==========>EDA<================

- The full form of EDA is Expletory Data Analysis.
- Reading data peform retrive data operations
- Categorical analysis
- Numerical analysis
- Outlier analysis
- Missing values analysis
- Data conversions
- Standaradization and Normalization of the data

In [1]:

```
#Step:1===> Import packages

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:

```
#Step:2===> Read the Data

file_name=('C:\\Users\\venka\\supraja\\train_ctrUa4K.csv')
loan_dataset=pd.read_csv(file_name)
loan_dataset
```

Out[2]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapp
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

- The following functions are used in EDA:
  - type
  - len
  - size
  - shape
  - columns
  - head
  - tail
  - take
  - iloc
  - loc
  - dtypes
  - info
  - isnull
  - type casting
  - unique
  - nunique

In [3]:

```
print("Type of File: ",type(loan_dataset))
print("Length of DataFrame: ",len(loan_dataset))
print("Size of DataFrame: ",loan_dataset.size)    #rows*columns
print("Shape of DataFrame: ",loan_dataset.shape)  #rows,columns
print("Columns of DataFrame: ",loan_dataset.columns)
```

```
Type of File:  <class 'pandas.core.frame.DataFrame'>
Length of DataFrame:  614
Size of DataFrame:  7982
Shape of DataFrame:  (614, 13)
Columns of DataFrame:  Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Educa
tion',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [4]:

```
print('\t first five rows')
loan_dataset.head()
```

```
	 first five rows
```

Out[4]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplic
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	

In [5]:

```
# Enter number inside the brackets which rows upto u want. (from first to last)(0 i
loan_dataset.head(5)
```

Out[5]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplic
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	

In [6]:

```
print('\t last five rows')
loan_dataset.tail(2)
```

```
         last five rows
```

Out[6]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapp
612	LP002984	Male	Yes	2	Graduate	No	7583	
613	LP002990	Female	No	0	Graduate	Yes	4583	

In [7]:

```
# Enter number inside the brackets which rows upto u want (from last to first)(-1 i
loan_dataset.tail(5)
```

Out[7]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapp
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	

In [8]:

```
loan_dataset.take([5,3]) # default axis=0

# axis=0   rows
# axis=1   columns
```

Out[8]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplic
5	LP001011	Male	Yes	2	Graduate	Yes	5417	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	

In [9]:

```
loan_dataset.take([5,3],axis=1)
```

Out[9]:

	Self_Employed	Dependents
0	No	0
1	No	1
2	Yes	0
3	No	0
4	No	0
...	...	...
609	No	0
610	No	3+
611	No	1
612	No	2
613	Yes	0

614 rows × 2 columns

In [10]:

```
# I want  400 500 600 rows  from 1,8,-1 columns?

loan_dataset.take([1,8,-1],axis=1).take([400,500,600])
```

Out[10]:

	Gender	LoanAmount	Loan_Status
400	Male	45.0	N
500	Female	113.0	Y
600	Female	350.0	N

loc-iloc

- So we have so many rows and columns is there
- Head will give only top of the rows
- Tail will give bottom of the table rows
- Take will works based upon axis and index

iloc:

In [11]:

```
# <dataframe>.iloc[rows,columns]
# <dataframe>.iloc[start:end,start:end]

# I want first 3 columns and 2 to 6th row  2,3,4,5,6
# python index start with zero
# <dataframe>.iloc[2:7,0:3]
```

In [12]:
```
loan_dataset.iloc[2:7,0:3]
```

Out[12]:

	Loan_ID	Gender	Married
2	LP001005	Male	Yes
3	LP001006	Male	Yes
4	LP001008	Male	No
5	LP001011	Male	Yes
6	LP001013	Male	Yes

In [13]:
```
loan_dataset.iloc[[450,500,550],[8]]
```

Out[13]:

	LoanAmount
450	125.0
500	113.0
550	NaN

Observations:

- I want 450 row , 500 row and 550 row from 'LoanAmount' column
- 'LoanAmount' column index is : 8
- we are counting for getting indexing value of 'LoanAmount'
    - drawback : we are counting the index of the column
    - If there are 100 columns are there, it is not possible to count.

loc:

- To avoid the count the index of columns we use loc
- loan_dataset.loc[rows,columns]
- But we can provide the column names directly

In [14]:
```
columns=['LoanAmount']
rows=[450,500,570]
loan_dataset.loc[rows,columns]
```

Out[14]:

	LoanAmount
450	125.0
500	113.0
570	186.0

In [15]:
```
loan_dataset.loc[[100,200,300,400,500,600],['ApplicantIncome','CoapplicantIncome',
```

Out[15]:

	ApplicantIncome	CoapplicantIncome	Property_Area
100	4288	3263.0	Urban
200	2600	2500.0	Semiurban
300	1800	2934.0	Urban
400	2889	0.0	Urban
500	645	3683.0	Rural
600	416	41667.0	Urban

In [16]:

```
#type casting to list

type(loan_dataset.columns)
# it is not a list
# we can type cast list
```

Out[16]:

```
pandas.core.indexes.base.Index
```

In [17]:

```
list(loan_dataset.columns)
```

Out[17]:

```
['Loan_ID',
 'Gender',
 'Married',
 'Dependents',
 'Education',
 'Self_Employed',
 'ApplicantIncome',
 'CoapplicantIncome',
 'LoanAmount',
 'Loan_Amount_Term',
 'Credit_History',
 'Property_Area',
 'Loan_Status']
```

In [18]:

```
cols_list=list(loan_dataset.columns)
cols_list.index('LoanAmount')
```

Out[18]:

```
8
```

dtypes:

In [19]:

```
loan_dataset.dtypes
```

Out[19]:

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

In [20]:
```python
# type of dataset.dtypes

type(loan_dataset.dtypes)
```

Out[20]:
```
pandas.core.series.Series
```

In [21]:
```python
# type cast to dictionary:
dict(loan_dataset.dtypes)
```

Out[21]:
```
{'Loan_ID': dtype('O'),
 'Gender': dtype('O'),
 'Married': dtype('O'),
 'Dependents': dtype('O'),
 'Education': dtype('O'),
 'Self_Employed': dtype('O'),
 'ApplicantIncome': dtype('int64'),
 'CoapplicantIncome': dtype('float64'),
 'LoanAmount': dtype('float64'),
 'Loan_Amount_Term': dtype('float64'),
 'Credit_History': dtype('float64'),
 'Property_Area': dtype('O'),
 'Loan_Status': dtype('O')}
```

In [22]:
```python
loan_dataset['Loan_ID'].dtypes
```

Out[22]:
```
dtype('O')
```

In [23]:
```python
loan_dataset.drop('Loan_ID',axis=1,inplace=True)
loan_dataset
```

Out[23]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncon
0	Male	No	0	Graduate	No	5849	(
1	Male	Yes	1	Graduate	No	4583	1508
2	Male	Yes	0	Graduate	Yes	3000	(
3	Male	Yes	0	Not Graduate	No	2583	2358
4	Male	No	0	Graduate	No	6000	(
...	...	...	...	...	...	...	
609	Female	No	0	Graduate	No	2900	(
610	Male	Yes	3+	Graduate	No	4106	(
611	Male	Yes	1	Graduate	No	8072	240
612	Male	Yes	2	Graduate	No	7583	(
613	Female	No	0	Graduate	Yes	4583	(

614 rows × 12 columns

Observation:

- After quick analysis Loan_ID is a applicant ID which data type is object, hence removing Loan_ID column.

Pre-Observations:

- ML models develop using maths
- we can't provide directly categorical columns data while train the model
- we need convert categorical data to numerical data at some point
- so will specifically works on categorical data
- so it is better to sepereate categorical and numerical columns
- I need two lists:
    - one: categorical_columns_list
    - second: Numerical_columns_list
- series type can convert into dictionary by using dictionary type casting
- Any thing either convert into list or dictionary
- If you give the dictionary or list It can make a datafarme

In [24]:
```
#Method:1

col_dict= dict(loan_dataset.dtypes)
cat_list, num_list =[],[]
for key,value in col_dict.items():
    if value=='O':
        cat_list.append(key)
    else:
        num_list.append(key)

print('Categorical column list:',cat_list)
print('Numericl column list:',num_list)
```

```
Categorical column list: ['Gender', 'Married', 'Dependents', 'Education', 'Self_Em
ployed', 'Property_Area', 'Loan_Status']
Numericl column list: ['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan
_Amount_Term', 'Credit_History']
```

In [25]:
```
#Method:2

cat1_list=[key for key,value in col_dict.items() if value=='O']

num1_list=[key for key,value in col_dict.items() if value!='O']

print(cat1_list)
print(num1_list)
```

```
['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Property_Area',
'Loan_Status']
['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term', 'Credit
_History']
```

Categorical Analysis:

In [26]:
```
cat_cols=loan_dataset.columns[loan_dataset.dtypes=='O']
cat_cols
```

Out[26]:
```
Index(['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
       'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [27]:
```
# Count the null values:

loan_dataset[cat_cols].isnull().sum()
```

Out[27]:
```
Gender           13
Married           3
Dependents       15
Education         0
Self_Employed    32
Property_Area     0
Loan_Status       0
dtype: int64
```

Observations:

- After Analyzing the categorical columns found that there are 13 missing values in Gender column , 3 missing values in Married column and 32 missing values in Self_employeed column
- Overall we have 48 missing values
- Hence filled Missing values with Mode values

In [28]:
```
loan_df=loan_dataset[cat_cols]
loan_df=loan_df.fillna(loan_df.mode().iloc[0])
loan_df[cat_cols]=loan_df
loan_df[cat_cols].isnull().sum()
```

Out[28]:
```
Gender           0
Married          0
Dependents       0
Education        0
Self_Employed    0
Property_Area    0
Loan_Status      0
dtype: int64
```

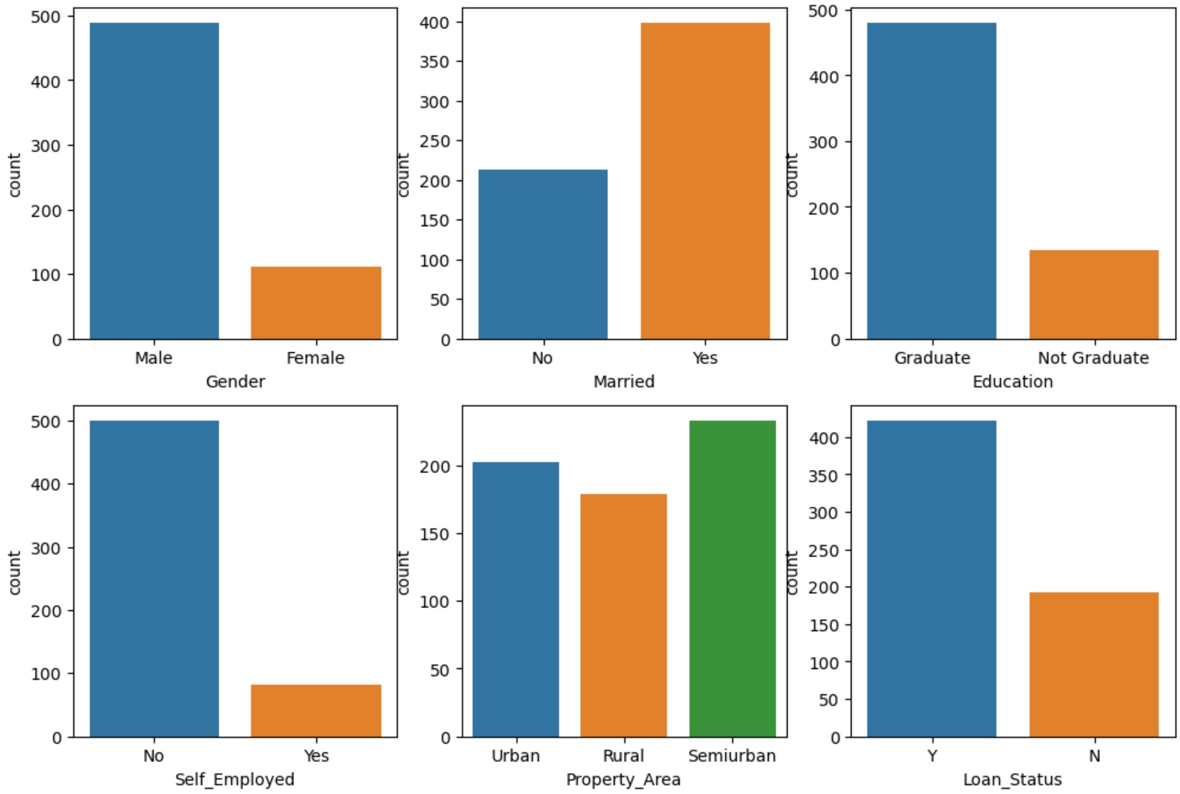
Creating Pie chart and Bar plot

In [29]:
```
import matplotlib.pyplot as plt

