EDA: exploratory data analysis

It is used by data scientists to analyze and investigate data set and summarize their main characteristic, often emplyoing data visulization methods

for prefporming any EDA firstly we have to import the packages which are requried for data frame analysis

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

After importing the packages we have to read the data file

In [8]:
```
import pandas as pd
file_name="C:\\Users\\91751\\OneDrive\\Documents\\aravind python\\test_sha.csv
pd.read_csv(file_name)
```

Out[8]:

Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	L
Male	Yes	0	Graduate	No	5720	0	
Male	Yes	1	Graduate	No	3076	1500	
Male	Yes	2	Graduate	No	5000	1800	
Male	Yes	2	Graduate	No	2340	2546	
Male	No	0	Not Graduate	No	3276	0	
...	...	...	...	...	...	...	...
Male	Yes	3+	Not Graduate	Yes	4009	1777	
Male	Yes	0	Graduate	No	4158	709	
Male	No	0	Graduate	No	3250	1993	
Male	Yes	0	Graduate	No	5000	2393	
Male	No	0	Graduate	Yes	9200	0	

columns

after reading the data we need to store it in variable

In [ ]:
```
import pandas as pd
file_name="C:\\Users\\91751\\OneDrive\\Documents\\aravind python\\test_sha.csv
pd.read_csv(file_name)
text_sha=pd.read_csv(file_name)
```

In [48]:
```
text_sha.shape

# 367 rows
# 12=colmns
```

Out[48]:
```
(367, 12)
```

to know how much rows and columns in data file we have to use (shape) fuction

we can also know the type file

In [49]:
```
type(text_sha.shape)
```

Out[49]:
```
tuple
```

we can also know the type of file

In [50]:
```
num=(367,12)
num[0],num[1]
```

Out[50]:
```
(367, 12)
```

printing in output in such a way that is much be easily understandable to others

In [54]:
```
print("number of rows are:",text_sha.shape[0])
print("numberof columns are:",text_sha.shape[1])
```

```
number of rows are: 367
numberof columns are: 12
```

If we want to know the length of the data,we can have to use the (len())function

In [55]:
```
len(text_sha)
```

Out[55]:
```
367
```

we can also know the size of the entries data file by using (size)function

In [56]:
```
text_sha.size
```

Out[56]:
```
4404
```

to know all the names of the columns we can use (column)function

In [57]:
```
text_sha.columns
```

Out[57]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area'],
      dtype='object')
```

If we want to see headling , then we have to use (head)function

In [58]:
```
text_sha.head()
```

Out[58]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coappl
0	LP001015	Male	Yes	0	Graduate	No	5720	
1	LP001022	Male	Yes	1	Graduate	No	3076	
2	LP001031	Male	Yes	2	Graduate	No	5000	
3	LP001035	Male	Yes	2	Graduate	No	2340	
4	LP001051	Male	No	0	Not Graduate	No	3276	

If we want to see end portion or tail portion of the data, then we have to use (tail)function

In [60]:
```
text_sha.tail()
```

Out[60]:

Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	L
Male	Yes	3+	Not Graduate	Yes	4009	1777	
Male	Yes	0	Graduate	No	4158	709	
Male	No	0	Graduate	No	3250	1993	
Male	Yes	0	Graduate	No	5000	2393	
Male	No	0	Graduate	Yes	9200	0	

If we provied indics in the head function ,then it will slow us the index number rows

In [61]:
```
text_sha.head(3)
```

Out[61]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coappl
0	LP001015	Male	Yes	0	Graduate	No	5720	
1	LP001022	Male	Yes	1	Graduate	No	3076	
2	LP001031	Male	Yes	2	Graduate	No	5000	

Take([]) this function shows us the list which are provided in the indicies

In [62]:
```
text_sha.take([0,1,2,3,4])
```

Out[62]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coappl
0	LP001015	Male	Yes	0	Graduate	No	5720	
1	LP001022	Male	Yes	1	Graduate	No	3076	
2	LP001031	Male	Yes	2	Graduate	No	5000	
3	LP001035	Male	Yes	2	Graduate	No	2340	
4	LP001051	Male	No	0	Not Graduate	No	3276	

Take ([]=axis=)this function provides the list of columns prvided as indices aand if the axis is 0 it will provide all columns and if the axis is 1, then it wil give only the indicsvalue columns

#[0,1,2]=======>represents-rows-or-columns

- based-on-axis
- axis=0====>rows
- axis=1===>columns

In [63]:
```
text_sha.take([0,1,2],axis=1)
```

Out[63]:

	Loan_ID	Gender	Married
0	LP001015	Male	Yes
1	LP001022	Male	Yes
2	LP001031	Male	Yes
3	LP001035	Male	Yes
4	LP001051	Male	No
...	...	...	...
362	LP002971	Male	Yes
363	LP002975	Male	Yes
364	LP002980	Male	No
365	LP002986	Male	Yes
366	LP002989	Male	No

367 rows × 3 columns

In [64]:
```
text_sha.take([5,8],axis=1).take([150,250,300])
```

Out[64]:

	Self_Employed	LoanAmount
150	No	180.0
250	No	142.0
300	Yes	75.0

In [67]:
```
dict1=text_sha.take([5,8],axis=1)
dict1.take([150,250,300])
```

Out[67]:

	Self_Employed	LoanAmount
150	No	180.0
250	No	142.0
300	Yes	75.0

In [69]:
```
text_sha.take([0,1,2])
```

Out[69]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coappl
0	LP001015	Male	Yes	0	Graduate	No	5720	
1	LP001022	Male	Yes	1	Graduate	No	3076	
2	LP001031	Male	Yes	2	Graduate	No	5000	

In [70]:
```
text_sha.take([100,200,300],axis=0)
```

Out[70]:

Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	L
Male	No	0	Graduate	No	4452	0	
Male	Yes	0	Graduate	Yes	8706	0	
Male	Yes	1	Graduate	Yes	7500	0	

In [71]:
```
text_sha.take([0,1,2],axis=1).take([1,2,3,4,5])
```

Out[71]:

	Loan_ID	Gender	Married
1	LP001022	Male	Yes
2	LP001031	Male	Yes
3	LP001035	Male	Yes
4	LP001051	Male	No
5	LP001054	Male	Yes

iloc[]is primarity interger position based (form 0 to length -1 of the axis),but mayalso be used with boolean array

In [72]:
```
text_sha.iloc[2:7,0:7]
```

Out[72]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
2	LP001031	Male	Yes	2	Graduate	No	5000
3	LP001035	Male	Yes	2	Graduate	No	2340
4	LP001051	Male	No	0	Not Graduate	No	3276
5	LP001054	Male	Yes	0	Not Graduate	Yes	2165
6	LP001055	Female	No	1	Not Graduate	No	2226

some examples of (iloc[])functions are below

In [76]:
```
text_sha.iloc[100:110,6:12]
```

Out[76]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Pro
100	4452	0	131.0	360.0	1.0	
101	2262	0	NaN	480.0	0.0	
102	3901	0	116.0	360.0	1.0	
103	2687	0	50.0	180.0	1.0	
104	2243	2233	107.0	360.0	NaN	
105	3417	1287	200.0	360.0	1.0	
106	1596	1760	119.0	360.0	0.0	
107	4513	0	120.0	360.0	1.0	
108	4500	0	140.0	360.0	1.0	
109	4523	1350	165.0	360.0	1.0	

In [78]:
```
index=[100,200,300]
text_sha.iloc[index]
```

Out[78]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coap
100	LP001534	Male	No	0	Graduate	No	4452	
200	LP002105	Male	Yes	0	Graduate	Yes	8706	
300	LP002644	Male	Yes	1	Graduate	Yes	7500	

Below we have saved rows and columns in variables

In [82]:
```
index_row=[100,200,300]              #<dataframe>.iloc[index_rows,index_col]
index_col=[4,11]
text_sha.iloc[index_row,index_col]
```

Out[82]:

	Education	Property_Area
100	Graduate	Rural
200	Graduate	Rural
300	Graduate	Urban

In [84]:
```
text_sha.iloc[[100,120,130,150,200],[5]]

# drawnback:we are counting the index of the colunms
# there are 100 columns
# it is not postive to count
```

Out[84]:

	Self_Employed
100	No
120	No
130	No
150	No
200	Yes

In [85]:
```
text_sha.columns
```

Out[85]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area'],
      dtype='object')
```

below we have used (loc[])function loc[]is primarily label basd,but may also be used with a boolena aray

- to -avoid-the-cunt-of-the-index-of-coumns-we-use-loc

In [89]:
```
columns=[]
text_sha.loc[[100,200,300],['Education']]
```

Out[89]:

	Education
100	Graduate
200	Graduate
300	Graduate

In [90]:
```
# we can also do

columns=['Education']
rows=[100,130,150,200]
text_sha.loc[rows,columns]
```

Out[90]:

	Education
100	Graduate
130	Graduate
150	Graduate
200	Graduate

In [92]:
```
text_sha.loc[[100,150,170,200],
         ['ApplicantIncome','Property_Area','Education']]
```

