.distplot(dataset['ApplicantIncome'])

# plot#2 plt.subplot: row=1, col=2, position=2
plt.subplot(1,2,2)
plt.title('After')
sns.distplot(dataset['ApplicantIncome_ss'])

plt.show()
```

```
C:\Users\rashi\AppData\Local\Temp/ipykernel_4156/2479568808.py:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataset['ApplicantIncome'])
C:\Users\rashi\AppData\Local\Temp/ipykernel_4156/2479568808.py:10: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataset['ApplicantIncome_ss'])
```



- So nature of data is not changed after scalling
- Only magnitude of big data is reduced
-

14.2 Feature Scaling of Data by Normalization (Min-Max Scaler)

- Data nature also remain same before and after scalling by Normalization
- Data will be reduced according to min and max values in the data
- Data range after scaling by Normalization is between 0 and 1

Normalization (Min-Max Scaling)

It is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1.

The formula for normalization is:

$$X_{new} = \frac{X_i - X_{min}}{X_{max} - X_{min}}$$

where:

- (X_{new}) is the normalized value,
- (X_i) is the original value,
- (X_{min}) is the minimum value of the feature,
- (X_{max}) is the maximum value of the feature.

In [19]:
```
dataset = pd.read_csv('loan.csv')
dataset.head(3)
```

Out[19]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	C
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	

In [21]:
```
dataset.isnull().sum()
```

Out[21]:
```
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
```

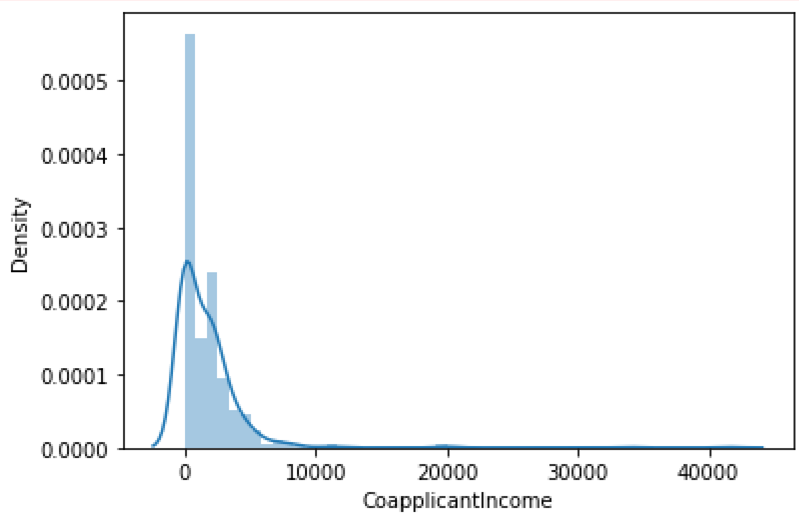
In [22]:
```
dataset.describe()
```

Out[22]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_Histo
count	614.000000	614.000000	592.000000	600.00000	564.0000
mean	5403.459283	1621.245798	146.412162	342.00000	0.8421
std	6109.041673	2926.248369	85.587325	65.12041	0.3648
min	150.000000	0.000000	9.000000	12.00000	0.0000
25%	2877.500000	0.000000	100.000000	360.00000	1.0000
50%	3812.500000	1188.500000	128.000000	360.00000	1.0000
75%	5795.000000	2297.250000	168.000000	360.00000	1.0000
max	81000.000000	41667.000000	700.000000	480.00000	1.0000

In [23]:
```
sns.distplot(dataset['CoapplicantIncome'])
plt.show()
```

```
C:\Users\rashi\AppData\Local\Temp/ipykernel_4156/3783729653.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataset['CoapplicantIncome'])
```



In [24]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [27]:
```
ms = MinMaxScaler()
ms.fit(dataset[['CoapplicantIncome']])
```

Out[27]:

```
▾ MinMaxScaler
MinMaxScaler()
```

In [30]:

