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

path=r"C:\Users\tanma\DATASCIENCE\EDA\vidya hackerthon data\loan_prediction.csv"
loan_prediction_df=pd.read_csv(path)
loan_prediction_df
```

Out[1]:

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmou |
|---|---|---|---|---|---|---|---|---|---|
| **0** | LP001002 | Male | No | 0 | Graduate | No | 5849 | 0.0 | Na |
| **1** | LP001003 | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 128 |
| **2** | LP001005 | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 66 |
| **3** | LP001006 | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 120 |
| **4** | LP001008 | Male | No | 0 | Graduate | No | 6000 | 0.0 | 141 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **609** | LP002978 | Female | No | 0 | Graduate | No | 2900 | 0.0 | 71 |
| **610** | LP002979 | Male | Yes | 3+ | Graduate | No | 4106 | 0.0 | 40 |
| **611** | LP002983 | Male | Yes | 1 | Graduate | No | 8072 | 240.0 | 253 |
| **612** | LP002984 | Male | Yes | 2 | Graduate | No | 7583 | 0.0 | 187 |
| **613** | LP002990 | Female | No | 0 | Graduate | Yes | 4583 | 0.0 | 133 |

614 rows × 13 columns

** Some common functions**

- len()
- shape
- size
- count()
- head()
- tail()

In [2]:
```
len(loan_prediction_df)
```

Out[2]: 614

In [3]:
```
loan_prediction_df.shape
```

Out[3]: (614, 13)

In [4]:
```
loan_prediction_df.count()
```

Out[4]:
```
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 [5]:
```
loan_prediction_df.size
```
Out[5]: 7982

In [6]:
```
loan_prediction_df.head()
```
Out[6]:

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmount |
|---|---|---|---|---|---|---|---|---|---|
| **0** | LP001002 | Male | No | 0 | Graduate | No | 5849 | 0.0 | NaN |
| **1** | LP001003 | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 128.0 |
| **2** | LP001005 | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 66.0 |
| **3** | LP001006 | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 120.0 |
| **4** | LP001008 | Male | No | 0 | Graduate | No | 6000 | 0.0 | 141.0 |

In [7]:
```
loan_prediction_df.tail(3)
```
Out[7]:

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmou |
|---|---|---|---|---|---|---|---|---|---|
| **611** | LP002983 | Male | Yes | 1 | Graduate | No | 8072 | 240.0 | 253 |
| **612** | LP002984 | Male | Yes | 2 | Graduate | No | 7583 | 0.0 | 187 |
| **613** | LP002990 | Female | No | 0 | Graduate | Yes | 4583 | 0.0 | 133 |

** Findings out details of a column**

- Finding out Column (df.columns)

-Data types of a column (.select_dtypes())

-How many are Categorical column (.select_dtypes(include='object'))

-How many are Numerical Column (.select_dtypes(exclude='object'))

-Findings out Null value of a column (df.isnull())

In [8]:
```
loan_prediction_df.columns
```
Out[8]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [9]:
```
loan_prediction_df.select_dtypes(include='object').columns
```
Out[9]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [10]:
```
loan_prediction_df.select_dtypes(exclude='object').columns
```
Out[10]:
```
Index(['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History'],
      dtype='object')
```

** Isnull means finding out null value (missing value) present in the data base or not **

-for this we have methods

-isnull()

-.isnull().sum()

here all are false means all has no null value in there

In [11]:
```
loan_prediction_df.isnull()
```

Out[11]:

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmoun |
|---|---|---|---|---|---|---|---|---|---|
| **0** | False | False | False | False | False | False | False | False | Tru |
| **1** | False | False | False | False | False | False | False | False | Fals |
| **2** | False | False | False | False | False | False | False | False | Fals |
| **3** | False | False | False | False | False | False | False | False | Fals |
| **4** | False | False | False | False | False | False | False | False | Fals |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | .. |
| **609** | False | False | False | False | False | False | False | False | Fals |
| **610** | False | False | False | False | False | False | False | False | Fals |
| **611** | False | False | False | False | False | False | False | False | Fals |
| **612** | False | False | False | False | False | False | False | False | Fals |
| **613** | False | False | False | False | False | False | False | False | Fals |

614 rows × 13 columns

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

Out[12]:
```
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
```

** To deal with null value we have 3 methods also**

- Fill the missing value with random number
- Method name : fillna -- (loan_prediction_df.fillna(20))
- Fill the missing values with random number on specific column

  df.['column_name'].fillna('update_value',inplace=True)
- bfill,ffill,pad,backfill

  df.fillna(method='backfill')
- bfill and backfill both are same
- pad and fill both are same
- Mean,Median,Mode

In [13]:
```
loan_prediction_df.isnull()
```

Out[13]:

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmoun |
|---|---|---|---|---|---|---|---|---|---|
| 0 | False | False | False | False | False | False | False | False | Tru |
| 1 | False | False | False | False | False | False | False | False | Fals |
| 2 | False | False | False | False | False | False | False | False | Fals |
| 3 | False | False | False | False | False | False | False | False | Fals |
| 4 | False | False | False | False | False | False | False | False | Fals |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | .. |
| 609 | False | False | False | False | False | False | False | False | Fals |
| 610 | False | False | False | False | False | False | False | False | Fals |
| 611 | False | False | False | False | False | False | False | False | Fals |
| 612 | False | False | False | False | False | False | False | False | Fals |
| 613 | False | False | False | False | False | False | False | False | Fals |

614 rows × 13 columns

In [14]:
```
loan_prediction_df.fillna(method='backfill')
```

Out[14]:

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAn |
|---|---|---|---|---|---|---|---|---|---|
| 0 | LP001002 | Male | No | 0 | Graduate | No | 5849 | 0.0 | |
| 1 | LP001003 | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | |
| 2 | LP001005 | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | |
| 3 | LP001006 | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | |
| 4 | LP001008 | Male | No | 0 | Graduate | No | 6000 | 0.0 | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 609 | LP002978 | Female | No | 0 | Graduate | No | 2900 | 0.0 | |
| 610 | LP002979 | Male | Yes | 3+ | Graduate | No | 4106 | 0.0 | |
| 611 | LP002983 | Male | Yes | 1 | Graduate | No | 8072 | 240.0 | |
| 612 | LP002984 | Male | Yes | 2 | Graduate | No | 7583 | 0.0 | |

In [15]:
```
loan_prediction_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            614 non-null    object 
 1   Gender             601 non-null    object 
 2   Married            611 non-null    object 
 3   Dependents         599 non-null    object 
 4   Education          614 non-null    object 
 5   Self_Employed      582 non-null    object 
 6   ApplicantIncome    614 non-null    int64  
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object 
 12  Loan_Status        614 non-null    object 
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

** Drop column**

In [16]:
```
loan_prediction_df.drop('Loan_ID',axis=1,inplace=True)
```

In [17]:
```
loan_prediction_df
```

Out[17]:

| | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_A |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Male | No | 0 | Graduate | No | 5849 | 0.0 | NaN | |
| **1** | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 128.0 | |
| **2** | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 66.0 | |
| **3** | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 120.0 | |
| **4** | Male | No | 0 | Graduate | No | 6000 | 0.0 | 141.0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **609** | Female | No | 0 | Graduate | No | 2900 | 0.0 | 71.0 | |
| **610** | Male | Yes | 3+ | Graduate | No | 4106 | 0.0 | 40.0 | |
| **611** | Male | Yes | 1 | Graduate | No | 8072 | 240.0 | 253.0 | |
| **612** | Male | Yes | 2 | Graduate | No | 7583 | 0.0 | 187.0 | |
| **613** | Female | No | 0 | Graduate | Yes | 4583 | 0.0 | 133.0 | |

614 rows × 12 columns

** Find out Duplicate values **

In [18]:
```
loan_prediction_df.drop_duplicates()
```

Out[18]:

| | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_A |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Male | No | 0 | Graduate | No | 5849 | 0.0 | NaN | |
| **1** | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 128.0 | |
| **2** | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 66.0 | |
| **3** | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 120.0 | |
| **4** | Male | No | 0 | Graduate | No | 6000 | 0.0 | 141.0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **609** | Female | No | 0 | Graduate | No | 2900 | 0.0 | 71.0 | |
| **610** | Male | Yes | 3+ | Graduate | No | 4106 | 0.0 | 40.0 | |
| **611** | Male | Yes | 1 | Graduate | No | 8072 | 240.0 | 253.0 | |
| **612** | Male | Yes | 2 | Graduate | No | 7583 | 0.0 | 187.0 | |
| **613** | Female | No | 0 | Graduate | Yes | 4583 | 0.0 | 133.0 | |

614 rows × 12 columns

** take-loc-iloc **

- loc and iloc helps to find out specic rows and columns of data

In [19]:
```
loan_prediction_df.take((101,203,311))
```

Out[19]:

| | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_A |
|---|---|---|---|---|---|---|---|---|---|
| **101** | Male | No | 0 | Graduate | No | 4843 | 3806.0 | 151.0 | |
| **203** | Male | Yes | 1 | Not Graduate | No | 3500 | 1083.0 | 135.0 | |
| **311** | Male | No | 0 | Not Graduate | No | 2927 | 2405.0 | 111.0 | |

In [ ]:

In [20]:
```
#read column
loan_prediction_df.take([5,6],axis=1)
```

Out[20]:

| | ApplicantIncome | CoapplicantIncome |
|---|---|---|
| **0** | 5849 | 0.0 |
| **1** | 4583 | 1508.0 |
| **2** | 3000 | 0.0 |
| **3** | 2583 | 2358.0 |
| **4** | 6000 | 0.0 |
| **...** | ... | ... |
| **609** | 2900 | 0.0 |
| **610** | 4106 | 0.0 |
| **611** | 8072 | 240.0 |
| **612** | 7583 | 0.0 |
| **613** | 4583 | 0.0 |

614 rows × 2 columns

In [21]:
```
# find out specific row with specific column

loan_prediction_df.take([101,201,301]).take([5,6],axis=1)
```

Out[21]:

| | ApplicantIncome | CoapplicantIncome |
|---|---|---|
| **101** | 4843 | 3806.0 |
| **201** | 4923 | 0.0 |
| **301** | 2875 | 1750.0 |

- categorical column*

In [22]:
```
loan_prediction_df['Property_Area']
```

Out[22]:
```
0          Urban
1          Rural
2          Urban
3          Urban
4          Urban
         ...
609        Rural
610        Rural
611        Urban
612        Urban
613    Semiurban
Name: Property_Area, Length: 614, dtype: object
```

In [23]:
```
loan_prediction_df['Dependents']
```

Out[23]:
```
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 [24]:
```
loan_prediction_df[['Education']]
```

Out[24]:

| | Education |
|---|---|
| **0** | Graduate |
| **1** | Graduate |
| **2** | Graduate |
| **3** | Not Graduate |
| **4** | Graduate |
| **...** | ... |
| **609** | Graduate |
| **610** | Graduate |
| **611** | Graduate |
| **612** | Graduate |
| **613** | Graduate |

614 rows × 1 columns

** unique() **

In [25]:
```
# unique mathod find out how many unique elements are present
loan_prediction_df['Education'].unique()
```

Out[25]:
```
array(['Graduate', 'Not Graduate'], dtype=object)
```

In [26]:
```
loan_prediction_df.select_dtypes(include='object').columns
```

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

In [27]:
```
loan_prediction_df.select_dtypes(exclude='object').columns
```

Out[27]:
```
Index(['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History'],
      dtype='object')
```

**nunique**

- It represents how many values are repeated.

In [28]:
```
loan_prediction_df['Self_Employed'].nunique()
```

Out[28]:
```
2
```

In [29]:
```
loan_prediction_df['Dependents'].nunique()
```
Out[29]: 4

In [30]:
```
loan_prediction_df['LoanAmount'].nunique()
```
Out[30]: 203

In [31]:
```
loan_prediction_df['Education'].nunique()
```
Out[31]: 2

In [32]:
```
loan_prediction_df['Gender'].nunique()
```
Out[32]: 2

In [33]:
```
loan_prediction_df[['Education']]
```
Out[33]:

| | Education |
|---|---|
| 0 | Graduate |
| 1 | Graduate |
| 2 | Graduate |
| 3 | Not Graduate |
| 4 | Graduate |
| ... | ... |
| 609 | Graduate |
| 610 | Graduate |
| 611 | Graduate |
| 612 | Graduate |
| 613 | Graduate |

614 rows × 1 columns

In [34]:
```
unique_labels= loan_prediction_df['Dependents'].unique()
for i in unique_labels:
    con=loan_prediction_df['Dependents']==i
    print(i," :",len(loan_prediction_df[con]))
```
```
0  : 345
1  : 102
2  : 101
3+  : 51
nan  : 0
```

In [35]:
```
unique_labels=loan_prediction_df['Dependents'].unique()
for i in unique_labels:
    con=loan_prediction_df['Dependents']==i
    print(i,":",len(loan_prediction_df[con]))
```
```
0 : 345
1 : 102
2 : 101
3+ : 51
nan : 0
```

In [36]:
```
#Q1)out of total observations How many Graduates & how many
#   are 2's dependent observations are there?
```

In [37]:
```
con=loan_prediction_df['Education']=='Graduate'
len(loan_prediction_df[con])
```

Out[37]: 480

In [38]:
```
con=loan_prediction_df['Dependents']=='2'
len(loan_prediction_df[con])
```

Out[38]: 101

In [ ]:

In [ ]:

**Frequency Table**

In [ ]:

In [39]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

unique_labels= loan_prediction_df['Dependents'].unique()
count=[]
for i in unique_labels:
    con=loan_prediction_df['Dependents']==i
    count.append(len(loan_prediction_df[con]))
Dependent_df=pd.DataFrame(zip(unique_labels,count),
                          columns=['Dependents','Count'])

Dependent_df
```

Out[39]:

| | Dependents | Count |
|---|---|---|
| **0** | 0 | 345 |
| **1** | 1 | 102 |
| **2** | 2 | 101 |
| **3** | 3+ | 51 |
| **4** | NaN | 0 |

In [40]:
```
unique_labels=loan_prediction_df['LoanAmount'].unique()
count=[]
for i in unique_labels:
    con=loan_prediction_df['LoanAmount']==i
    count.append(len(loan_prediction_df[con]))

