loan prediction

Objective of the Article

- This article is about who are applying for loan

import packages:

In [131…

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

Read Data:

- To read the data there are three steps
  - step1 : Have to enter file location
  - step2 : Have to enter file name
  - step3 : Have to add type of file is .csv , .txt ,.xlsx

In [132…

```python
file_name='C:\\Users\\Bhanu Kumar\\Desktop\\EDA\\train_ctrUa4K.csv'
loan_data=pd.read_csv(file_name)
loan_data
```

Out[132]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_
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	
...	...	...	...	...	...	...	...	...	...	
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	

614 rows × 13 columns

Analyzing the data:

- type
- len
- size
- shape
- columns
- head
- tail
- take
- iloc
- loc
- type casting
  - columns

- dtypes
- cat and numerical
- isnull

Type

In [133…
```
type(loan_data)
```
Out[133]:
```
pandas.core.frame.DataFrame
```

shape

In [134…
```
loan_data.shape
```
Out[134]:
```
(614, 13)
```

- 614 are columns which can be denoted by axis=0
- 13 are rows which can be denoted by axis=1

In [135…
```
a=(614,13)
print('no.of columns',a[0])
print('no of rows',a[1])
```
```
no.of columns 614
no of rows 13
```

Type of datashape

In [136…
```
type(loan_data.shape)
```
Out[136]:
```
tuple
```

length

In [137…
```
len(loan_data)
```
Out[137]:
```
614
```

size

In [138…
```
loan_data.size
```
Out[138]:
```
7982
```

In [139…
```
values=len(loan_data)*(loan_data.size)
print('Total no of values in loan prediction :',values)
```
```
Total no of values in loan prediction : 4900948
```

columns

In [140…
```
loan_data.columns
```
Out[140]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

Head

In [141…
```
loan_data.head()
```
Out[141]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Te
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	36(
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	36(
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	36(
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	36(
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	36(

Tail

In [142…
```
loan_data.tail()
```

Out[142]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	3
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	1
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	3
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	3
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	3

Take

In [143…
```
loan_data.take([100,200,300])
```

Out[143]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_
100	LP001345	Male	Yes	2	Not Graduate	No	4288	3263.0	133.0	1
200	LP001674	Male	Yes	1	Not Graduate	No	2600	2500.0	90.0	3
300	LP001964	Male	Yes	0	Not Graduate	No	1800	2934.0	93.0	3

In [144…
```
loan_data.take([10,11],axis=1)
```

Out[144]:

	Credit_History	Property_Area
0	1.0	Urban
1	1.0	Rural
2	1.0	Urban
3	1.0	Urban
4	1.0	Urban
...	...	...
609	1.0	Rural
610	1.0	Rural
611	1.0	Urban
612	1.0	Urban
613	0.0	Semiurban

614 rows × 2 columns

loc&iloc

In [145…
```
loan_data.iloc[100:300,0:3]
```

Out[145]:

	Loan_ID	Gender	Married
100	LP001345	Male	Yes
101	LP001349	Male	No
102	LP001350	Male	Yes
103	LP001356	Male	Yes
104	LP001357	Male	NaN
...	...	...	...
295	LP001949	Male	Yes
296	LP001953	Male	Yes
297	LP001954	Female	Yes
298	LP001955	Female	No
299	LP001963	Male	Yes

200 rows × 3 columns

In [146…
```
loan_data.iloc[[600,601,602],[1,2,3]]
```

Out[146]:

	Gender	Married	Dependents
600	Female	No	3+
601	Male	Yes	0
602	Male	Yes	3+

In [147…
```
loan_data.loc[[100,200,300],['Credit_History', 'Property_Area', 'Loan_Status']]
```

Out[147]:

	Credit_History	Property_Area	Loan_Status
100	1.0	Urban	Y
200	1.0	Semiurban	Y
300	0.0	Urban	N

obsevation for head ,tail ,take,loc,iloc

- Head
  - top 5 rows
- Tail
  - last 5 rows
- Take
  - It represents the rows or columns based on axis
- loc
  - It represents same as take but here we can call rows and columns with names
- iloc
  - It represents rows and columns by there index values

Datatypes

In [148…
```
loan_data.dtypes
```

Out[148]:
```
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 [149…
```
dict(loan_data.dtypes)
```

Out[149]:
```
{'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')}
```

- There are in series and converted in dictionary by using type casting
- By this we form a dataframe

cat_list, num_list

In [150…
```
col_val=dict(loan_data.dtypes)
cat_list=[keys for keys,values in col_val.items() if values=='O']
num_list=[keys for keys,values in col_val.items() if values !='O']
cat_list,num_list
```

Out[150]:
```
([
'Loan_ID',
  'Gender',
  'Married',
  'Dependents',
  'Education',
  'Self_Employed',
  'Property_Area',
  'Loan_Status'],
 ['ApplicantIncome',
  'CoapplicantIncome',
  'LoanAmount',
  'Loan_Amount_Term',
  'Credit_History'])
```

isnull

In [151…
```
loan_data.isnull()
```

Out[151]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Te
0	False	False	False	False	False	False	False	False	True	Fa
1	False	False	False	False	False	False	False	False	False	Fa
2	False	False	False	False	False	False	False	False	False	Fa
3	False	False	False	False	False	False	False	False	False	Fa
4	False	False	False	False	False	False	False	False	False	Fa
...	...	...	...	...	...	...	...	...	...	
609	False	False	False	False	False	False	False	False	False	Fa
610	False	False	False	False	False	False	False	False	False	Fa
611	False	False	False	False	False	False	False	False	False	Fa
612	False	False	False	False	False	False	False	False	False	Fa
613	False	False	False	False	False	False	False	False	False	Fa

614 rows × 13 columns

In [152…
```
loan_data.isnull().sum()
```

Out[152]:
```
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
```

observation of isnull

- null means empty value
- In the data where it denotes true there is no value in the area
- which means missing value

Missing values

In [153…
```
amt_med=loan_data['LoanAmount'].median()
amt_term_med=loan_data['Loan_Amount_Term'].median()
cre_med=loan_data['Credit_History'].median()
print(amt_med,amt_term_med,cre_med)
```

```
128.0 360.0 1.0
```

In [154…
```
loan_data['LoanAmount']=loan_data['LoanAmount'].fillna(amt_med)
loan_data['Loan_Amount_Term']=loan_data['Loan_Amount_Term'].fillna(amt_term_med)
loan_data['Credit_History']=loan_data['Credit_History'].fillna(cre_med)
```

In [155…
```
gen_mode=loan_data['Gender'].mode()
marr_mode=loan_data['Married'].mode()
de_mode=loan_data['Dependents'].mode()
self_mode=loan_data['Self_Employed'].mode()
print(gen_mode)
print('married:',marr_mode)
print('dependents:',de_mode)
```

```
print('self_employeed:',self_mode)
```

```
0    Male
Name: Gender, dtype: object
married: 0    Yes
Name: Married, dtype: object
dependents: 0    0
Name: Dependents, dtype: object
self_employeed: 0    No
Name: Self_Employed, dtype: object
```

In [156…

```
loan_data['Gender']=loan_data['Gender'].fillna('Male')
loan_data['Married']=loan_data['Married'].fillna('Yes')
loan_data['Dependents']=loan_data['Dependents'].fillna('0')
loan_data['Self_Employed']=loan_data['Self_Employed'].fillna('No')
```

In [157…

```
loan_data.isnull().sum()
```

Out[157]:

```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

EDA Univariate Analysis

- Analyzing the dataset by taking one variable
- univariate analysis can be done for both Categorical and Numerical variables
- Categorical variable:
  - count plot
  - Bar plot
  - pie plot
- Numerical variable:
  - Histogram
  - Box plot

Categorical variable

Bar-plot

- Categorical analysis of data
  - we read one categorical column
  - unique and nunique operation
  - we get value counts
  - we have to create data frame
  - we have to mention about x-axis & y-axis along with dataset name

Difference betweeen bar plot and countplot

- In bar plot we have to form Data frame
- In countplot no need of Dataframe
- In bar plot we have to metion x,y-axis ,data
- In countplot directly mention column name from given data set

In [158…

```
cat_list[1:]
```

Out[158]:
```
['Gender',
 'Married',
 'Dependents',
 'Education',
 'Self_Employed',
 'Property_Area',
 'Loan_Status']
```

In [159…
```
loan_data['Gender'].value_counts()
names=loan_data['Gender'].value_counts().keys()
values=loan_data['Gender'].value_counts().to_list()
gen_df=pd.DataFrame(zip(names,values),
                    columns=['type','count'])
print(gen_df)
print('--------------------------------------------------------------')
loan_data['Married'].value_counts()
names=loan_data['Married'].value_counts().keys()
values=loan_data['Married'].value_counts().to_list()
marr_df=pd.DataFrame(zip(names,values),
                    columns=['type','count'])
print(marr_df)
print('------------------------------------------------------------------')
loan_data['Dependents'].value_counts()
names=loan_data['Dependents'].value_counts().keys()
values=loan_data['Dependents'].value_counts().to_list()
dep_df=pd.DataFrame(zip(names,values),
                    columns=['type','count'])
print(dep_df)
print('--------------------------------------------------------------')
loan_data['Education'].value_counts()
names=loan_data['Education'].value_counts().keys()
values=loan_data['Education'].value_counts().to_list()
edu_df=pd.DataFrame(zip(names,values),
                    columns=['type','count'])
print(edu_df)
print('--------------------------------------------------------------')
loan_data['Self_Employed'].value_counts()
names=loan_data['Self_Employed'].value_counts().keys()
values=loan_data['Self_Employed'].value_counts().to_list()
sel_empl_df=pd.DataFrame(zip(names,values),
                    columns=['type','count'])
print(sel_empl_df)
print('--------------------------------------------------------------')
loan_data['Property_Area'].value_counts()
names=loan_data['Property_Area'].value_counts().keys()
values=loan_data['Property_Area'].value_counts().to_list()
pro_df=pd.DataFrame(zip(names,values),
                    columns=['type','count'])
print(pro_df)
print('--------------------------------------------------------------')
loan_data['Loan_Status'].value_counts()
names=loan_data['Loan_Status'].value_counts().keys()
values=loan_data['Loan_Status'].value_counts().to_list()
lo_st_df=pd.DataFrame(zip(names,values),
                    columns=['type','count'])
print(lo_st_df)
```