```
dataset['CoapplicantIncome_min'] = pd.DataFrame(ms.transform(dataset[['CoapplicantI
dataset.head(3)
```

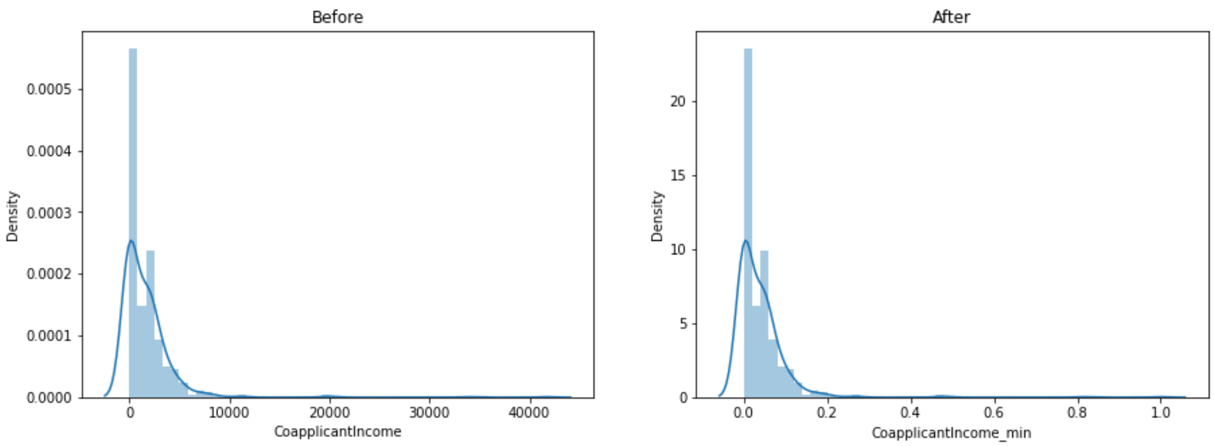
Out[30]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	C
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	

In [31]:

```
plt.figure(figsize=(15,5))
# plot#1 plt.subplot: row=1, col=2, position=1
plt.subplot(1,2,1)
plt.title('Before')
sns.distplot(dataset['CoapplicantIncome'])

# plot#2 plt.subplot: row=1, col=2, position=2
plt.subplot(1,2,2)
plt.title('After')
sns.distplot(dataset['CoapplicantIncome_min'])

plt.show()
```

```
C:\Users\rashi\AppData\Local\Temp/ipykernel_4156/710565463.py:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataset['CoapplicantIncome'])
C:\Users\rashi\AppData\Local\Temp/ipykernel_4156/710565463.py:10: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataset['CoapplicantIncome_min'])
```



So you can see the nature of data is not changed through scalling of data by normalization also

In [ ]:

15. Handling Duplicate Data

- The repetition of same data of one row is repeated in another row is called duplicate data

In [1]:
```
import pandas as pd
```

In [8]:
```
data = {'name':['a','b','c','d','a','c'], "eng":[8,7,5,8,8,5], "Urdu":[2,3,4,5,2,6]
data
```

Out[8]:
```
{'name': ['a', 'b', 'c', 'd', 'a', 'c'],
 'eng': [8, 7, 5, 8, 8, 5],
 'Urdu': [2, 3, 4, 5, 2, 6]}
```

In [9]:
```
df = pd.DataFrame(data)
df
```

Out[9]:

	name	eng	Urdu
0	a	8	2
1	b	7	3
2	c	5	4
3	d	8	5
4	a	8	2
5	c	5	6

- You can see that row number 0 and 4 have duplicate data
- row 2 and 5 are not duplicate, even the two values are identical, but to call a data duplicate exact data has to be there

In [14]:
```
# To identify the duplicate data
df.duplicated()
```

Out[14]:
```
0    False
1    False
2    False
3    False
4    False
5    False
dtype: bool
```

In [23]:
```
df['duplicate'] = df.duplicated()
df
```

Out[23]:

	name	eng	Urdu	duplicated	duplicate
0	a	8	2	False	False
1	b	7	3	False	False
2	c	5	4	False	False
3	d	8	5	False	False
4	a	8	2	False	True
5	c	5	6	False	False

In [24]:

```
df.drop('duplicate', axis=1, inplace=True)
```

In [25]:

```
df
```

Out[25]:

	name	eng	Urdu	duplicated
0	a	8	2	False
1	b	7	3	False
2	c	5	4	False
3	d	8	5	False
4	a	8	2	False
5	c	5	6	False

- Some ML algo also get train on duplicated data such as when we doing classification, so we should remove duplicate before data training

In [27]:

```
# To remove duplicated data
df.drop_duplicates()
```

Out[27]:

	name	eng	Urdu	duplicated
0	a	8	2	False
1	b	7	3	False
2	c	5	4	False
3	d	8	5	False
5	c	5	6	False

You can see that row 4 is deleted

In [29]:

```
df.drop('duplicated', axis=1, inplace=True)
```

In [30]:
```
df
```

Out[30]:

	name	eng	Urdu
0	a	8	2
1	b	7	3
2	c	5	4
3	d	8	5
4	a	8	2
5	c	5	6

Lets practice on orginal data

In [32]:
```
dataset = pd.read_csv('loan.csv')
dataset.head(3)
```

Out[32]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	C
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	

In [34]:
```
dataset.duplicated().sum()
```

Out[34]: 0

No duplicate is present in the data

Other way to see duplicates in the data:

In [36]:
```
dataset.shape
```

Out[36]: (614, 13)

In [38]:
```
dataset.drop_duplicates(inplace=True)
```

In [40]:
```
dataset.shape
```

Out[40]: (614, 13)

So you can see that the number of rows and columns are same before and after removing duplicates, so no duplicates are present in the data

16. Data Type Transformation(Replace and Data Type Change)

- 3+ is categorical (object) data while other rows in this column contain numerical data
- We will remove 3+ and convert its data type from object data type to int dtype

In [2]:
```python
import pandas as pd
```

In [4]:
```python
dataset = pd.read_csv('loan.csv')
dataset.head(3)
```

Out[4]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
0	LP001002	Male	No	0	Graduate	No	5849
1	LP001003	Male	Yes	1	Graduate	No	4583
2	LP001005	Male	Yes	0	Graduate	Yes	3000

- 3+ is present in 'Dependents' column

In [6]:
```python
dataset.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
```

- 'Dependent' column has null value, b/c RangeIndex:614, wherease (Dependents 599 non-null object) - as shown above

In [7]:
```
dataset.isnull().sum()
```

Out[7]:
```
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
```

In [9]:
```
dataset['Dependents'].value_counts()
```

Out[9]:
```
0     345
1     102
2     101
3+     51
Name: Dependents, dtype: int64
```

In [12]:
```
# Remove null values in Dependents column
dataset['Dependents'].fillna(dataset['Dependents'].mode()[0], inplace=True)
```

In [13]:
```
dataset.isnull().sum()
```

Out[13]:
```
Loan_ID               0
Gender               13
Married               3
Dependents            0
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
```

Replace 3+ with 3

In [15]:
```
dataset['Dependents'].replace('3+', '3', inplace=True)
```

In [16]:
```
dataset['Dependents'].value_counts()
```

Out[16]:
```
0    360
1    102
2    101
3     51
Name: Dependents, dtype: int64
```

In [17]:
```
dataset.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         614 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
```

Now change the datatype

In [19]:
```
dataset['Dependents'] = dataset['Dependents'].astype("int64")
```

In [20]:
```
dataset.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         614 non-null    int64  
 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(2), object(7)
memory usage: 62.5+ KB
```

In [22]:
```
dataset['Dependents'].value_counts()
```

Out[22]:
```
0    360
1    102
2    101
3     51
Name: Dependents, dtype: int64
```

In [ ]:

17. Function (Transformer)

- to conver the non-normal distribution data into normal distribution data

In [18]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pylab as plt
import numpy as np
```

In [3]:
```
dataset = pd.read_csv('loan.csv')
```

In [4]:
```
dataset.head(3)
```

Out[4]:

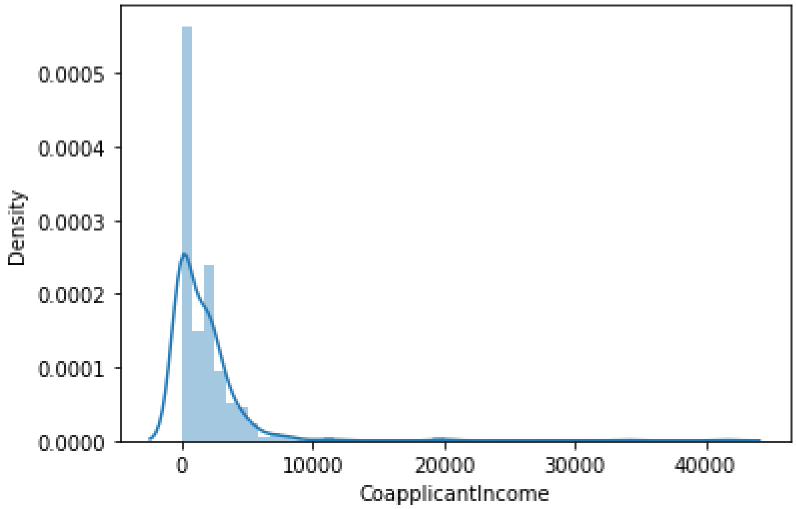
	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	C
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	

In [7]:
```
dataset.isnull().sum()
```

Out[7]:
```
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
```

In [8]:
```
sns.distplot(dataset['CoapplicantIncome'])
plt.show()
```

```
C:\Users\rashi\AppData\Local\Temp/ipykernel_4868/3783729653.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataset['CoapplicantIncome'])
```



- You can see the data is not normally distributed as it has long tail on right side

17.1 Remove Outlier

We will remove the outlier by IQR method

In [9]:
```python
q1 = dataset['CoapplicantIncome'].quantile(0.25)
q3 = dataset['CoapplicantIncome'].quantile(0.75)
iqr = q3 - q1
```

In [12]:
```python
min_r = q1-(1.5*iqr)
max_r = q3 + (1.5*iqr)
min_r, max_r
```

Out[12]:
```
(-3445.875, 5743.125)
```

In [15]:
```python
dataset = dataset[dataset['CoapplicantIncome'] <= max_r]
dataset
```

Out[15]:

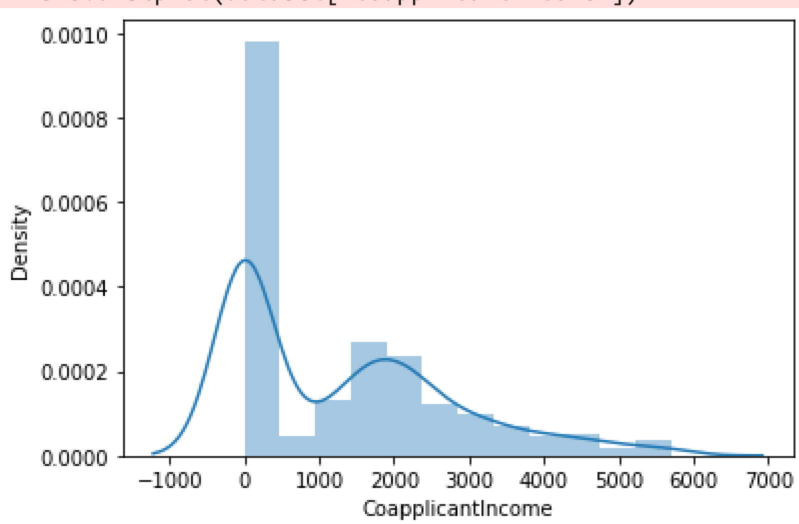
	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
0	LP001002	Male	No	0	Graduate	No	5849
1	LP001003	Male	Yes	1	Graduate	No	4583
2	LP001005	Male	Yes	0	Graduate	Yes	3000
3	LP001006	Male	Yes	0	Not Graduate	No	2583
4	LP001008	Male	No	0	Graduate	No	6000
...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900
610	LP002979	Male	Yes	3+	Graduate	No	4106
611	LP002983	Male	Yes	1	Graduate	No	8072
612	LP002984	Male	Yes	2	Graduate	No	7583
613	LP002990	Female	No	0	Graduate	Yes	4583