LoanAmount_df=pd.DataFrame(zip(unique_labels,count),
                          columns=['LoanAmount','Count'])
LoanAmount_df
```

Out[40]:

| | LoanAmount | Count |
|---|---|---|
| **0** | NaN | 0 |
| **1** | 128.0 | 11 |
| **2** | 66.0 | 4 |
| **3** | 120.0 | 20 |
| **4** | 141.0 | 2 |
| **...** | ... | ... |
| **199** | 292.0 | 1 |
| **200** | 142.0 | 1 |
| **201** | 350.0 | 1 |
| **202** | 496.0 | 1 |
| **203** | 253.0 | 1 |

204 rows × 2 columns

In [41]:
```
unique_labels=loan_prediction_df['Gender'].unique()
count=[]
for i in unique_labels:
    con=loan_prediction_df['Gender']==i
    count.append(len(loan_prediction_df[con]))
Gender_df=pd.DataFrame(zip(unique_labels,count),
                           columns=['gender','count'])
Gender_df
```

Out[41]:

| | gender | count |
|---|---|---|
| **0** | Male | 489 |
| **1** | Female | 112 |
| **2** | NaN | 0 |

In [ ]:

**what is differnce between between unique() & value_counts()**

In [ ]:

In [42]:
```python
LoanAmount_vc=loan_prediction_df['LoanAmount'].value_counts()
LoanAmount_vc
```

Out[42]:
```
LoanAmount
120.0    20
110.0    17
100.0    15
160.0    12
187.0    12
         ..
240.0     1
214.0     1
59.0      1
166.0     1
253.0     1
Name: count, Length: 203, dtype: int64
```

In [43]:
```python
LoanAmount_vc.keys()
```

Out[43]:
```
Index([120.0, 110.0, 100.0, 160.0, 187.0, 128.0, 113.0, 130.0,  95.0,  96.0,
       ...
       304.0, 279.0, 280.0,  42.0,  72.0, 240.0, 214.0,  59.0, 166.0, 253.0],
      dtype='float64', name='LoanAmount', length=203)
```

In [44]:
```python
LoanAmount_vc.values
```

Out[44]:
```
array([20, 17, 15, 12, 12, 11, 11, 10,  9,  9,  8,  8,  8,  7,  7,  7,  7,
        7,  7,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  5,  5,  5,  5,  5,
        5,  5,  5,  5,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,
        4,  4,  4,  4,  4,  4,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,
        3,  3,  3,  3,  3,  3,  3,  3,  3,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  2,  2,  2,  2,  2,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1],
      dtype=int64)
```

**Frequency Table using value.counts() methods**

In [45]:
```python
LoanAmount_vc=loan_prediction_df['LoanAmount'].value_counts()
l1=LoanAmount_vc.keys()
l2=LoanAmount_vc.values
LoanAmount_vc_df=pd.DataFrame(zip(l1,l2),
                              columns=['loanamount','count'])
LoanAmount_vc_df
```

Out[45]:

| | loanamount | count |
|---|---|---|
| **0** | 120.0 | 20 |
| **1** | 110.0 | 17 |
| **2** | 100.0 | 15 |
| **3** | 160.0 | 12 |
| **4** | 187.0 | 12 |
| **...** | ... | ... |
| **198** | 240.0 | 1 |
| **199** | 214.0 | 1 |
| **200** | 59.0 | 1 |
| **201** | 166.0 | 1 |
| **202** | 253.0 | 1 |

203 rows × 2 columns

In [46]:
```python
Gender_vc=loan_prediction_df['Gender'].value_counts()
l1=Gender_vc.keys()
l2=Gender_vc.values
Gender_vc_df=pd.DataFrame(zip(l1,l2),
                          columns=['Gender','count'])
Gender_vc_df
```

Out[46]:

| | Gender | count |
|---|---|---|
| **0** | Male | 489 |
| **1** | Female | 112 |

In [47]:
```python
Dependents_vc=loan_prediction_df['Dependents'].value_counts()
l1=Dependents_vc.keys()
l2=Dependents_vc.values
Dependents_vc_df=pd.DataFrame(zip(l1,l2),
            columns=['Dependents','count'])
Dependents_vc_df
```

Out[47]:

| | Dependents | count |
|---|---|---|
| **0** | 0 | 345 |
| **1** | 1 | 102 |
| **2** | 2 | 101 |
| **3** | 3+ | 51 |

In [ ]:

#### Bar chart

- in order to draw bar chart
- we required one categorical colun
- we required one numerical column
- package: matplotlib
- dataframe: continent_vc_df

In [48]:
```python
LoanAmount_df
```

Out[48]:

| | LoanAmount | Count |
|---|---|---|
| **0** | NaN | 0 |
| **1** | 128.0 | 11 |
| **2** | 66.0 | 4 |
| **3** | 120.0 | 20 |
| **4** | 141.0 | 2 |
| **...** | ... | ... |
| **199** | 292.0 | 1 |
| **200** | 142.0 | 1 |
| **201** | 350.0 | 1 |
| **202** | 496.0 | 1 |
| **203** | 253.0 | 1 |

204 rows × 2 columns

In [49]:
```
Dependent_df
```

Out[49]:

| | Dependents | Count |
|---|---|---|
| **0** | 0 | 345 |
| **1** | 1 | 102 |
| **2** | 2 | 101 |
| **3** | 3+ | 51 |
| **4** | NaN | 0 |

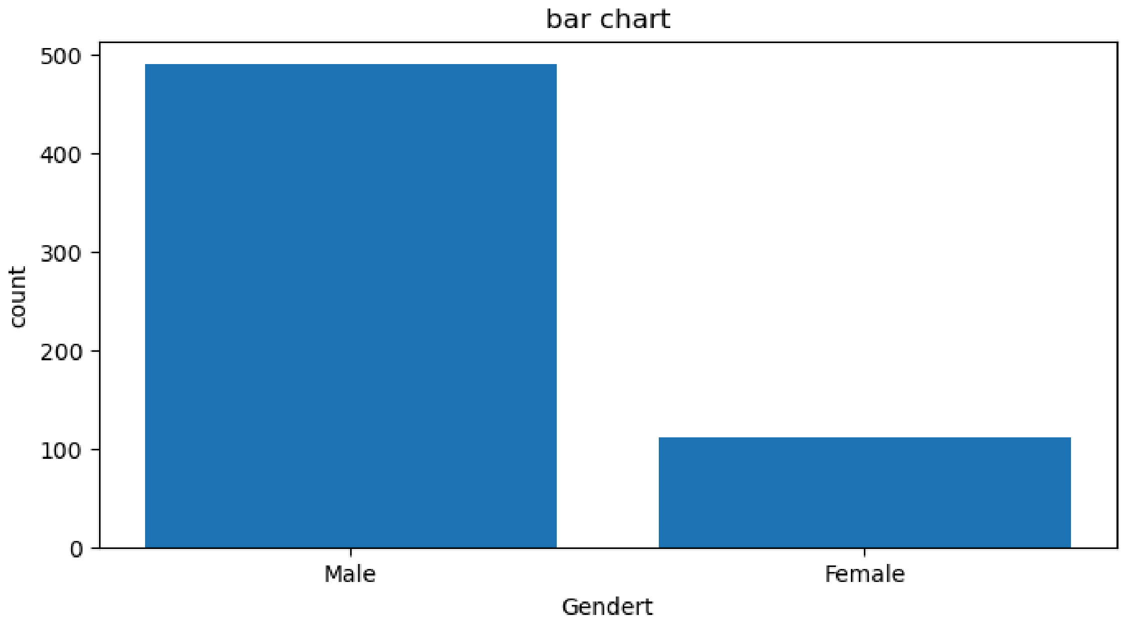
In [50]:
```
LoanAmount_vc_df
```

Out[50]:

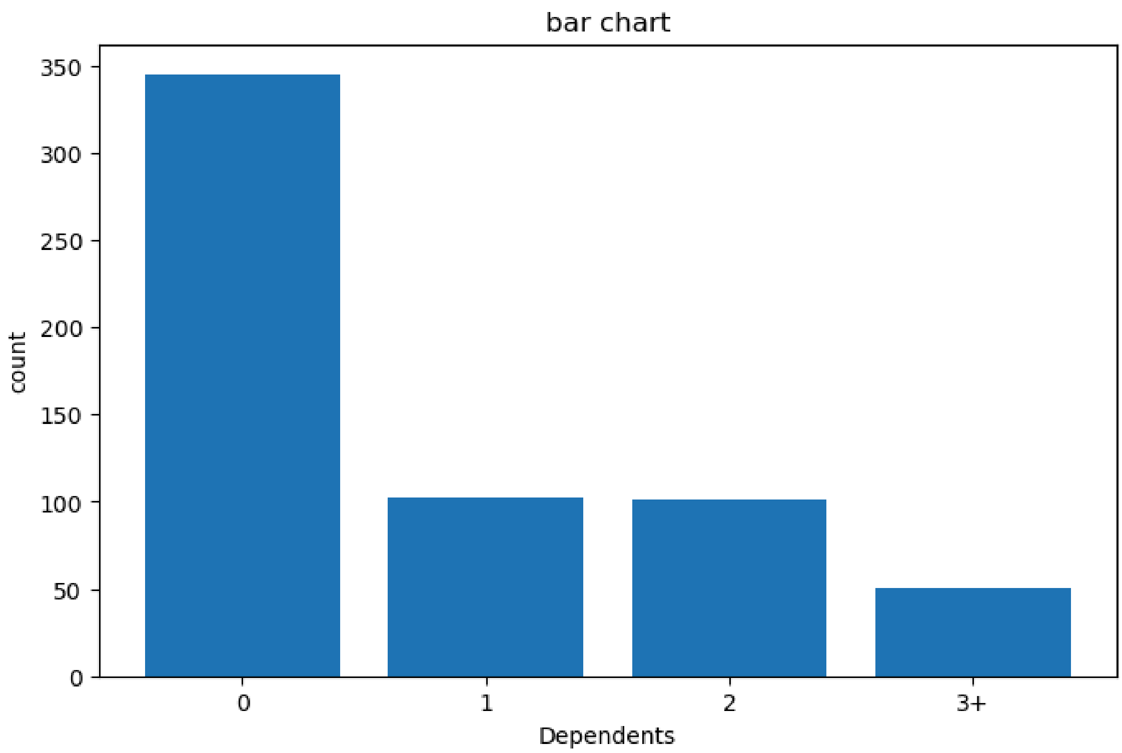
| | loanamount | count |
|---|---|---|
| **0** | 120.0 | 20 |
| **1** | 110.0 | 17 |
| **2** | 100.0 | 15 |
| **3** | 160.0 | 12 |
| **4** | 187.0 | 12 |
| **...** | ... | ... |
| **198** | 240.0 | 1 |
| **199** | 214.0 | 1 |
| **200** | 59.0 | 1 |
| **201** | 166.0 | 1 |
| **202** | 253.0 | 1 |

203 rows × 2 columns

In [51]:
```
plt.figure(figsize=(8,4))
plt.bar('Gender','count',
        data=Gender_vc_df)
plt.xlabel('Gendert')
plt.ylabel('count')
plt.title("bar chart")
plt.show()
```



In [52]:
```
plt.figure(figsize=(8,5))
plt.bar('Dependents','count',
       data=Dependents_vc_df)
plt.xlabel('Dependents')
plt.ylabel('count')
plt.title('bar chart')
plt.show()
```




- we read the data
- we read categorical column
- we made frequency table by using value counts
- we plot the bar chart using matplotlib
- But matplotlib required 3 arguments
- x label: categorical column (width)
- y label: numerical column (height)
- data ( frquency table name)

**Count plot**

- count plot can use bt seaborn package
- It requires only entire dataframe and categorical column
- entire dataframe name: Visadf
- categorical column name: contnent
- order: In which order you want plot

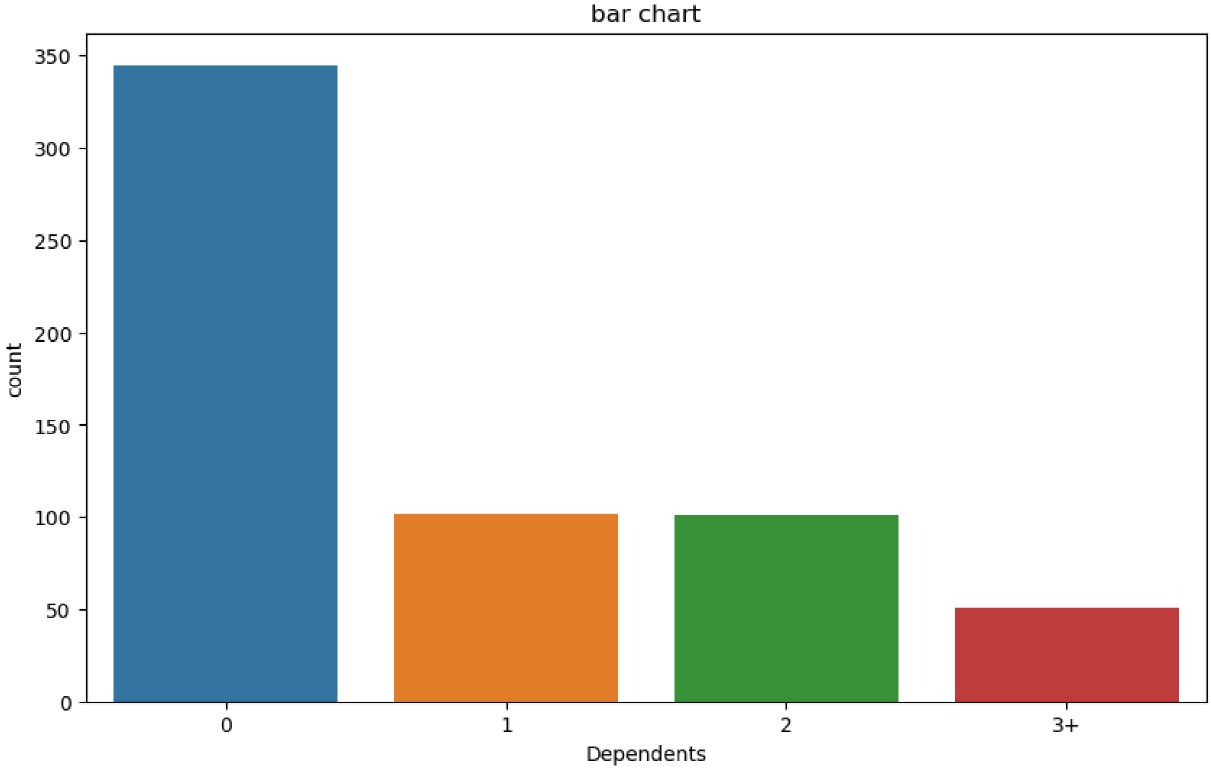
In [53]:
```
loan_prediction_df['Dependents'].value_counts().keys()
```

Out[53]:
```
Index(['0', '1', '2', '3+'], dtype='object', name='Dependents')
```

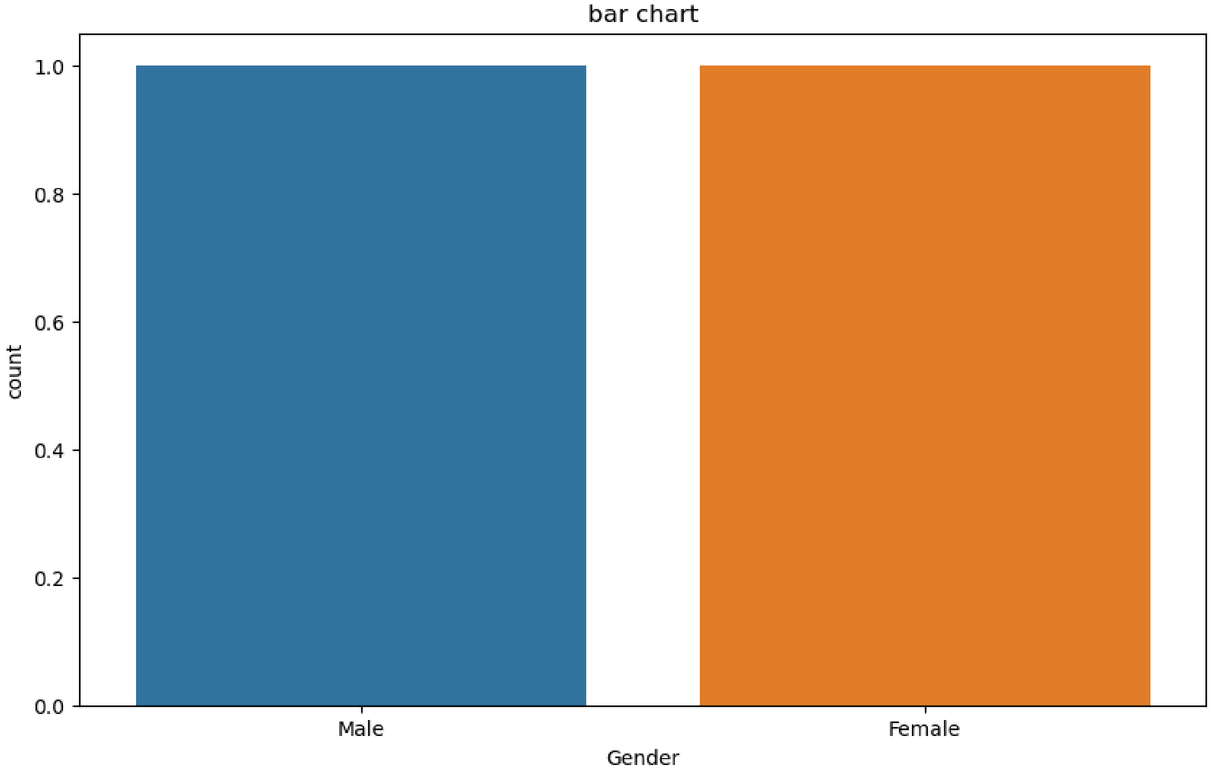
In [54]:
```
loan_prediction_df['Gender'].value_counts().keys()
```

Out[54]:
```
Index(['Male', 'Female'], dtype='object', name='Gender')
```

In [55]:
```
plt.figure(figsize=(10,6))
l=loan_prediction_df['Dependents'].value_counts().keys()
sns.countplot(data=loan_prediction_df,
             x='Dependents',
             order=l)
plt.xlabel('Dependents')
plt.ylabel('count')
plt.title('bar chart')
plt.show()
```

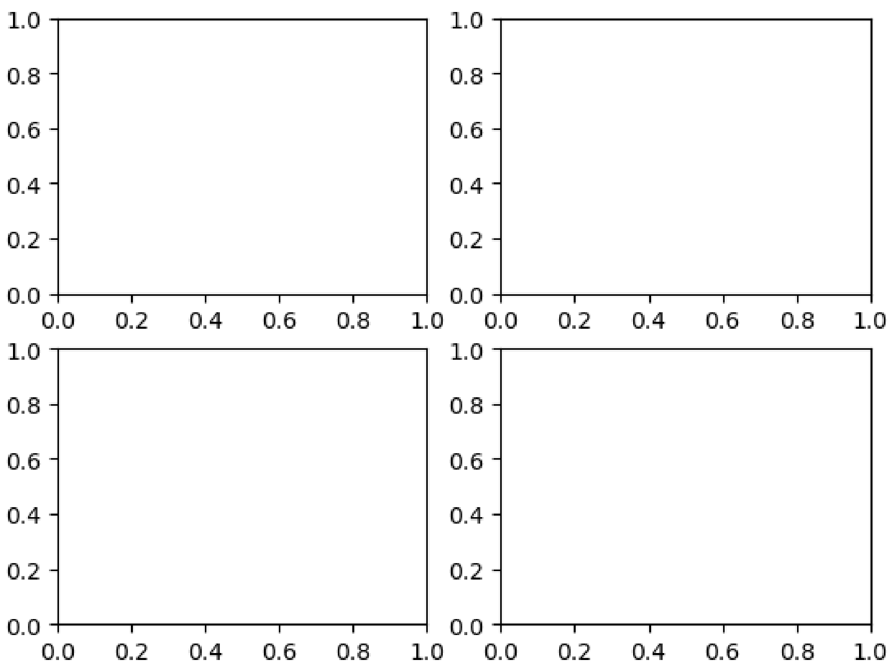


In [56]:
```
plt.figure(figsize=(10,6))
l=loan_prediction_df['Gender'].value_counts().keys()
sns.countplot(data=Gender_vc_df,
              x='Gender',order=l)
plt.xlabel('Gender')
plt.ylabel('count')
plt.title('bar chart')
plt.show()
```




In [57]:
```
plt.subplot(2,2,1)
plt.subplot(2,2,2)
plt.subplot(2,2,3)
plt.subplot(2,2,4)
```

Out[57]: <Axes: >



**Relative frequency**

In [58]:
```python
loan_prediction_df['Gender'].value_counts(normalize=True)
```

Out[58]:
```
Gender
Male      0.813644
Female    0.186356
Name: proportion, dtype: float64
```

In [59]:
```python
loan_prediction_df['Dependents'].value_counts(normalize=True)
```

Out[59]:
```
Dependents
0     0.575960
1     0.170284
2     0.168614
3+    0.085142
Name: proportion, dtype: float64
```

#### pie chat

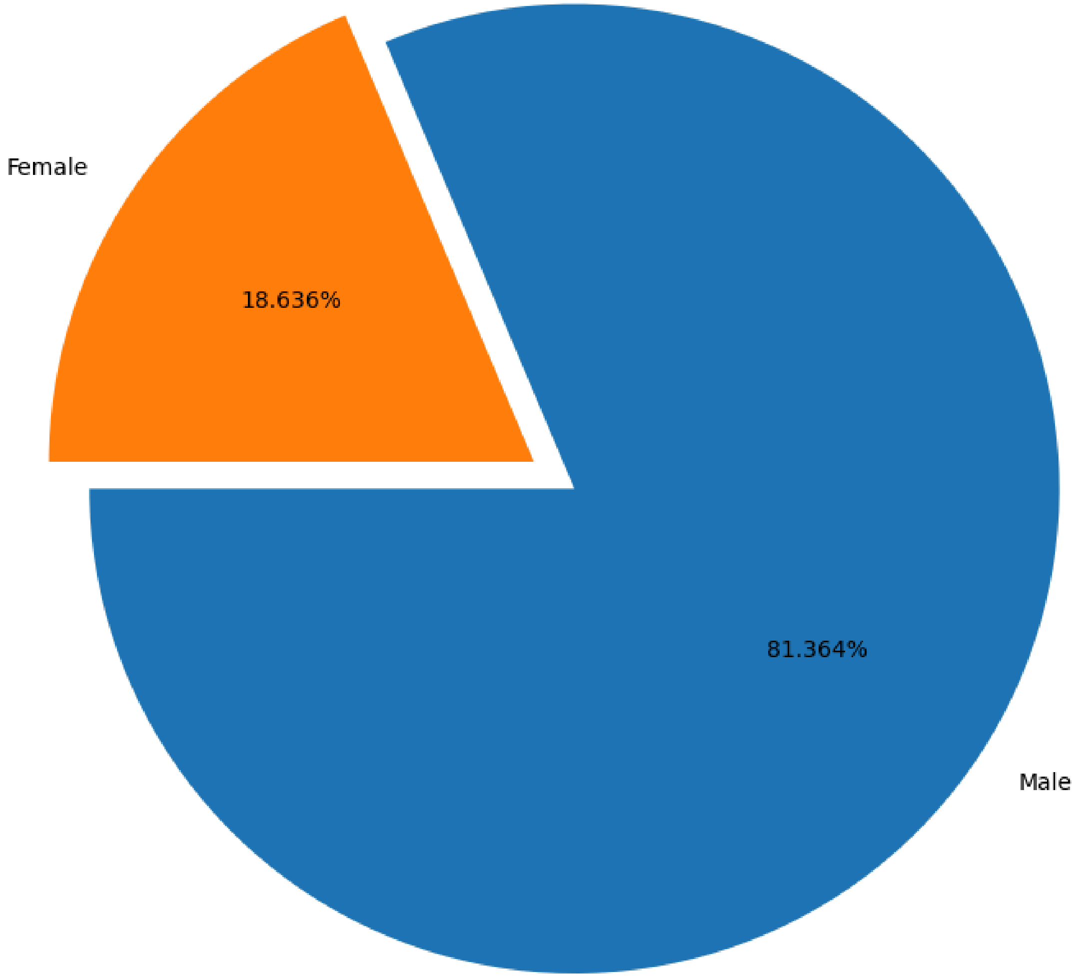
- count plot can use bt seaborn package
- It requires only entire dataframe and categorical column
- en tire dataframe name: Visadf
- categorical column name: contnent

In [60]:
```python
keys=loan_prediction_df['Gender'].value_counts().keys()
values=loan_prediction_df['Gender'].value_counts().values
values
```

Out[60]:
```
array([489, 112], dtype=int64)
```

In [61]:
```
plt.pie(values,
        labels=keys,
       autopct='%0.3f%%',
       explode=[0.1,0.1],
       startangle=180,
       radius=2)
```

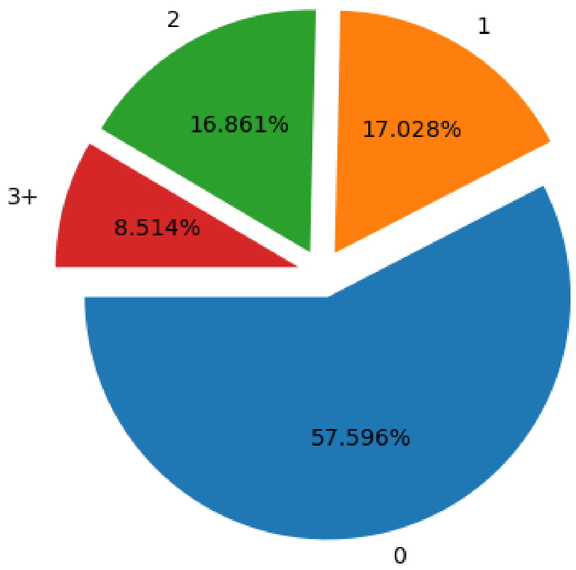
Out[61]:
```
([<matplotlib.patches.Wedge at 0x1fd5bb56a90>,
  <matplotlib.patches.Wedge at 0x1fd5bba9810>],
 [Text(1.9169599091112846, -1.2709306459677712, 'Male'),
  Text(-1.9169599091112854, 1.2709306459677696, 'Female')],
 [Text(1.0834990790629, -0.7183521042426533, '81.364%'),
  Text(-1.083499079062902, 0.718352104242652 4, '18.636%')])
```




In [62]:
```
keys=loan_prediction_df['Dependents'].value_counts().keys()
values=loan_prediction_df['Dependents'].value_counts().values
values
```

Out[62]:
```
array([345, 102, 101,  51], dtype=int64)
```

In [63]:
```
plt.pie(values,
       labels=keys,
       autopct="%0.3f%%",
       explode=[0.1,0.1,0.1,0.1],
        startangle=180,
        radius=1)
plt.show()
```




**Numerical analysis**

In [64]:
```
loan_prediction_df.select_dtypes(exclude='object').columns
```

Out[64]:
```
Index(['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History'],
      dtype='object')
```

In [65]:
```
loan_prediction_df['Loan_Amount_Term']
```

Out[65]:
```
0      360.0
1      360.0
2      360.0
3      360.0
4      360.0
       ...
609    360.0
610    180.0
611    360.0
612    360.0
613    360.0
Name: Loan_Amount_Term, Length: 614, dtype: float64
```

In [66]:
```
loan_prediction_df['ApplicantIncome']
```

Out[66]:
```
0      5849
1      4583
2      3000
3      2583
4      6000
       ...
609    2900
610    4106
611    8072
612    7583
613    4583
Name: ApplicantIncome, Length: 614, dtype: int64
```

In [67]:
```
loan_prediction_df['CoapplicantIncome']
```

Out[67]:
```
0         0.0
1      1508.0
2         0.0
3      2358.0
4         0.0
        ...
609       0.0
610       0.0
611     240.0
612       0.0
613       0.0
Name: CoapplicantIncome, Length: 614, dtype: float64
```

In [68]:
```
len(loan_prediction_df['CoapplicantIncome'])
```

Out[68]: 614

In [69]:
```
loan_prediction_df['CoapplicantIncome'].mean()
```

Out[69]: 1621.2457980271008

In [70]:
```
np.mean(loan_prediction_df['CoapplicantIncome'])
```

Out[70]: 1621.2457980271008

In [71]:
```
np.median(loan_prediction_df['CoapplicantIncome'])
```

Out[71]: 1188.5

In [72]:
```
loan_prediction_df['CoapplicantIncome'].max()
```

Out[72]: 41667.0

In [73]:
```
np.min(loan_prediction_df['CoapplicantIncome'])
```

Out[73]: 0.0

**standard deviation**

In [74]:
```
loan_prediction_df['CoapplicantIncome'].std()
```

Out[74]: 2926.2483692241917

In [75]:
```
wage_count=round(loan_prediction_df['CoapplicantIncome'].count(),2)
wage_max=round(loan_prediction_df['CoapplicantIncome'].max(),2)
wage_min=round(loan_prediction_df['CoapplicantIncome'].min(),2)
wage_mean=round(loan_prediction_df['CoapplicantIncome'].mean(),2)
wage_median=round(loan_prediction_df['CoapplicantIncome'].median(),2)
wage_std=round(loan_prediction_df['CoapplicantIncome'].std(),2)