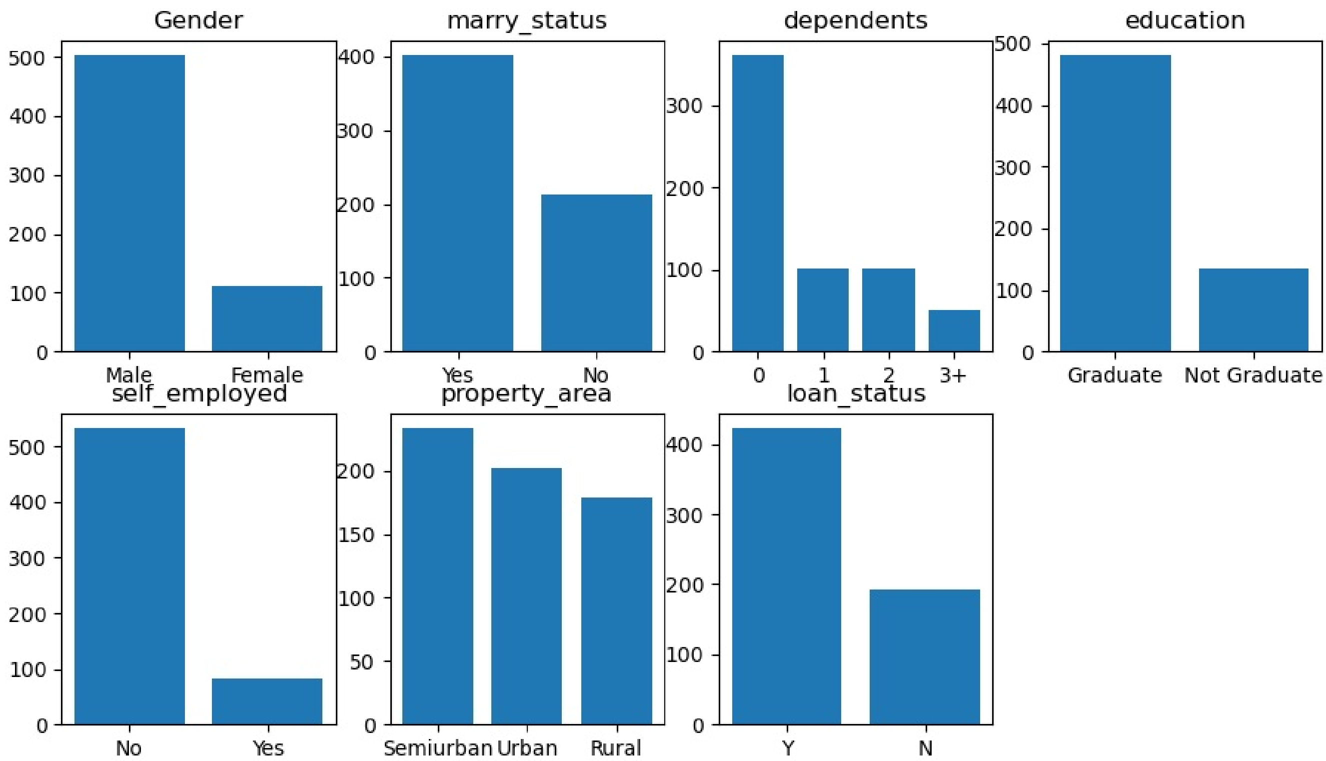
```
     type  count
0    Male    502
1  Female    112
--------------------------------------------------------------
  type  count
0  Yes    401
1   No    213
------------------------------------------------------------------
  type  count
0    0    360
1    1    102
2    2    101
3   3+     51
--------------------------------------------------------------
           type  count
0      Graduate    480
1  Not Graduate    134
--------------------------------------------------------------
  type  count
0   No    532
1  Yes     82
--------------------------------------------------------------
        type  count
0  Semiurban    233
1      Urban    202
2      Rural    179
--------------------------------------------------------------
  type  count
0    Y    422
1    N    192
```

In [160…
```
plt.figure(figsize=(11,6))
```

```
plt.subplot(2,4,1)
plt.bar('type','count',data=gen_df)
plt.title('Gender')
plt.subplot(2,4,2)
plt.bar('type','count',data=marr_df)
plt.title('marry_status')
plt.subplot(2,4,3)
plt.bar('type','count',data=dep_df)
plt.title('dependents')
plt.subplot(2,4,4)
plt.bar('type','count',data=edu_df)
plt.title('education')
plt.subplot(2,4,5)
plt.bar('type','count',data=sel_empl_df)
plt.title('self_employed')
plt.subplot(2,4,6)
plt.bar('type','count',data=pro_df)
plt.title('property_area')
plt.subplot(2,4,7)
plt.bar('type','count',data=lo_st_df)
plt.title('loan_status')
```

Out[160]:

```
Text(0.5, 1.0, 'loan_status')
```



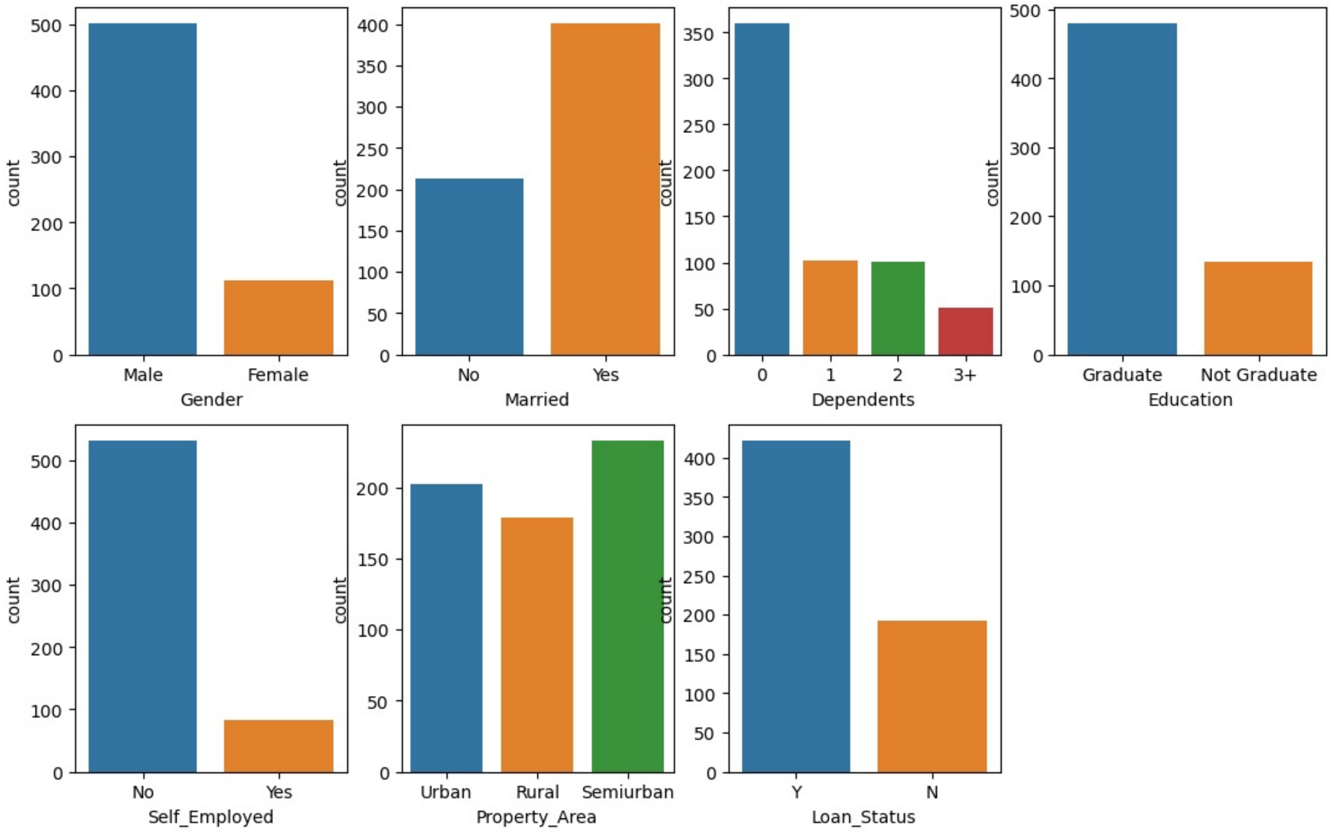
Count plot

In [161…

```
plt.figure(figsize=(13,8))
plt.subplot(2,4,1)
sns.countplot(x='Gender',data=loan_data)
plt.subplot(2,4,2)
sns.countplot(x='Married',data=loan_data)
plt.subplot(2,4,3)
sns.countplot(x='Dependents',data=loan_data)
plt.subplot(2,4,4)
sns.countplot(x='Education',data=loan_data)
plt.subplot(2,4,5)
sns.countplot(x='Self_Employed',data=loan_data)
plt.subplot(2,4,6)
sns.countplot(x='Property_Area',data=loan_data)
plt.subplot(2,4,7)
sns.countplot(x='Loan_Status',data=loan_data)
```