loan_dataset['Gender'].value_counts()
names1=loan_dataset['Gender'].value_counts().keys()
values1=loan_dataset['Gender'].value_counts().to_list()

loan_dataset['Married'].value_counts()
names2=loan_dataset['Married'].value_counts().keys()
values2=loan_dataset['Married'].value_counts().to_list()

loan_dataset['Education'].value_counts()
names3=loan_dataset['Education'].value_counts().keys()
values3=loan_dataset['Education'].value_counts().to_list()

loan_dataset['Self_Employed'].value_counts()
names4=loan_dataset['Self_Employed'].value_counts().keys()
```

```
values4=loan_dataset['Self_Employed'].value_counts().to_list()


loan_dataset['Property_Area'].value_counts()
names5=loan_dataset['Property_Area'].value_counts().keys()
values5=loan_dataset['Property_Area'].value_counts().to_list()

loan_dataset['Loan_Status'].value_counts()
names6=loan_dataset['Loan_Status'].value_counts().keys()
values6=loan_dataset['Loan_Status'].value_counts().to_list()

fig,axes=plt.subplots(2,3,figsize=(12,8))

axes[0,0].pie(values1,labels=names1,autopct='%0.2f%%')
axes[0,0].set_title('Gender')
axes[0,1].pie(values2,labels=names2,autopct='%0.2f%%')
axes[0,1].set_title('Married')
axes[0,2].pie(values3,labels=names3,autopct='%0.2f%%')
axes[0,2].set_title('Education')
axes[1,0].pie(values4,labels=names4,autopct='%0.2f%%')
axes[1,0].set_title('Self_Employed')
axes[1,1].pie(values5,labels=names5,autopct='%0.2f%%')
axes[1,1].set_title('Property_Area')
axes[1,2].pie(values6,labels=names6,autopct='%0.2f%%')
axes[1,2].set_title('Loan_status')

plt.show()
```

In [30]:
```
import matplotlib.pyplot as plt
import seaborn as sns

loan_dataset['Gender'].value_counts()
names1=loan_dataset['Gender'].value_counts().keys()
values1=loan_dataset['Gender'].value_counts().to_list()

loan_dataset['Married'].value_counts()
names2=loan_dataset['Married'].value_counts().keys()
values2=loan_dataset['Married'].value_counts().to_list()
```

```
loan_dataset['Education'].value_counts()
names3=loan_dataset['Education'].value_counts().keys()
values3=loan_dataset['Education'].value_counts().to_list()

loan_dataset['Self_Employed'].value_counts()
names4=loan_dataset['Self_Employed'].value_counts().keys()
values4=loan_dataset['Self_Employed'].value_counts().to_list()


loan_dataset['Property_Area'].value_counts()
names5=loan_dataset['Property_Area'].value_counts().keys()
values5=loan_dataset['Property_Area'].value_counts().to_list()

loan_dataset['Loan_Status'].value_counts()
names6=loan_dataset['Loan_Status'].value_counts().keys()
values6=loan_dataset['Loan_Status'].value_counts().to_list()

fig,axes = plt.subplots(2,3,figsize=(12,8))

sns.countplot(data=loan_dataset,x='Gender',ax=axes[0,0])
sns.countplot(data=loan_dataset,x='Married',ax=axes[0,1])
sns.countplot(data=loan_dataset,x='Education',ax=axes[0,2])
sns.countplot(data=loan_dataset,x='Self_Employed',ax=axes[1,0])
sns.countplot(data=loan_dataset,x='Property_Area',ax=axes[1,1])
sns.countplot(data=loan_dataset,x='Loan_Status',ax=axes[1,2])
plt.show()
```



Observations:

we can make the following observations: 1) Gender: 81.36% of the individuals in the dataset are male, while the remaining 18.64% are female. This indicates higher representation of males in dataset. 2) Marital Status: 65.14% of the candidates in the dataset are married, while the remaining 34.86% are unmarried. This indicates majority of the candidates in the dataset are married. 3) Education:78.18% of the candidates in the dataset have a graduate education, while the remaining 21.82% do't have a graduate degree. This indicates that most of the candidates in the dataset are graduates. 4) Self-Employment: 85.91% of the candidates in the dataset are not selfemployed, while the remaining 14.09% are self-employed. This indicates that a majority of the candidates in the dataset are not self-employed. 5) Property Area: 37.95% of the candidates reside in semiurban areas, 32.90% in urban areas, and the remaining 29.15% in rural areas. Higher proportion of candidates residing in semiurban and urban areas. 6) Loan Status: In the total dataset, approximately 68.73% have been approved for a loan, while the remaining 31.27% have been rejected.

Numerical Analysis

In [31]:
```
file_name=('C:\\Users\\venka\\supraja\\train_ctrUa4K.csv')
loan_dataset=pd.read_csv(file_name)
loan_dataset
```

Out[31]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapp
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 [32]:
```
loan_dataset['Dependents']
```

Out[32]:
```
0       0
1       1
2       0
3       0
4       0
       ..
609     0
610    3+
611     1
612     2
613     0
Name: Dependents, Length: 614, dtype: object
```

In [33]:
```
loan_dataset['Dependents']=loan_dataset['Dependents'].replace('3+',3)
```

```
loan_dataset['Dependents']
```

Out[33]:

```
0      0
1      1
2      0
3      0
4      0
      ..
609    0
610    3
611    1
612    2
613    0
Name: Dependents, Length: 614, dtype: object
```

Observation:

- Generalizing the Dependents column as it contains 3+ which is object.
- So we replacing it with value 3.

In [34]:

```
loan_dataset["Dependents"] = loan_dataset["Dependents"].replace("3+", 3)
loan_dataset["Dependents"]=loan_dataset["Dependents"].fillna(loan_dataset["Depende
loan_dataset["Dependents"] = loan_dataset["Dependents"].astype("int")
loan_dataset["Dependents"].dtypes
```

Out[34]:

```
dtype('int32')
```

In [35]:

```
num_cols=loan_dataset.columns[loan_dataset.dtypes!='O']
num_cols
```

Out[35]:

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

In [36]:

```
loan_dataset[num_cols].isnull().sum()
```

Out[36]:

```
Dependents            0
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
dtype: int64
```

Observation:

- After analysis of data 86 missing values are found in which LoanAmount has 22 missing values,Loan_Amount_Term has 14 missing values and Credit_History has 50 msissing values
- Hence filling all missing values with mode values.

In [37]:

```
loan_df=loan_dataset[num_cols].fillna(loan_dataset.mode().iloc[0])
loan_dataset[num_cols]=loan_df
loan_dataset[num_cols].isnull().sum()
```

```
Out[37]: Dependents           0
         ApplicantIncome      0
         CoapplicantIncome    0
         LoanAmount           0
         Loan_Amount_Term     0
         Credit_History       0
         dtype: int64
```

```
In [38]: loan_dataset
```

Out[38]:

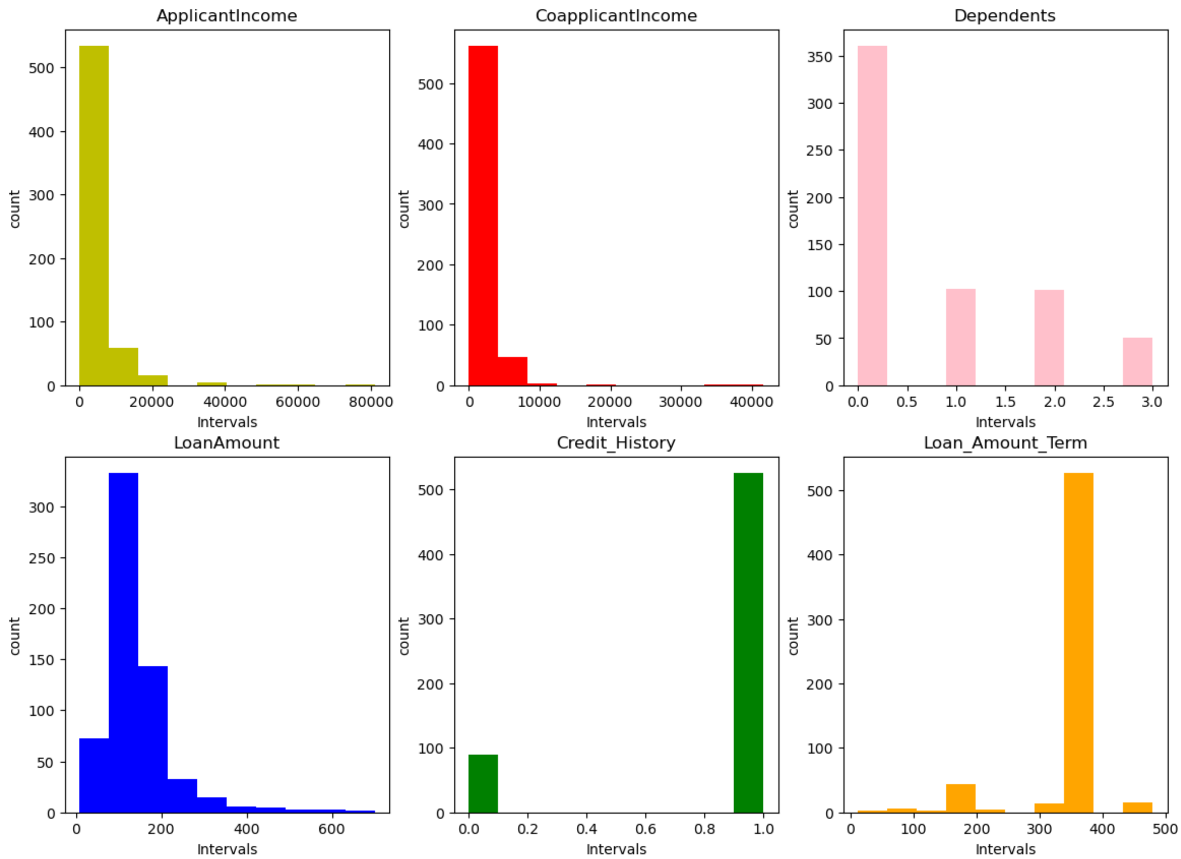
	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapp
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	
4	LP001008	Male	No	0	Graduate	No	6000	
...	...	...	...	...	...	...	...	
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

Histogram:

```
In [39]: fig,axes = plt.subplots(2,3, figsize=(14,10))

axes[0,0].hist(loan_dataset['ApplicantIncome'], bins=10,color='y')
axes[0,0].set_xlabel("Intervals")
axes[0,0].set_ylabel("count")
axes[0,0].set_title("ApplicantIncome")

axes[0,1].hist(loan_dataset['CoapplicantIncome'], bins=10,color='r')
axes[0,1].set_xlabel("Intervals")
axes[0,1].set_ylabel("count")
axes[0,1].set_title("CoapplicantIncome")

axes[1,0].hist(loan_dataset['LoanAmount'], bins=10,color='b')
axes[1,0].set_xlabel("Intervals")
axes[1,0].set_ylabel("count")
axes[1,0].set_title("LoanAmount")

axes[1,1].hist(loan_dataset['Credit_History'], bins=10,color='g')
axes[1,1].set_xlabel("Intervals")
axes[1,1].set_ylabel("count")
axes[1,1].set_title("Credit_History")

axes[0,2].hist(loan_dataset['Dependents'], bins=10,color='pink')
```

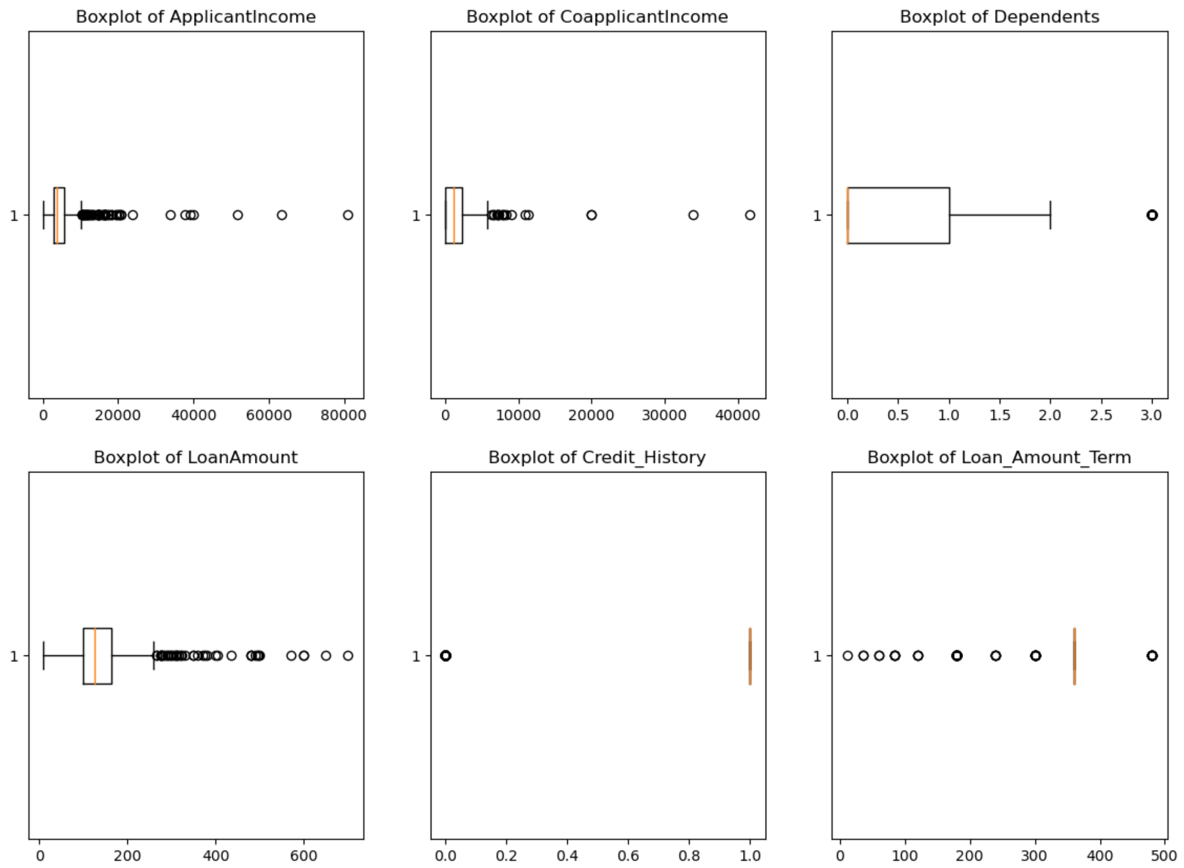
```
axes[0,2].set_xlabel("Intervals")
axes[0,2].set_ylabel("count")
axes[0,2].set_title("Dependents")

axes[1,2].hist(loan_dataset['Loan_Amount_Term'], bins=10,color='orange')
axes[1,2].set_xlabel("Intervals")
axes[1,2].set_ylabel("count")
axes[1,2].set_title("Loan_Amount_Term")


plt.show()
```



Box Plot:

In [40]:
```
fig,axes = plt.subplots(2,3, figsize=(14,10))

axes[0, 0].boxplot(loan_dataset['ApplicantIncome'],vert=False)
axes[0, 0].set_title("Boxplot of ApplicantIncome")

axes[0, 1].boxplot(loan_dataset['CoapplicantIncome'],vert=False)
axes[0, 1].set_title("Boxplot of CoapplicantIncome")

axes[0, 2].boxplot(loan_dataset['Dependents'],vert=False)
axes[0, 2].set_title("Boxplot of Dependents")

axes[1, 0].boxplot(loan_dataset['LoanAmount'],vert=False)
axes[1, 0].set_title("Boxplot of LoanAmount")

axes[1, 1].boxplot(loan_dataset['Credit_History'],vert=False)
axes[1, 1].set_title("Boxplot of Credit_History")

axes[1, 2].boxplot(loan_dataset['Loan_Amount_Term'],vert=False)
axes[1, 2].set_title("Boxplot of Loan_Amount_Term")
```

```python
plt.show()
```



Observations:

1) Dependents column: The maximum outlier value is observed at 3.0. 2) ApplicantIncome column: The presence of outliers is observed at the higher end of the income scale, specifically in the range of 20000 to 40000. Exceeding the income levels of the majority. 3) CoapplicantIncome column: outliers are observed atthe higher end of the income scale, specifically in the range of 10000 to 20000. 4) LoanAmountTerm column: Outliers are identified at the higher end of the loan amount term scale, specifically in the range of 400 to 500. 5) LoanAmount: Outliers are identified at higher end of the loan amount scale, specifically in the range of 200 to 600.

Describe() using for outliers analysis:

In [41]:
```python
loan_dataset.describe()
```

Out[41]:

	Dependents	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credi
count	614.000000	614.000000	614.000000	614.000000	614.000000	6
mean	0.744300	5403.459283	1621.245798	145.465798	342.410423	
std	1.009623	6109.041673	2926.248369	84.180967	64.428629	
min	0.000000	150.000000	0.000000	9.000000	12.000000	
25%	0.000000	2877.500000	0.000000	100.250000	360.000000	
50%	0.000000	3812.500000	1188.500000	125.000000	360.000000	
75%	1.000000	5795.000000	2297.250000	164.750000	360.000000	
max	3.000000	81000.000000	41667.000000	700.000000	480.000000	

percentile and **quantiles**

- Perentile: 1 to 100
- quantile: 25 50 75

In [42]:

```
data=loan_dataset['ApplicantIncome']
np.percentile(data,[25,50,75])
```

Out[42]:

```
array([2877.5, 3812.5, 5795. ])
```

In [43]:

```
np.quantile(data,[0.25,0.50,0.75])
```

Out[43]:

```
array([2877.5, 3812.5, 5795. ])
```

Step1: Find Q1:25 Q2:50 Q3:75

- Step-2: IQR : (Q3-Q1)
- Step-3: Q1-1.5*IQR : Lower bound
- Step-4: Q3+1.5*IQR : Upper bound

In [44]:

```
Q1=np.quantile(data,0.25)
Q2=np.quantile(data,0.50)
Q3=np.quantile(data,0.75)
IQR=Q3-Q1
LB=Q1-1.5*IQR
UB=Q3+1.5*IQR
print(Q1,Q2,Q3,IQR,LB,UB)
```

```
2877.5 3812.5 5795.0 2917.5 -1498.75 10171.25
```

In [45]:

```
# outliers are less than LB
# outlier are greater than UB
# c1:  data < lb
# c2:  data > ub
# c1 oR c2
data= loan_dataset['ApplicantIncome']
                   #,'CoapplicantIncome','LoanAmount','Loan_Amount_Term','Credit_H
con1=data<LB
```

```
con2=data>UB
con1|con2
```

Out[45]:

```
0      False
1      False
2      False
3      False
4      False
       ...
609    False
610    False
611    False
612    False
613    False
Name: ApplicantIncome, Length: 614, dtype: bool
```

Fill the outliers with medain values

In [46]:

```
median=loan_dataset['ApplicantIncome'].median()

cond=loan_dataset['ApplicantIncome']>UB

loan_dataset['ApplicantIncome']=np.where(cond,median,loan_dataset['ApplicantIncome