l=[wage_count,wage_max,wage_min,wage_mean,wage_median,wage_std]
cols=['CoapplicantIncome']
index=['count','min','max','mean','median','std']
pd.DataFrame(l,columns=cols,index=index)
```

Out[75]:

| | CoapplicantIncome |
|---|---|
| **count** | 614.00 |
| **min** | 41667.00 |
| **max** | 0.00 |
| **mean** | 1621.25 |
| **median** | 1188.50 |
| **std** | 2926.25 |

**percentile-quantile**

- perecntile and quantile available in numpy
- np.percentile()
- column name
- percentile value between 0 to 100
- np.quantile()
- column name
- 0 to 1
- In quantile 0.25 means 25 in percentile

In [ ]:

```

```

In [76]:

```
np.percentile(loan_prediction_df['CoapplicantIncome'],25)
```

Out[76]:

```
0.0
```

In [77]:

```
np.quantile(loan_prediction_df['LoanAmount'],0.25)
```

Out[77]:

```
nan
```

In [78]:

```
wage_count=round(loan_prediction_df['CoapplicantIncome'].count(),2)
wage_max=round(loan_prediction_df['CoapplicantIncome'].max(),2)
wage_min=round(loan_prediction_df['CoapplicantIncome'].min(),2)
wage_mean=round(loan_prediction_df['CoapplicantIncome'].mean(),2)
wage_median=round(loan_prediction_df['CoapplicantIncome'].median(),2)
wage_std=round(loan_prediction_df['CoapplicantIncome'].std(),2)
wage_25=np.percentile(loan_prediction_df['CoapplicantIncome'],25)
wage_50=np.percentile(loan_prediction_df['CoapplicantIncome'],50)
wage_75=np.percentile(loan_prediction_df['CoapplicantIncome'],75)

l=[wage_count,wage_max,wage_min,
wage_mean,wage_median,wage_std,
wage_25,wage_50,wage_75]
cols=['CoapplicantIncome']
index=['count','max','min',
'mean','median','std',
'25%','50%','75%']
pd.DataFrame(l,columns=cols,index=index)
```

Out[78]:

| | CoapplicantIncome |
|---|---|
| **count** | 614.00 |
| **max** | 41667.00 |
| **min** | 0.00 |
| **mean** | 1621.25 |
| **median** | 1188.50 |
| **std** | 2926.25 |
| **25%** | 0.00 |
| **50%** | 1188.50 |
| **75%** | 2297.25 |

In [79]:
```
loan_prediction_df.describe()
```

Out[79]:

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

In [80]:
```
cols=loan_prediction_df.select_dtypes(exclude='object').columns
l=[]
for i in cols:
    wage_count=round(loan_prediction_df[i].count(),2)
    wage_max=round(loan_prediction_df[i].max(),2)
    wage_min=round(loan_prediction_df[i].min(),2)
    wage_mean=round(loan_prediction_df[i].mean(),2)
    wage_median=round(loan_prediction_df[i].median(),2)
    wage_std=round(loan_prediction_df[i].std(),2)
    wage_25=np.percentile(loan_prediction_df[i],25)
    wage_50=np.percentile(loan_prediction_df[i],50)
    wage_75=np.percentile(loan_prediction_df[i],75)

    l.append([wage_count,wage_max,wage_min,
    wage_mean,wage_median,wage_std,
    wage_25,wage_50,wage_75])


print(l)
index=['count','max','min',
        'mean','median','std',
        '25%','50%','75%']
pd.DataFrame(zip(l[0],l[1],l[2],l[3],l[4]),columns=cols,index=index)
```