Out[161]:

```
<Axes: xlabel='Loan_Status', ylabel='count'>
```



observation:

- Bar graphs ,countplot are a very common type of graph used in data visualization and are used to represent one variable
- This all slide shows simple bar diagrams ,now we can observe each and every categorical column
- Gender:
  - Observing the diagram we can say that no of males and females who are needed loan
  - by seeing we can determine that males(502) are more rather than females(112)
- Married:
  - observing the diagram we can say that no. of candidates who wanted loan are married & unmarried
  - married candiates are more those who wanted money
  - total number of 614 candiates 401 are married and 213 are unmarried
- Dependents:
  - This means total number of people are dependent on loan pursuing candidates
  - By observing the plot we can say there are 4 types of dependents
  - 0 dependent person total are (360) for loan pursuing candidate
  - 1 dependent person total are(102) for loan pursuing candidate
  - 2 dependent person total are (101) for loan pursuing candidate
  - 3 dependent person total are (51) for loan pursuing candidate
  - we can find seeing the plot who are more 0 dependent person are more who needs loan
- Graduate:
  - The Total number of candiates whether graduated or not graduated
  - those who need loan are Graduated candidates more than not graduated people
- self employed:
  - By seeing the graph we can identify how many candiates those who are self employed needs loan
  - those who are not self employed needs loan 532 are not self employed and 82 are employed person who needed loan
- Property area:
  - This part of plot deals with which area the
- loan status:

- This plot deals with how many person acquiried for loan
- In the total 614 applicants 422 persons are getting loan 212 are not suittable for acquiring loan

pie-plot

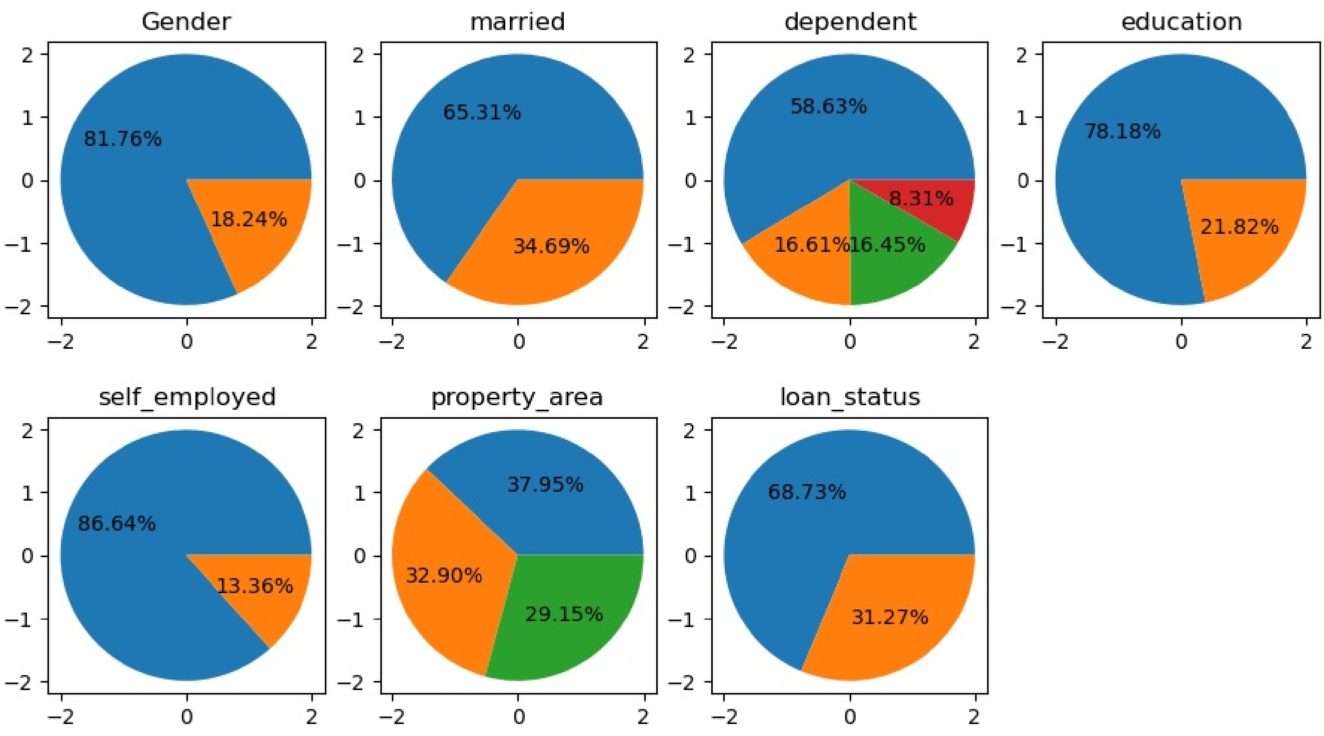
In [162…

```
gen=loan_data['Gender'].value_counts(normalize=True)
marr=loan_data['Married'].value_counts(normalize=True)
depend=loan_data['Dependents'].value_counts(normalize=True)
edu=loan_data['Education'].value_counts(normalize=True)
sel_empl=loan_data['Self_Employed'].value_counts(normalize=True)
pro_area=loan_data['Property_Area'].value_counts(normalize=True)
loan_sta=loan_data['Loan_Status'].value_counts(normalize=True)
print(gen)
print('----------------------------')
print(marr)
print('----------------------------')
print(depend)
print('----------------------------')
print(edu)
print('----------------------------')
print(sel_empl)
print('----------------------------')
print(pro_area)
print('----------------------------')
print(loan_sta)
```

```
Male      0.81759
Female    0.18241
Name: Gender, dtype: float64
----------------------------
Yes    0.653094
No     0.346906
Name: Married, dtype: float64
----------------------------
0     0.586319
1     0.166124
2     0.164495
3+    0.083062
Name: Dependents, dtype: float64
----------------------------
Graduate        0.781759
Not Graduate    0.218241
Name: Education, dtype: float64
----------------------------
No     0.86645
Yes    0.13355
Name: Self_Employed, dtype: float64
----------------------------
Semiurban    0.379479
Urban        0.328990
Rural        0.291531
Name: Property_Area, dtype: float64
----------------------------
Y    0.687296
N    0.312704
Name: Loan_Status, dtype: float64
```

In [163…

```
plt.figure(figsize=(11,6))
plt.subplot(2,4,1)
plt.pie(gen,autopct='%0.2f%%',radius=2,frame=True)
plt.title('Gender')
plt.subplot(2,4,2)
plt.pie(marr,autopct='%0.2f%%',radius=2,frame=True)
plt.title('married')
plt.subplot(2,4,3)
plt.pie(depend,autopct='%0.2f%%',radius=2,frame=True)
plt.title('dependent')
plt.subplot(2,4,4)
plt.pie(edu,autopct='%0.2f%%',radius=2,frame=True)
plt.title('education')
plt.subplot(2,4,5)
plt.pie(sel_empl,autopct='%0.2f%%',radius=2,frame=True)
plt.title('self_employed')
plt.subplot(2,4,6)
plt.pie(pro_area,autopct='%0.2f%%',radius=2,frame=True)
plt.title('property_area')
plt.subplot(2,4,7)
plt.pie(loan_sta,autopct='%0.2f%%',radius=2,frame=True)
plt.title('loan_status')
plt.show()
```



observations for pie plots

- pie chart is a circualr graphic that displays numeric proportions by dividing a circle into proportional slices.
- Here in the pie chart we find each percentage of dataset column
- now lets see the proportional slices for each categorical list in loan prediction
- Gender
    - Here the 82% of circle covered by males and 18% by females
    - By observing the no. of females and males who are needed loan
- Married
    - Here the pie represents the total percentage of members there maritial status
    - by observing the pie chart divided into parts one for married and another for nonmarried
    - 65% are married and 35% are unmarried
- Dependent
    - The no of dependents for the person who are applying for loan
    - here we can see the most of people are 0 (58%)dependents are more by comparing 1(16%), 2(16%),3(8%) dependents for loan applicants
        - Education
    - By observing the chart we can identify there qualificaton how many are graduated and non graduated
    - most 78% persons are graduated who wants loan
- Self-employed
    - The Charts itself explain that 86% of people are not self-employed
- Property-area
    - most of the people who wants loan are from semi-urban (37%)
    - next urban(32%)
    - from rural(29%)
- loan-status
    - This is deciding column whether they get the loan or not
    - out of 614 applicants in that 68% getting loan and 32% not getting loan

Numerical variable

-

In [164…

```
num_list
```

Out[164]:

```
['ApplicantIncome',
 'CoapplicantIncome',
 'LoanAmount',
 'Loan_Amount_Term',
 'Credit_History']
```