loan_dataset['ApplicantIncome']
```

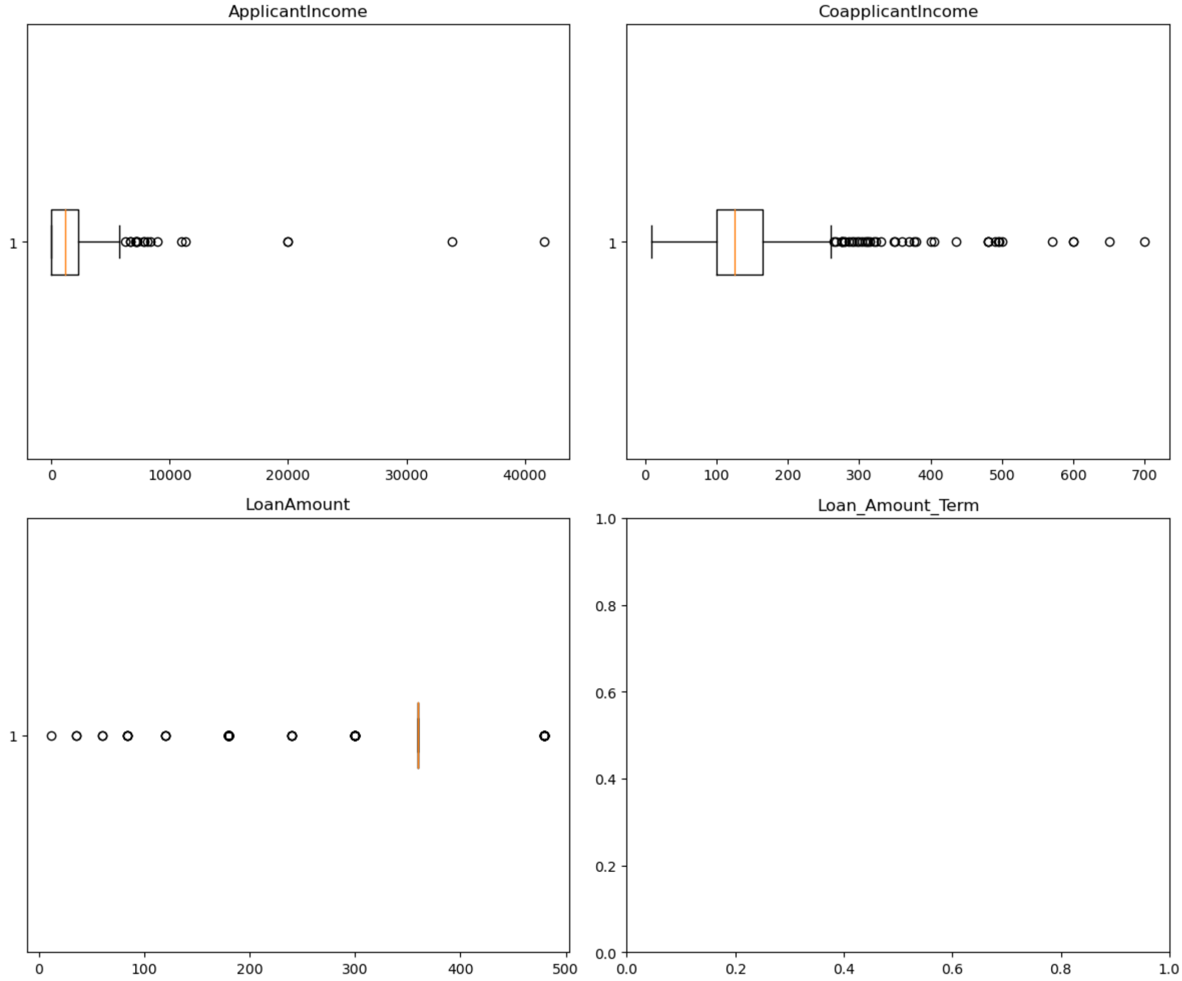
Out[46]:

```
0      5849.0
1      4583.0
2      3000.0
3      2583.0
4      6000.0
        ...
609    2900.0
610    4106.0
611    8072.0
612    7583.0
613    4583.0
Name: ApplicantIncome, Length: 614, dtype: float64
```

In [47]:

```
plt.figure(figsize=(12, 10))
plt.boxplot(loan_dataset['ApplicantIncome'],vert=False)
plt.subplot(2,2,1)
plt.title('ApplicantIncome')
plt.boxplot(loan_dataset['CoapplicantIncome'],vert=False)
plt.subplot(2,2,2)
plt.title('CoapplicantIncome')
plt.boxplot(loan_dataset['LoanAmount'],vert=False)
plt.subplot(2,2,3)
plt.title('LoanAmount')
plt.boxplot(loan_dataset['Loan_Amount_Term'],vert=False)
plt.subplot(2,2,4)
plt.title('Loan_Amount_Term')
plt.tight_layout()
plt.show()
```

```
C:\Users\venka\AppData\Local\Temp\ipykernel_13540\2183012049.py:3: MatplotlibDepre
cationWarning: Auto-removal of overlapping axes is deprecated since 3.6 and will b
e removed two minor releases later; explicitly call ax.remove() as needed.
  plt.subplot(2,2,1)
```



Bivariate analysis

In [48]:
```
loan_dataset['Property_Area'].value_counts()
```

Out[48]:
```
Semiurban    233
Urban        202
Rural        179
Name: Property_Area, dtype: int64
```

In [49]:
```
con1=loan_dataset['Property_Area']=='Urban'
Urban=loan_dataset[con1]
len(Urban)
```

Out[49]:
```
202
```

In [50]:
```
# How many loanstatus=Y in total loan datasets
len(loan_dataset[loan_dataset['Loan_Status']=='Y'])
```

Out[50]:
```
422
```

In [51]:
```
#Q) Out of Urban (202) applicants,  how many are eligible for loan?

# Method-1:

con1=loan_dataset['Property_Area']=='Urban'
Urban=loan_dataset[con1]

con2=Urban['Loan_Status']=='Y'
print("Eligible for loan in urban area:",len(Urban[con2]))
```

```
Eligible for loan in urban area: 133
```

In [52]:
```python
# Method-2:
# I will retrive two data frames from original dataframe
# Property_Area= Urban
# loan status= Y

con1=loan_dataset['Property_Area']=='Urban'
con2=loan_dataset['Loan_Status']=='Y'

loan_dataset[con1&con2]
```

Out[52]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapp
0	LP001002	Male	No	0	Graduate	No	5849.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583.0	
4	LP001008	Male	No	0	Graduate	No	6000.0	
5	LP001011	Male	Yes	2	Graduate	Yes	5417.0	
...	...	...	...	...	...	...	...	
594	LP002938	Male	Yes	0	Graduate	Yes	3812.5	
599	LP002948	Male	Yes	2	Graduate	No	5780.0	
602	LP002953	Male	Yes	3	Graduate	No	5703.0	
611	LP002983	Male	Yes	1	Graduate	No	8072.0	
612	LP002984	Male	Yes	2	Graduate	No	7583.0	

133 rows × 13 columns

In [53]:
```python
Property_Area=['Semiurban','Urban','Rural']
for i in Property_Area:
    count=len(loan_dataset[(loan_dataset['Property_Area']==i)&
                           (loan_dataset['Loan_Status']=='Y')])
    print("No. of applicants are eligible to loan in {}: {}".format(i,count))
```

```
No. of applicants are eligible to loan in Semiurban: 179
No. of applicants are eligible to loan in Urban: 133
No. of applicants are eligible to loan in Rural: 110
```

In [54]:
```python
# Method:3
Y,N=[],[]
Property_Area=loan_dataset['Property_Area'].unique()
for i in Property_Area:
    con1=(loan_dataset['Property_Area']==i)
    con2=(loan_dataset['Loan_Status']=='Y')
    con3=(loan_dataset['Loan_Status']=='N')
    count=len(loan_dataset[con1&con2])
    count_cer=len(loan_dataset[con1&con3])
    Y.append(count)
    N.append(count_cer)
print("No. of applicants are eligible to loan {} ".format(Y))
print("No. of applicants are eligible to loan {}".format(N))
```