```
[[614, 81000, 150, 5403.46, 3812.5, 6109.04, 2877.5, 3812.5, 5795.0], [614, 41667.0, 0.0, 1621.2
5, 1188.5, 2926.25, 0.0, 1188.5, 2297.25], [592, 700.0, 9.0, 146.41, 128.0, 85.59, nan, nan, na
n], [600, 480.0, 12.0, 342.0, 360.0, 65.12, nan, nan, nan], [564, 1.0, 0.0, 0.84, 1.0, 0.36, nan,
nan, nan]]
```

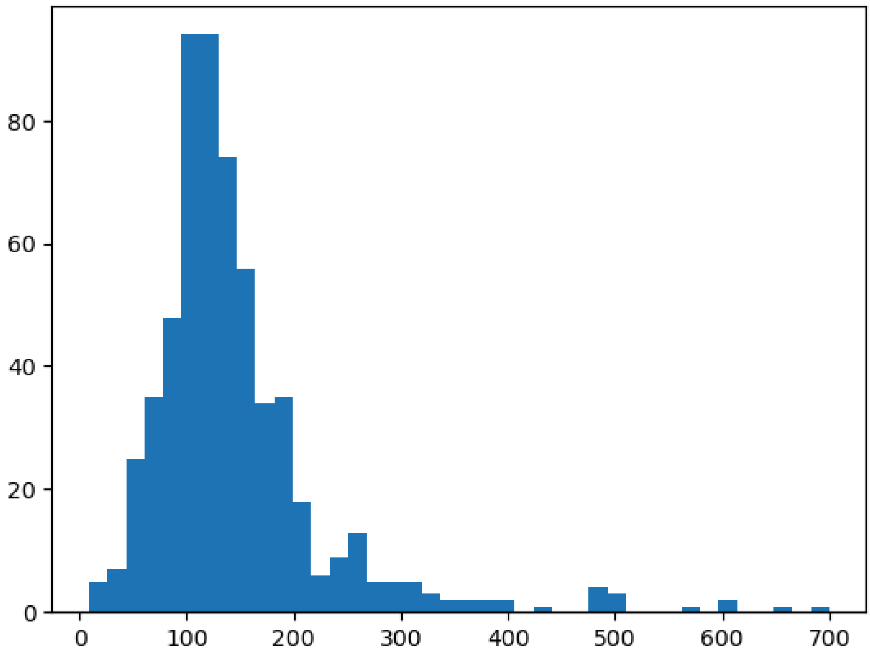
Out[80]:

| | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_Amount_Term | Credit_History |
|---|---|---|---|---|---|
| **count** | 614.00 | 614.00 | 592.00 | 600.00 | 564.00 |
| **max** | 81000.00 | 41667.00 | 700.00 | 480.00 | 1.00 |
| **min** | 150.00 | 0.00 | 9.00 | 12.00 | 0.00 |
| **mean** | 5403.46 | 1621.25 | 146.41 | 342.00 | 0.84 |
| **median** | 3812.50 | 1188.50 | 128.00 | 360.00 | 1.00 |
| **std** | 6109.04 | 2926.25 | 85.59 | 65.12 | 0.36 |
| **25%** | 2877.50 | 0.00 | NaN | NaN | NaN |
| **50%** | 3812.50 | 1188.50 | NaN | NaN | NaN |
| **75%** | 5795.00 | 2297.25 | NaN | NaN | NaN |

In [ ]:
```

```

**Histogram**

In [81]:
```
f,i,n=plt.hist(loan_prediction_df['LoanAmount'],bins=40)
```




In [82]:
```
cols=loan_prediction_df.select_dtypes(exclude='object').columns
#L=[]
for i in cols:
    plt.hist(loan_prediction_df[i])
    plt.show()
```

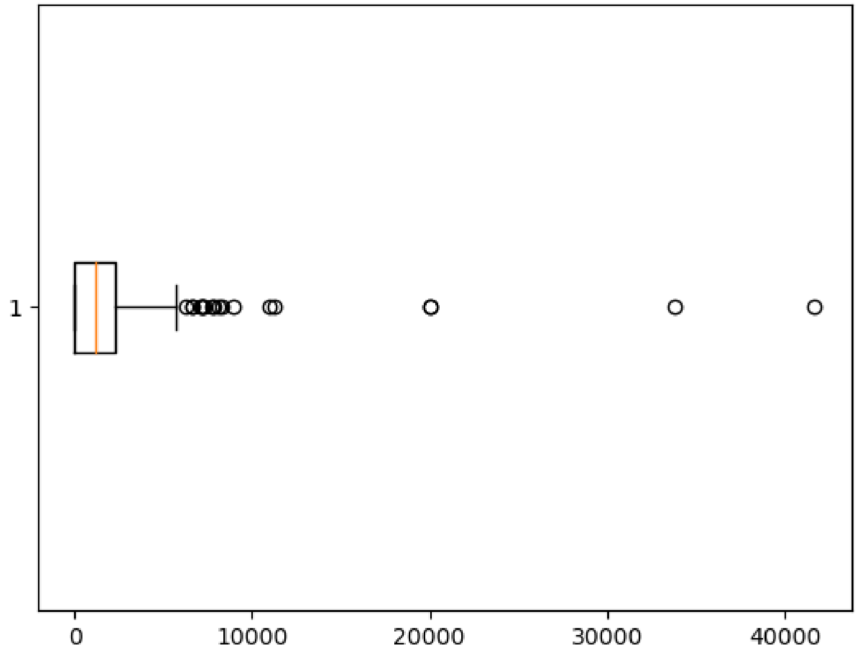

In [ ]:

**Boxplot**

- Boxplot is used to identify outliers
- In box plot we have
- Q1: 25p value
- Q2: 50p value
- Q3: 75p value
- IQR: Q3-Q1
- Mild outliers Q1-1.5IQR and Q3+1.5IQR
- huge outliers Q1-3IQR and Q3+3IQR

In [83]:
```python
plt.boxplot(x=loan_prediction_df['CoapplicantIncome'],
            vert=False)
```

Out[83]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x1fd5c1c43d0>,
  <matplotlib.lines.Line2D at 0x1fd5c1b8150>],
 'caps': [<matplotlib.lines.Line2D at 0x1fd5c1ba7d0>,
  <matplotlib.lines.Line2D at 0x1fd5c1b9ad0>],
 'boxes': [<matplotlib.lines.Line2D at 0x1fd5c1c6910>],
 'medians': [<matplotlib.lines.Line2D at 0x1fd5c1baa10>],
 'fliers': [<matplotlib.lines.Line2D at 0x1fd5c1baad0>],
 'means': []}
```




**Outlier Analysis**

- step1: Find the Q1,Q2,and Q3
  - np.percentile(column data,q)
- step2: Calculate lower boundary and upper boundary
  - IQR= Q3-Q1
- step3:Calculate lower boundary and upper boundary
  - lb:Q1-1.5IQR
  - ub:Q3+1.5IQR
- step4:Find the Outliersdf
  - c1:column data<lb
  - c2:column data>ub
  - c:apply the main condition
  - main data[c]

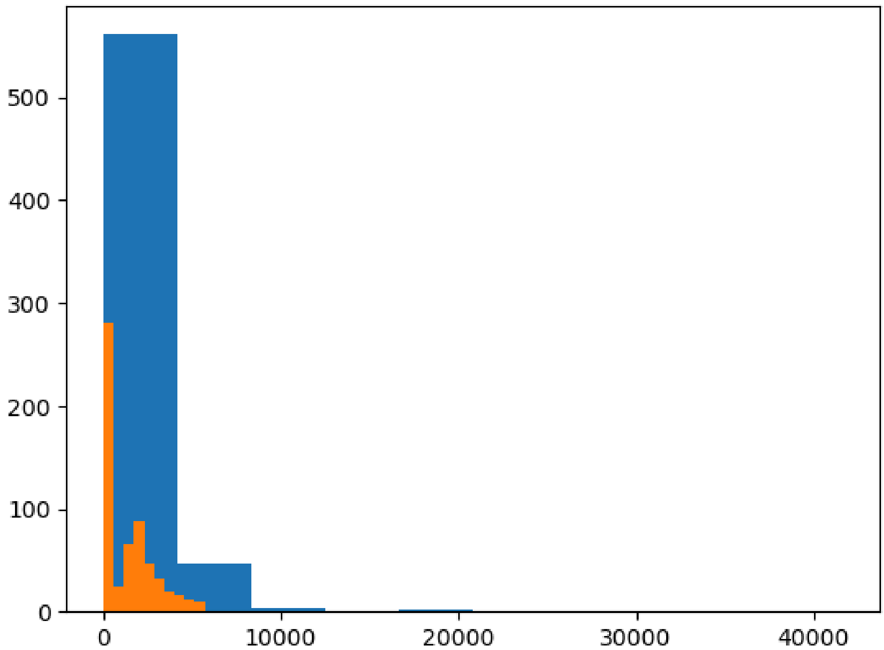
In [84]:
```
Q1=np.percentile(loan_prediction_df['CoapplicantIncome'],25)
Q2=np.percentile(loan_prediction_df['CoapplicantIncome'],50)
Q3=np.percentile(loan_prediction_df['CoapplicantIncome'],75)

IQR=Q3-Q1
lb=Q1-1.5*IQR
ub=Q3+1.5*IQR

c1=loan_prediction_df['CoapplicantIncome']<lb
c2=loan_prediction_df['CoapplicantIncome']>ub
con=c1|c2

outliers_df=loan_prediction_df[con]
outliers_df

c1=loan_prediction_df['CoapplicantIncome']>lb
c2=loan_prediction_df['CoapplicantIncome']<ub
con=c1&c2
non_outliers_df=loan_prediction_df[c1&c2]
non_outliers_df
```

Out[84]:

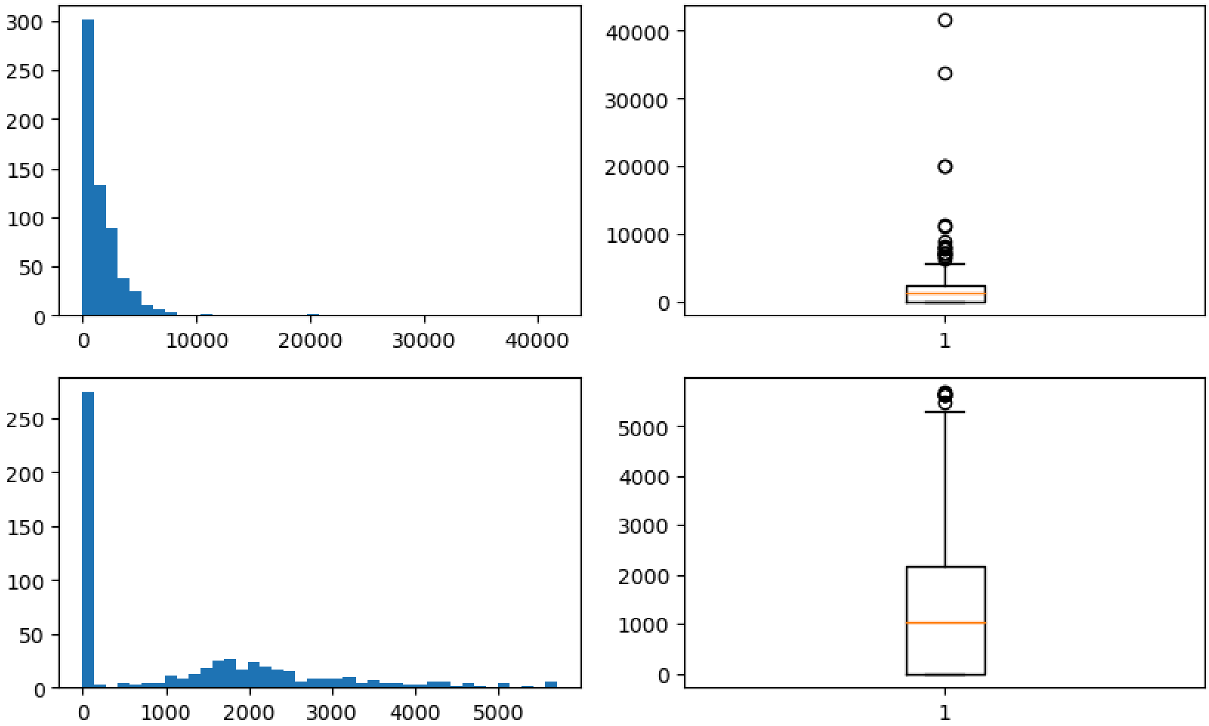
| | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_A |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Male | No | 0 | Graduate | No | 5849 | 0.0 | NaN | |
| **1** | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 128.0 | |
| **2** | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 66.0 | |
| **3** | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 120.0 | |
| **4** | Male | No | 0 | Graduate | No | 6000 | 0.0 | 141.0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **609** | Female | No | 0 | Graduate | No | 2900 | 0.0 | 71.0 | |
| **610** | Male | Yes | 3+ | Graduate | No | 4106 | 0.0 | 40.0 | |
| **611** | Male | Yes | 1 | Graduate | No | 8072 | 240.0 | 253.0 | |
| **612** | Male | Yes | 2 | Graduate | No | 7583 | 0.0 | 187.0 | |
| **613** | Female | No | 0 | Graduate | Yes | 4583 | 0.0 | 133.0 | |

596 rows × 12 columns

In [85]:
```
plt.hist(loan_prediction_df['CoapplicantIncome'])
plt.hist(non_outliers_df['CoapplicantIncome'])
plt.show()
```



In [86]:
```
plt.figure(figsize=(10,6))
plt.subplot(2,2,1)
plt.hist(loan_prediction_df['CoapplicantIncome'],bins=40)
plt.subplot(2,2,2)
plt.boxplot(loan_prediction_df['CoapplicantIncome'])
plt.subplot(2,2,3)
plt.hist(non_outliers_df['CoapplicantIncome'],bins=40)
plt.subplot(2,2,4)
plt.boxplot(non_outliers_df['CoapplicantIncome'])
plt.show()
```




In [ ]:

**Categorical vs Categorical**

In [87]:
```
loan_prediction_df.select_dtypes(include='object').columns
```

Out[87]:
```
Index(['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
       'Property_Area', 'Loan_Status'],
      dtype='object')
```

**comparing with Gender and loan_Status**

In [88]:
```
c1=loan_prediction_df['Gender']=='Male'
c2=loan_prediction_df['Loan_Status']=='Y'
c3=loan_prediction_df['Loan_Status']=='N'

cert_con=c1&c2
den_con=c1&c3

Y_count=len(loan_prediction_df[cert_con])
N_count=len(loan_prediction_df[den_con])

print(f"there are {Y_count} got certified for loan ")
print(f"there are {N_count} got denied  for loan")
```

```
there are 339 got certified for loan
there are 150 got denied  for loan
```

** Makeing Database from this loan_Status**

In [89]:
```
# step-1: make unique lables
labels=loan_prediction_df['Gender'].unique()
# step-2: create empty two lists
Y_count=[]
N_count=[]
# step-3: iterate through loop
for i in labels:
    c1=loan_prediction_df['Gender']==i
    c2=loan_prediction_df['Loan_Status']=='Y'
    c3=loan_prediction_df['Loan_Status']=='N'

    Y_con=c1&c2
    N_con=c1&c3

    Y_count.append(len(loan_prediction_df[Y_con]))
    N_count.append(len(loan_prediction_df[N_con]))

cols=['Gender','Y','N']
d1=pd.DataFrame(zip(labels,
                Y_count,
                 N_count), columns=cols)
d1
#d1.set_index('Gender')
```

Out[89]:

| | Gender | Y | N |
|---|---|---|---|
| **0** | Male | 339 | 150 |
| **1** | Female | 75 | 37 |
| **2** | NaN | 0 | 0 |

In [90]:
```
loan_prediction_df
```

Out[90]:

| | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_A |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Male | No | 0 | Graduate | No | 5849 | 0.0 | NaN | |
| **1** | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 128.0 | |
| **2** | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 66.0 | |
| **3** | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 120.0 | |
| **4** | Male | No | 0 | Graduate | No | 6000 | 0.0 | 141.0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **609** | Female | No | 0 | Graduate | No | 2900 | 0.0 | 71.0 | |
| **610** | Male | Yes | 3+ | Graduate | No | 4106 | 0.0 | 40.0 | |
| **611** | Male | Yes | 1 | Graduate | No | 8072 | 240.0 | 253.0 | |
| **612** | Male | Yes | 2 | Graduate | No | 7583 | 0.0 | 187.0 | |
| **613** | Female | No | 0 | Graduate | Yes | 4583 | 0.0 | 133.0 | |

614 rows × 12 columns

**pd.crosstab**

In [91]:
```
col1=[loan_prediction_df['Loan_Status']]
col2=loan_prediction_df['Gender']
result1=pd.crosstab(col2,col1)
result1
```

Out[91]:

| Loan_Status | N | Y |
|---|---|---|
| **Gender** | | |
| **Female** | 37 | 75 |
| **Male** | 150 | 339 |

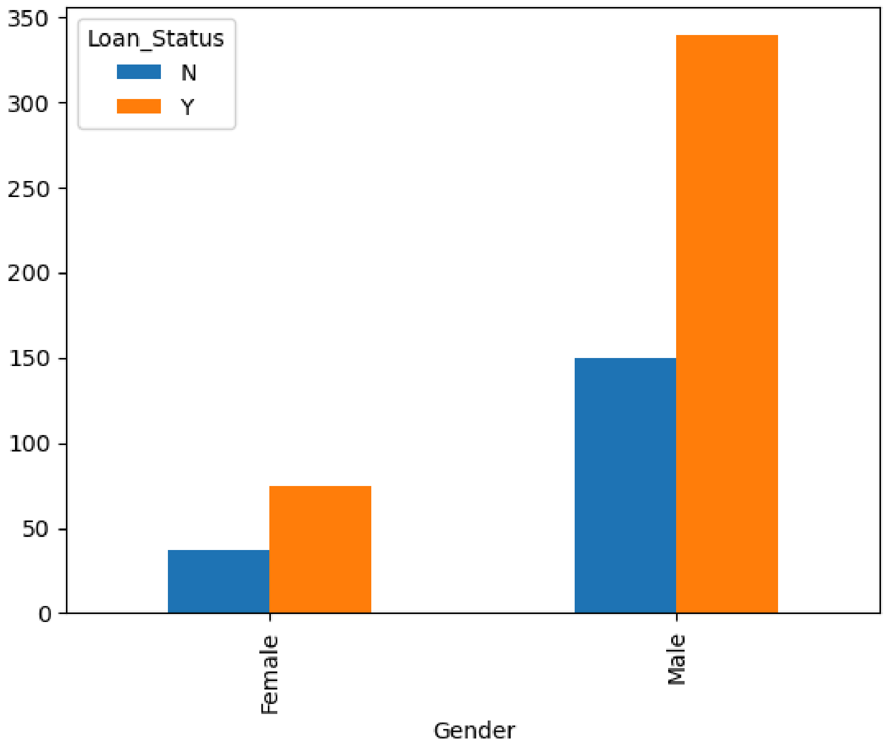
In [92]:
```
col1=[loan_prediction_df['Gender'],loan_prediction_df['Loan_Status']]
col2=loan_prediction_df['Education']
result2=pd.crosstab(col1,col2)
result2
```

Out[92]:

| | Education | Graduate | Not Graduate |
|---|---|---|---|
| **Gender** | **Loan_Status** | | |
| **Female** | **N** | 31 | 6 |
| | **Y** | 61 | 14 |
| **Male** | **N** | 105 | 45 |
| | **Y** | 271 | 68 |