In [165…

```
data1=loan_data['ApplicantIncome']
dict2={}
dict2['p_mean']=round(loan_data['ApplicantIncome'].mean(),2)
dict2['p_median']=round(loan_data['ApplicantIncome'].median(),2)
```

```
dict2['p_std']=round(loan_data['ApplicantIncome'].std(),2)
dict2['p_max']=round(loan_data['ApplicantIncome'].max(),2)
dict2['p_min']=round(loan_data['ApplicantIncome'].min(),2)
dict2['p_count']=round(loan_data['ApplicantIncome'].count(),2)
dict2['25%']=round(np.percentile(data1,25),2)
dict2['50%']=round(np.percentile(data1,50),2)
dict2['75%']=round(np.percentile(data1,75),2)

print(pd.DataFrame(dict2,index=['ApplicantIncome']))
print('--------------------------------------------------------------------------------------')
data2=loan_data['CoapplicantIncome']
dict3={}
dict3['p_mean']=round(loan_data['CoapplicantIncome'].mean(),2)
dict3['p_median']=round(loan_data['CoapplicantIncome'].median(),2)
dict3['p_std']=round(loan_data['CoapplicantIncome'].std(),2)
dict3['p_max']=round(loan_data['CoapplicantIncome'].max(),2)
dict3['p_min']=round(loan_data['CoapplicantIncome'].min(),2)
dict3['p_count']=round(loan_data['CoapplicantIncome'].count(),2)
dict3['25%']=round(np.percentile(data2,25),2)
dict3['50%']=round(np.percentile(data2,50),2)
dict3['75%']=round(np.percentile(data2,75),2)

print(pd.DataFrame(dict3,index=['CoapplicantIncome']))
print('----------------------------------------------------------------------------------------------')
data3=loan_data['LoanAmount']
dict4={}
dict4['p_mean']=round(loan_data['LoanAmount'].mean(),2)
dict4['p_median']=round(loan_data['LoanAmount'].median(),2)
dict4['p_std']=round(loan_data['LoanAmount'].std(),2)
dict4['p_max']=round(loan_data['LoanAmount'].max(),2)
dict4['p_min']=round(loan_data['LoanAmount'].min(),2)
dict4['p_count']=round(loan_data['LoanAmount'].count(),2)
dict4['25%']=round(np.percentile(data3,25),2)
dict4['50%']=round(np.percentile(data3,50),2)
dict4['75%']=round(np.percentile(data3,75),2)

print(pd.DataFrame(dict4,index=['loan_amount']))
print('-----------------------------------------------------------------------------------------------------')
data4=loan_data['Loan_Amount_Term']
dict3={}
dict3['p_mean']=round(loan_data['Loan_Amount_Term'].mean(),2)
dict3['p_median']=round(loan_data['Loan_Amount_Term'].median(),2)
dict3['p_std']=round(loan_data['Loan_Amount_Term'].std(),2)
dict3['p_max']=round(loan_data['Loan_Amount_Term'].max(),2)
dict3['p_min']=round(loan_data['Loan_Amount_Term'].min(),2)
dict3['p_count']=round(loan_data['Loan_Amount_Term'].count(),2)
dict3['25%']=round(np.percentile(data4,25),2)
dict3['50%']=round(np.percentile(data4,50),2)
dict3['75%']=round(np.percentile(data4,75),2)

print(pd.DataFrame(dict3,index=['Loan_amount_term']))
print('-----------------------------------------------------------------------------------------------------')
data5=loan_data['Credit_History']
dict4={}
dict4['p_mean']=round(loan_data['Credit_History'].mean(),2)
dict4['p_median']=round(loan_data['Credit_History'].median(),2)
dict4['p_std']=round(loan_data['Credit_History'].std(),2)
dict4['p_max']=round(loan_data['Credit_History'].max(),2)
dict4['p_min']=round(loan_data['Credit_History'].min(),2)
dict4['p_count']=round(loan_data['Credit_History'].count(),2)
dict4['25%']=round(np.percentile(data5,25),2)
dict4['50%']=round(np.percentile(data5,50),2)
dict4['75%']=round(np.percentile(data5,75),2)

print(pd.DataFrame(dict4,index=['Credit_History']))
```

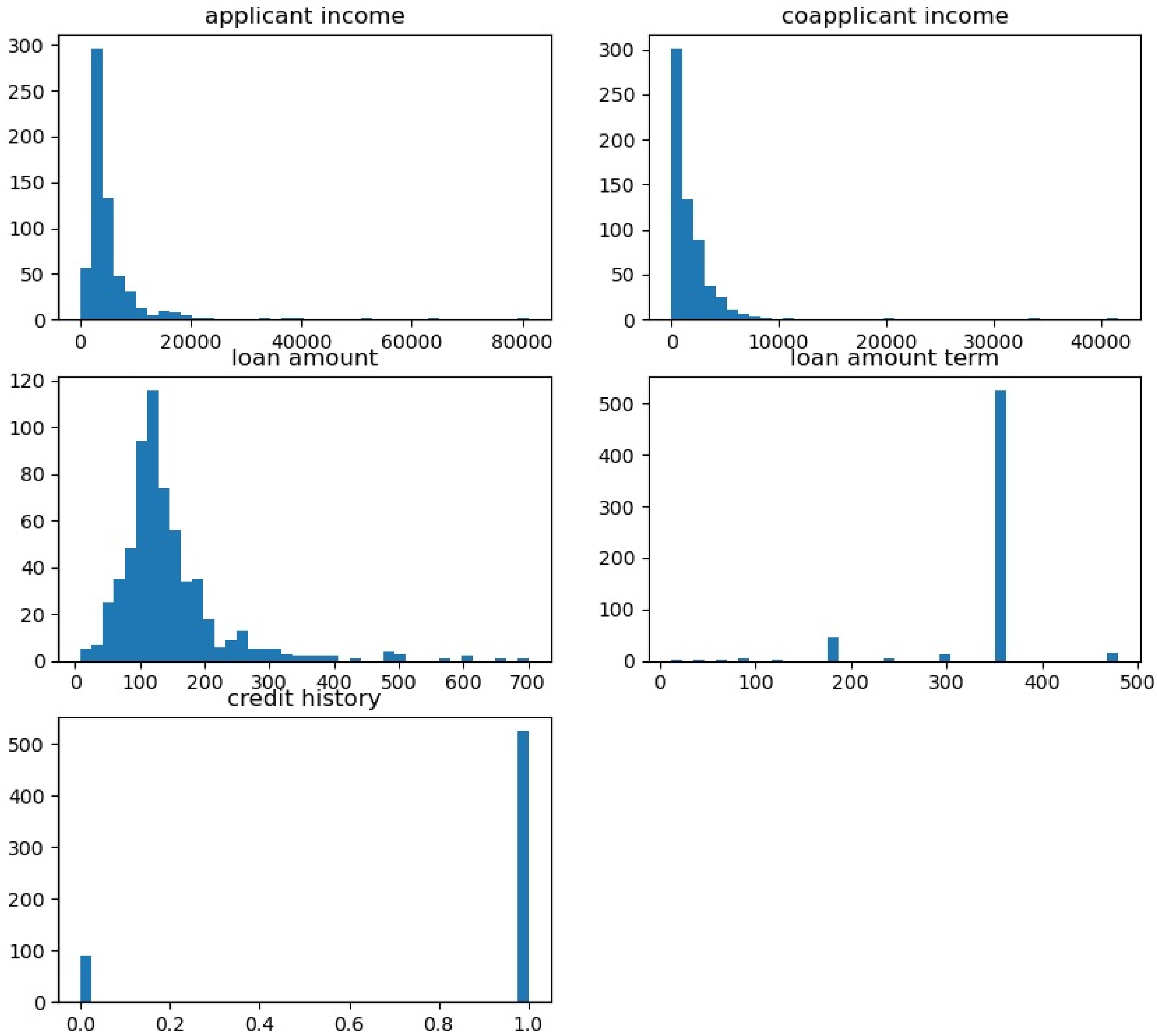
```
                  p_mean  p_median    p_std  p_max  p_min  p_count     25%  \
ApplicantIncome  5403.46    3812.5  6109.04  81000    150      614  2877.5   

                   50%     75%  
ApplicantIncome  3812.5  5795.0  
-------------------------------------------------------------------------------------
                    p_mean  p_median    p_std    p_max  p_min  p_count  25%  \
CoapplicantIncome  1621.25    1188.5  2926.25  41667.0    0.0      614  0.0   

                     50%      75%  
CoapplicantIncome  1188.5  2297.25  
---------------------------------------------------------------------------------------------
             p_mean  p_median  p_std  p_max  p_min  p_count     25%    50%  \
loan_amount  145.75     128.0  84.11  700.0    9.0      614  100.25  128.0   

                75%  
loan_amount  164.75  
----------------------------------------------------------------------------------------------------
                  p_mean  p_median  p_std  p_max  p_min  p_count    25%  \
Loan_amount_term  342.41     360.0  64.43  480.0   12.0      614  360.0   

                    50%    75%  
Loan_amount_term  360.0  360.0  
----------------------------------------------------------------------------------------------------
                p_mean  p_median  p_std  p_max  p_min  p_count  25%  50%  75%
Credit_History    0.86       1.0   0.35    1.0    0.0      614  1.0  1.0  1.0
```

Histogram

- It helps in understanding the distribution of numerical data into series of the interval
- The histogram is composed of a number of bars
- The height of the bars reflects the total count of data elements whose value falls within the frequency