```
No. of applicants are eligible to loan [133, 110, 179]
No. of applicants are eligible to loan [69, 69, 54]
```

In [55]:
```
# Method:1
pd.DataFrame(zip(Y,N),
            columns=['Eligible','Not Eligible'],
            index=loan_dataset['Property_Area'].unique())
```

Out[55]:

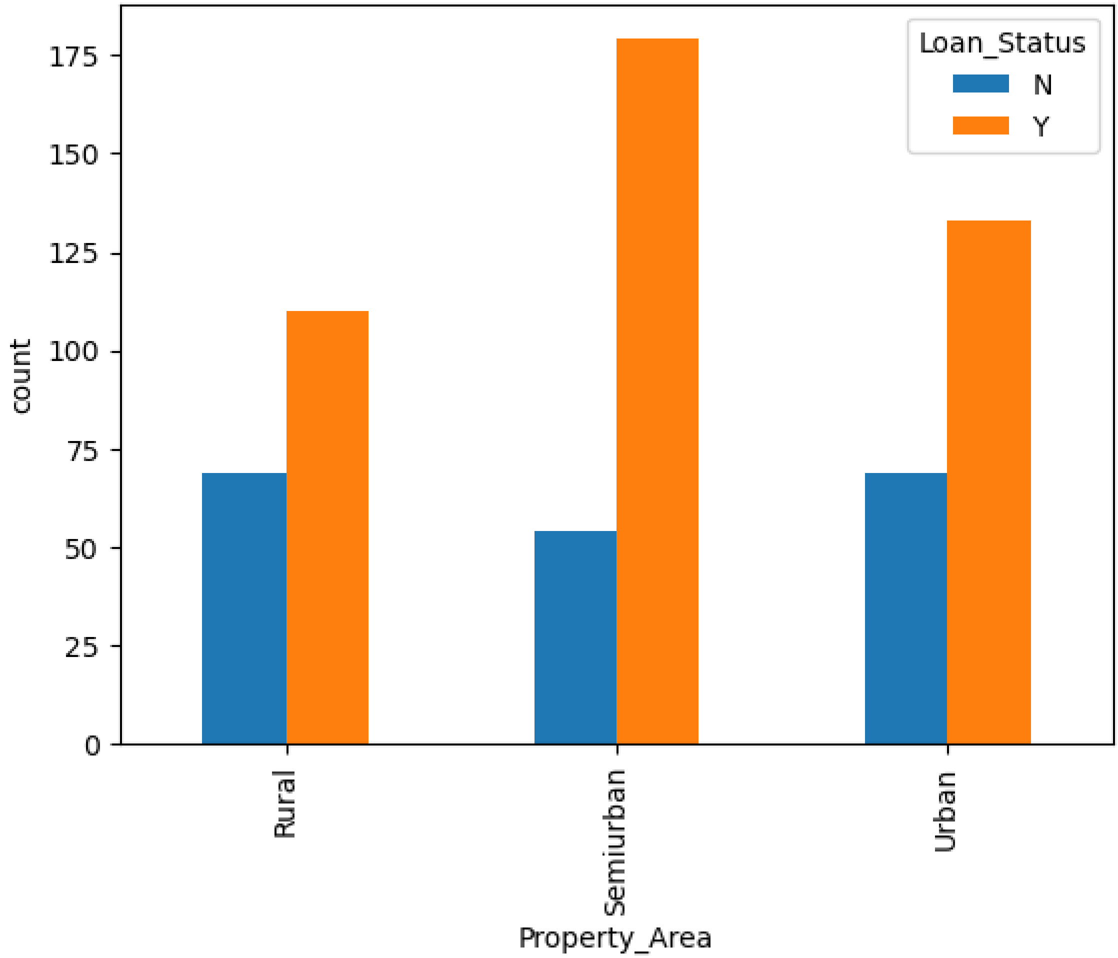
	Eligible	Not Eligible
Urban	133	69
Rural	110	69
Semiurban	179	54

In [56]:
```
#Method:2
col1=loan_dataset['Property_Area']
col2=loan_dataset['Loan_Status']
result=pd.crosstab(col1,col2)
result
```

Out[56]:

Loan_Status	N	Y
Property_Area		
Rural	69	110
Semiurban	54	179
Urban	69	133

In [57]:
```
result.plot(kind='bar')
plt.ylabel("count")
plt.show()
```



In [58]:
```
data1=loan_dataset[loan_dataset['Loan_Status']=='Y']
data1=data1[['Gender','Loan_Status']]
data1
```

Out[58]:

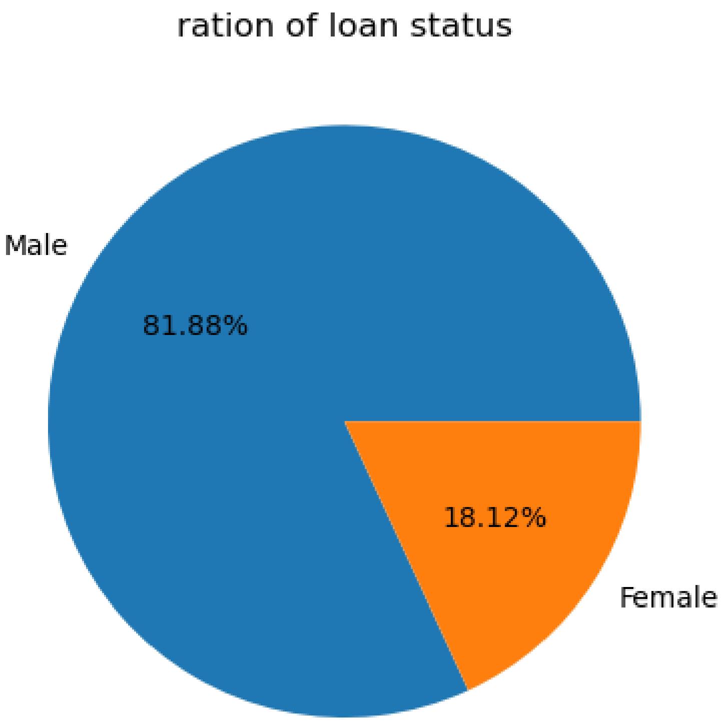
	Gender	Loan_Status
0	Male	Y
2	Male	Y
3	Male	Y
4	Male	Y
5	Male	Y
...	...	...
608	Male	Y
609	Female	Y
610	Male	Y
611	Male	Y
612	Male	Y

422 rows × 2 columns

In [59]:
```
data2=dict(data1['Gender'].value_counts())
data2
```

Out[59]: {'Male': 339, 'Female': 75}

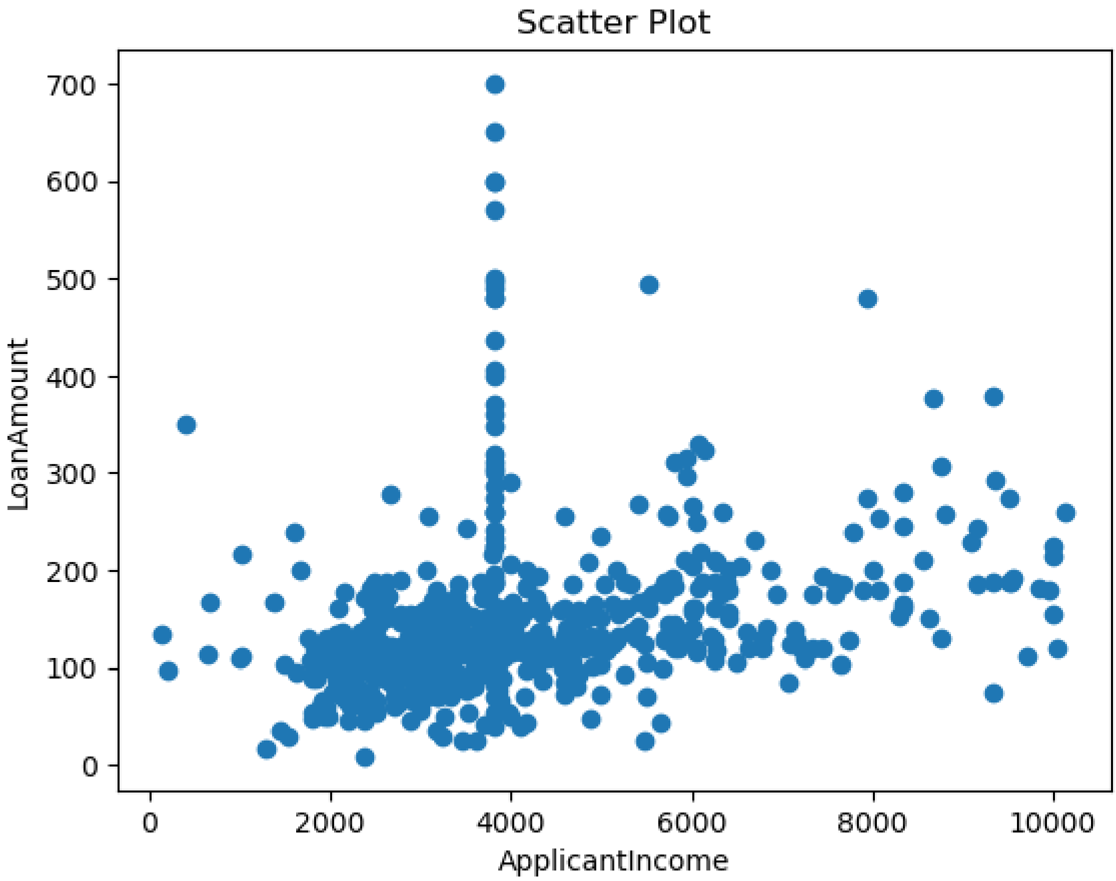
In [60]:
```
values=list(data2.values())
keys=list(data2.keys())
plt.pie(x=values,labels=keys,autopct="%0.2f%%")
plt.title("ration of loan status")
plt.show()
```



Scatter- Plots

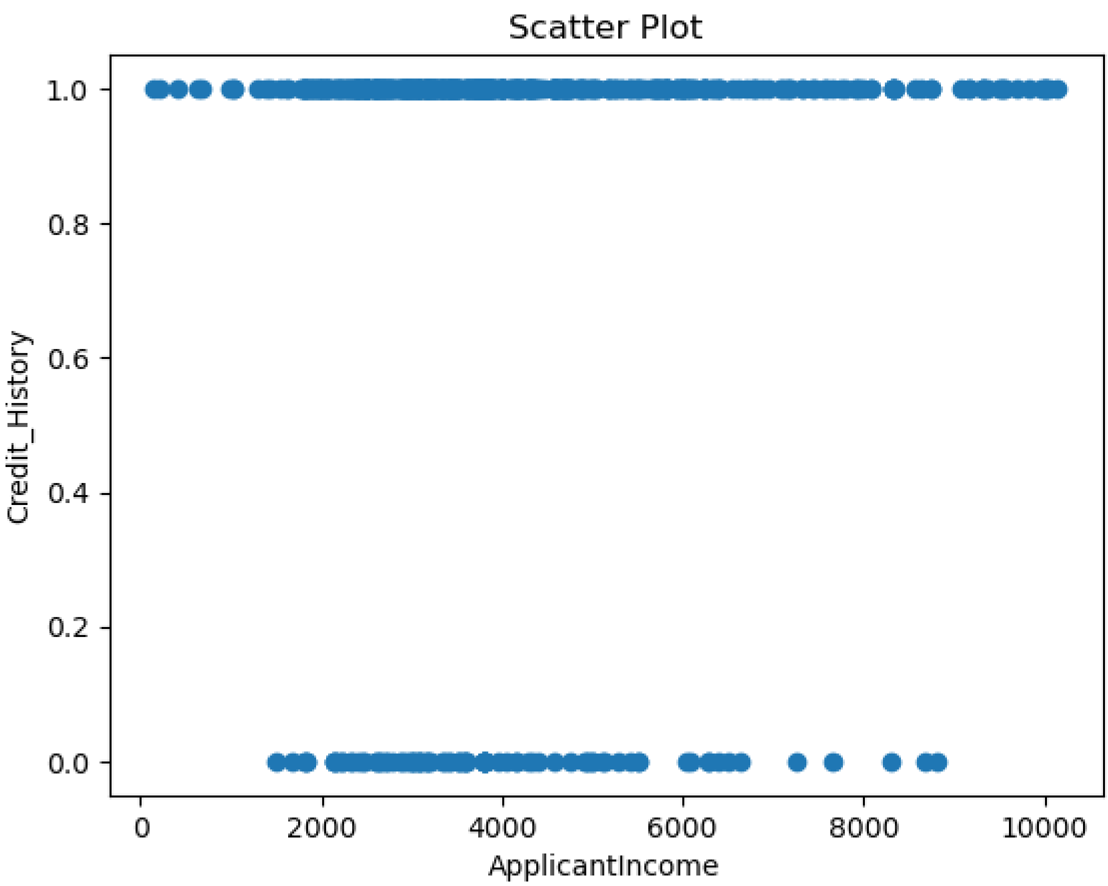
- It provides the relation between two numerical variables
- One variable will with another variable

In [61]:
```
# Provide the scatter plots between
# ApplicantIncome vs Loan amount
col3=loan_dataset['ApplicantIncome']
col4=loan_dataset['LoanAmount']
plt.scatter(col3,col4)
plt.xlabel('ApplicantIncome')
plt.ylabel('LoanAmount')
plt.title('Scatter Plot')
plt.show()
```



In [62]:

```
# Provide the scatter plots between
# ApplicantIncome vs Credit_History
col3=loan_dataset['ApplicantIncome']
col4=loan_dataset['Credit_History']
plt.scatter(col3,col4)
plt.xlabel('ApplicantIncome')
plt.ylabel('Credit_History')
plt.title('Scatter Plot')
plt.show()
```



Pearson-Correlation-Coeffiecient(r)

q) difference between correlation and covariance

- Covariance: will exalain wether the both features have relation or not
- Correlation: will explain how much the relation between two features

In [63]:
```
loan_dataset.corr()
```

```
C:\Users\venka\AppData\Local\Temp\ipykernel_13540\4246285894.py:1: FutureWarning:
The default value of numeric_only in DataFrame.corr is deprecated. In a future ver
sion, it will default to False. Select only valid columns or specify the value of
numeric_only to silence this warning.
  loan_dataset.corr()
```

Out[63]:

	Dependents	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amoun
Dependents	1.000000	0.117502	0.030430	0.163017	-0.
ApplicantIncome	0.117502	1.000000	-0.171521	0.293497	-0.
CoapplicantIncome	0.030430	-0.171521	1.000000	0.189723	-0.
LoanAmount	0.163017	0.293497	0.189723	1.000000	0.
Loan_Amount_Term	-0.103864	-0.035532	-0.059383	0.037152	1.
Credit_History	-0.040160	0.043470	0.011134	-0.000250	-0.

Heat - Map:

In [64]:
```
corr_values=loan_dataset.corr()
corr_values
```