Out[92]:

	ApplicantIncome	Property_Area	Education
100	4452	Rural	Graduate
150	3071	Urban	Graduate
170	3958	Rural	Graduate
200	8706	Rural	Graduate

In [95]:
```
rows=[100,120,140,150,200]
columns=['ApplicantIncome','Property_Area','Education']
text_sha.loc[[100,120,140,150,200],
         ['ApplicantIncome','Property_Area','Education']]
```

Out[95]:

	ApplicantIncome	Property_Area	Education
100	4452	Rural	Graduate
120	3125	Urban	Graduate
140	4727	Rural	Graduate
150	3071	Urban	Graduate
200	8706	Rural	Graduate

Here dtypes will gives the category of all the iteams in the list

In [97]:
```
text_sha.dtypes
```

Out[97]:
```
Loan_ID               object
Gender                object
Married               object
Dependents            object
Education             object
Self_Employed         object
ApplicantIncome        int64
CoapplicantIncome      int64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area         object
dtype: object
```

In [99]:
```
type(text_sha.dtypes)
```

Out[99]:
```
pandas.core.series.Series
```

Below type casting by coverting list to dict

In [100]:
```
dict(text_sha.dtypes)
```

Out[100]:
```
{'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('int64'),
 'LoanAmount': dtype('float64'),
 'Loan_Amount_Term': dtype('float64'),
 'Credit_History': dtype('float64'),
 'Property_Area': dtype('O')}
```

Below dictionary values are in the forem of key and values

In [104]:
```
cols_dict=dict(text_sha.dtypes)
for key, value in cols_dict.items():
    print(key,'==========>',value)
```

```
Loan_ID ==========> object
Gender ==========> object
Married ==========> object
Dependents ==========> object
Education ==========> object
Self_Employed ==========> object
ApplicantIncome ==========> int64
CoapplicantIncome ==========> int64
LoanAmount ==========> float64
Loan_Amount_Term ==========> float64
Credit_History ==========> float64
Property_Area ==========> object
```

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

In [2]:
```
import pandas as pd
file_name="C:\\Users\\91751\\OneDrive\\Documents\\aravind python\\test_sha.csv
pd.read_csv(file_name)
```

Out[2]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coap
0	LP001015	Male	Yes	0	Graduate	No	5720	
1	LP001022	Male	Yes	1	Graduate	No	3076	
2	LP001031	Male	Yes	2	Graduate	No	5000	
3	LP001035	Male	Yes	2	Graduate	No	2340	
4	LP001051	Male	No	0	Not Graduate	No	3276	
...	...	...	...	...	...	...	...	
362	LP002971	Male	Yes	3+	Not Graduate	Yes	4009	
363	LP002975	Male	Yes	0	Graduate	No	4158	
364	LP002980	Male	No	0	Graduate	No	3250	
365	LP002986	Male	Yes	0	Graduate	No	5000	
366	LP002989	Male	No	0	Graduate	Yes	9200	

367 rows × 12 columns

In [ ]:
```
import pandas as pd
file_name="C:\\Users\\91751\\OneDrive\\Documents\\aravind python\\test_sha.csv
pd.read_csv(file_name)
text_sha=pd.read_csv(file_name)
```

In [6]:
```
pd.read_csv(file_name)
text_sha=pd.read_csv(file_name)

cols_dict=dict(text_sha.dtypes)
for key,value in cols_dict.items():
    if value=='O':
        print(key)
```

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

In [10]:
```
cols_dict=dict(text_sha.dtypes)
cat_list=[]
for key,value in cols_dict.items():
    if value=='O':
         cat_list.append(key)

cat_list
```

Out[10]:
```
['Loan_ID',
 'Gender',
 'Married',
 'Dependents',
 'Education',
 'Self_Employed',
 'Property_Area']
```

[info()]function will give whole info of the datafile

In [12]:
```
text_sha.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 367 entries, 0 to 366
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            367 non-null    object 
 1   Gender             356 non-null    object 
 2   Married            367 non-null    object 
 3   Dependents         357 non-null    object 
 4   Education          367 non-null    object 
 5   Self_Employed      344 non-null    object 
 6   ApplicantIncome    367 non-null    int64  
 7   CoapplicantIncome  367 non-null    int64  
 8   LoanAmount         362 non-null    float64
 9   Loan_Amount_Term   361 non-null    float64
 10  Credit_History     338 non-null    float64
 11  Property_Area      367 non-null    object 
dtypes: float64(3), int64(2), object(7)
memory usage: 34.5+ KB
```

[isnull()]it means whenever true means

- null valuee is there
- there is no value value is missed

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

Out[13]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapp
0	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	
362	False	False	False	False	False	False	False	
363	False	False	False	False	False	False	False	
364	False	False	False	False	False	False	False	
365	False	False	False	False	False	False	False	
366	False	False	False	False	False	False	False	

367 rows × 12 columns

when we want to read two columns at a time

In [15]:
```
list1=['Education','LoanAmount']
text_sha[list1]
```

Out[15]:

	Education	LoanAmount
0	Graduate	110.0
1	Graduate	126.0
2	Graduate	208.0
3	Graduate	100.0
4	Not Graduate	78.0
...	...	...
362	Not Graduate	113.0
363	Graduate	115.0
364	Graduate	126.0
365	Graduate	158.0
366	Graduate	98.0

367 rows × 2 columns

In [17]:
```
text_sha['Property_Area'].unique()
```

Out[17]:
```
array(['Urban', 'Semiurban', 'Rural'], dtype=object)
```

In [18]:
```
text_sha['Property_Area'].nunique()
```

Out[18]:
```
3
```

In [32]:
```
text_sha_values=text_sha['Property_Area'].value_counts()
```

In [33]:
```
text_sha_values
```

Out[33]:
```
Urban        140
Semiurban    116
Rural        111
Name: Property_Area, dtype: int64
```

In [34]:
```
dict(text_sha_values)
```

Out[34]:
```
{'Urban': 140, 'Semiurban': 116, 'Rural': 111}
```

In [38]:
```
#series type convert into dictionary   typecasting
text_sha_dict=dict(text_sha_values)

#keep all values in one list
# keep all values in another list
text_sha_dict
```

Out[38]:
```
{'Urban': 140, 'Semiurban': 116, 'Rural': 111}
```

foe creating a DataFrame we have to do

In [47]:
```
text_sha_dict.keys()
```

Out[47]:
```
dict_keys(['Urban', 'Semiurban', 'Rural'])
```

In [48]:
```
text_sha_dict.values()
```

Out[48]:
```
dict_values([140, 116, 111])
```

In [51]:
```
text_sha_keys=list(dict(text_sha_values).keys())
text_sha_keys
```

Out[51]:
```
['Urban', 'Semiurban', 'Rural']
```

In [52]:
```
text_sha_values=list(dict(text_sha_values).values())
text_sha_values
```

Out[52]:
```
[140, 116, 111]
```

In [53]:
```
pd.DataFrame(zip(text_sha_keys,text_sha_values))
```

Out[53]:

	0	1
0	Urban	140
1	Semiurban	116
2	Rural	111

In [55]:
```
pd.DataFrame(zip(text_sha_keys,text_sha_values),
             columns=['Property_Area','loan amount'])
```

Out[55]:

	Property_Area	loan amount
0	Urban	140
1	Semiurban	116
2	Rural	111

In [57]:
```
text_sha_df=pd.DataFrame(zip(text_sha_keys,text_sha_values),
                         columns=['Property_Area','loan amount'])
text_sha_df
```

Out[57]:

	Property_Area	loan amount
0	Urban	140
1	Semiurban	116
2	Rural	111

saving the dataframe

In [58]:
```
# save the dataframe
text_sha_df.to_csv("text_sha_info.csv")
# file extension . csv
# location= same as python file location
```