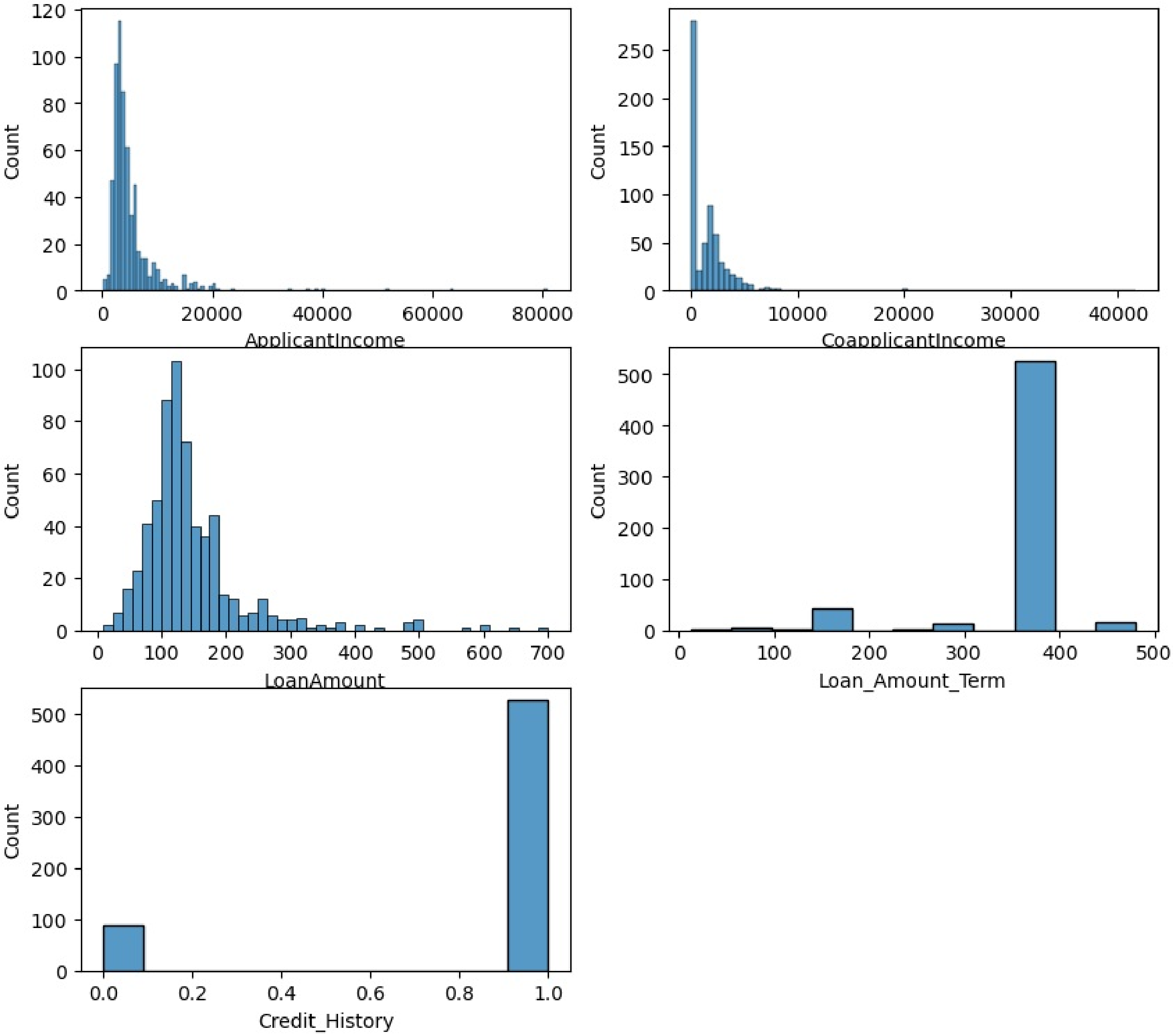
In [166…

```
plt.figure(figsize=(10,9))
plt.subplot(3,2,1)
data= loan_data['ApplicantIncome']
plt.hist(data,bins=40)
plt.title('applicant income')
plt.subplot(3,2,2)
data= loan_data['CoapplicantIncome']
plt.hist(data,bins=40)
plt.title('coapplicant income')
plt.subplot(3,2,3)
data= loan_data['LoanAmount']
plt.hist(data,bins=40)
plt.title('loan amount')
plt.subplot(3,2,4)
data= loan_data['Loan_Amount_Term']
plt.hist(data,bins=40)
plt.title('loan amount term')
plt.subplot(3,2,5)
data= loan_data['Credit_History']
plt.hist(data,bins=40)
plt.title('credit history')
plt.show()
```



In [167…

```
plt.figure(figsize=(10,9))
plt.subplot(3,2,1)
sns.histplot(data1)
plt.subplot(3,2,2)
sns.histplot(data2)
plt.subplot(3,2,3)
sns.histplot(data3)
plt.subplot(3,2,4)
sns.histplot(data4)
plt.subplot(3,2,5)
sns.histplot(data5)
plt.show()
```



Observations of histogram

- Applicant-income
  - By observing the graph we can say that it is postive skewed
  - because the graph leads to postive side
  - The min value of income is 150 and max value is 81000
  - max no of data placed in 50% of percentile data which is median 3812
  - max number of applicants lies in 50% percentile = 296 frequency
  - 296 applicant of lies at 2171 applicant income
- Co applicant income
  - By observing the graph we can say that it is postive skewed
  - The leads to postive side
  - The min value of co applicant income is 0 and max value of co applicant income is 41667
  - max number of applicants in 50% data median=1188 , the frequency of this was 301
  - 301 applicants are lie at co applicant income at min value
- loan amount
  - By observing the graph it is also postive skewed as because it leads to postive side
  - The max value of loan amount 700 and min value of laon amount is 9
  - 50%== median value
  - 116 number of applicants lies at loan amount 112
- loan amount term
  - By observing the graph we can it is not postive , negative and no skewed
  - It just lies as bar graph
  - max value of loan amount term 480 and min value of loan amount term is 12
  - 526 applicants were placed in 363 loan amount term
- credit history
  - By observing the graph we can it is not postive , negative and no skewed
  - It just lies as bar graph
  - max value of credit history 1 and min value of credit history is 0

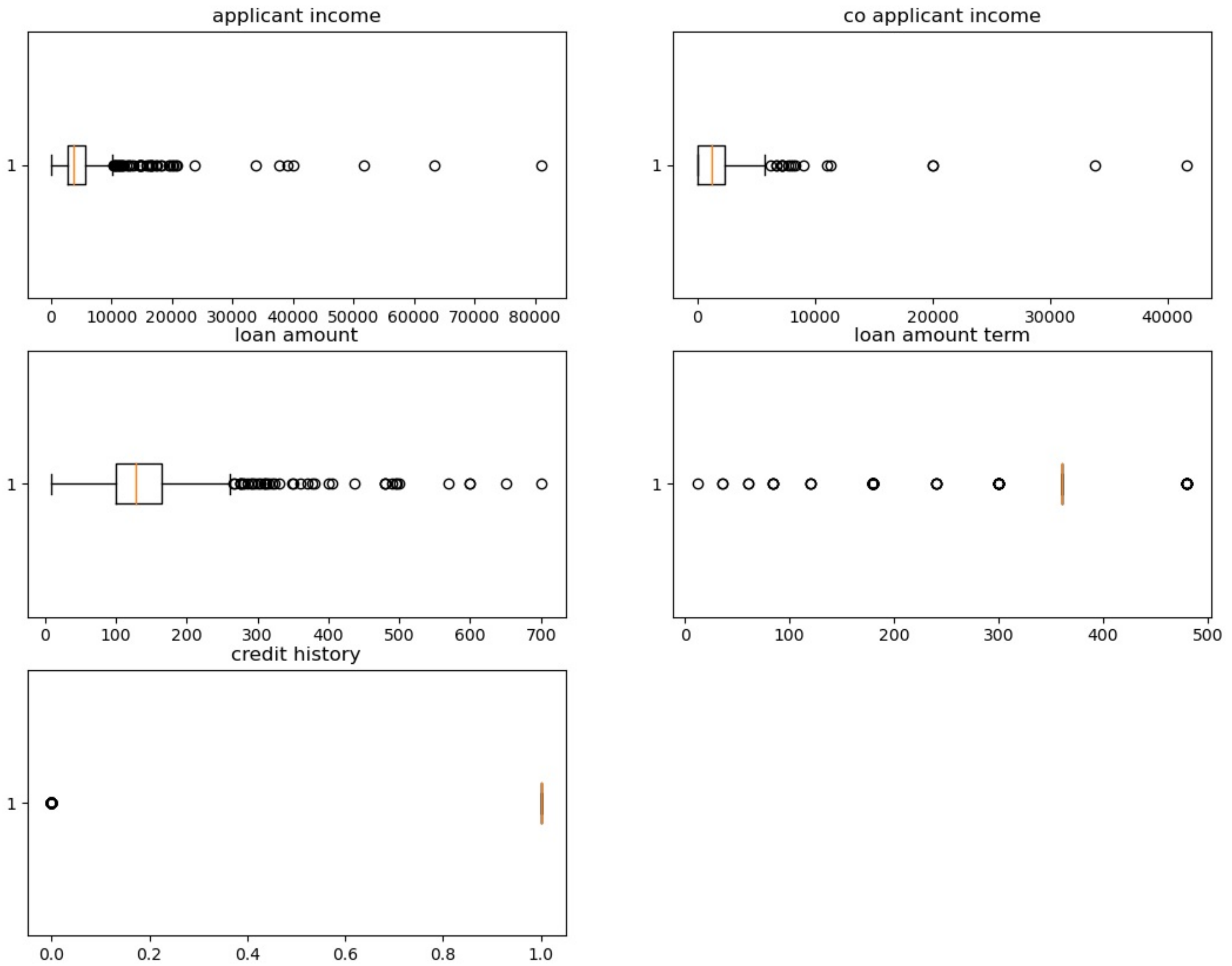
- 525 applicants were placed in 1 credit history