```
C:\Users\venka\AppData\Local\Temp\ipykernel_13540\3249102990.py:1: FutureWarning:
The default value of numeric_only in DataFrame.corr is deprecated. In a future ver
sion, it will default to False. Select only valid columns or specify the value of
numeric_only to silence this warning.
  corr_values=loan_dataset.corr()
```

Out[64]:

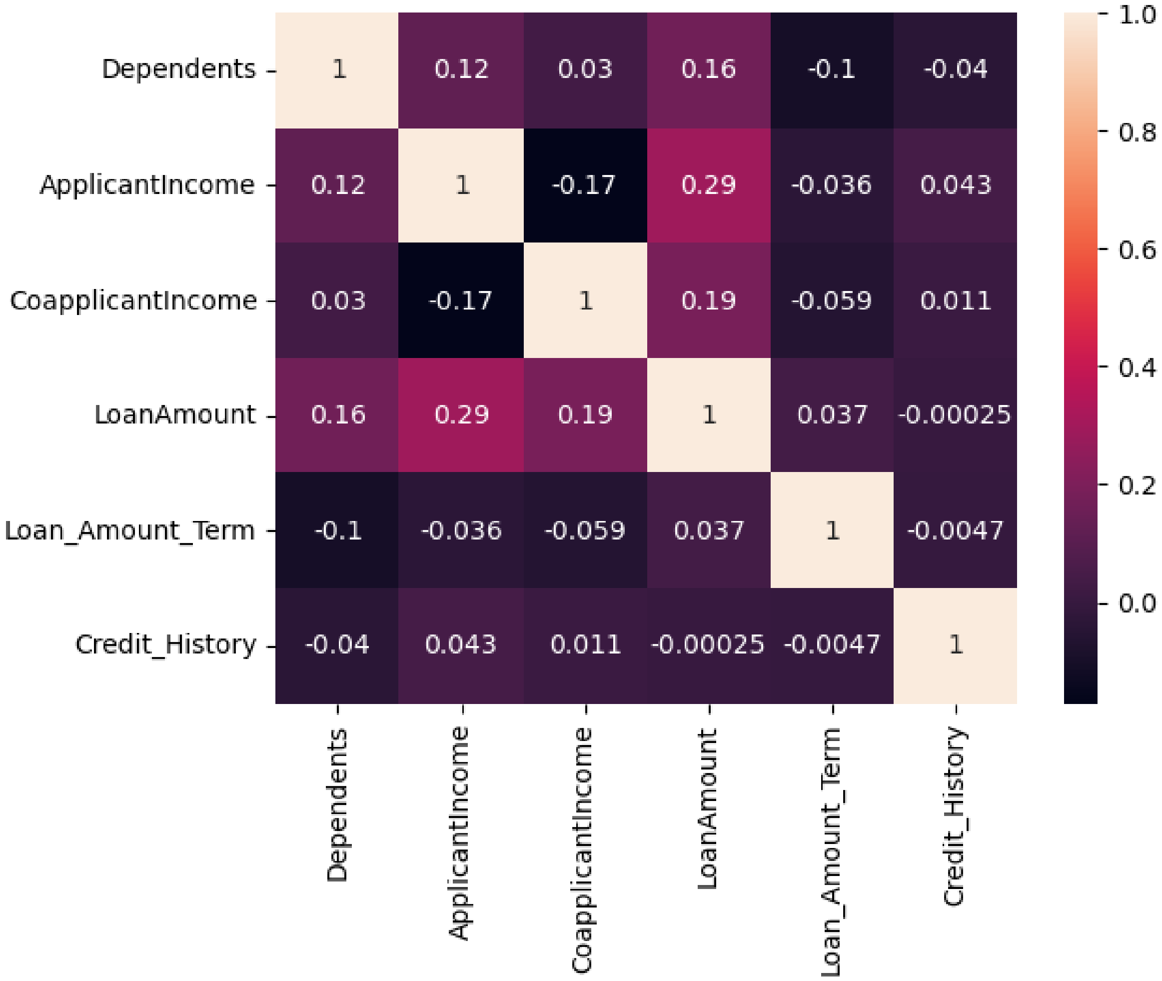
	Dependents	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amoun
Dependents	1.000000	0.117502	0.030430	0.163017	-0.
ApplicantIncome	0.117502	1.000000	-0.171521	0.293497	-0.
CoapplicantIncome	0.030430	-0.171521	1.000000	0.189723	-0.
LoanAmount	0.163017	0.293497	0.189723	1.000000	0.
Loan_Amount_Term	-0.103864	-0.035532	-0.059383	0.037152	1.
Credit_History	-0.040160	0.043470	0.011134	-0.000250	-0.

In [65]:
```
corr_values.values
```

Out[65]:
```
array([[ 1.00000000e+00,  1.17501896e-01,  3.04299667e-02,
         1.63017413e-01, -1.03864123e-01, -4.01598012e-02],
       [ 1.17501896e-01,  1.00000000e+00, -1.71520809e-01,
         2.93496557e-01, -3.55322845e-02,  4.34697037e-02],
       [ 3.04299667e-02, -1.71520809e-01,  1.00000000e+00,
         1.89722837e-01, -5.93830879e-02,  1.11339177e-02],
       [ 1.63017413e-01,  2.93496557e-01,  1.89722837e-01,
         1.00000000e+00,  3.71517394e-02, -2.49920117e-04],
       [-1.03864123e-01, -3.55322845e-02, -5.93830879e-02,
         3.71517394e-02,  1.00000000e+00, -4.70498328e-03],
       [-4.01598012e-02,  4.34697037e-02,  1.11339177e-02,
        -2.49920117e-04, -4.70498328e-03,  1.00000000e+00]])
```

In [66]:
```
sns.heatmap(corr_values,
            annot=True)
```

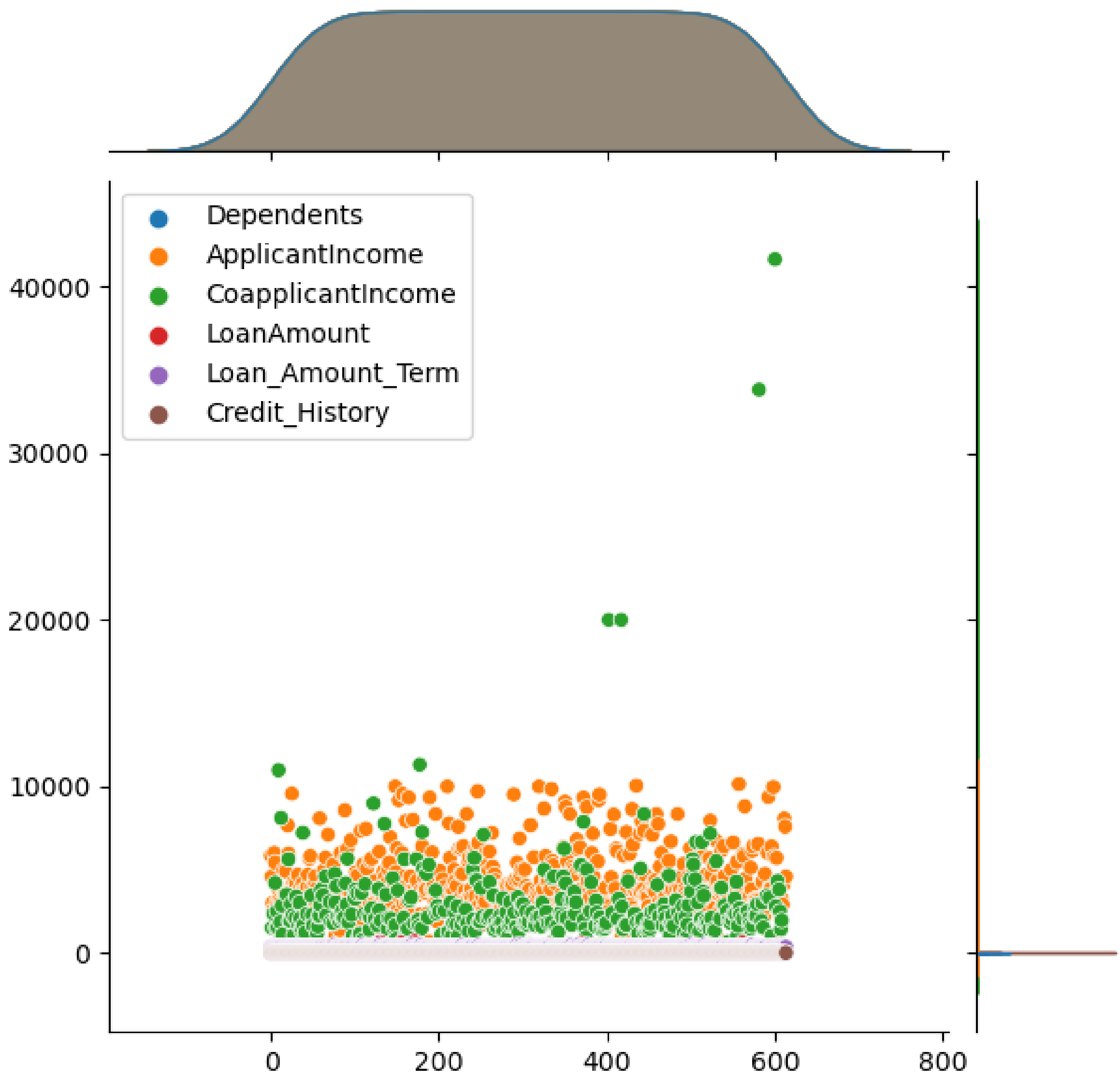
Out[66]:
```
<Axes: >
```



Joint Plot:

In [67]:
```
column1=loan_dataset['ApplicantIncome']
column2=loan_dataset['LoanAmount']
sns.jointplot(loan_dataset)
```

Out[67]: <seaborn.axisgrid.JointGrid at 0x1cdfa1480d0>



Converting Categorical columns to Numerical columns:

- Before we develop ML algorithm , It is very important to do
- ML algorithms developed Maths

In [68]:
```
from sklearn.preprocessing import MinMaxScaler,LabelEncoder,StandardScaler
mns=MinMaxScaler()
le=LabelEncoder()
sc=StandardScaler()
```

In [69]:
```
from sklearn.preprocessing import LabelEncoder  # Import the package
le=LabelEncoder()  # save the package


for i in cat_cols[:]:
    loan_dataset[i]=le.fit_transform(loan_dataset[i]) # Apply fit transform


loan_dataset
```

Out[69]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapp
0	LP001002	1	0	0	0	0	5849.0	
1	LP001003	1	1	1	0	0	4583.0	
2	LP001005	1	1	0	0	1	3000.0	
3	LP001006	1	1	0	1	0	2583.0	
4	LP001008	1	0	0	0	0	6000.0	
...	...	...	...	...	...	...	...	
609	LP002978	0	0	0	0	0	2900.0	
610	LP002979	1	1	3	0	0	4106.0	
611	LP002983	1	1	1	0	0	8072.0	
612	LP002984	1	1	2	0	0	7583.0	
613	LP002990	0	0	0	0	1	4583.0	

614 rows × 13 columns

Standardzing Method:

In [70]:

```
#Method:1
mean=loan_dataset['ApplicantIncome'].mean()
std=loan_dataset['ApplicantIncome'].std()
nr=loan_dataset['ApplicantIncome']-mean
loan_dataset['ApplicantIncome2']=nr/std
```

In [71]:

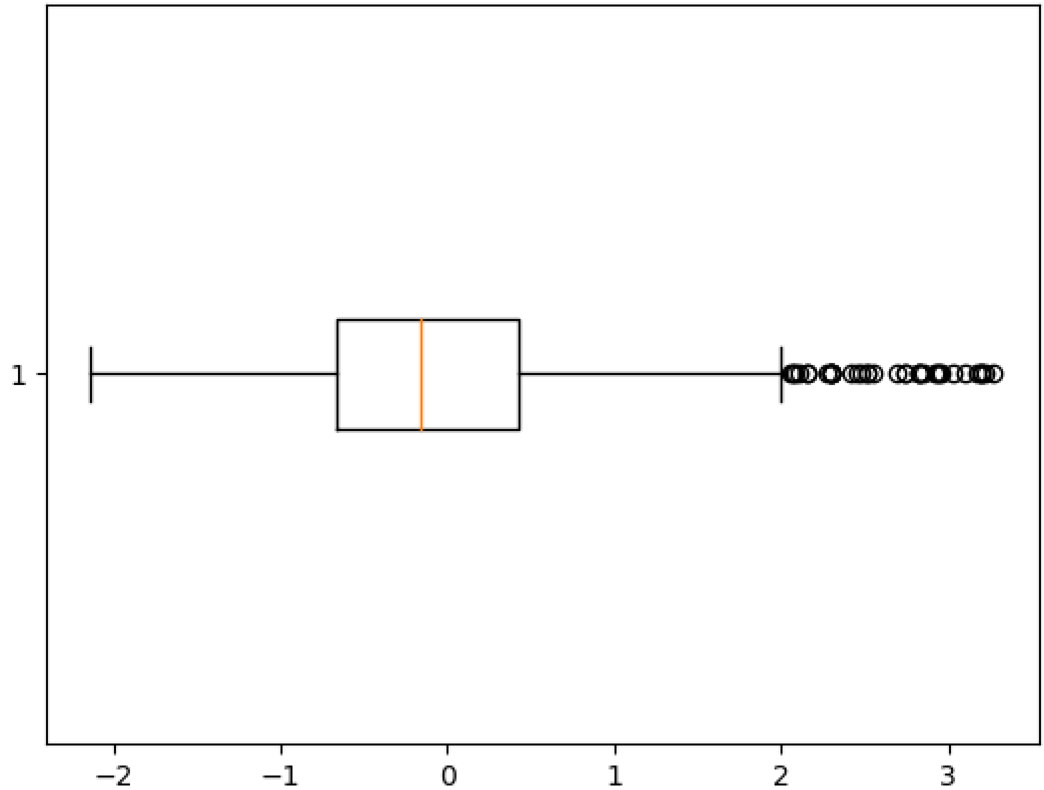
```
loan_dataset['ApplicantIncome2']
```

Out[71]:

```
0      0.946447
1      0.261643
2     -0.594631
3     -0.820194
4      1.028125
         ...
609   -0.648723
610    0.003625
611    2.148909
612    1.884400
613    0.261643
Name: ApplicantIncome2, Length: 614, dtype: float64
```

In [72]:

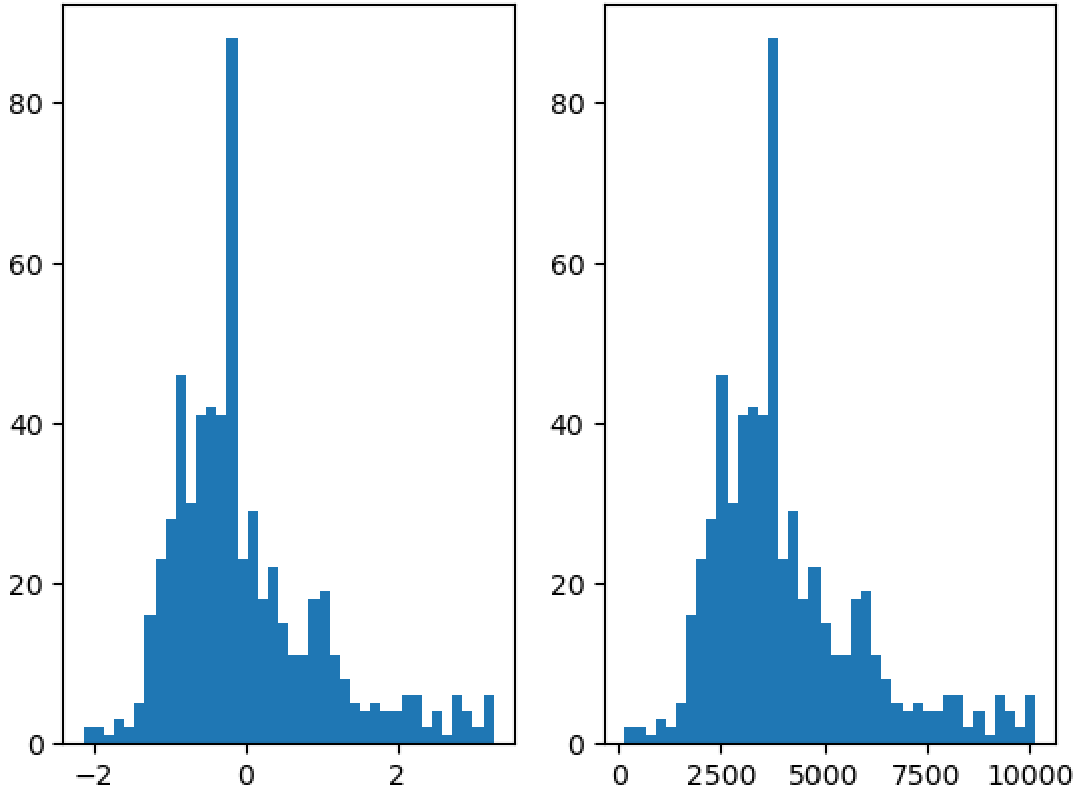
```
plt.boxplot(loan_dataset['ApplicantIncome2'],vert=False)
plt.show()
```



In [73]:
```
con1=loan_dataset['ApplicantIncome2']<3
con2=loan_dataset['ApplicantIncome2']>-3
count=len(loan_dataset[con1&con2])
count
```

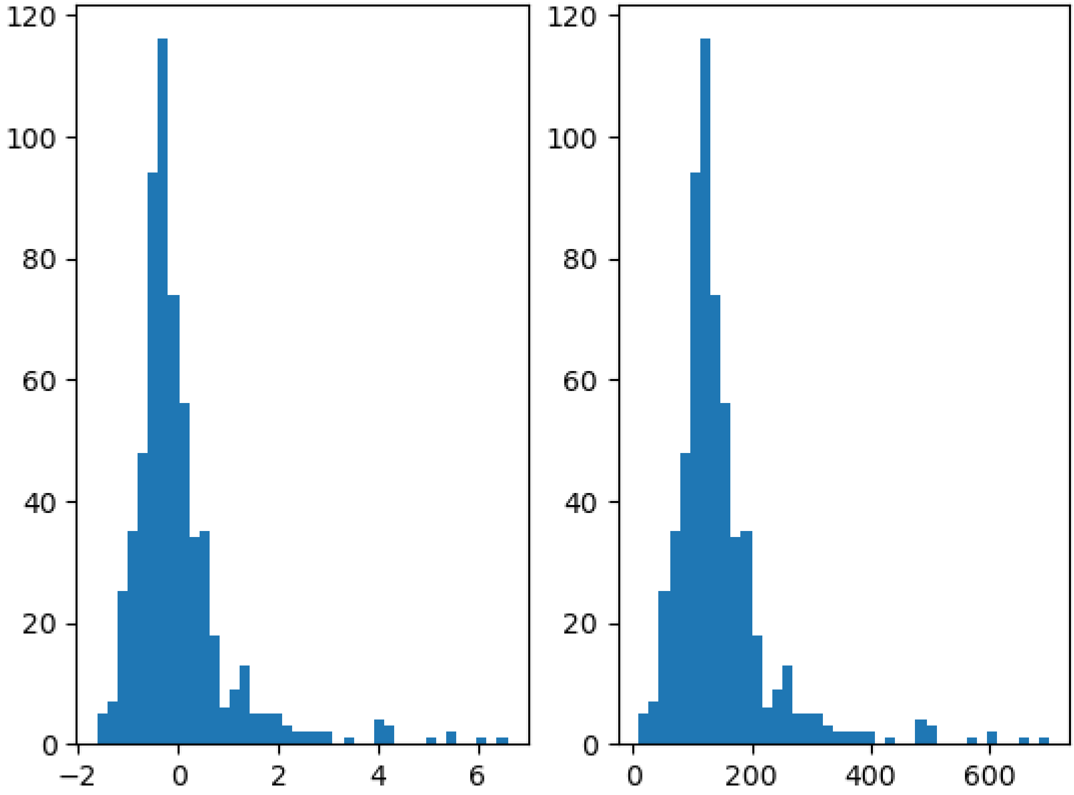
Out[73]:
```
606
```

In [74]:
```
import matplotlib.pyplot as plt
plt.subplot(1,2,1)
plt.hist(loan_dataset['ApplicantIncome2'],bins=40)

plt.subplot(1,2,2)
plt.hist(loan_dataset['ApplicantIncome'],bins=40)

plt.show()
```



In [75]:
```
#Method:2

from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
loan_dataset['LoanAmount2']=ss.fit_transform(loan_dataset[['LoanAmount']])
```

In [76]:
```
import matplotlib.pyplot as plt
plt.subplot(1,2,1)
plt.hist(loan_dataset['LoanAmount2'],bins=40)

plt.subplot(1,2,2)
plt.hist(loan_dataset['LoanAmount'],bins=40)