categorical columns plot

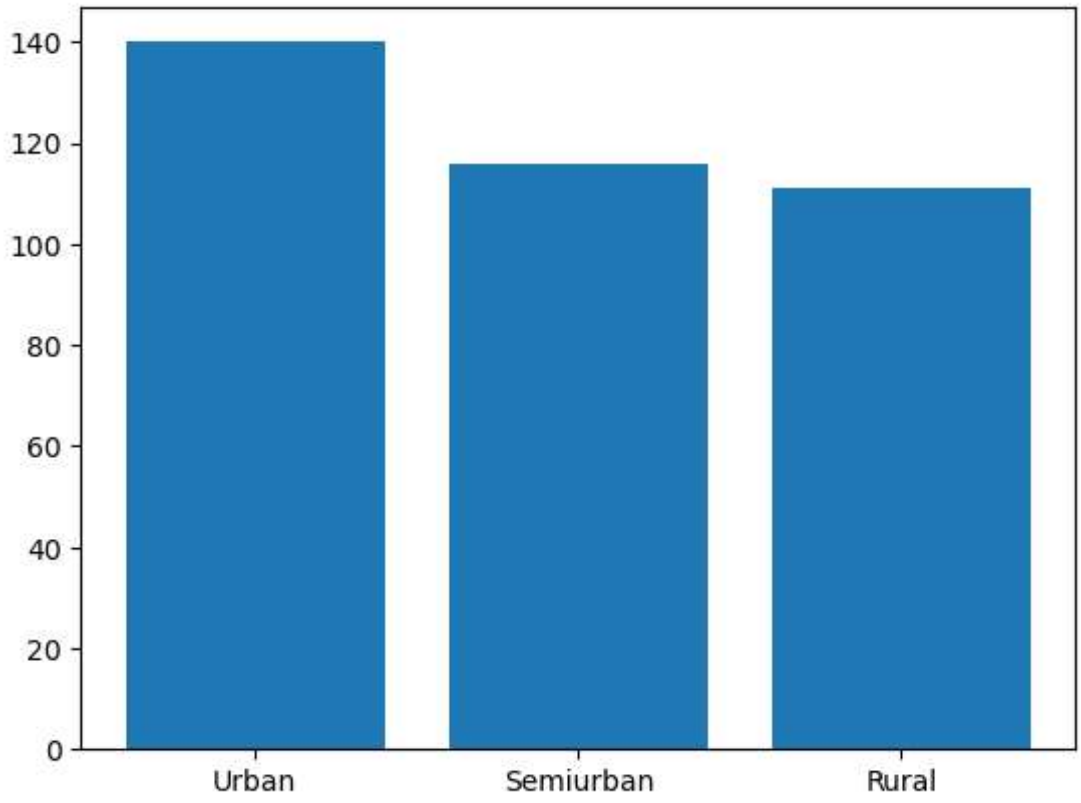
- Bar plot
- Pie chart

bar-plot

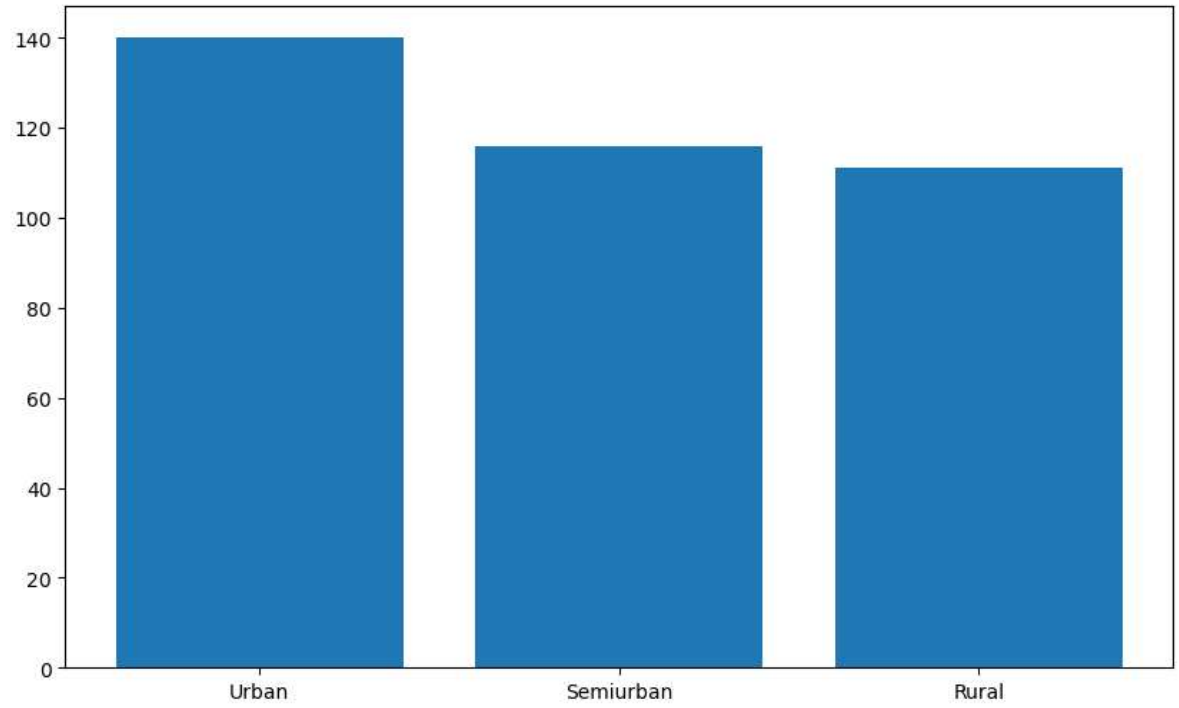
- categorical columns foem freaquncy table (data frame)
- x-axis:categorical column
- y-axis:numerical column
- data:name of the data frame

In [60]:
```
# x-axis:property_area
# y-axis:loan amount
import matplotlib.pyplot as plt

plt.bar('Property_Area','loan amount',data=text_sha_df)
```

Out[60]: <BarContainer object of 3 artists>

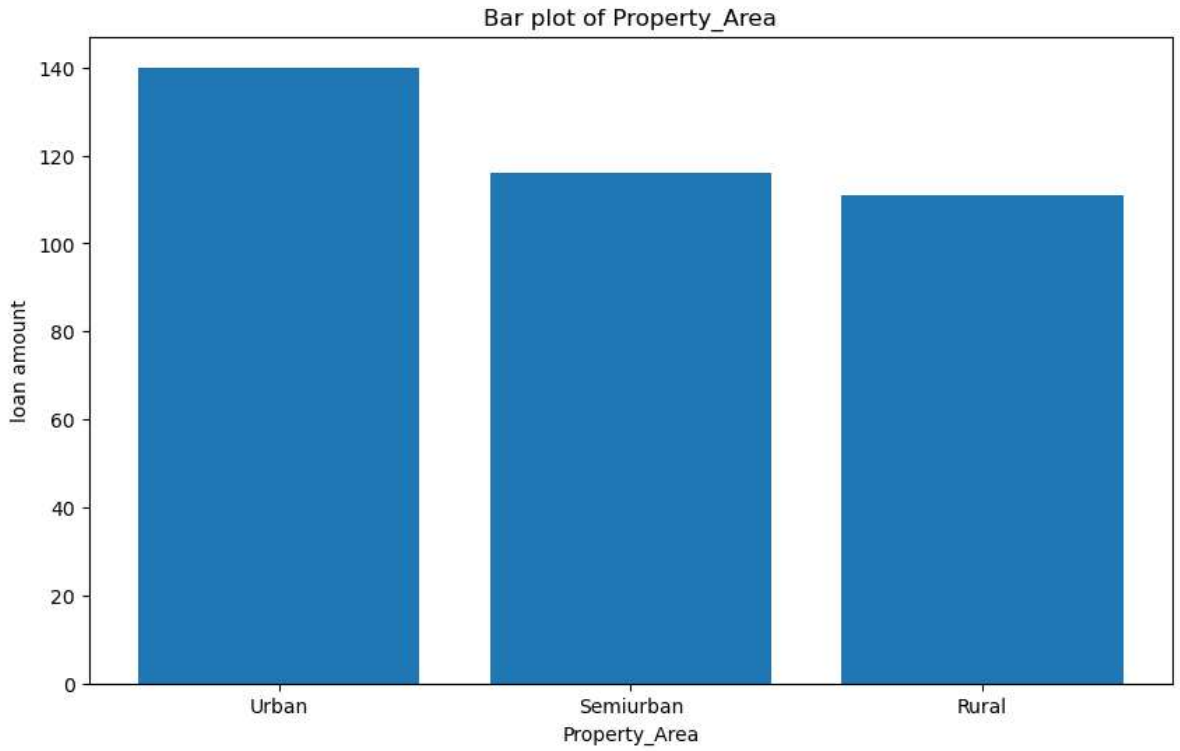


In [66]:
```
# x-axis:property_area
#y-axis:loan amount
import matplotlib.pyplot as plt

plt.figure(figsize=(10,6))   # 11 units:horizontal 5 units:vertical
plt.bar('Property_Area','loan amount',data=text_sha_df)
plt.show()
```