In [93]:
```
result1.plot(kind='bar')
```

Out[93]: <Axes: xlabel='Gender'>



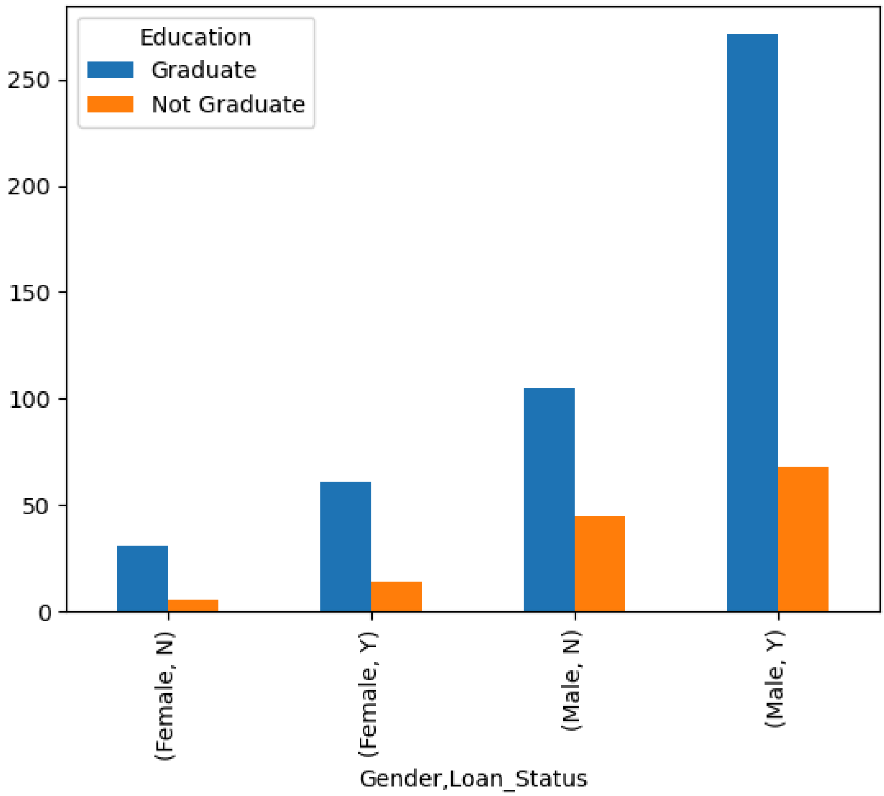

In [94]:
```
pd.DataFrame(result2)
```

Out[94]:

| | Education | Graduate | Not Graduate |
|---|---|---|---|
| **Gender** | **Loan_Status** | | |
| **Female** | **N** | 31 | 6 |
| | **Y** | 61 | 14 |
| **Male** | **N** | 105 | 45 |
| | **Y** | 271 | 68 |

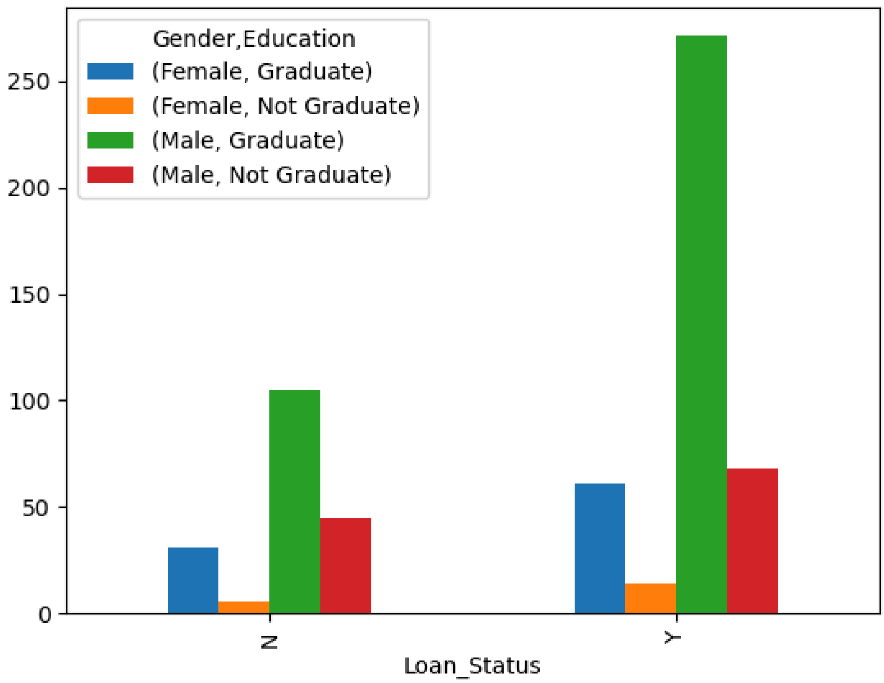
In [95]:
```
pd.DataFrame(result2).plot(kind='bar')
```

Out[95]: <Axes: xlabel='Gender,Loan_Status'>



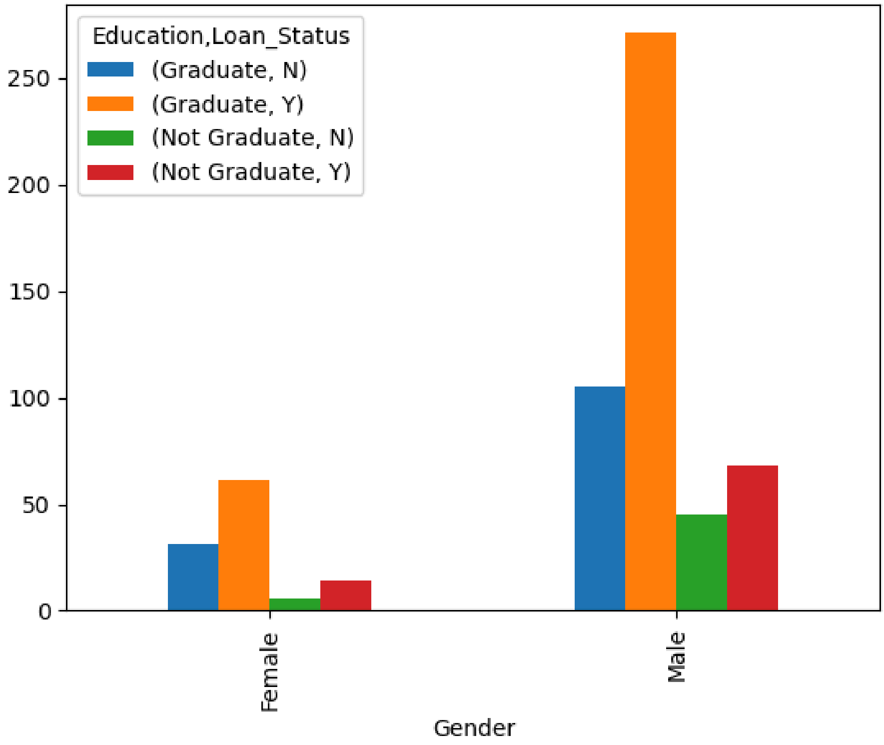

In [96]:
```
col1=[loan_prediction_df['Gender'],
      loan_prediction_df['Education']]
col2=loan_prediction_df['Loan_Status']
result2=pd.crosstab(col2,col1)
result2.plot(kind='bar')
```

Out[96]: <Axes: xlabel='Loan_Status'>



In [97]:
```
col1=loan_prediction_df['Gender']
col2=loan_prediction_df['Education']
col3=loan_prediction_df['Loan_Status']
r1=pd.crosstab(col1,[col2,col3])
r1.plot(kind='bar')
```

Out[97]: <Axes: xlabel='Gender'>

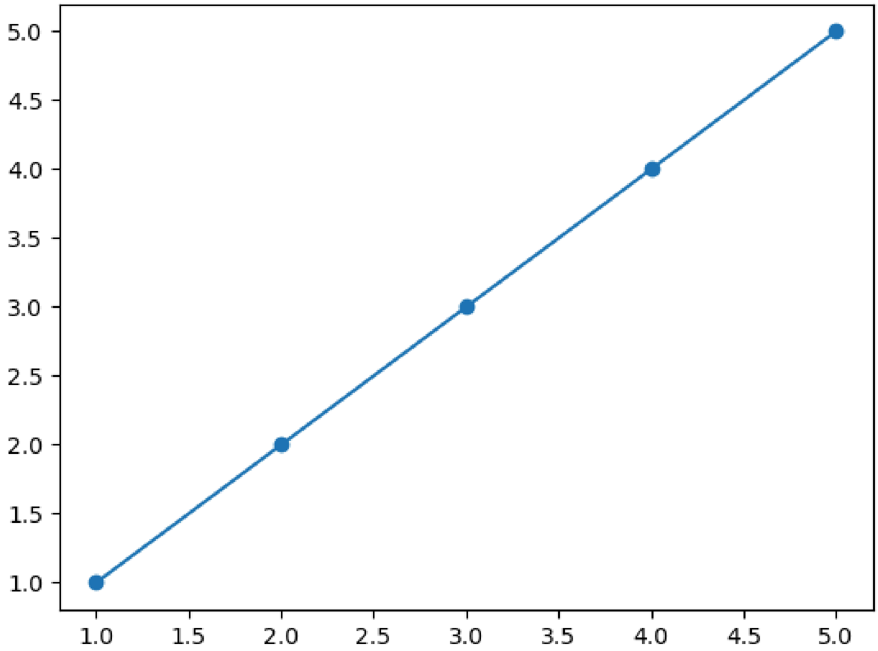



In [ ]:

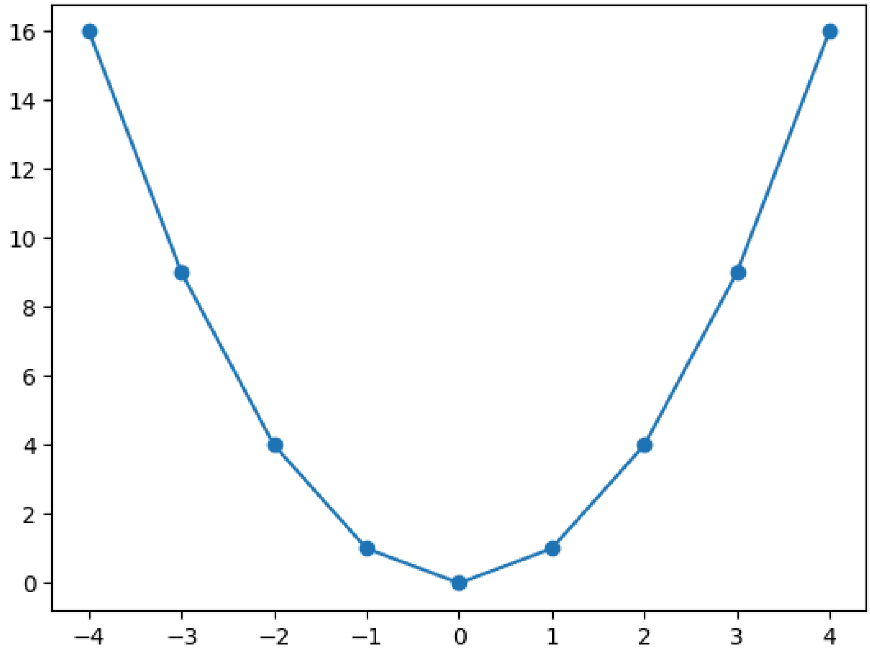
**Numerical vs Numerical**

**scatter Diagram**

In [98]:
```
x=[1,2,3,4,5]
y=[1,2,3,4,5]
plt.scatter(x,y)
plt.plot(x,y)
plt.show()
```




In [99]:
```
x=[i for i in range(-4,5)]
y=[i*i for i in x]
plt.scatter(x,y)
plt.plot(x,y)
plt.show()
```

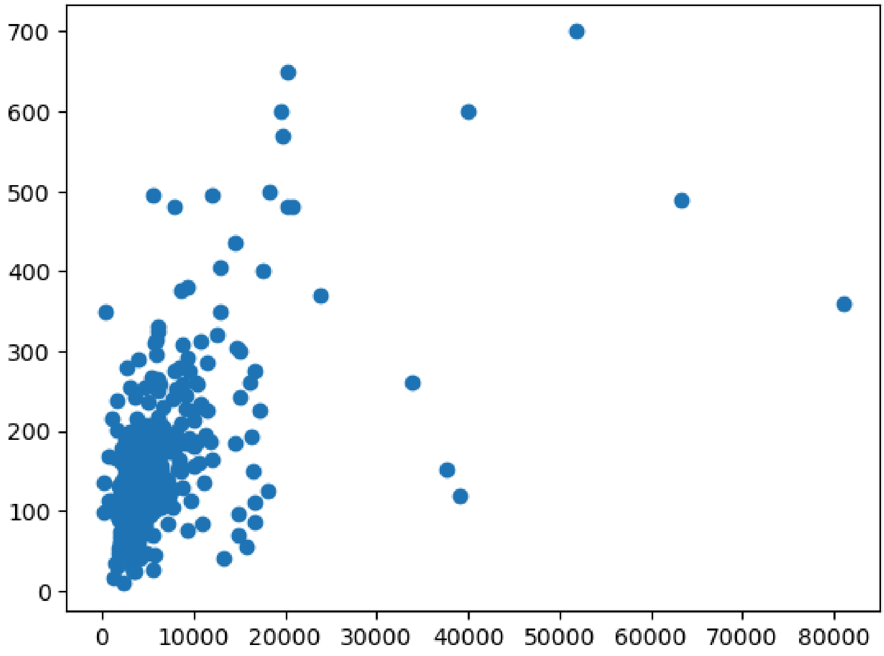



**Scatter plot always happens in numerical column**

In [100]:
```
cols=loan_prediction_df.select_dtypes(exclude='object')
cols.columns
```

Out[100]:
```
Index(['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History'],
      dtype='object')
```

In [101]:
```
col1=loan_prediction_df['ApplicantIncome']
col2=loan_prediction_df['LoanAmount']
plt.scatter(col1,col2)
plt.show()
```




** Pearson Coefficient Correlation**

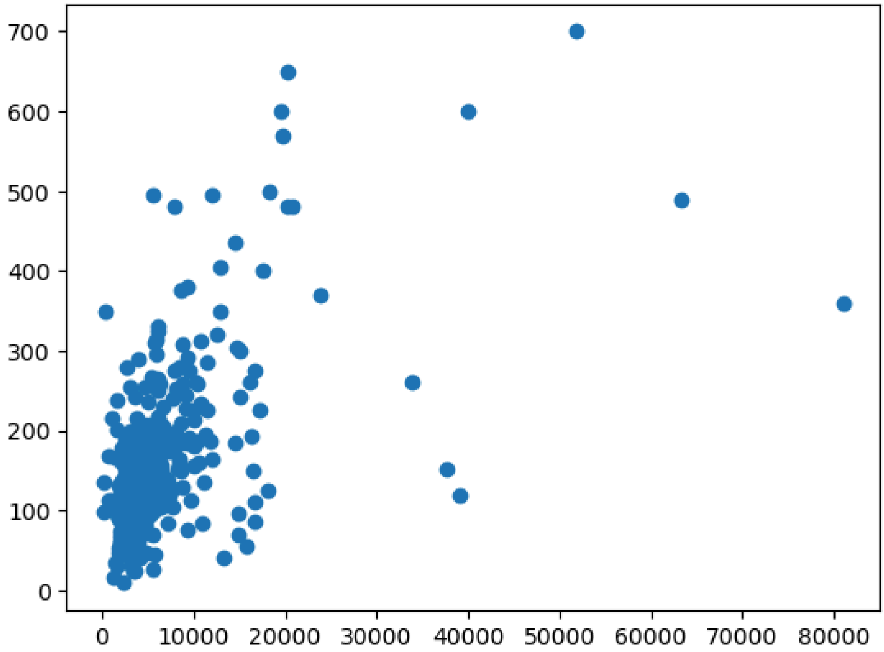
- r varies from -1 to 1
- -1 to 0 : Negative relation
- 0 to 1: Postive relation
- 0: No relation
- when you do this python
- It gives the matrix
- in Visa data we have 3 numerical columns are there
- python will give a matrix w.r.t 3 numerical columns
- The values in each field tells about the relation between
- the variables

In [102]:
```
loan_prediction_df.corr(numeric_only=True)
```

Out[102]:

| | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_Amount_Term | Credit_History |
|---|---|---|---|---|---|
| **ApplicantIncome** | 1.000000 | -0.116605 | 0.570909 | -0.045306 | -0.014715 |
| **CoapplicantIncome** | -0.116605 | 1.000000 | 0.188619 | -0.059878 | -0.002056 |
| **LoanAmount** | 0.570909 | 0.188619 | 1.000000 | 0.039447 | -0.008433 |
| **Loan_Amount_Term** | -0.045306 | -0.059878 | 0.039447 | 1.000000 | 0.001470 |
| **Credit_History** | -0.014715 | -0.002056 | -0.008433 | 0.001470 | 1.000000 |