plt.show()
```



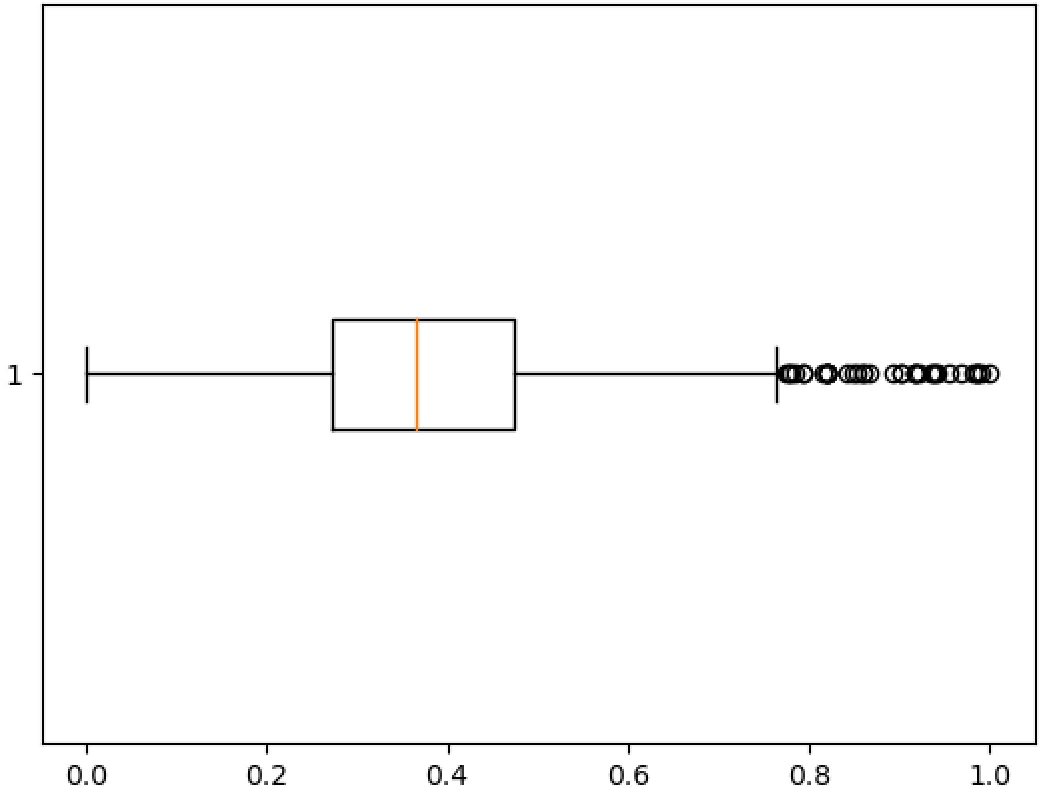
Normalization Method:

In [77]:
```
#Method:1
min_value=loan_dataset['ApplicantIncome'].min()
max_value=loan_dataset['ApplicantIncome'].max()
nr=loan_dataset['ApplicantIncome']-min_value
dr=max_value-min_value
loan_dataset['ApplicantIncome_new']=nr/dr
```

In [78]:
```
loan_dataset['ApplicantIncome_new']
```

Out[78]:
```
0      0.570528
1      0.443788
2      0.285314
3      0.243568
4      0.585644
         ...
609    0.275303
610    0.396036
611    0.793072
612    0.744119
613    0.443788
Name: ApplicantIncome_new, Length: 614, dtype: float64
```

In [79]:
```
plt.boxplot(loan_dataset['ApplicantIncome_new'],vert=False)
plt.show()
```

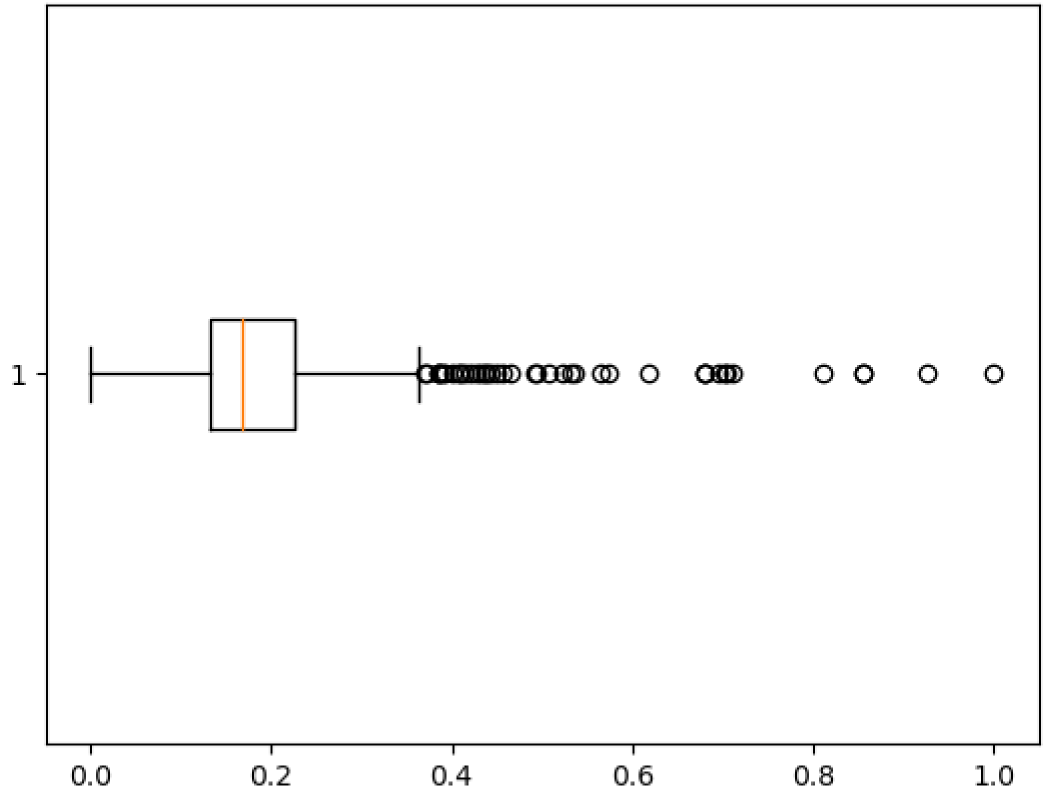


In [80]:
```
#Method:2
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler()
loan_dataset['LoanAmount_new']=mms.fit_transform(loan_dataset[['LoanAmount']])
```

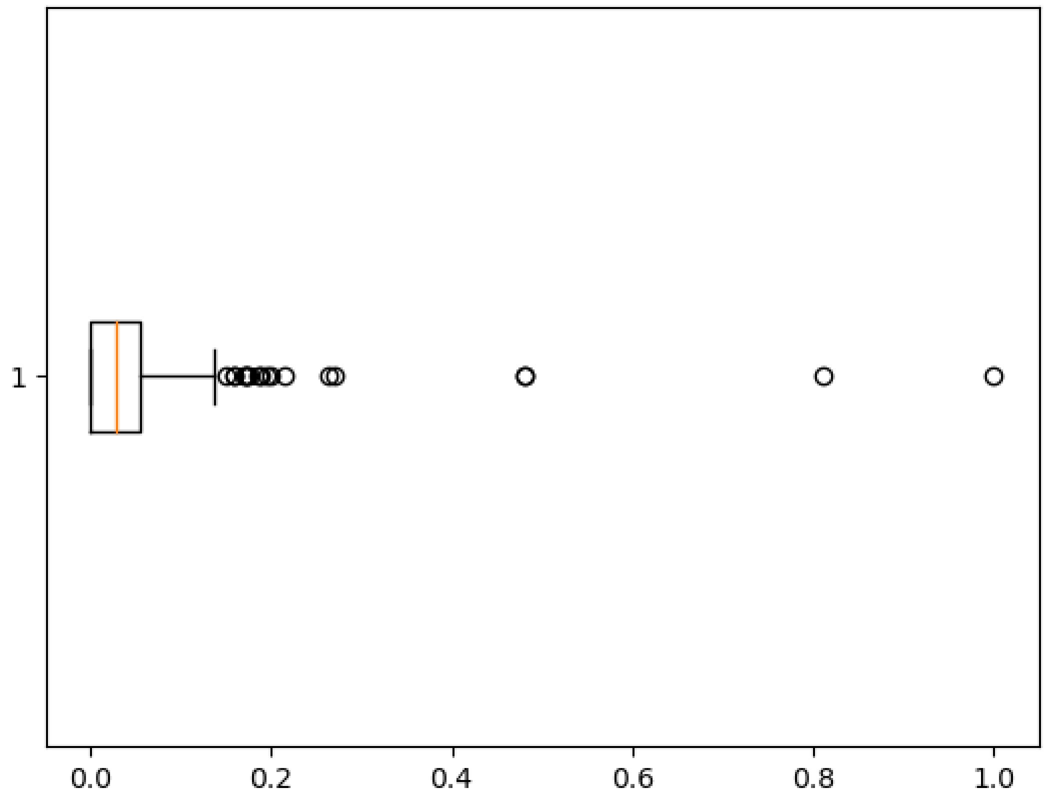
In [81]:
```
loan_dataset['LoanAmount_new']
```

Out[81]:
```
0      0.160637
1      0.172214
2      0.082489
3      0.160637
4      0.191027
         ...
609    0.089725
610    0.044863
611    0.353111
612    0.257598
613    0.179450
Name: LoanAmount_new, Length: 614, dtype: float64
```

In [82]:
```
plt.boxplot(loan_dataset['LoanAmount_new'],vert=False)
plt.show()
```

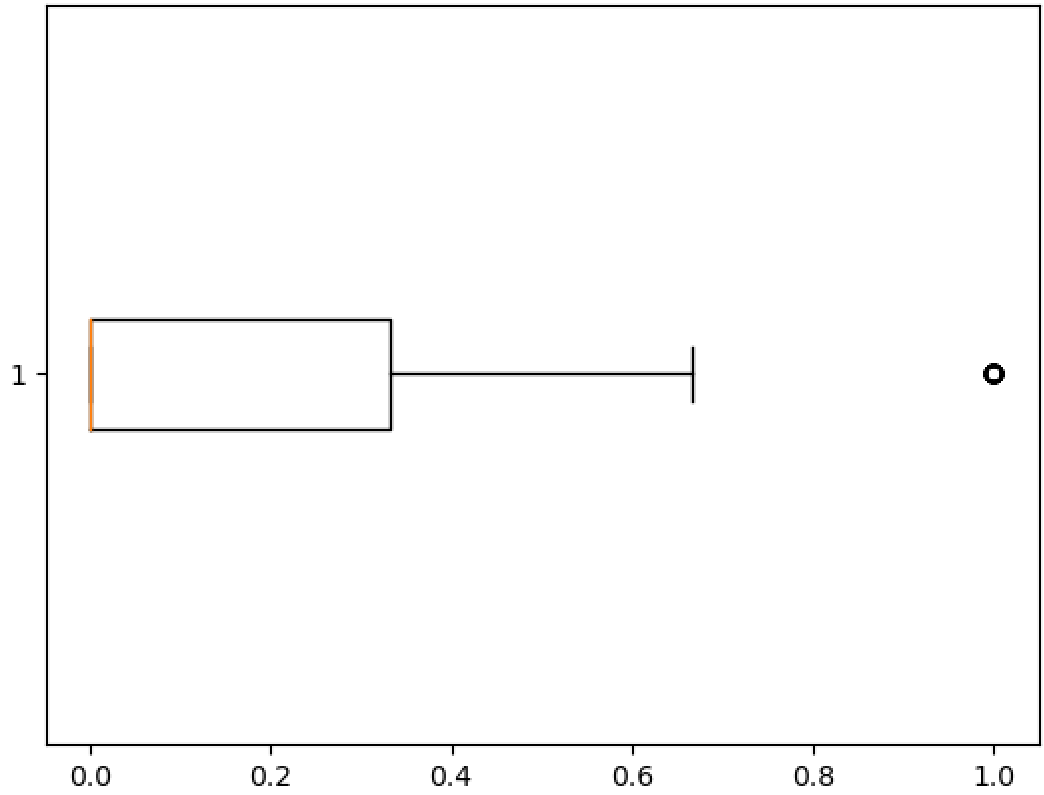


In [88]:
```
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler()
loan_dataset['CoapplicantIncome_new']=mms.fit_transform(loan_dataset[['CoapplicantI

plt.boxplot(loan_dataset['CoapplicantIncome_new'],vert=False)
plt.show()
```



In [90]:
```
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler()
loan_dataset['Dependents_new']=mms.fit_transform(loan_dataset[['Dependents']])
```

```
plt.boxplot(loan_dataset['Dependents_new'],vert=False)
plt.show()
```



In [92]:
```
loan_dataset['ApplicantIncome'].values.reshape(-1,1).ndim
```

Out[92]:
```
2
```

In [ ]:
```
loan_dataset['ApplicantIncome'].values.reshape(-1,1)
```

In [93]:
```
max_id=loan_dataset['ApplicantIncome'].idxmax()
min_id=loan_dataset['ApplicantIncome_new'].idxmin()
print(max_id,min_id)
```

```
557 216
```

In [94]:
```
loan_dataset['ApplicantIncome'].iloc[[max_id,min_id]]

# here 557 is the id
# here 216  is the id
```

Out[94]:
```
557    10139.0
216      150.0
Name: ApplicantIncome, dtype: float64
```

In [95]:
```
loan_dataset['ApplicantIncome_new'].iloc[[max_id,min_id]]
```

Out[95]:
```
557    1.0
216    0.0
Name: ApplicantIncome_new, dtype: float64
```