saving the bar plot image in the form of jpgor png

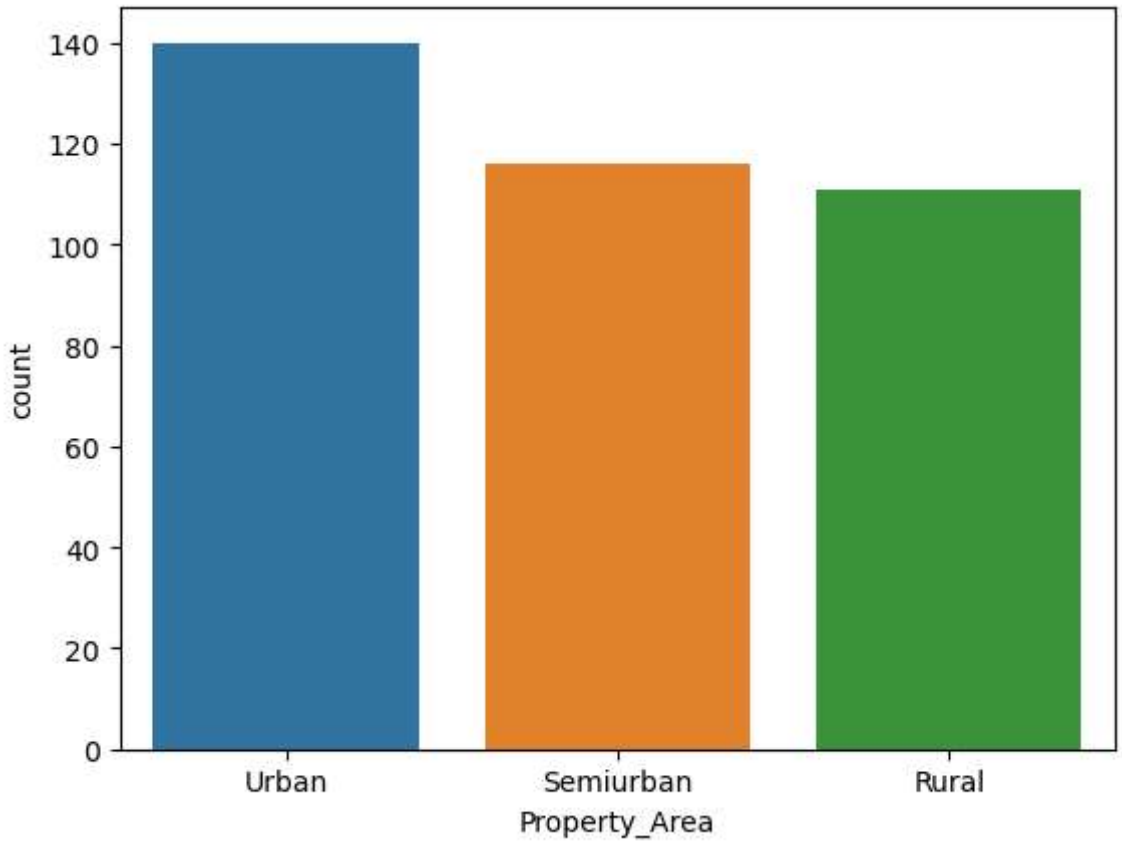
In [67]:
```
# X-axis: Property_Area
# Y- axis: loan amount
import matplotlib.pyplot as plt
plt.figure(figsize=(10,6))
plt.bar('Property_Area',
 'loan amount',
 data=text_sha_df)
plt.xlabel("Property_Area")
plt.ylabel("loan amount")
plt.title("Bar plot of Property_Area")
plt.savefig('barplot_Property_Area.png') # save image in the form of png or jp
# figure name; barplot_Property_Area
# figure extensions: png
# where it will save: where my python file existed
```



Bar plot in seaborn type

In [69]:
```
import seaborn as sns
sns.countplot(data=text_sha,x='Property_Area')
```

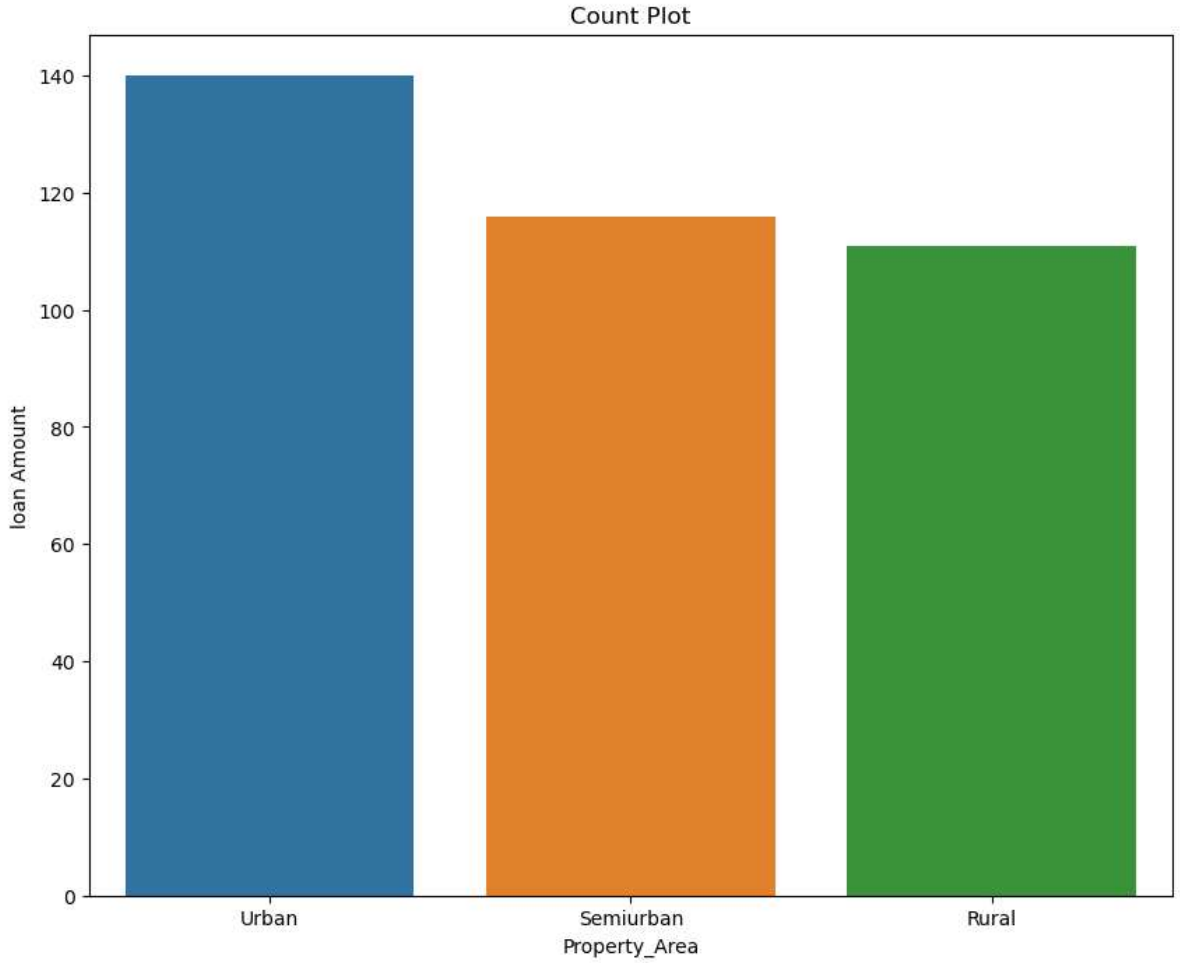
Out[69]:
```
<Axes: xlabel='Property_Area', ylabel='count'>
```



In [70]:
```
order_keys=text_sha['Property_Area'].value_counts().keys()
order_keys
```

Out[70]:
```
Index(['Urban', 'Semiurban', 'Rural'], dtype='object')
```

In [71]:
```
plt.figure(figsize=(10,8))
sns.countplot(data=text_sha,
 x='Property_Area',
 order=order_keys)
plt.xlabel('Property_Area')
plt.ylabel('loan Amount')
plt.title('Count Plot')
plt.show()
```



pie-chart

showing the data in pie chart form

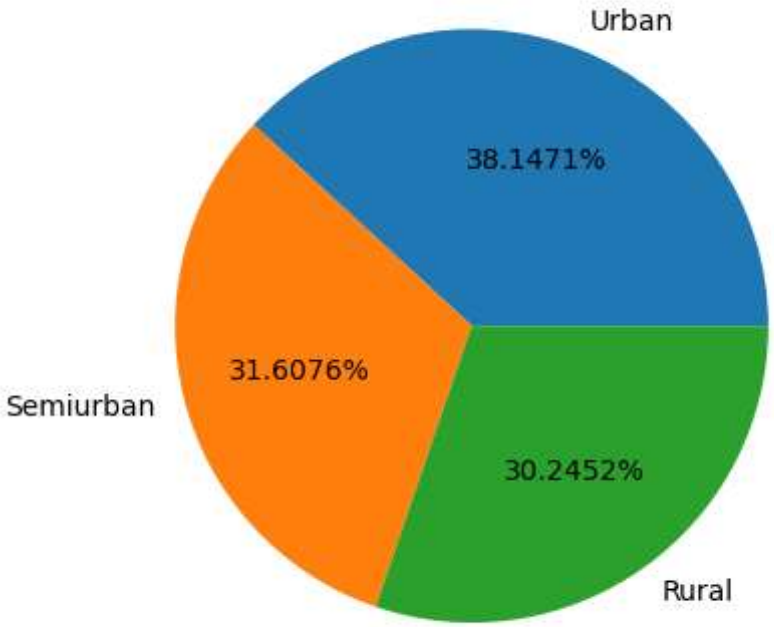
In [72]:
```
text_sha['Property_Area'].value_counts()
```

Out[72]:
```
Urban        140
Semiurban    116
Rural        111
Name: Property_Area, dtype: int64
```