596 rows × 13 columns

In [16]:

```
sns.distplot(dataset['CoapplicantIncome'])
plt.show()
```

```
C:\Users\rashi\AppData\Local\Temp/ipykernel_4868/3783729653.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataset['CoapplicantIncome'])
```



17.2 Function Transformation

- To make the data normally distributed, we will use function transformation

In [17]:
```
from sklearn.preprocessing import FunctionTransformer
```

In [21]:
```
ft = FunctionTransformer(func=np.log1p)
```

In [22]:
```
ft.fit(dataset[['CoapplicantIncome']])
```

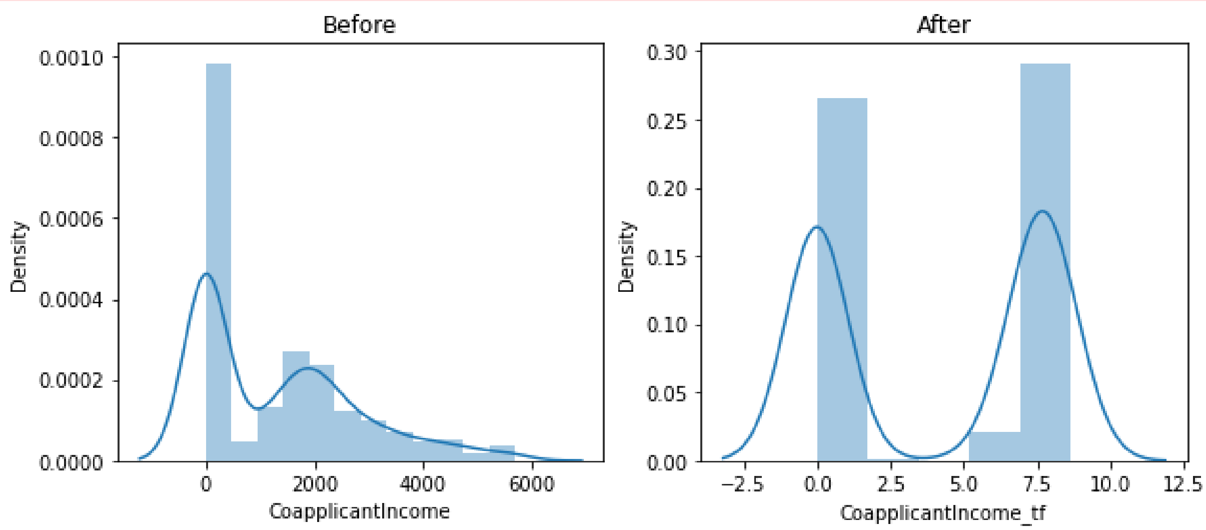
Out[22]:
```
▾ FunctionTransformer
FunctionTransformer(func=<ufunc 'log1p'>)
```

In [25]:
```
dataset['CoapplicantIncome_tf'] = ft.transform(dataset[['CoapplicantIncome']])
dataset['CoapplicantIncome_tf']
```

Out[25]:
```
0      0.000000
1      7.319202
2      0.000000
3      7.765993
4      0.000000
         ...
609    0.000000
610    0.000000
611    5.484797
612    0.000000
613    0.000000
Name: CoapplicantIncome_tf, Length: 596, dtype: float64
```

In [28]:
```
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
sns.distplot(dataset['CoapplicantIncome'])
plt.title("Before")
plt.subplot(1,2,2)
sns.distplot(dataset['CoapplicantIncome_tf'])
plt.title("After")
plt.show()
```

```
C:\Users\rashi\AppData\Local\Temp/ipykernel_4868/3310440801.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataset['CoapplicantIncome'])
C:\Users\rashi\AppData\Local\Temp/ipykernel_4868/3310440801.py:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dataset['CoapplicantIncome_tf'])
```



18. Feature Selection Techniques

- Data consist of many columns representing number of features, so we will select only those columns which are important for ML model building
- Feature selection is we select certian columns from many available columns
- columbn = feature
- A features is an attribute that has an impact on a problem or is useful for the problem, and choosing the important features for the mdoel is known as feature selection
- One should have domain knowledge in order to select appropriate features from data