Box- plot

- It is visualization medium for numerical data and easy to indentify if there any outliers is present in data
- Box plot displays five-number set of data
- minimum, maxmium , quartile1, quartile2,quartile3

In [168…

```
plt.figure(figsize=(13,10))
plt.subplot(3,2,1)
plt.boxplot(data1,vert=False)
plt.title('applicant income')
plt.subplot(3,2,2)
plt.boxplot(data2,vert=False)
plt.title('co applicant income')
plt.subplot(3,2,3)
plt.boxplot(data3,vert=False)
plt.title('loan amount')
plt.subplot(3,2,4)
plt.boxplot(data4,vert=False)
plt.title('loan amount term')
plt.subplot(3,2,5)
plt.boxplot(data5,vert=False)
plt.title('credit history')
plt.show()
```



In [169…

```
Q1=round(np.quantile(data1,0.25),2)
Q2=round(np.quantile(data1,0.50),2)
Q3=round(np.quantile(data1,0.75),2)
IQR=(Q3-Q1)
LB=Q1-1.5*IQR
UB=Q3+1.5*IQR

print("Q1:",Q1)
print("Q2:",Q2)
print("Q3:",Q3)
print("IQR:",IQR)
print("LB:",LB)
print("UB:",UB)
print('-------------------------------------------')
Q1=round(np.quantile(data2,0.25),2)
```

```
Q2=round(np.quantile(data2,0.50),2)
Q3=round(np.quantile(data2,0.75),2)
IQR=(Q3-Q1)
LB=Q1-1.5*IQR
UB=Q3+1.5*IQR

print("Q1:",Q1)
print("Q2:",Q2)
print("Q3:",Q3)
print("IQR:",IQR)
print("LB:",LB)
print("UB:",UB)
print('--------------------------------------------')
Q1=round(np.quantile(data3,0.25),2)
Q2=round(np.quantile(data3,0.50),2)
Q3=round(np.quantile(data3,0.75),2)
IQR=(Q3-Q1)
LB=Q1-1.5*IQR
UB=Q3+1.5*IQR

print("Q1:",Q1)
print("Q2:",Q2)
print("Q3:",Q3)
print("IQR:",IQR)
print("LB:",LB)
print("UB:",UB)
print('---------------------------------------------')
Q1=round(np.quantile(data4,0.25),2)
Q2=round(np.quantile(data4,0.50),2)
Q3=round(np.quantile(data4,0.75),2)
IQR=(Q3-Q1)
LB=Q1-1.5*IQR
UB=Q3+1.5*IQR

print("Q1:",Q1)
print("Q2:",Q2)
print("Q3:",Q3)
print("IQR:",IQR)
print("LB:",LB)
print("UB:",UB)
print('------------------------------------------------')
Q1=round(np.quantile(data5,0.25),2)
Q2=round(np.quantile(data5,0.50),2)
Q3=round(np.quantile(data5,0.75),2)
IQR=(Q3-Q1)
LB=Q1-1.5*IQR
UB=Q3+1.5*IQR

print("Q1:",Q1)
print("Q2:",Q2)
print("Q3:",Q3)
print("IQR:",IQR)
print("LB:",LB)
print("UB:",UB)
```

```
Q1: 2877.5
Q2: 3812.5
Q3: 5795.0
IQR: 2917.5
LB: -1498.75
UB: 10171.25
------------------------------------------
Q1: 0.0
Q2: 1188.5
Q3: 2297.25
IQR: 2297.25
LB: -3445.875
UB: 5743.125
--------------------------------------------
Q1: 100.25
Q2: 128.0
Q3: 164.75
IQR: 64.5
LB: 3.5
UB: 261.5
---------------------------------------------
Q1: 360.0
Q2: 360.0
Q3: 360.0
IQR: 0.0
LB: 360.0
UB: 360.0
------------------------------------------------
Q1: 1.0
Q2: 1.0
Q3: 1.0
IQR: 0.0
LB: 1.0
UB: 1.0
```

Dealing the outliers

In [170…
```
med=loan_data['ApplicantIncome'].median()
cond=loan_data['ApplicantIncome']>UB
loan_data['ApplicantIncome']=np.where(cond,med,loan_data['ApplicantIncome'])
loan_data['ApplicantIncome']
```

Out[170]:
```
0      3812.5
1      3812.5
2      3812.5
3      3812.5
4      3812.5
        ...
609    3812.5
610    3812.5
611    3812.5
612    3812.5
613    3812.5
Name: ApplicantIncome, Length: 614, dtype: float64
```

Observation of box plot

- The yellow line indicates the median Q2-Q3
- left part indicates the low value outliers and right side indicates high value outliers
- we can also see in graphs some polts both outliers missing
- we can also fill the outliers with median value

supervised and un supervised

- Before going into 'Bivariate analysis' we have to know about unsupervised and supervised alogorithm
- supervised alogorithm
  - It have target column or output label
  - If the data set has categorical output label then called as classification alogorithm
  - If the data set has numerical output label then called as Regression alogorithm
- unsupervised alogorithm
  - It doesnt have output label
  - It is just an information

EDA Bivariate Analysis

- The analysis done by two variables
- numerical vs numerical variable
- categorical vs catgorical variable

In [171…
```
cat_list[1:]
```

Out[171]:
```
['Gender',
 'Married',
 'Dependents',
 'Education',
 'Self_Employed',
 'Property_Area',
 'Loan_Status']
```

catgorical vs catgorical variable

In [184…
```
col1=loan_data['Gender']
col2=loan_data['Loan_Status']
result1=pd.crosstab(col1,col2)
print(result1)
print('---------------------------------------------------------')
col3=loan_data['Married']
col2=loan_data['Loan_Status']
result2=pd.crosstab(col3,col2)
print(result2)
print('--------------------------------------------------------------')
col4=loan_data['Dependents']
col2=loan_data['Loan_Status']
result3=pd.crosstab(col4,col2)
print(result3)
print('----------------------------------------------------------------')
col5=loan_data['Education']
col2=loan_data['Loan_Status']
result4=pd.crosstab(col5,col2)
print(result4)
print('----------------------------------------------------------------')
```

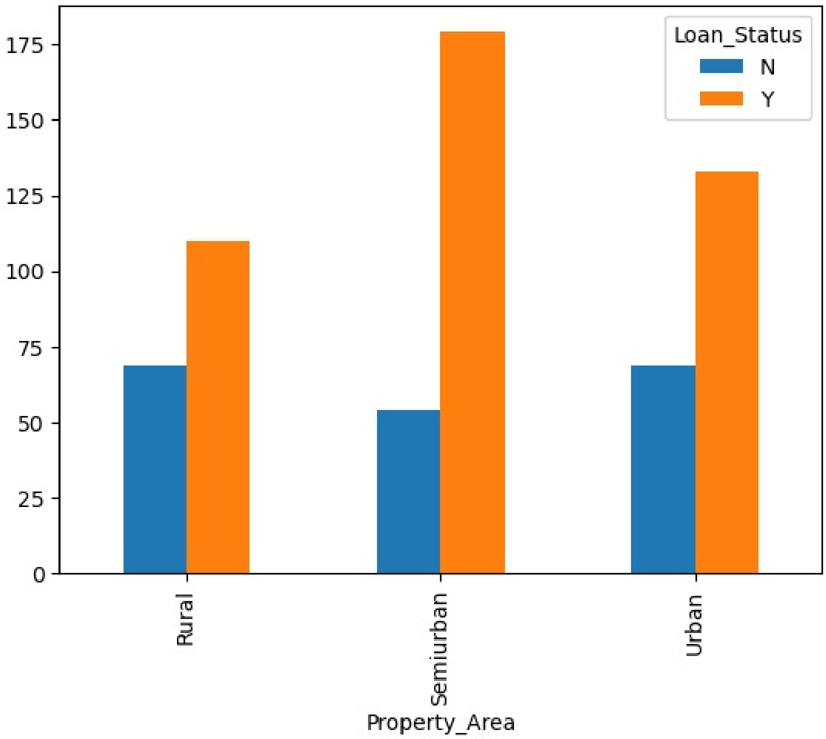
```
col6=loan_data['Self_Employed']
col2=loan_data['Loan_Status']
result5=pd.crosstab(col6,col2)
print(result5)
print('----------------------------------------------------------------')
col7=loan_data['Property_Area']
col2=loan_data['Loan_Status']
result6=pd.crosstab(col7,col2)
print(result6)
```

```
Loan_Status    N    Y
Gender
Female        37   75
Male         155  347
--------------------------------------------------------
Loan_Status    N    Y
Married
No            79  134
Yes          113  288
-------------------------------------------------------------
Loan_Status    N    Y
Dependents
0            113  247
1             36   66
2             25   76
3+            18   33
---------------------------------------------------------------
Loan_Status     N    Y
Education
Graduate      140  340
Not Graduate   52   82
---------------------------------------------------------------
Loan_Status      N    Y
Self_Employed
No             166  366
Yes             26   56
---------------------------------------------------------------
Loan_Status     N    Y
Property_Area
Rural          69  110
Semiurban      54  179
Urban          69  133
```