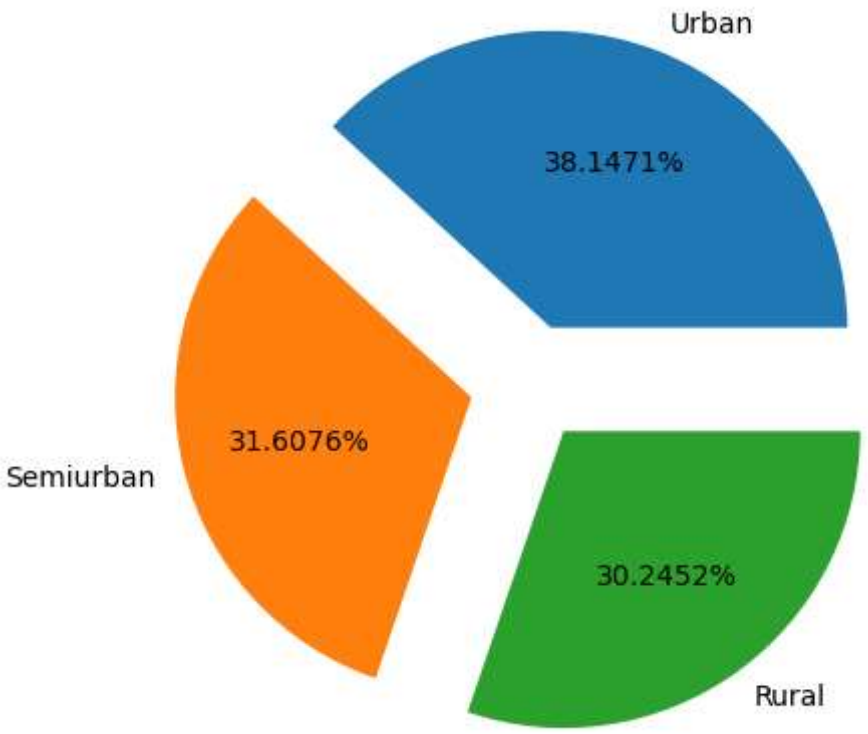
In [73]:
```
text_sha['Property_Area'].value_counts()
name=text_sha['Property_Area'].value_counts().keys()
values=text_sha['Property_Area'].value_counts().to_list()
name,values
```

Out[73]:
```
(Index(['Urban', 'Semiurban', 'Rural'], dtype='object'), [140, 116, 111])
```

In [75]:
```
plt.pie(x=values,labels=name,autopct='%0.4f%%')
plt.show()
```

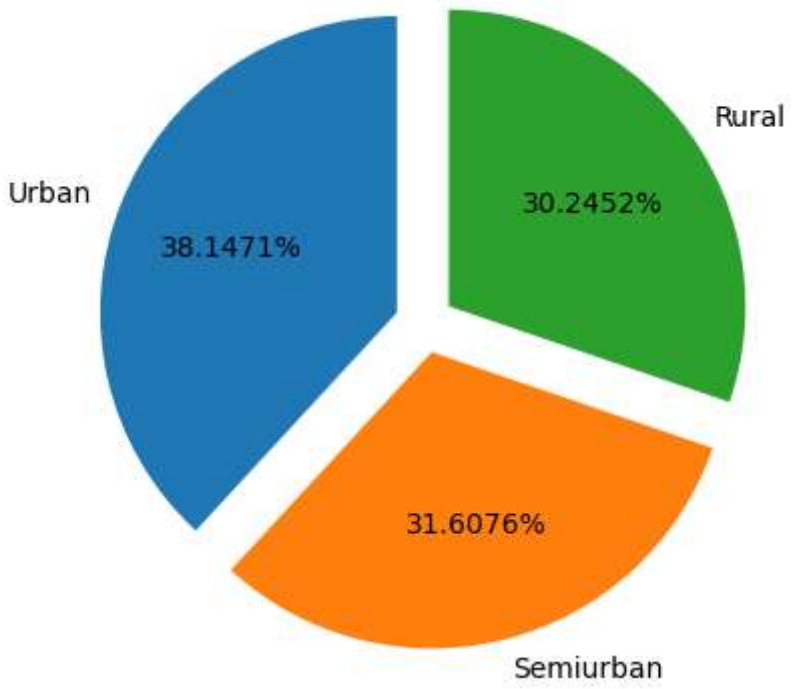


In [80]:
```
plt.pie(x=values,
        labels=name,
         autopct='%0.4f%%',
           explode=[0.2,0.2,0.2])
plt.show()
```



separated-pie-chart-90-degree-angle-rotation

In [82]:
```
plt.pie(x=values,
        labels=name,
         autopct='%0.4f%%',
          explode=[0.1,0.1,0.1],
         startangle=90)
plt.show()
```



In [83]:
```
text_sha['ApplicantIncome']
```

Out[83]:
```
0      5720
1      3076
2      5000
3      2340
4      3276
       ...
362    4009
363    4158
364    3250
365    5000
366    9200
Name: ApplicantIncome, Length: 367, dtype: int64
```

In [84]:
```
text_sha['ApplicantIncome'].mean()
```

Out[84]:
```
4805.599455040872
```

we can round up the decimal values by using round()function

In [85]:
```
round(text_sha['ApplicantIncome'].mean(),3)
```

Out[85]: 4805.599

In [86]:
```
text_sha['ApplicantIncome'].median()
```

Out[86]: 3786.0

In [87]:
```
text_sha['ApplicantIncome'].max()
```

Out[87]: 72529

In [88]:
```
text_sha['ApplicantIncome'].min()
```

Out[88]: 0

In [89]:
```
text_sha['ApplicantIncome'].std()
```

Out[89]: 4910.685398980398

In [90]:
```
text_sha['ApplicantIncome'].count()
```

Out[90]: 367

merging all the values in one dict

In [91]:
```
p_mean=round(text_sha['ApplicantIncome'].mean(),2)
p_median=round(text_sha['ApplicantIncome'].median(),2)
p_std=round(text_sha['ApplicantIncome'].std(),2)
p_max=round(text_sha['ApplicantIncome'].max(),2)
p_min=round(text_sha['ApplicantIncome'].min(),2)
p_count=round(text_sha['ApplicantIncome'].count(),2)
dict1={'mean':p_mean,
 'median':p_median,
 'std':p_std,
 'max':p_max,
 'min':p_min,
 'count':p_count}
dict1
pd.DataFrame(dict1,index=['0'])
```

Out[91]:

	mean	median	std	max	min	count
0	4805.6	3786.0	4910.69	72529	0	367

In [92]:
```
p_mean=round(text_sha['ApplicantIncome'].mean(),2)
p_median=round(text_sha['ApplicantIncome'].median(),2)
p_std=round(text_sha['ApplicantIncome'].std(),2)
p_max=round(text_sha['ApplicantIncome'].max(),2)
p_min=round(text_sha['ApplicantIncome'].min(),2)
p_count=round(text_sha['ApplicantIncome'].count(),2)
dict1={'mean':p_mean,
 'median':p_median,
 'std':p_std,
 'max':p_max,
 'min':p_min,
 'count':p_count}
dict1
pd.DataFrame(dict1,index=['values'])
```

Out[92]:

	mean	median	std	max	min	count
values	4805.6	3786.0	4910.69	72529	0	367

creating-a-data frame

In [93]:
```
list1=[p_mean,p_median,p_std,p_max,p_min,p_count]
pd.DataFrame(list1)
```

Out[93]:

	0
0	4805.60
1	3786.00
2	4910.69
3	72529.00
4	0.00
5	367.00

In [95]:
```
list1=[p_mean,p_median,p_std,p_max,p_min,p_count]
pd.DataFrame(list1,columns=['ApplicantIncome'],
             index=['mean','median','std','max','min','count'])
```

Out[95]:

	ApplicantIncome
mean	4805.60
median	3786.00
std	4910.69
max	72529.00
min	0.00
count	367.00

In [96]:
```
text_sha.describe()
```

Out[96]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	367.000000	367.000000	362.000000	361.000000	338.000000
mean	4805.599455	1569.577657	136.132597	342.537396	0.825444
std	4910.685399	2334.232099	61.366652	65.156643	0.380150
min	0.000000	0.000000	28.000000	6.000000	0.000000
25%	2864.000000	0.000000	100.250000	360.000000	1.000000
50%	3786.000000	1025.000000	125.000000	360.000000	1.000000
75%	5060.000000	2430.500000	158.000000	360.000000	1.000000
max	72529.000000	24000.000000	550.000000	480.000000	1.000000

Percentile-and-Quatiles

- percentile gives all the values in range between 1 to 100
- Quatiles gives all the values in range bewteen 25 50 75

In [98]:
```
import numpy as np
data=text_sha['ApplicantIncome']
per_25=np.percentile(data,25)
round(per_25,2)
```

Out[98]:
```
2864.0
```

In [99]:
```
data=text_sha['ApplicantIncome']
np.percentile(data,[25,50,75])
```

Out[99]:
```
array([2864., 3786., 5060.])
```

In [100]:
```
data=text_sha['ApplicantIncome']
per_25=round(np.percentile(data,25),2)
per_50=round(np.percentile(data,50),2)
per_75=round(np.percentile(data,75),2)
print(per_25,per_50,per_75)
```

```
2864.0 3786.0 5060.0
```