In [103]:
```
# check the scatter plot between Applicantincome
# with LoanAmount
# we are seeing the relation is 0.570909

col1=loan_prediction_df['ApplicantIncome']
col2=loan_prediction_df['LoanAmount']
plt.scatter(col1,col2)
plt.show()
```




**Heat map**

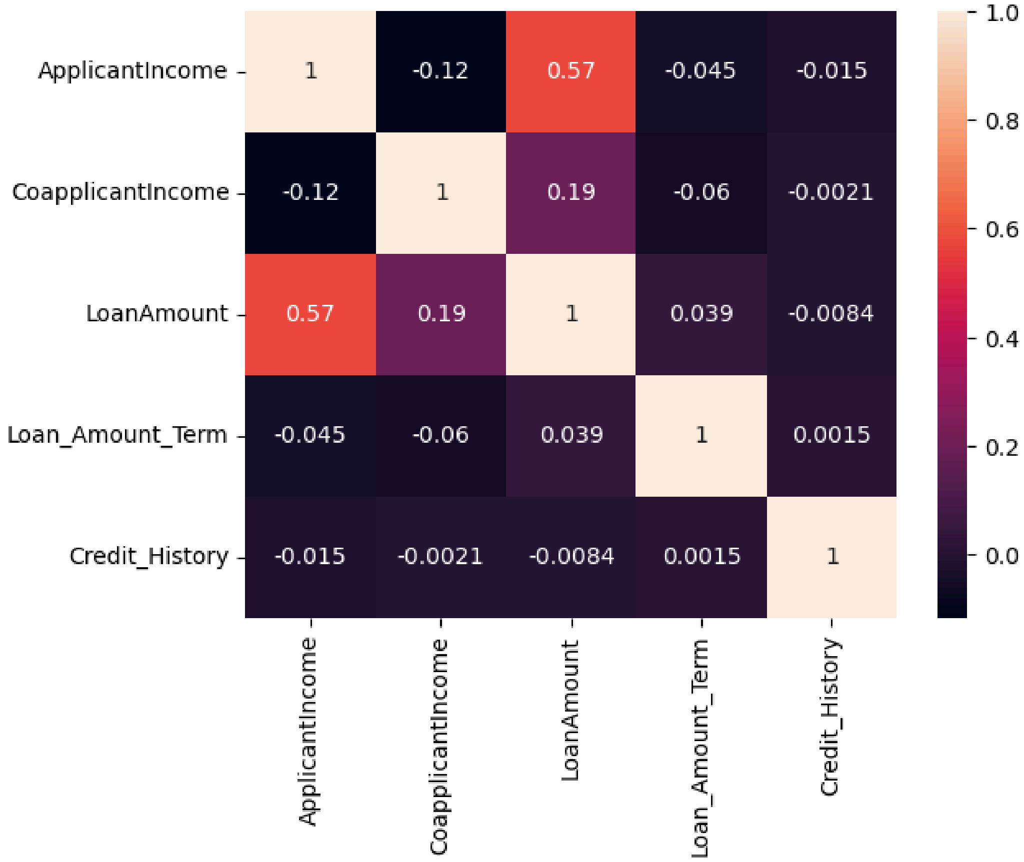
- heat map is useful to visulization of matrix
- it is under seaborn pacakges
- heat map will varies the values and gives the color about the - -
- values
- Spot issues and opportunities for improvement
- Provide a great first step for further user behavior
- research.
- Convey data in an easy-to-understand and interesting way
- Analyze user behavior on websites and mobile apps
- Provide useful insights into where users click, scroll,
- and move their cursors.
- Analyze a company's existing data and update it to reflect
- growth and other specific efforts.
- Visually appeal to team members and clients of the business
- or company.
- Give a visual representation of specifically collected and
- combined user data from your website visitors.
- View the popular and less popular parts of your website in
- color coding laid over your page.

In [104]:
```
corr_loan=loan_prediction_df.corr(numeric_only=True)
corr_loan
```

Out[104]:

| | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_Amount_Term | Credit_History |
|---|---|---|---|---|---|
| **ApplicantIncome** | 1.000000 | -0.116605 | 0.570909 | -0.045306 | -0.014715 |
| **CoapplicantIncome** | -0.116605 | 1.000000 | 0.188619 | -0.059878 | -0.002056 |
| **LoanAmount** | 0.570909 | 0.188619 | 1.000000 | 0.039447 | -0.008433 |
| **Loan_Amount_Term** | -0.045306 | -0.059878 | 0.039447 | 1.000000 | 0.001470 |
| **Credit_History** | -0.014715 | -0.002056 | -0.008433 | 0.001470 | 1.000000 |

In [105]:
```
sns.heatmap(corr_loan,annot=True)
plt.show()
```




**SESSION--7**

***CONVERT CATEGORICAL DATA TO NUMERICAL DATA***

*** MAP METHOD***

- In Machine learning it is very important to convert categorical data to numerical data
- Machine learning models develop by Maths
- Machine learning takes the input in the form of Numbers only
- To convert the we have some encoding techniques
- Label Encoder

- `map`

- `np.where`

- `using sklearn package: LabelEncoder`

- One hot encoder

- `using pandas package: pd.get_dummies`

- Before applying map method first get the unique labels of the
- column.
- For example Loan_Status is a categorical column
- It has two unique labels are there
- Y
- N
- Create a dictionary key as label, value as number
- d={'Y':0,'N:1}
- This dictionary we need to map the Loan_Status column

In [106]:
```
#checking categorical columns are
loan_prediction_df.select_dtypes(include='object').columns
```

Out[106]:
```
Index(['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
       'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [107]:
```
loan_prediction_df['Loan_Status'].unique()
```

Out[107]:
```
array(['Y', 'N'], dtype=object)
```

In [108]:
```
d={'Y':0,'N':1}
loan_prediction_df['Loan_Status']=loan_prediction_df['Loan_Status'].map(d)
loan_prediction_df
```

Out[108]:

| | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_A |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Male | No | 0 | Graduate | No | 5849 | 0.0 | NaN | |
| **1** | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 128.0 | |
| **2** | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 66.0 | |
| **3** | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 120.0 | |
| **4** | Male | No | 0 | Graduate | No | 6000 | 0.0 | 141.0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **609** | Female | No | 0 | Graduate | No | 2900 | 0.0 | 71.0 | |
| **610** | Male | Yes | 3+ | Graduate | No | 4106 | 0.0 | 40.0 | |
| **611** | Male | Yes | 1 | Graduate | No | 8072 | 240.0 | 253.0 | |
| **612** | Male | Yes | 2 | Graduate | No | 7583 | 0.0 | 187.0 | |
| **613** | Female | No | 0 | Graduate | Yes | 4583 | 0.0 | 133.0 | |

614 rows × 12 columns

In [109]:
```
d={}
labels=loan_prediction_df['Property_Area'].unique()
for i in range(len(labels)):
    d[labels[i]]=i
    loan_prediction_df['Property_Area']=loan_prediction_df['Property_Area'].map(d)

loan_prediction_df
```

Out[109]:

| | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_A |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Male | No | 0 | Graduate | No | 5849 | 0.0 | NaN | |
| **1** | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 128.0 | |
| **2** | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 66.0 | |
| **3** | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 120.0 | |
| **4** | Male | No | 0 | Graduate | No | 6000 | 0.0 | 141.0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **609** | Female | No | 0 | Graduate | No | 2900 | 0.0 | 71.0 | |
| **610** | Male | Yes | 3+ | Graduate | No | 4106 | 0.0 | 40.0 | |
| **611** | Male | Yes | 1 | Graduate | No | 8072 | 240.0 | 253.0 | |
| **612** | Male | Yes | 2 | Graduate | No | 7583 | 0.0 | 187.0 | |
| **613** | Female | No | 0 | Graduate | Yes | 4583 | 0.0 | 133.0 | |

614 rows × 12 columns

In [110]:
```
cat_cols=loan_prediction_df.select_dtypes(include='object').columns

d={}
for j in cat_cols[1:]:
    labels=loan_prediction_df[j].unique()
    for i in range(len(labels)):
        d[labels[i]]=i
        loan_prediction_df[j]=loan_prediction_df[j].map(d)
loan_prediction_df
```

Out[110]:

| | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_A |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Male | 2 | 4.0 | NaN | 2 | 5849 | 0.0 | NaN | |
| **1** | Male | 2 | NaN | NaN | 2 | 4583 | 1508.0 | 128.0 | |
| **2** | Male | 2 | 4.0 | NaN | 2 | 3000 | 0.0 | 66.0 | |
| **3** | Male | 2 | 4.0 | 4.0 | 2 | 2583 | 2358.0 | 120.0 | |
| **4** | Male | 2 | 4.0 | NaN | 2 | 6000 | 0.0 | 141.0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **609** | Female | 2 | 4.0 | NaN | 2 | 2900 | 0.0 | 71.0 | |
| **610** | Male | 2 | NaN | NaN | 2 | 4106 | 0.0 | 40.0 | |
| **611** | Male | 2 | NaN | NaN | 2 | 8072 | 240.0 | 253.0 | |
| **612** | Male | 2 | NaN | NaN | 2 | 7583 | 0.0 | 187.0 | |
| **613** | Female | 2 | 4.0 | NaN | 2 | 4583 | 0.0 | 133.0 | |

614 rows × 12 columns

**Label Encoder**

## Label Encoding is a technique used to convert categorical columns into numerical ones so that they can be fitted by machine learning models which only take numerical data123. It is an important pre-processing step in a machine-learning project1. By

## using the LabelEncoder class from scikit-learn, you can easily encode your categorical data and prepare it for further analysis or input into machine learning algorithms

- LabelEncoder is pacakge avialabel in sklearn
- Sickit learn heart of ML
- Read the package
- Save the package

In [111]:
```
from sklearn.preprocessing import LabelEncoder # read the pacakge
le=LabelEncoder()
loan_prediction_df['Married']=le.fit_transform(loan_prediction_df['Married'])
loan_prediction_df
```

Out[111]:

| | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_A |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Male | 0 | 4.0 | NaN | 2 | 5849 | 0.0 | NaN | |
| **1** | Male | 0 | NaN | NaN | 2 | 4583 | 1508.0 | 128.0 | |
| **2** | Male | 0 | 4.0 | NaN | 2 | 3000 | 0.0 | 66.0 | |
| **3** | Male | 0 | 4.0 | 4.0 | 2 | 2583 | 2358.0 | 120.0 | |
| **4** | Male | 0 | 4.0 | NaN | 2 | 6000 | 0.0 | 141.0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **609** | Female | 0 | 4.0 | NaN | 2 | 2900 | 0.0 | 71.0 | |
| **610** | Male | 0 | NaN | NaN | 2 | 4106 | 0.0 | 40.0 | |
| **611** | Male | 0 | NaN | NaN | 2 | 8072 | 240.0 | 253.0 | |
| **612** | Male | 0 | NaN | NaN | 2 | 7583 | 0.0 | 187.0 | |
| **613** | Female | 0 | 4.0 | NaN | 2 | 4583 | 0.0 | 133.0 | |

614 rows × 12 columns

In [112]:
```
cat_cols=loan_prediction_df.select_dtypes(include='object').columns
cat_cols
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
for i in cat_cols:
    loan_prediction_df[i]=le.fit_transform(loan_prediction_df[i])
loan_prediction_df
```

Out[112]:

| | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_A |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 4.0 | NaN | 2 | 5849 | 0.0 | NaN | |
| **1** | 1 | 0 | NaN | NaN | 2 | 4583 | 1508.0 | 128.0 | |
| **2** | 1 | 0 | 4.0 | NaN | 2 | 3000 | 0.0 | 66.0 | |
| **3** | 1 | 0 | 4.0 | 4.0 | 2 | 2583 | 2358.0 | 120.0 | |
| **4** | 1 | 0 | 4.0 | NaN | 2 | 6000 | 0.0 | 141.0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **609** | 0 | 0 | 4.0 | NaN | 2 | 2900 | 0.0 | 71.0 | |
| **610** | 1 | 0 | NaN | NaN | 2 | 4106 | 0.0 | 40.0 | |
| **611** | 1 | 0 | NaN | NaN | 2 | 8072 | 240.0 | 253.0 | |
| **612** | 1 | 0 | NaN | NaN | 2 | 7583 | 0.0 | 187.0 | |
| **613** | 0 | 0 | 4.0 | NaN | 2 | 4583 | 0.0 | 133.0 | |

614 rows × 12 columns

In [113]:
```
path=r"C:\Users\tanma\DATASCIENCE\EDA\vidya hackerthon data\loan_prediction.csv"
loan_prediction_df=pd.read_csv(path)


from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
loan_prediction_df['Married']=le.fit_transform(loan_prediction_df['Married'])
le.inverse_transform(loan_prediction_df['Married'])
```

Out[113]:
```
array(['No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes',
       'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'No', 'Yes', 'Yes',
       'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No',
       'Yes', 'No', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No',
       'No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No',
       'Yes', 'No', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes',
       'Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No',
       'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'Yes',
       'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes',
       'Yes', 'Yes', 'Yes', 'No', 'No', 'No', 'Yes', 'Yes', 'Yes', 'Yes',
       'Yes', 'No', 'Yes', 'Yes', nan, 'Yes', 'Yes', 'No', 'Yes', 'Yes',
       'No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No',
       'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes',
       'No', 'No', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes',
       'Yes', 'No', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes',
       'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes',
       'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No',
       'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes',
       'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes',
```

In [ ]:

**np.where**

- np.where required 3 arguments
- condition
- True
- False
- It is applicable only for binary labels
- case status has only two labels Certified and Denied
- if case_status==Certified replace that as 0, otherwise 1

In [114]:
```
loan_prediction_df.select_dtypes(include='object').columns
```

Out[114]:
```
Index(['Loan_ID', 'Gender', 'Dependents', 'Education', 'Self_Employed',
       'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [115]:
```
loan_prediction_df['Education'].unique()
```

Out[115]:
```
array(['Graduate', 'Not Graduate'], dtype=object)
```

In [116]:

```python
path=r"C:\Users\tanma\DATASCIENCE\EDA\vidya hackerthon data\loan_prediction.csv"
loan_prediction_df=pd.read_csv(path)


con=loan_prediction_df['Education']=='Graduate'
loan_prediction_df['Education']=np.where(con,0,1)
loan_prediction_df
```

Out[116]:

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmou |
|---|---|---|---|---|---|---|---|---|---|
| **0** | LP001002 | Male | No | 0 | 0 | No | 5849 | 0.0 | Na |
| **1** | LP001003 | Male | Yes | 1 | 0 | No | 4583 | 1508.0 | 128 |
| **2** | LP001005 | Male | Yes | 0 | 0 | Yes | 3000 | 0.0 | 66 |
| **3** | LP001006 | Male | Yes | 0 | 1 | No | 2583 | 2358.0 | 120 |
| **4** | LP001008 | Male | No | 0 | 0 | No | 6000 | 0.0 | 141 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **609** | LP002978 | Female | No | 0 | 0 | No | 2900 | 0.0 | 71 |
| **610** | LP002979 | Male | Yes | 3+ | 0 | No | 4106 | 0.0 | 40 |
| **611** | LP002983 | Male | Yes | 1 | 0 | No | 8072 | 240.0 | 253 |
| **612** | LP002984 | Male | Yes | 2 | 0 | No | 7583 | 0.0 | 187 |
| **613** | LP002990 | Female | No | 0 | 0 | Yes | 4583 | 0.0 | 133 |

614 rows × 13 columns

**one hot encoder**

- one hot encoder name says at a time one will On and other will Off
- For example case status has two labels
- Certified
- Denied
- When you apply one hot encoding on case status , it creates two more extra columns
- case_status_Certified
- case_status_Denied

**Advantages**

- When you develop ML model it is very impoartnt the columns should -- be independent
- each other
- So here case status creating two extra columns
- Which are independent each other, which means the row values at a -- time only one
- column has 1
- Columns are independent each other
- Whcih means 90 degrees phase shift
- Whcih means perpendicular each other
- Whcih mean orthoganal each other

**Disadvantage**

- The Distavantage is if a column has 100 unique lables , 100 new columns will be
- created
- The data will become sparse , which means huge
- Columns are more means, Dimesnions are more
- The processing time is more
- The memory consumption is more
- Curse of Dimensionality

In [117]:
```
path=r"C:\Users\tanma\DATASCIENCE\EDA\vidya hackerthon data\loan_prediction.csv"
loan_prediction_df=pd.read_csv(path)

pd.get_dummies(loan_prediction_df,
               columns=['Education','Loan_Status'],
dtype='int')
```