In [187…

```
result6.plot(kind='bar')
plt.figure(figsize=(0.3,0.1))
```

Out[187]: <Figure size 30x10 with 0 Axes>



<Figure size 30x10 with 0 Axes>

observation of bivariate bar graph

- Clear we can see that we are using columns
- orange is yes and blue is no
- how are applicable for laon status in rural ,semi urban , urban

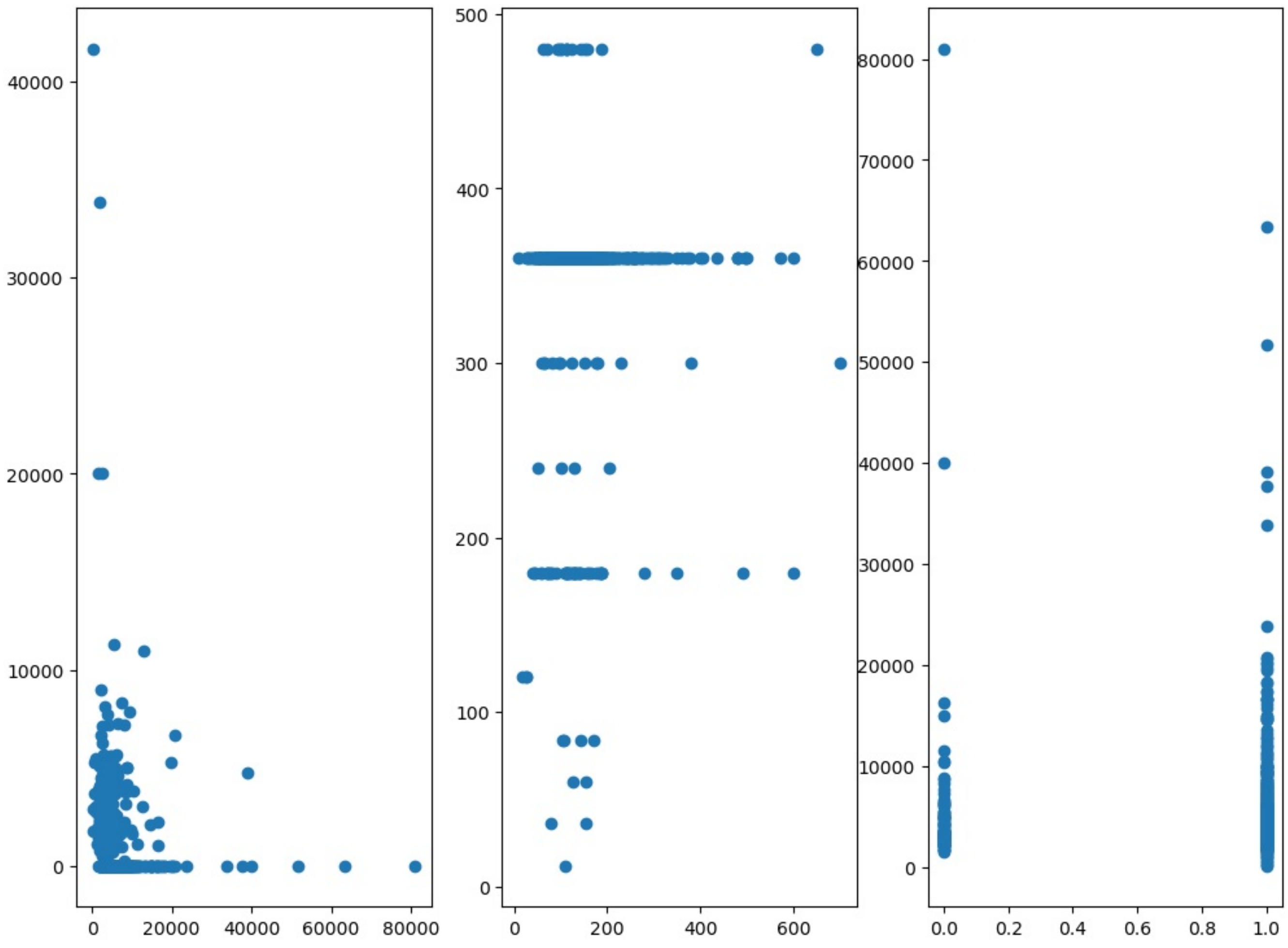
numerical vs numerical

```
In [69]: num_list
```

```
Out[69]: ['ApplicantIncome',
 'CoapplicantIncome',
 'LoanAmount',
 'Loan_Amount_Term',
 'Credit_History']
```

```
In [77]: plt.figure(figsize=(12,9))
plt.subplot(1,3,1)
col1=loan_data['ApplicantIncome']
col2=loan_data['CoapplicantIncome']
plt.scatter(col1,col2)
plt.subplot(1,3,2)
col1=loan_data['LoanAmount']
col2=loan_data['Loan_Amount_Term']
plt.scatter(col1,col2)
plt.subplot(1,3,3)
col1=loan_data['Credit_History']
col2=loan_data['ApplicantIncome']
plt.scatter(col1,col2)
```

```
Out[77]: <matplotlib.collections.PathCollection at 0x2a90a9b4e80>
```



observation of scatter plot

- applicants & co income
  - Here we can observe that max number points in the place of o the garph show linear relationship
- loan amount & loan amount term
  - we can see that max number points lies at in between 400 and 300
  - this also show linear relationship between both numerical variable
- credit history & applicants income
  - here we can see clearly that max number of point at 1
  - this also shows linear relations ship between them

heat map

```
In [78]: corr_val=loan_data.corr()
corr_val
```

```
C:\Users\Bhanu Kumar\AppData\Local\Temp\ipykernel_4744\3272850232.py:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
  corr_val=loan_data.corr()
```

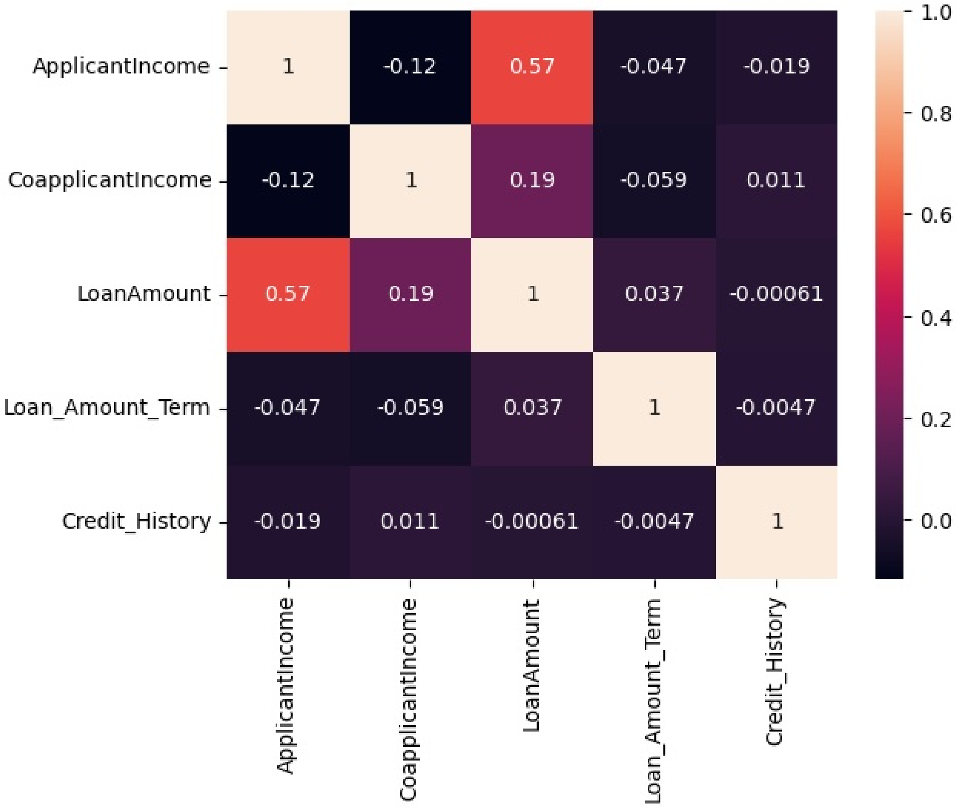
Out[78]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
ApplicantIncome	1.000000	-0.116605	0.565181	-0.046531	-0.018615
CoapplicantIncome	-0.116605	1.000000	0.189218	-0.059383	0.011134
LoanAmount	0.565181	0.189218	1.000000	0.036960	-0.000607
Loan_Amount_Term	-0.046531	-0.059383	0.036960	1.000000	-0.004705
Credit_History	-0.018615	0.011134	-0.000607	-0.004705	1.000000