In [101]:
```
np.quantile(data,0.25)
```

Out[101]:
```
2864.0
```

In [102]:
```
data=text_sha['ApplicantIncome']
quan_25=round(np.quantile(data,0.25),2)
quan_50=round(np.quantile(data,0.50),2)
quan_75=round(np.quantile(data,0.75),2)
print(quan_25,quan_50,quan_75)
```

```
2864.0 3786.0 5060.0
```

In [105]:
```
dict2={}
dict2['p_mean']=round(text_sha['ApplicantIncome'].mean(),2)
dict2['p_median']=round(text_sha['ApplicantIncome'].median(),2)
dict2['p_std']=round(text_sha['ApplicantIncome'].std(),2)
dict2['p_max']=round(text_sha['ApplicantIncome'].max(),2)
dict2['p_min']=round(text_sha['ApplicantIncome'].min(),2)
dict2['p_count']=round(text_sha['ApplicantIncome'].count(),2)
dict2['per_25']=round(np.percentile(data,25),2)
dict2['per_50']=round(np.percentile(data,50),2)
dict2['per_75']=round(np.percentile(data,75),2)


pd.DataFrame(dict2,index=['ApplicantIncome'])
```

Out[105]:

	p_mean	p_median	p_std	p_max	p_min	p_count	per_25	per_50	per_75
ApplicantIncome	4805.6	3786.0	4910.69	72529	0	367	2864.0	3786.0	5060.0

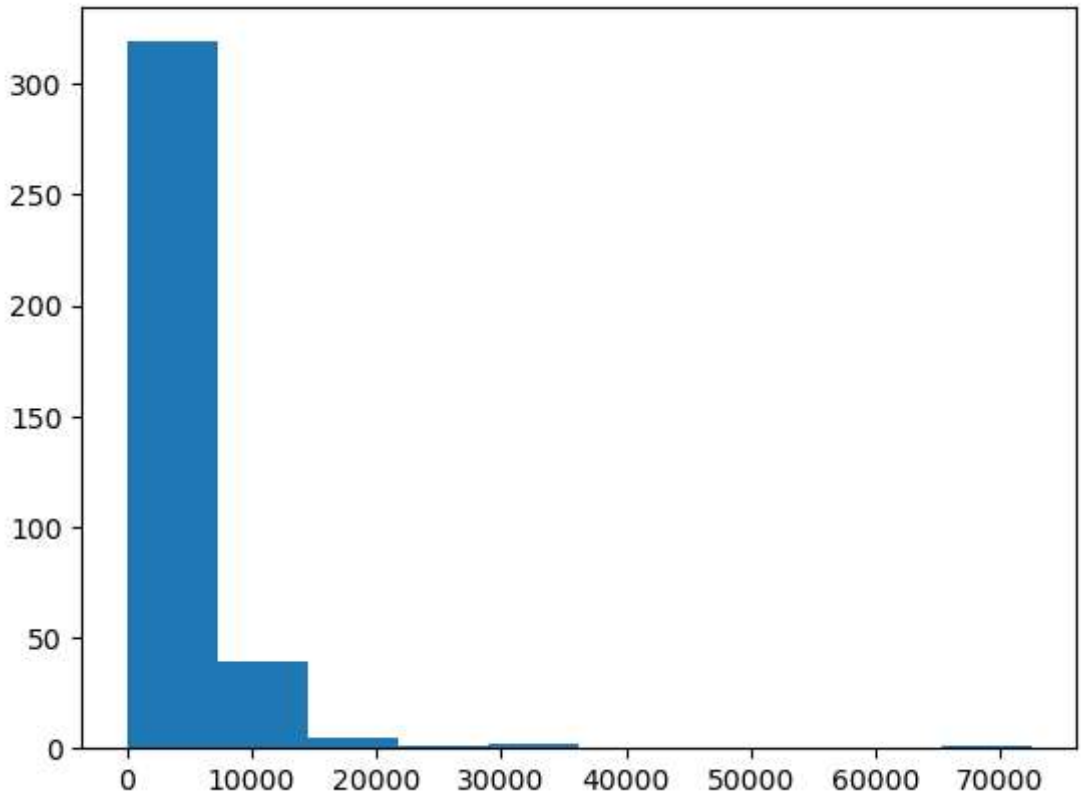
In [106]:
```
dict2.keys()
dict2.values()
pd.DataFrame(dict2.values(),columns=['AppliantIncome'],
             index=dict2.keys())
```

Out[106]:

	AppliantIncome
p_mean	4805.60
p_median	3786.00
p_std	4910.69
p_max	72529.00
p_min	0.00
p_count	367.00
per_25	2864.00
per_50	3786.00
per_75	5060.00

In [107]:
```
data=text_sha['ApplicantIncome']
plt.hist(data)
```

Out[107]:
```
(array([319.,  39.,   5.,   1.,   2.,   0.,   0.,   0.,   0.,   1.]),
 array([    0. ,  7252.9, 14505.8, 21758.7, 29011.6, 36264.5, 43517.4,
        50770.3, 58023.2, 65276.1, 72529. ]),
 <BarContainer object of 10 artists>)
```

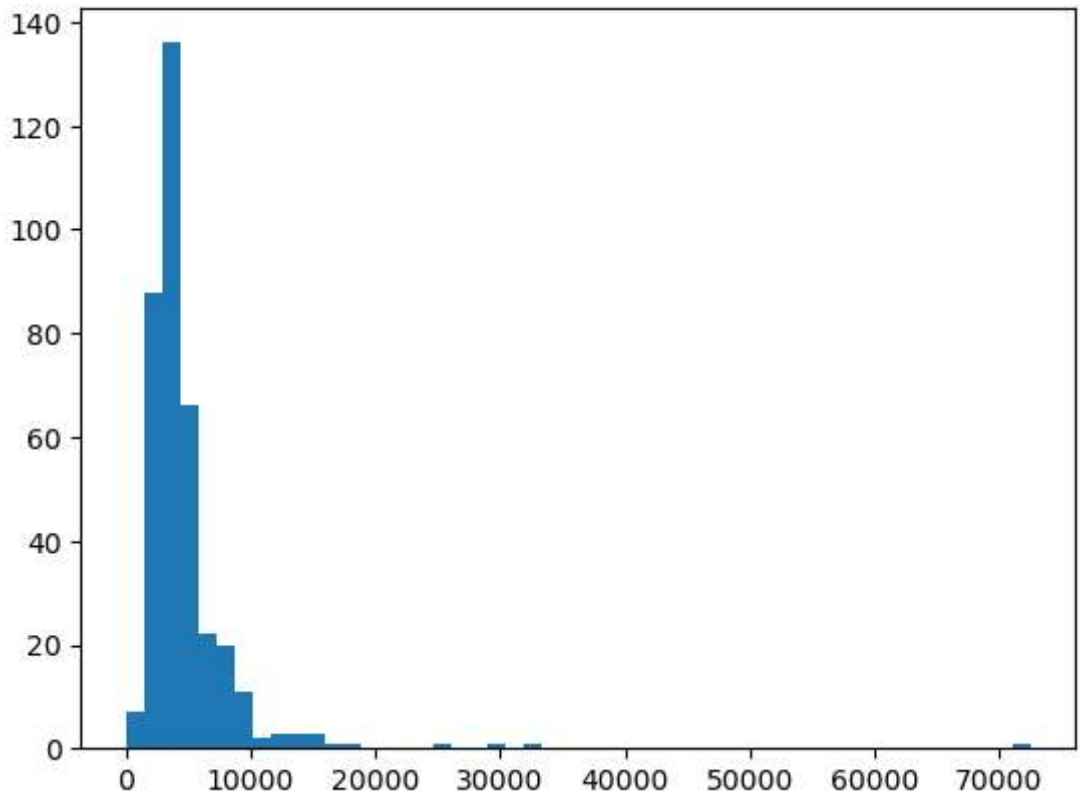


By-providing-bins-in-histrogram

In [ ]:

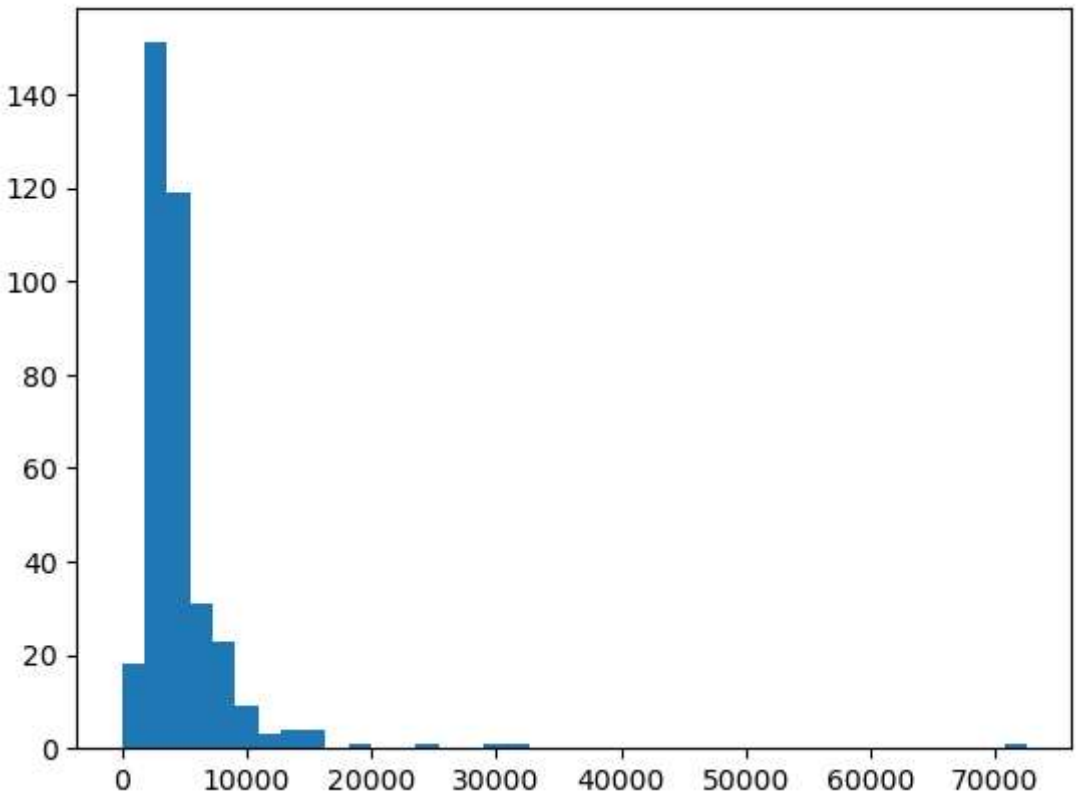
In [109]:
```
data=text_sha['ApplicantIncome']
plt.hist(data,bins=50)
```

Out[109]:
```
(array([  7.,  88., 136.,  66.,  22.,  20.,  11.,   2.,   3.,   3.,   3.,
          1.,   1.,   0.,   0.,   0.,   0.,   1.,   0.,   0.,   1.,   0.,
          1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          0.,   0.,   0.,   0.,   0.,   1.]),
 array([    0.  ,  1450.58,  2901.16,  4351.74,  5802.32,  7252.9 ,
         8703.48, 10154.06, 11604.64, 13055.22, 14505.8 , 15956.38,
        17406.96, 18857.54, 20308.12, 21758.7 , 23209.28, 24659.86,
        26110.44, 27561.02, 29011.6 , 30462.18, 31912.76, 33363.34,
        34813.92, 36264.5 , 37715.08, 39165.66, 40616.24, 42066.82,
        43517.4 , 44967.98, 46418.56, 47869.14, 49319.72, 50770.3 ,
        52220.88, 53671.46, 55122.04, 56572.62, 58023.2 , 59473.78,
        60924.36, 62374.94, 63825.52, 65276.1 , 66726.68, 68177.26,
        69627.84, 71078.42, 72529.  ]),
 <BarContainer object of 50 artists>)
```



using-frequncy,interval,n-in-hist

In [111]:
```
data=text_sha['ApplicantIncome']
frequency,interval,n=plt.hist(data,bins=40)
```



In [112]:
```
frequency
```

Out[112]:
```
array([ 18., 151., 119.,  31.,  23.,   9.,   3.,   4.,   4.,   0.,   1.,
         0.,   0.,   1.,   0.,   0.,   1.,   1.,   0.,   0.,   0.,   0.,
         0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
         0.,   0.,   0.,   0.,   0.,   0.,   1.])
```

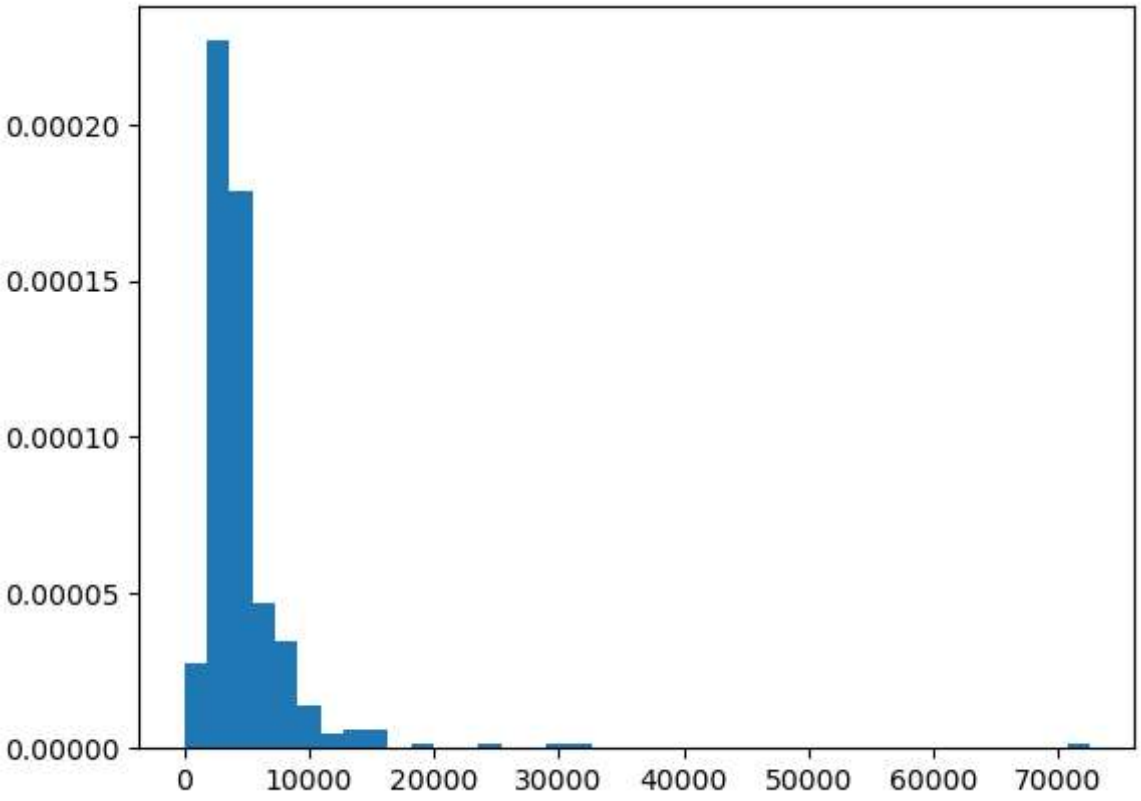
In [113]:
```
interval
```

Out[113]:
```
array([    0.   ,  1813.225,  3626.45 ,  5439.675,  7252.9  ,  9066.125,
       10879.35 , 12692.575, 14505.8  , 16319.025, 18132.25 , 19945.475,
       21758.7  , 23571.925, 25385.15 , 27198.375, 29011.6  , 30824.825,
       32638.05 , 34451.275, 36264.5  , 38077.725, 39890.95 , 41704.175,
       43517.4  , 45330.625, 47143.85 , 48957.075, 50770.3  , 52583.525,
       54396.75 , 56209.975, 58023.2  , 59836.425, 61649.65 , 63462.875,
       65276.1  , 67089.325, 68902.55 , 70715.775, 72529.   ])
```