18.1 Feature Selection by Forward Elimination

- In this example, we will select feature even if we don't have domain knowledge.
- From following example, consider these points:

1. From layer 1, we will select only that feature which will have highest accuracy, for example feature 2 has highest accuracy
2. Then we will merge all remaining features, with the highest selected features, ie feature 3 (accuracy = 75%)
3. From layer 2, we will select features set, for example feature set (feature 3 + feature 1), only if it will have higher accuracy than 75%, otherwise we will move to the next step with feature 3 only
4.

forward-elimination

18.2 Backward Elimination

- it move oppostie, first groups of multiple features are carried further having good accuracy score, then remove one feature and move and so on

backward-elimination

18.3 Implementation

In [12]:
```
import pandas as pd
from mlxtend.feature_selection import SequentialFeatureSelector
```

In [15]:
```
dataset = pd.read_csv('diabetes.csv')
dataset.head(3)
```

Out[15]:

	Glucose	BloodPressure	SkinThickness	BMI	Age	Outcome
0	148	72	35	33.6	50	1
1	85	66	29	26.6	31	0
2	183	64	0	23.3	32	1

In [18]:
```
x = dataset.iloc[:,:-1]
x.head(3)
```

Out[18]:

	Glucose	BloodPressure	SkinThickness	BMI	Age
0	148	72	35	33.6	50
1	85	66	29	26.6	31
2	183	64	0	23.3	32

In [19]:
```
y = dataset['Outcome']
y.head(3)
```

Out[19]:
```
0    1
1    0
2    1
Name: Outcome, dtype: int64
```

In [22]:
```
x.shape
```

Out[22]:
```
(768, 5)
```

There are 5 features

In [20]:
```
from sklearn.linear_model import LogisticRegression
```

In [21]:
```
lr = LogisticRegression()
```

In [24]:
```
#fs = SequentialFeatureSelector(estimator, k_feature, )
fs = SequentialFeatureSelector(lr, k_features=5, forward=True)
fs.fit(x,y)
```

Out[24]:
```
▸ SequentialFeatureSelector
▸ estimator: LogisticRegression
    ▸ LogisticRegression
```

In [25]:
```
fs.feature_names
```

Out[25]: ['Glucose', 'BloodPressure', 'SkinThickness', 'BMI', 'Age']

In [27]:
```
fs.k_feature_names_
```

Out[27]: ('Glucose', 'BloodPressure', 'SkinThickness', 'BMI', 'Age')

In [28]:
```
fs.k_score_
```

Out[28]: 0.7682794329853152

In [ ]:
```
5 - 0.7682794329853152
```

Now we will select 4 features and see accuracy and then 3 features and see its accuracy and so on..

In [29]:
```
fs = SequentialFeatureSelector(lr, k_features=4, forward=True)
fs.fit(x,y)
```

Out[29]:
```
▸ SequentialFeatureSelector
▸ estimator: LogisticRegression
    ▸ LogisticRegression
```

In [30]:
```
fs.k_feature_names_
```

Out[30]: ('Glucose', 'BloodPressure', 'BMI', 'Age')

In [31]:
```
fs.k_score_
```

Out[31]: 0.7682709447415329

In [32]:
```
fs = SequentialFeatureSelector(lr, k_features=3, forward=True)
fs.fit(x,y)
```

Out[32]:
```
▸ SequentialFeatureSelector
▸ estimator: LogisticRegression
    ▸ LogisticRegression
```

In [33]:
```
fs.k_feature_names_
```

Out[33]: ('Glucose', 'BMI', 'Age')

In [34]:
```
fs.k_score_
```

Out[34]: 0.7683048977166624

In [35]:
```
fs = SequentialFeatureSelector(lr, k_features=2, forward=True)
fs.fit(x,y)
```

Out[35]:

```
▸ SequentialFeatureSelector
▸ estimator: LogisticRegression
    ▸ LogisticRegression
```

In [36]:
```
fs.k_feature_names_
```

Out[36]:
```
('Glucose', 'BMI')
```

In [37]:
```
fs.k_score_
```

Out[37]:
```
0.7591206179441474
```