Out[117]:

| | Loan_ID | Gender | Married | Dependents | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_A |
|---|---|---|---|---|---|---|---|---|---|
| **0** | LP001002 | Male | No | 0 | No | 5849 | 0.0 | NaN | |
| **1** | LP001003 | Male | Yes | 1 | No | 4583 | 1508.0 | 128.0 | |
| **2** | LP001005 | Male | Yes | 0 | Yes | 3000 | 0.0 | 66.0 | |
| **3** | LP001006 | Male | Yes | 0 | No | 2583 | 2358.0 | 120.0 | |
| **4** | LP001008 | Male | No | 0 | No | 6000 | 0.0 | 141.0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **609** | LP002978 | Female | No | 0 | No | 2900 | 0.0 | 71.0 | |
| **610** | LP002979 | Male | Yes | 3+ | No | 4106 | 0.0 | 40.0 | |
| **611** | LP002983 | Male | Yes | 1 | No | 8072 | 240.0 | 253.0 | |
| **612** | LP002984 | Male | Yes | 2 | No | 7583 | 0.0 | 187.0 | |
| **613** | LP002990 | Female | No | 0 | Yes | 4583 | 0.0 | 133.0 | |

614 rows × 15 columns

In [118]:
```
path=r"C:\Users\tanma\DATASCIENCE\EDA\vidya hackerthon data\loan_prediction.csv"
loan_prediction_df=pd.read_csv(path)

loan_prediction_df.drop('Loan_Status',axis=1,inplace=True)
pd.get_dummies(loan_prediction_df,dtype='int')
```

Out[118]:

| | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_Amount_Term | Credit_History | Loan_ID_LP001002 | Loan_ID_l |
|---|---|---|---|---|---|---|---|
| **0** | 5849 | 0.0 | NaN | 360.0 | 1.0 | 1 | |
| **1** | 4583 | 1508.0 | 128.0 | 360.0 | 1.0 | 0 | |
| **2** | 3000 | 0.0 | 66.0 | 360.0 | 1.0 | 0 | |
| **3** | 2583 | 2358.0 | 120.0 | 360.0 | 1.0 | 0 | |
| **4** | 6000 | 0.0 | 141.0 | 360.0 | 1.0 | 0 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **609** | 2900 | 0.0 | 71.0 | 360.0 | 1.0 | 0 | |
| **610** | 4106 | 0.0 | 40.0 | 180.0 | 1.0 | 0 | |
| **611** | 8072 | 240.0 | 253.0 | 360.0 | 1.0 | 0 | |
| **612** | 7583 | 0.0 | 187.0 | 360.0 | 1.0 | 0 | |
| **613** | 4583 | 0.0 | 133.0 | 360.0 | 0.0 | 0 | |

614 rows × 634 columns

**session--8**

**Standardization**

- Standardization means scaling the data into one scale
- We have different columns has different units so that the value will vary
- One column has very huge values
- Another column has very less values
- So it is important to scale all type of units under one scale
- We have 2 procedures

- Standrdization
- Z-score:
- the values ranges from -3 to 3
- Normalization
- Min max scalar

- `values ranges from 0 to 1`

In [119]:
```
loan_prediction_df.select_dtypes(exclude='object')
```

Out[119]:

| | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan_Amount_Term | Credit_History |
|---|---|---|---|---|---|
| **0** | 5849 | 0.0 | NaN | 360.0 | 1.0 |
| **1** | 4583 | 1508.0 | 128.0 | 360.0 | 1.0 |
| **2** | 3000 | 0.0 | 66.0 | 360.0 | 1.0 |
| **3** | 2583 | 2358.0 | 120.0 | 360.0 | 1.0 |
| **4** | 6000 | 0.0 | 141.0 | 360.0 | 1.0 |
| **...** | ... | ... | ... | ... | ... |
| **609** | 2900 | 0.0 | 71.0 | 360.0 | 1.0 |
| **610** | 4106 | 0.0 | 40.0 | 180.0 | 1.0 |
| **611** | 8072 | 240.0 | 253.0 | 360.0 | 1.0 |
| **612** | 7583 | 0.0 | 187.0 | 360.0 | 1.0 |
| **613** | 4583 | 0.0 | 133.0 | 360.0 | 0.0 |

614 rows × 5 columns

In [120]:
```
# step-1:Take the prevalaing wage column
# Z-score = x-mean/sigma
# Step-2: Calculate mean of prevailing wage
# step-3: Calculate std of prewage
# Step-4: Nr: Pwage-mean
# Step-5: pwage_zscore=Nr/Dr
```

In [ ]:
```

```

In [121]:
```
coincome=loan_prediction_df['CoapplicantIncome']
coincome_mean=loan_prediction_df['CoapplicantIncome'].mean()
coincome_std=loan_prediction_df['CoapplicantIncome'].std()
nr=coincome-coincome_mean
loan_prediction_df['CoapplicantIncome_z']=nr/coincome_std
```

In [122]:
```
loan_prediction_df[['CoapplicantIncome','CoapplicantIncome_z']]
```

Out[122]:

| | CoapplicantIncome | CoapplicantIncome_z |
|---|---|---|
| 0 | 0.0 | -0.554036 |
| 1 | 1508.0 | -0.038700 |
| 2 | 0.0 | -0.554036 |
| 3 | 2358.0 | 0.251774 |
| 4 | 0.0 | -0.554036 |
| ... | ... | ... |
| 609 | 0.0 | -0.554036 |
| 610 | 0.0 | -0.554036 |
| 611 | 240.0 | -0.472019 |
| 612 | 0.0 | -0.554036 |
| 613 | 0.0 | -0.554036 |

614 rows × 2 columns

In [123]:
```
path=r"C:\Users\tanma\DATASCIENCE\EDA\vidya hackerthon data\loan_prediction.csv"
loan_prediction_df=pd.read_csv(path)
loan_prediction_df
```

Out[123]:

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmou |
|---|---|---|---|---|---|---|---|---|---|
| 0 | LP001002 | Male | No | 0 | Graduate | No | 5849 | 0.0 | Na |
| 1 | LP001003 | Male | Yes | 1 | Graduate | No | 4583 | 1508.0 | 128 |
| 2 | LP001005 | Male | Yes | 0 | Graduate | Yes | 3000 | 0.0 | 66 |
| 3 | LP001006 | Male | Yes | 0 | Not Graduate | No | 2583 | 2358.0 | 120 |
| 4 | LP001008 | Male | No | 0 | Graduate | No | 6000 | 0.0 | 141 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 609 | LP002978 | Female | No | 0 | Graduate | No | 2900 | 0.0 | 71 |
| 610 | LP002979 | Male | Yes | 3+ | Graduate | No | 4106 | 0.0 | 40 |
| 611 | LP002983 | Male | Yes | 1 | Graduate | No | 8072 | 240.0 | 253 |
| 612 | LP002984 | Male | Yes | 2 | Graduate | No | 7583 | 0.0 | 187 |
| 613 | LP002990 | Female | No | 0 | Graduate | Yes | 4583 | 0.0 | 133 |

614 rows × 13 columns

In [124]:
```
income=loan_prediction_df['ApplicantIncome']
income_mean=loan_prediction_df['ApplicantIncome'].mean()
income_std=loan_prediction_df['ApplicantIncome'].std()
nr=income-income_mean
loan_prediction_df['ApplicantIncome_z']=nr/income_std
```

In [125]:
```
loan_prediction_df[['ApplicantIncome','ApplicantIncome_z']]
```

Out[125]:

| | ApplicantIncome | ApplicantIncome_z |
|---|---|---|
| **0** | 5849 | 0.072931 |
| **1** | 4583 | -0.134302 |
| **2** | 3000 | -0.393427 |
| **3** | 2583 | -0.461686 |
| **4** | 6000 | 0.097649 |
| **...** | ... | ... |
| **609** | 2900 | -0.409796 |
| **610** | 4106 | -0.212383 |
| **611** | 8072 | 0.436818 |
| **612** | 7583 | 0.356773 |
| **613** | 4583 | -0.134302 |

614 rows × 2 columns

In [126]:
```
loan_prediction_df['ApplicantIncome'].max(),loan_prediction_df['ApplicantIncome_z'].max()

#99.7% data between  -3 to 3
```

Out[126]: `(81000, 12.374533479765521)`

In [127]:
```
loan_prediction_df['ApplicantIncome'].min(),loan_prediction_df['ApplicantIncome_z'].min()
```

Out[127]: `(150, -0.8599481824249576)`

In [128]:
```
loan_prediction_df['ApplicantIncome_z'].idxmax()
```

Out[128]: `409`

In [129]:
```
## find out some specific rows values
loan_prediction_df.iloc[[ 601,605,305,411]]
```

Out[129]:

| | Loan_ID | Gender | Married | Dependents | Education | Self_Employed | ApplicantIncome | CoapplicantIncome | LoanAmou |
|---|---|---|---|---|---|---|---|---|---|
| **601** | LP002950 | Male | Yes | 0 | Not Graduate | NaN | 2894 | 2792.0 | 155 |
| **605** | LP002960 | Male | Yes | 0 | Not Graduate | No | 2400 | 3800.0 | Na |
| **305** | LP001990 | Male | No | 0 | Not Graduate | No | 2000 | 0.0 | Na |
| **411** | LP002319 | Male | Yes | 0 | Graduate | NaN | 6256 | 0.0 | 160 |

In [130]:
```
cols=['ApplicantIncome','ApplicantIncome_z']
ids=[301,340]
loan_prediction_df[['ApplicantIncome','ApplicantIncome_z']].iloc[[301,340]]
loan_prediction_df[cols].iloc[ids]
```

Out[130]:

| | ApplicantIncome | ApplicantIncome_z |
|---|---|---|
| **301** | 2875 | -0.413888 |
| **340** | 2647 | -0.451210 |

In [ ]:
```

```

**StandardScalar**

In [ ]:
```
#read the package
#save the package
#apply fit transform

from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
loan_prediction_df['ApplicantIncome_ss']=ss.fit_transform(loan_prediction_df[['ApplicantIncome']])
```

In [134]:
```
cols=['ApplicantIncome','ApplicantIncome_z','ApplicantIncome_ss']
loan_prediction_df[cols]
```

Out[134]:

| | ApplicantIncome | ApplicantIncome_z | ApplicantIncome_ss |
|---|---|---|---|
| **0** | 5849 | 0.072931 | 0.072991 |
| **1** | 4583 | -0.134302 | -0.134412 |
| **2** | 3000 | -0.393427 | -0.393747 |
| **3** | 2583 | -0.461686 | -0.462062 |
| **4** | 6000 | 0.097649 | 0.097728 |
| **...** | ... | ... | ... |
| **609** | 2900 | -0.409796 | -0.410130 |
| **610** | 4106 | -0.212383 | -0.212557 |
| **611** | 8072 | 0.436818 | 0.437174 |
| **612** | 7583 | 0.356773 | 0.357064 |
| **613** | 4583 | -0.134302 | -0.134412 |

614 rows × 3 columns

**Normalization**

*minmaxScaler*

In [135]:
```
path=r"C:\Users\tanma\DATASCIENCE\EDA\vidya hackerthon data\loan_prediction.csv"
loan_prediction_df=pd.read_csv(path)

# x-x_min/(x_max-x_min)
# step-1: Read the pawge column
# step-2: Find the min value of the pwage column
# step-2: Find the max value of the pwage column
# Step-3: nr= datacolumn-min value
# Step-4: dr= max_value-min_value
# Step-5: nr/dr

pincome=loan_prediction_df['ApplicantIncome']
pincome_min=loan_prediction_df['ApplicantIncome'].min()
pincome_max=loan_prediction_df['ApplicantIncome'].max()
nr=pincome-pincome_min
dr=pincome_max-pincome_min
loan_prediction_df['ApplicantIncome_norm']=nr/dr
```

In [136]:
```
loan_prediction_df['ApplicantIncome_norm'].min(),loan_prediction_df['ApplicantIncome_norm'].max()
```

Out[136]: (0.0, 1.0)

**minmaxscaler**

In [137]:
```
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler()
loan_prediction_df['ApplicantIncome_mms']=ss.fit_transform(loan_prediction_df[['ApplicantIncome']]
```

In [138]:
```
loan_prediction_df['ApplicantIncome_mms']
```

Out[138]:
```
0      0.072991
1     -0.134412
2     -0.393747
3     -0.462062
4      0.097728
         ...
609   -0.410130
610   -0.212557
611    0.437174
612    0.357064
613   -0.134412
Name: ApplicantIncome_mms, Length: 614, dtype: float64
```

**session-9**

**Data Transformation Techniques**

- Generally used for to convert Normal distribution
- Because all statistical math analysis by assumption Data follows Normal distribution
- It is also avoid skew ness also
- We have some important transformation
- Log transformation
- Exponential transformation
- Reciprocal transformation
- Square root transformation
- Power transformaton

In [139]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [140]:
```
dict1={'Names':['Ramesh','Suresh',np.nan,'Mahesh'],
         'Age':[31,32,33,np.nan],
         'City':[np.nan,'Hyd','Mumbai','Chennai']}
```

In [141]:
```
data1=pd.DataFrame(dict1)
```

In [142]:
```
data1.isnull()
```

Out[142]:

| | Names | Age | City |
|---|---|---|---|
| **0** | False | False | True |
| **1** | False | False | False |
| **2** | True | False | False |
| **3** | False | True | False |

In [143]:
```
data1.isnull().sum()
 # every column has one missing value is there
```

Out[143]:
```
Names     1
Age       1
City      1
dtype: int64
```

In [144]:
```
data1.isnull().sum()/len(data1)
```

Out[144]:
```
Names     0.25
Age       0.25
City      0.25
dtype: float64
```

In [145]:
```
data1.isnull().sum()*100/len(data1)
```

Out[145]:
```
Names     25.0
Age       25.0
City      25.0
dtype: float64
```

In [147]:
```
dict2={'Names':['Ramesh','Suresh',None,'Mahesh'],
       'Age':[31,32,33,None],
        'City':[None,'Hyd','Mumbai','Chennai']}
data2=pd.DataFrame(dict2)
data2
```

Out[147]:

| | Names | Age | City |
|---|---|---|---|
| **0** | Ramesh | 31.0 | None |
| **1** | Suresh | 32.0 | Hyd |
| **2** | None | 33.0 | Mumbai |
| **3** | Mahesh | NaN | Chennai |

In [148]:
```
 data2.isnull().sum()
```

Out[148]:
```
Names     1
Age       1
City      1
dtype: int64
```

In [149]:
```
dict3={'Names':['Ramesh','Suresh','Null','Mahesh'],
     'Age':[31,32,33,'Null'],
     'City':['Null','Hyd','Mumbai','Chennai']}
data3=pd.DataFrame(dict3)
data3
```

Out[149]:

| | Names | Age | City |
|---|---|---|---|
| **0** | Ramesh | 31 | Null |
| **1** | Suresh | 32 | Hyd |
| **2** | Null | 33 | Mumbai |
| **3** | Mahesh | Null | Chennai |

In [150]:
```
 Method-1
 fill the missing values with random number
 dataframe name=data1
 method name:fillna
 data1.fillna(40
```

```
  Cell In[150], line 1
    **Method-1**
    ^
SyntaxError: invalid syntax
```

**Method-1**

- fill the missing values with random number
- dataframe name=data1
- method name:fillna

In [151]:
```
data1.fillna(40)
```
Out[151]:

| | Names | Age | City |
|---|---|---|---|
| **0** | Ramesh | 31.0 | 40 |
| **1** | Suresh | 32.0 | Hyd |
| **2** | 40 | 33.0 | Mumbai |
| **3** | Mahesh | 40.0 | Chennai |

### Method-2

- fill the missing values with random numbers on specific column
- dataframe name=data1-
- method name:fillna

In [154]:
```
data1['Names'].fillna('Sathish',inplace=True)
data1
```
Out[154]:

| | Names | Age | City |
|---|---|---|---|
| **0** | Ramesh | 31.0 | NaN |
| **1** | Suresh | 32.0 | Hyd |
| **2** | Sathish | 33.0 | Mumbai |
| **3** | Mahesh | NaN | Chennai |

### Method-3

- bfill
- ffill
- pad
- backfill

In [155]:
```
data1.fillna(method='backfill')
```
Out[155]:

| | Names | Age | City |
|---|---|---|---|
| **0** | Ramesh | 31.0 | Hyd |
| **1** | Suresh | 32.0 | Hyd |
| **2** | Sathish | 33.0 | Mumbai |
| **3** | Mahesh | NaN | Chennai |