In [80]:

```
sns.heatmap(corr_val,annot=True)
```

Out[80]: <Axes: >



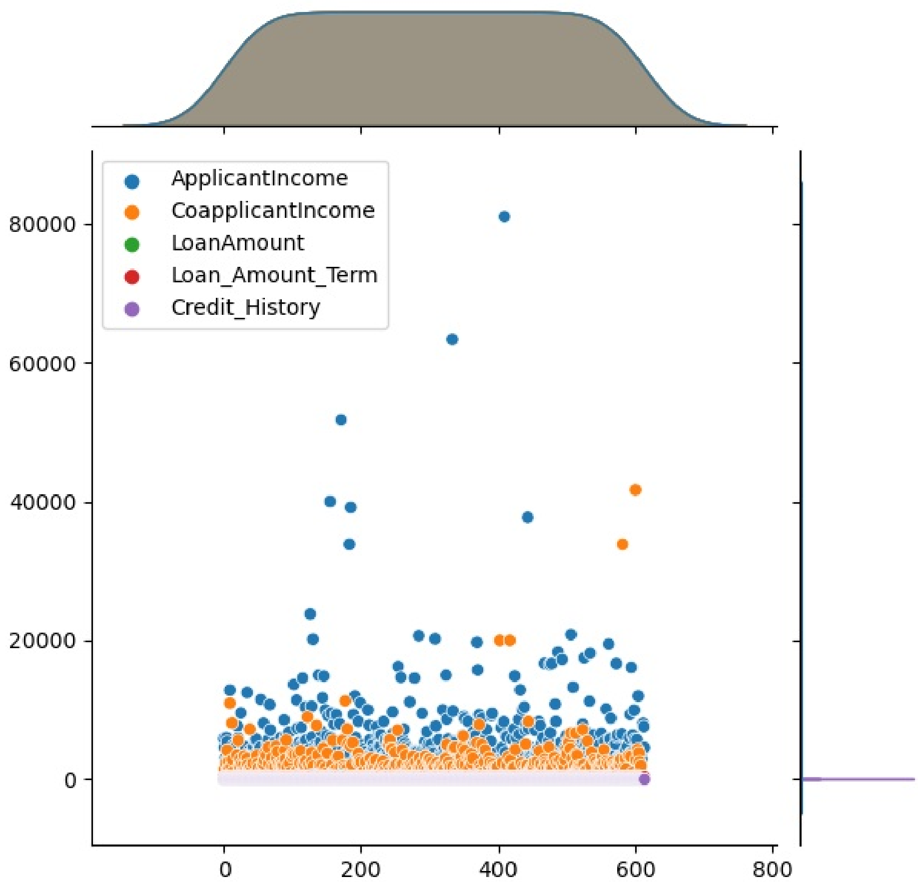
observation of heat map

- first we have to find corr() values which means pair wise relation of all the columns
- It is use whether there is any null value is column if it is present it just shows null set
- The heat describes the different colour based on the values

In [82]:

```
sns.jointplot(loan_data)
```

Out[82]: <seaborn.axisgrid.JointGrid at 0x2a90a938490>



converting the catergorical to numerical

- there are 5 ways to do
    - map
    - np.where
        - pd.getDummies
        - LabelEncoder

map

In [94]:
```
import numpy as np
import pandas as pd
file_name='C:\\Users\\Bhanu Kumar\\Desktop\\EDA\\train_ctrUa4K.csv'
loan_data=pd.read_csv(file_name)
```

In [95]:
```
loan_data['Loan_Status'].unique()
dict1={'Y':0,'N':1}
loan_data['Loan_Status'].map(dict1)
```

Out[95]:
```
0      0
1      1
2      0
3      0
4      0
      ..
609    0
610    0
611    0
612    0
613    1
Name: Loan_Status, Length: 614, dtype: int64
```

Drawback of map

- we have to create dictionary manually
- if unique value present more it is difficult to use map method

np.where

In [97]:
```
import numpy as np
import pandas as pd
file_name='C:\\Users\\Bhanu Kumar\\Desktop\\EDA\\train_ctrUa4K.csv'
loan_data=pd.read_csv(file_name)
```

In [98]:
```
cond=loan_data['Loan_Status']=='Y'

loan_data['Loan_Status']=np.where(cond,0,1)

loan_data
```

Out[98]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_T
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	36
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	36
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	36
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	36
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	36
...	...	...	...	...	...	...	...	...	...	
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	36
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	18
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	36
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	36
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	36

614 rows × 13 columns

Drawbacks of np.where

- It can change the binary value only

pd.get dummies

In [ ]:
```
import numpy as np
import pandas as pd
file_name='C:\\Users\\Bhanu Kumar\\Desktop\\EDA\\train_ctrUa4K.csv'
loan_data=pd.read_csv(file_name)
```

In [99]:
```
loan_data['Property_Area'].unique()
```

Out[99]:
```
array(['Urban', 'Rural', 'Semiurban'], dtype=object)
```

In [100…
```
pd.get_dummies(loan_data['Property_Area'])
```

Out[100]:

	Rural	Semiurban	Urban
0	0	0	1
1	1	0	0
2	0	0	1
3	0	0	1
4	0	0	1
...	...	...	...
609	1	0	0
610	1	0	0
611	0	0	1
612	0	0	1
613	0	1	0

614 rows × 3 columns

Drawback of get_dummies

- Based on the columns unique value it create new columns in the data set
- so it is good for less unique values

label encoder

In [104…
```
import numpy as np
import pandas as pd
file_name='C:\\Users\\Bhanu Kumar\\Desktop\\EDA\\train_ctrUa4K.csv'
loan_data=pd.read_csv(file_name)
```

In [105…
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
loan_data['Property_Area_new']=le.fit_transform(loan_data['Property_Area'])
```

In [106…
```
loan_data[['Property_Area_new','Property_Area']]
```

Out[106]:

	Property_Area_new	Property_Area
0	2	Urban
1	0	Rural
2	2	Urban
3	2	Urban
4	2	Urban
...	...	...
609	0	Rural
610	0	Rural
611	2	Urban
612	2	Urban
613	1	Semiurban

614 rows × 2 columns

In [108…
```
loan_data['Loan_Status']=le.fit_transform(loan_data['Loan_Status'])
```

In [111…
```
converted =loan_data['Loan_Status'].values
old=le.inverse_transform(loan_data['Loan_Status'])
```

In [115…
```
print(converted[:10])
print(old[:10])
```

```
[1 0 1 1 1 1 1 0 1 0]
['Y' 'N' 'Y' 'Y' 'Y' 'Y' 'Y' 'N' 'Y' 'N']
```

observations of label encoder

- IN the label encoder we can compelte catergorical columns
- we are using fit transform auto matic to change the value by its self
- here we can also inverse the by using inverse transform

Normalization

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

In [121…
```
import numpy as np
import pandas as pd
file_name='C:\\Users\\Bhanu Kumar\\Desktop\\EDA\\train_ctrUa4K.csv'
loan_data=pd.read_csv(file_name)
```

In [124…
```
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler()
mms.fit_transform(loan_data[['ApplicantIncome','CoapplicantIncome','LoanAmount','Loan_Amount_Term','Credit_Hist
```

Out[124]:
```
array([[0.07048856, 0.        ,        nan, 0.74358974, 1.        ],
       [0.05482993, 0.03619171, 0.17221418, 0.74358974, 1.        ],
       [0.03525046, 0.        , 0.08248915, 0.74358974, 1.        ],
       ...,
       [0.09798392, 0.00575995, 0.35311143, 0.74358974, 1.        ],
       [0.09193568, 0.        , 0.25759768, 0.74358974, 1.        ],
       [0.05482993, 0.        , 0.17945007, 0.74358974, 0.        ]])
```

In [125…
```
min1=loan_data['ApplicantIncome'].min()
max1=loan_data['ApplicantIncome'].max()
nr=loan_data['ApplicantIncome']-min1
dr=max1-min1
loan_data['ApplicantIncome']=nr/dr
```

In [126…
```
loan_data['ApplicantIncome']
```

Out[126]:
```
0      0.070489
1      0.054830
2      0.035250
3      0.030093
4      0.072356
         ...
609    0.034014
610    0.048930
611    0.097984
612    0.091936
613    0.054830
Name: ApplicantIncome, Length: 614, dtype: float64
```

Observation of normalization

- The normalization scale down the values between 0 and 1
- WE can do by using formula
- sklearn proscesssing fit tansform the values to min max by importing min max scaler

standardization

$$Z = \frac{x - \mu}{\sigma}$$

In [128…
```
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
ss.fit_transform(loan_data[['ApplicantIncome','CoapplicantIncome','LoanAmount','Loan_Amount_Term','Credit_Histo
```

Out[128]:
```
array([[ 0.07299082, -0.55448733,         nan,  0.27664167,  0.43286074],
       [-0.13441195, -0.03873155, -0.21530913,  0.27664167,  0.43286074],
       [-0.39374734, -0.55448733, -0.94032807,  0.27664167,  0.43286074],
       ...,
       [ 0.43717437, -0.47240418,  1.24642259,  0.27664167,  0.43286074],
       [ 0.35706382, -0.55448733,  0.47462824,  0.27664167,  0.43286074],
       [-0.13441195, -0.55448733, -0.15683986,  0.27664167, -2.31021182]])
```

In [129…
```
mean1=loan_data['ApplicantIncome'].mean()
std1=loan_data['ApplicantIncome'].std()
nr=loan_data['ApplicantIncome']-mean1
loan_data['ApplicantIncome']=nr/std1
```

In [130…
```
loan_data['ApplicantIncome']
```

```
In [130]: loan_data['ApplicantIncome']
```

```
Out[130]: 0      0.072931
1     -0.134302
2     -0.393427
3     -0.461686
4      0.097649
         ...
609   -0.409796
610   -0.212383
611    0.436818
612    0.356773
613   -0.134302
Name: ApplicantIncome, Length: 614, dtype: float64
```

observation of starndization

- Standardization involves transforming the features such that they have a mean of zero and a standard deviation of one

DIfference between normalization and Standardization

- Normalization is used when the data doesn't have Gaussian distribution whereas Standardization is used on data having Gaussian distribution.
- Normalization scales in a range of [0,1] or [-1,1]. Standardization is not bounded by range.
- Normalization is highly affected by outliers. Standardization is slightly affected by outliers.