In [ ]:
```
 # names index 2 is missed value
 # it will replace by index 3 value
# age index 3 is missed value
 # we dont have index 4,so the value is NaN
 # city index 0 has missed value
 # it replace with index 1 value
```
In [156]:
```
data1.fillna(method='bfill')
```
Out[156]:

| | Names | Age | City |
|---|---|---|---|
| **0** | Ramesh | 31.0 | Hyd |
| **1** | Suresh | 32.0 | Hyd |
| **2** | Sathish | 33.0 | Mumbai |
| **3** | Mahesh | NaN | Chennai |

In [157]:
```
data1.fillna(method='ffill')
```

Out[157]:

| | Names | Age | City |
|---|---|---|---|
| 0 | Ramesh | 31.0 | NaN |
| 1 | Suresh | 32.0 | Hyd |
| 2 | Sathish | 33.0 | Mumbai |
| 3 | Mahesh | 33.0 | Chennai |

In [158]:
```
data1.fillna(method='pad')
```

Out[158]:

| | Names | Age | City |
|---|---|---|---|
| 0 | Ramesh | 31.0 | NaN |
| 1 | Suresh | 32.0 | Hyd |
| 2 | Sathish | 33.0 | Mumbai |
| 3 | Mahesh | 33.0 | Chennai |

#### Method-4

- mean
- median
- mode

In [159]:
```
data1
```

Out[159]:

| | Names | Age | City |
|---|---|---|---|
| 0 | Ramesh | 31.0 | NaN |
| 1 | Suresh | 32.0 | Hyd |
| 2 | Sathish | 33.0 | Mumbai |
| 3 | Mahesh | NaN | Chennai |

In [160]:
```
#mean
age_mean=data1['Age'].mean()
age_mean
```

Out[160]:
```
32.0
```

In [161]:
```
 data1['Age'].fillna(age_mean)
```

Out[161]:
```
0    31.0
1    32.0
2    33.0
3    32.0
Name: Age, dtype: float64
```

In [162]:
```
#median
age_median=data1['Age'].median()
age_median
```

Out[162]:
```
32.0
```

In [163]:
```
data1['Age'].fillna(age_median)
```

Out[163]:
```
0    31.0
1    32.0
2    33.0
3    32.0
Name: Age, dtype: float64
```

In [165]:
```python
#mode

age_mode=data1['Age'].mode()
age_mode
```

Out[165]:
```
0    31.0
1    32.0
2    33.0
Name: Age, dtype: float64
```

**KNN imputer**

- KNN: k nearest neighbours
- In the knn imputer instead of taking mean of all the values
- it will choose neighbours data
- will take those mean only
- Method-6
- KNN imputer
- n_neighbors is a parameter can choose
- if we dont to choose by default it will take as 5

In [166]:
```python
from sklearn.impute import KNNImputer
knn=KNNImputer(n_neighbors=2)
knn.fit_transform(data1[['Age']])
```

Out[166]:
```
array([[31.],
       [32.],
       [33.],
       [32.]])
```

In [167]:
```python
data1
```

Out[167]:

| | Names | Age | City |
|---|---|---|---|
| **0** | Ramesh | 31.0 | NaN |
| **1** | Suresh | 32.0 | Hyd |
| **2** | Sathish | 33.0 | Mumbai |
| **3** | Mahesh | NaN | Chennai |

**Method-6**

- based on other columns
- sometimes all above methods will not provide good justification
- at that time we need to check other columns dependency also
- most of the time we will pick a column which have greatest correlati

**EDA_session:10 Data transformation techniques**

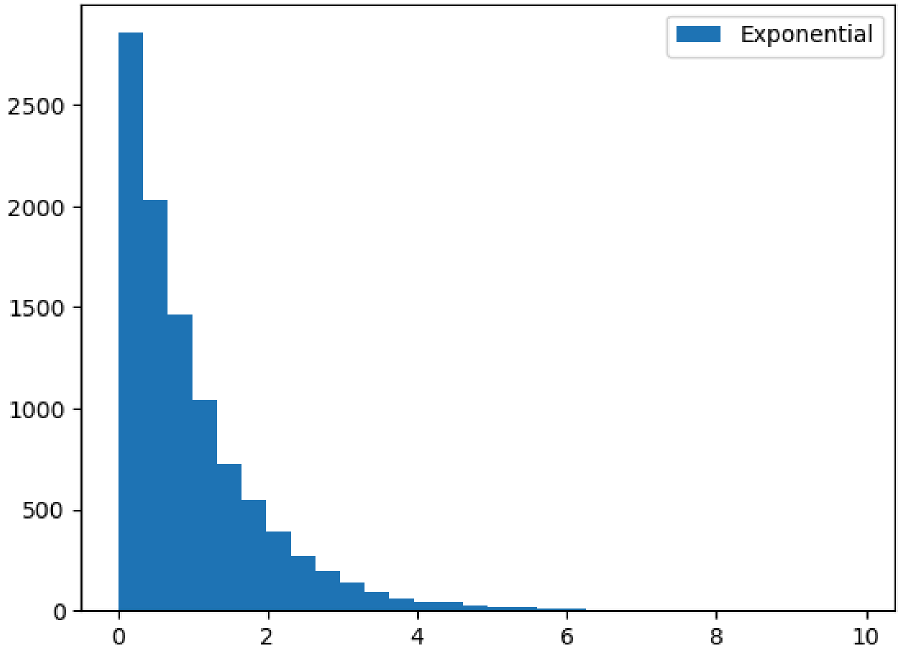
- Generally used for to convrt normal distribution
- Because all statistical math analysis by assumption data follows normal distribution
- it is also avoid skewness also
- we have some important transformation
- log transformation
- exponential transformation
- reciprocal transformation
- square root transformation
- power transformation

$Exponential - \mathbf{d}ata$

In [168]:
```
exp_data=np.random.exponential(size=10000)
exp_data[:10]
```

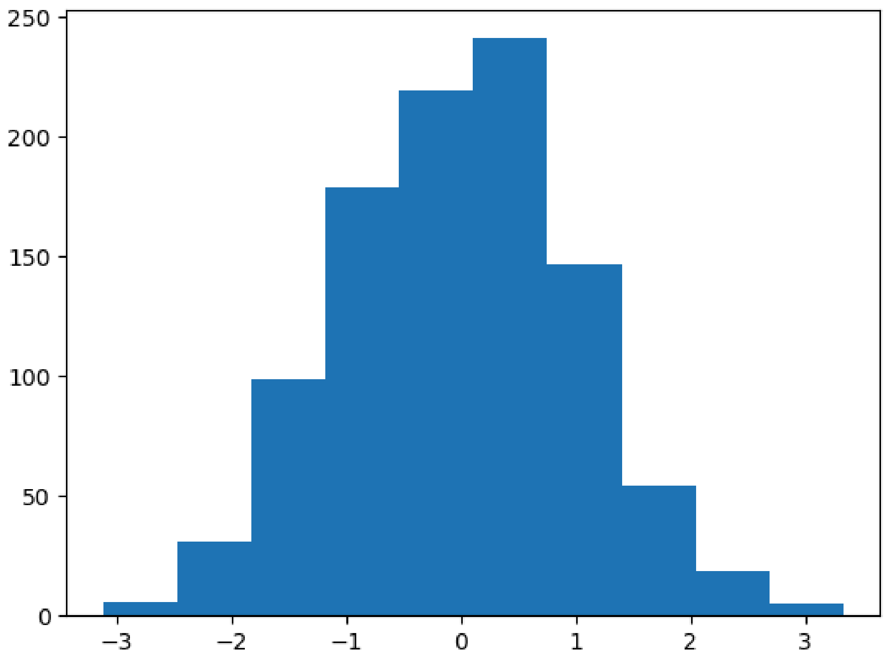
Out[168]:
```
array([0.11159125, 0.81642271, 0.97756268, 0.1380922 , 0.00522318,
       0.17717701, 1.1625165 , 0.34377777, 2.06549647, 0.54824368])
```

In [169]:
```
plt.hist(exp_data,bins=30,label='Exponential')
plt.legend()
plt.show()
```




In [170]:
```
norm_data=np.random.normal(size=1000)
plt.hist(norm_data)
```

Out[170]:
```
(array([  6.,  31.,  99., 179., 219., 241., 147.,  54.,  19.,   5.]),
 array([-3.11396248, -2.46946786, -1.82497324, -1.18047862, -0.535984  ,
         0.10851062,  0.75300524,  1.39749986,  2.04199448,  2.6864891 ,
         3.33098372]),
 <BarContainer object of 10 artists>)
```




**step-3**

- log transformation

- In [76]:
- np.log is used for log transformation
- generally log transformation will not convert data into normal
- it avoids skew ness
- np.log means natural logarithm base

In [171]:
```
x=2
import numpy as np
np.log(2)
```

Out[171]:
```
0.6931471805599453
```

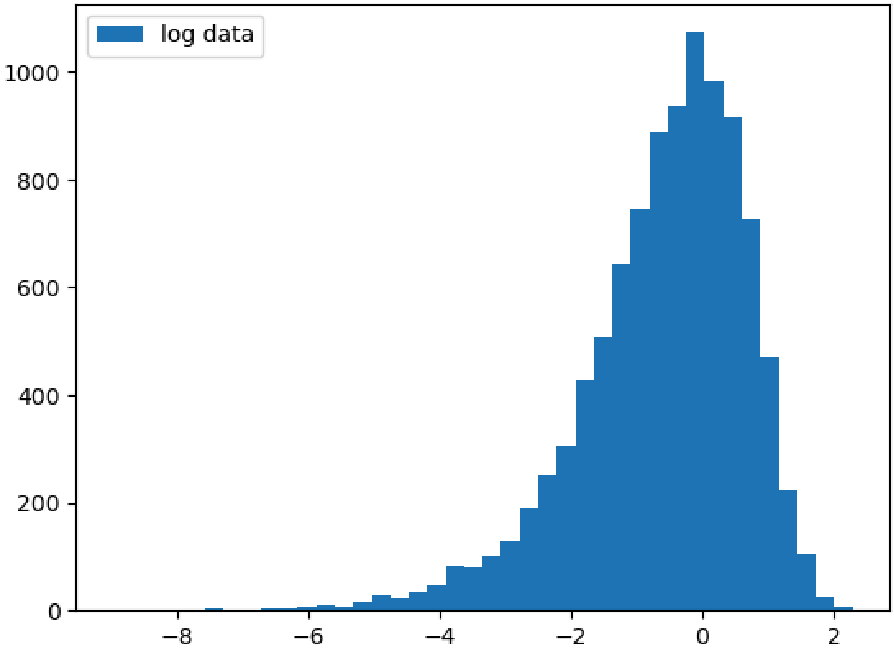
In [173]:
```
log_data=np.log(exp_data)
log_data[:10]
```

Out[173]:
```
array([-2.19291261, -0.20282304, -0.02269287, -1.97983372, -5.25464932,
       -1.73060601,  0.15058705, -1.06775984,  0.72537062, -0.60103542])
```

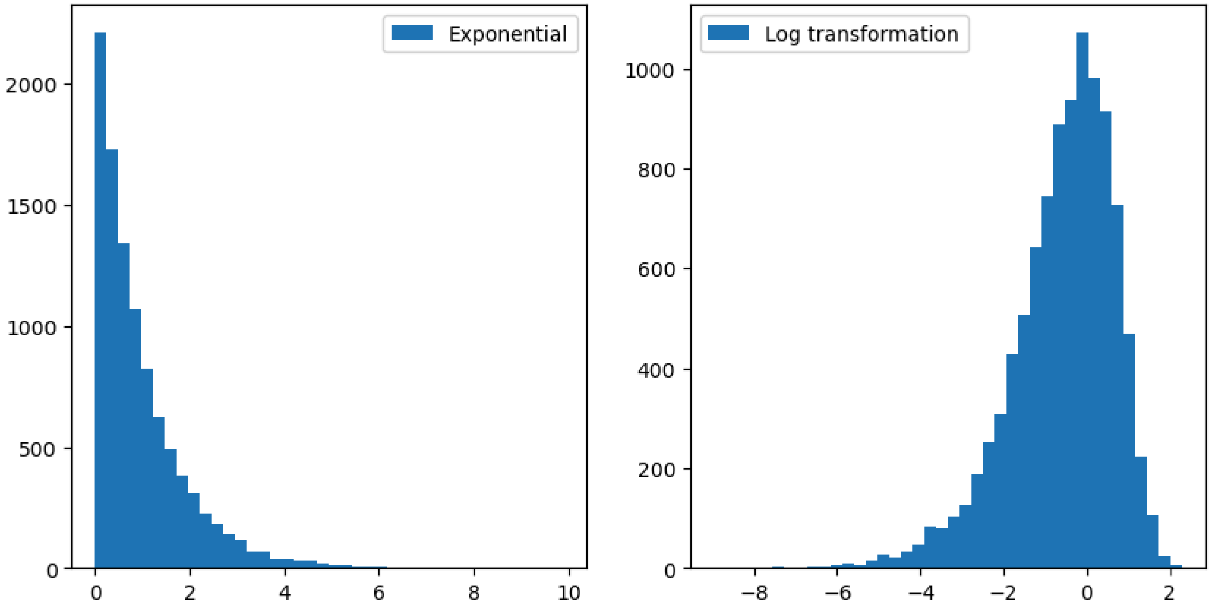
In [174]:
```
exp_data[:10]
```

Out[174]:
```
array([0.11159125, 0.81642271, 0.97756268, 0.1380922 , 0.00522318,
       0.17717701, 1.1625165 , 0.34377777, 2.06549647, 0.54824368])
```

In [175]:
```
plt.hist(log_data,bins=40,label='log data')
plt.legend()
plt.show()
```



In [176]:
```
plt.figure(figsize=(10,5))
plt.subplot(1,2,1).hist(exp_data,
                        bins=40,
                        label='Exponential')
plt.legend()
plt.subplot(1,2,2).hist(log_data,
                         bins=40,
                         label='Log transformation')
plt.legend()
plt.show()
```




$step$ **− 4 Reciprocol transformation**

- reciprocol transformaton fails when data has zero valu

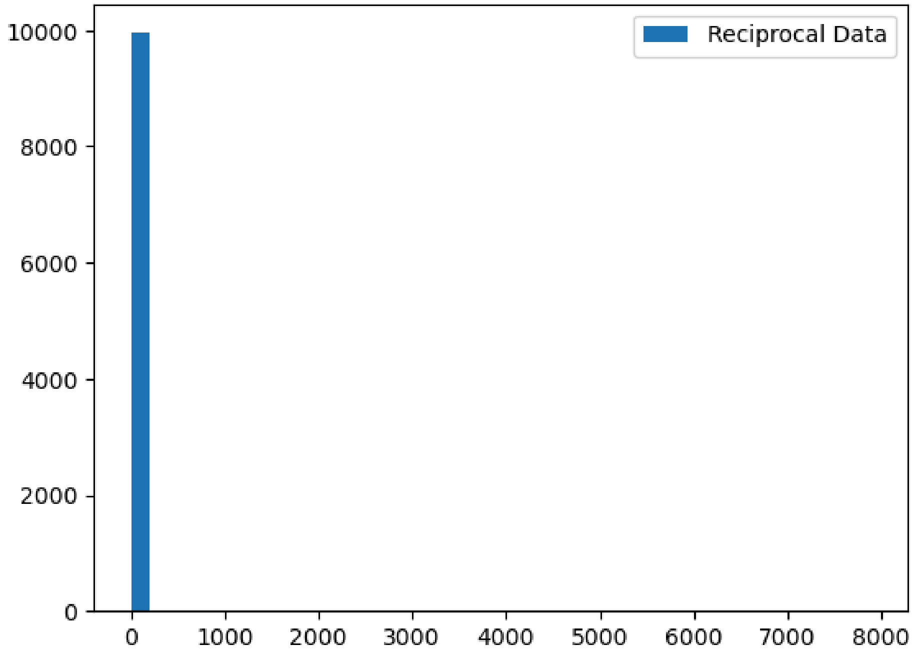
In [178]:
```
x=3
np.reciprocal(x)
```

Out[178]: 0

In [179]:
```
1/0.77
```

Out[179]: 1.2987012987012987

In [180]:
```
rec_data=np.reciprocal(exp_data)
plt.hist(rec_data,bins=40,label='Reciprocal Data')
plt.legend()
plt.show()
```




In [182]:
```
exp_data,rec_data
```

Out[182]:
```
(array([0.11159125, 0.81642271, 0.97756268, ..., 1.06779374, 1.08804838,
        0.80334309]),
 array([8.96127587, 1.22485569, 1.02295231, ..., 0.93651045, 0.91907678,
        1.24479815]))
```

In [183]:
```
 exp_data[:2]
```

Out[183]:
```
array([0.11159125, 0.81642271])
```

**step-5**

*** Square root transformatio***

In [184]:
```
print(25**2)
print(25**(1/2))
print(np.sqrt(25))
```

```
625
5.0
5.0
```

In [186]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```

```

In [ ]:

```

```