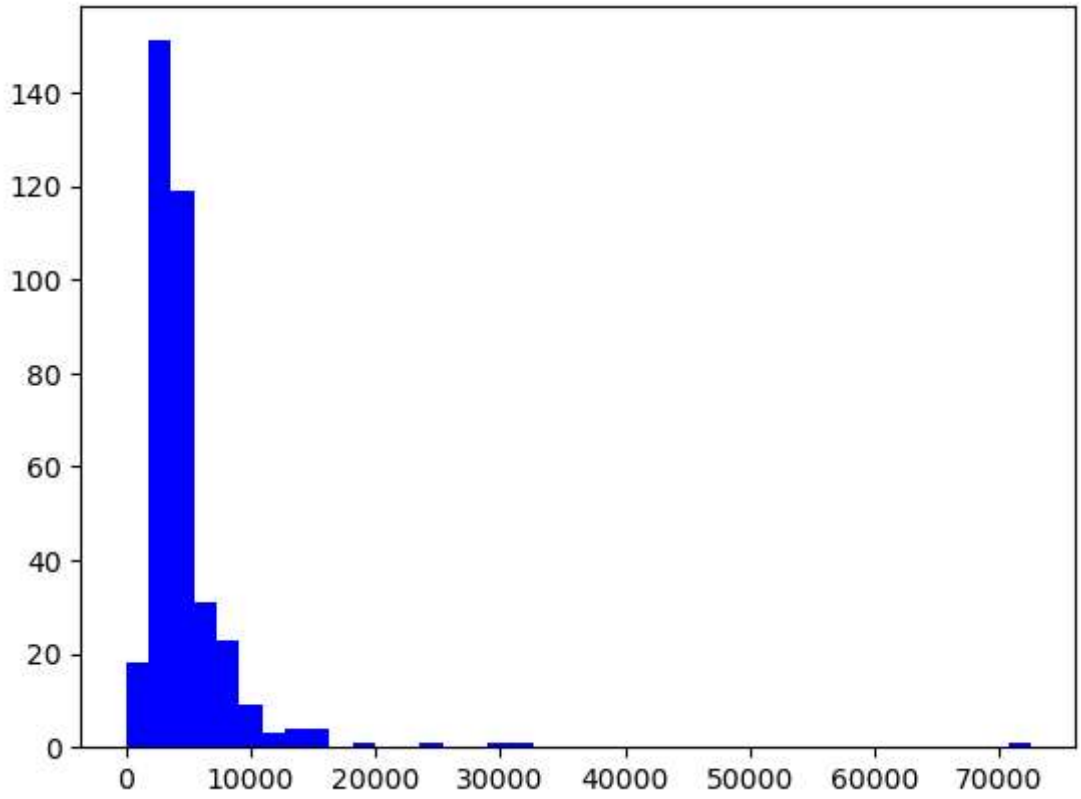
In [114]:
```
n
```

Out[114]:
```
<BarContainer object of 40 artists>
```

In [115]:
```
var=text_sha['ApplicantIncome']
plt.hist(data,bins=40,density=True)
plt.show()
```



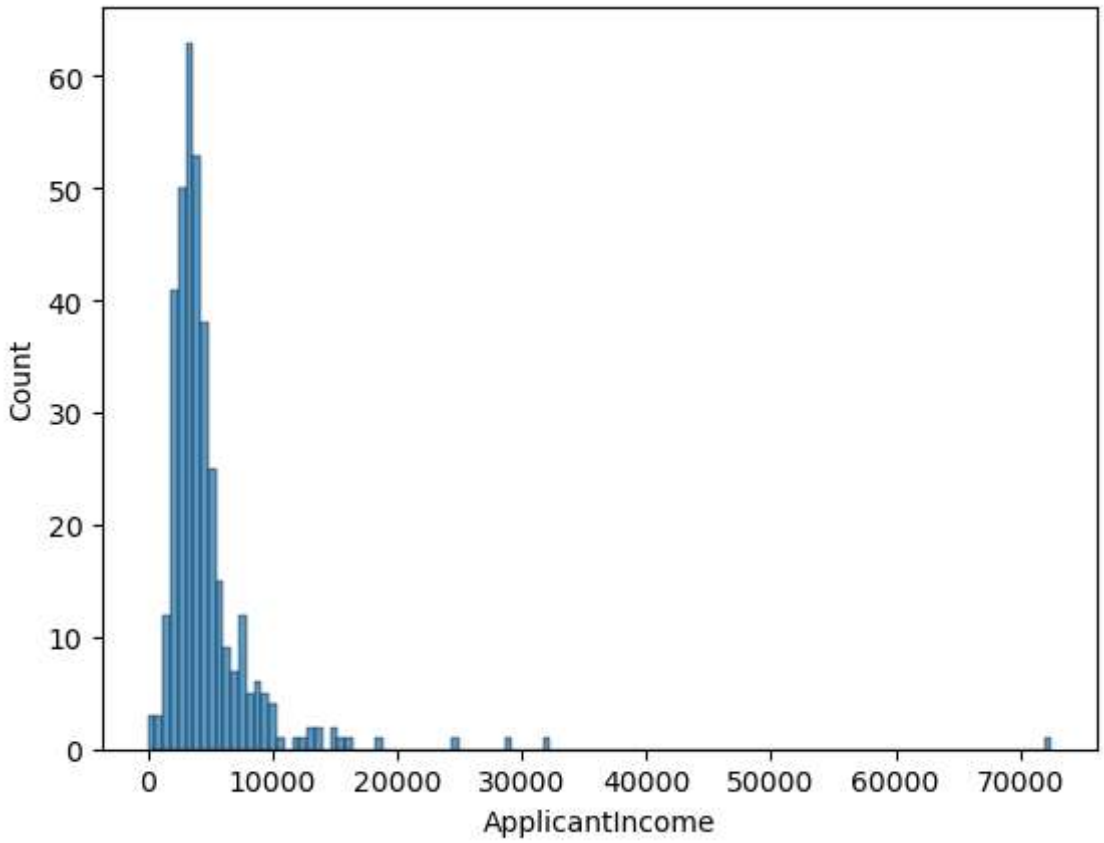
In [117]:
```
var=text_sha['ApplicantIncome']
plt.hist(data,bins=40,color='b')
plt.show()
```



Histrogram-by-seaborn-'sns'

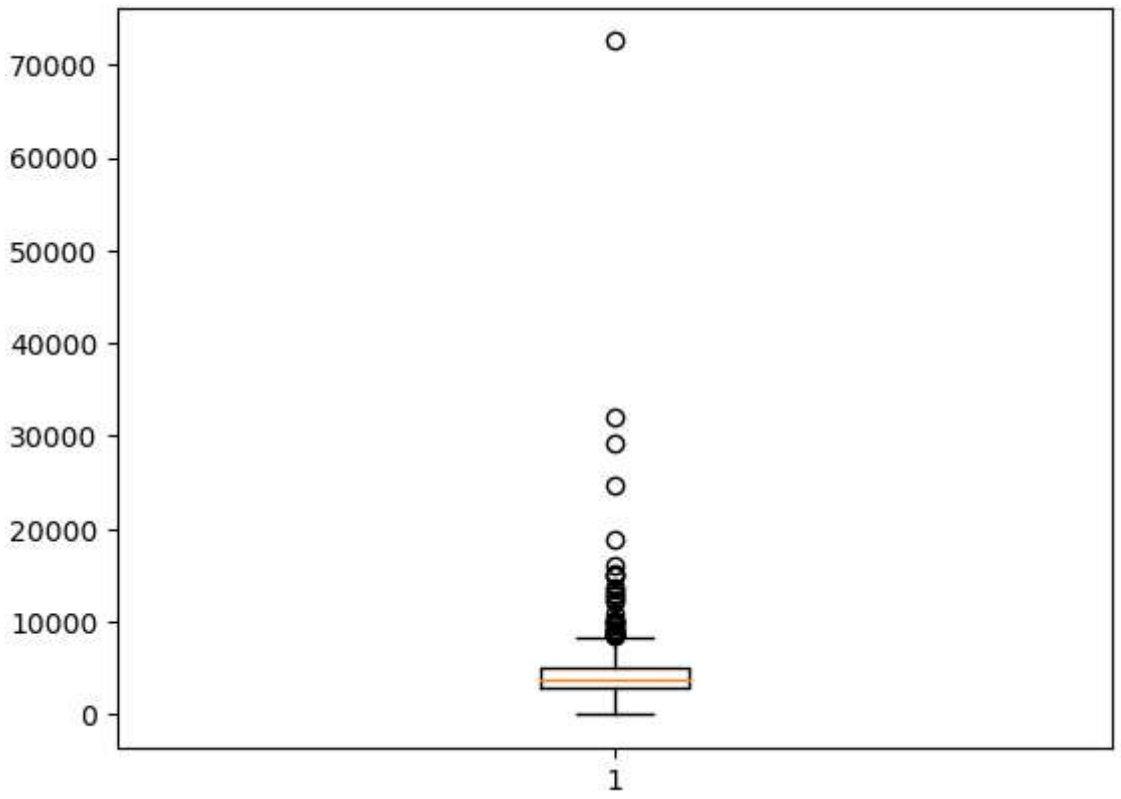
In [118]:
```
sns.histplot(data)
```

Out[118]:
```
<Axes: xlabel='ApplicantIncome', ylabel='Count'>
```



Box-plot

In [119]:
```
plt.boxplot(data)
plt.show()
```



- step-1:find Q1:25 Q2:50 Q3:75
- step-2:IQR:(Q3-Q1)
- step-3:Q1-1.5*IQR:lower bound _ step-4:Q3+1.5*IQR:upper bound

In [120]:
```
Q1=round(np.quantile(data,0.25),2)
Q2=round(np.quantile(data,0.50),2)
Q3=round(np.quantile(data,0.75),2)
IQR=round((Q3-Q1),2)
LB=round(Q1-1.5*IQR,2)
UB=round(Q3+1.5*IQR,2)
print("Q1:",Q1)
print("Q2:",Q2)
print("Q3:",Q3)
print("IQR:",IQR)
print("LB:",LB)
print("UB:",UB)
```

```
Q1: 2864.0
Q2: 3786.0
Q3: 5060.0
IQR: 2196.0
LB: -430.0
UB: 8354.0
```

outliers

In [121]:
```
# outliers are very less than LB
# outliers are greater than UB

#c1:data<lb
#c2:data>ub

data=text_sha['ApplicantIncome']
con1=data<LB
con2=data>UB

con1|con2    #it will return bool values

# in order to get the data

outliers_data=text_sha[con1|con2]
```

In [122]:
```
len(outliers_data)
```

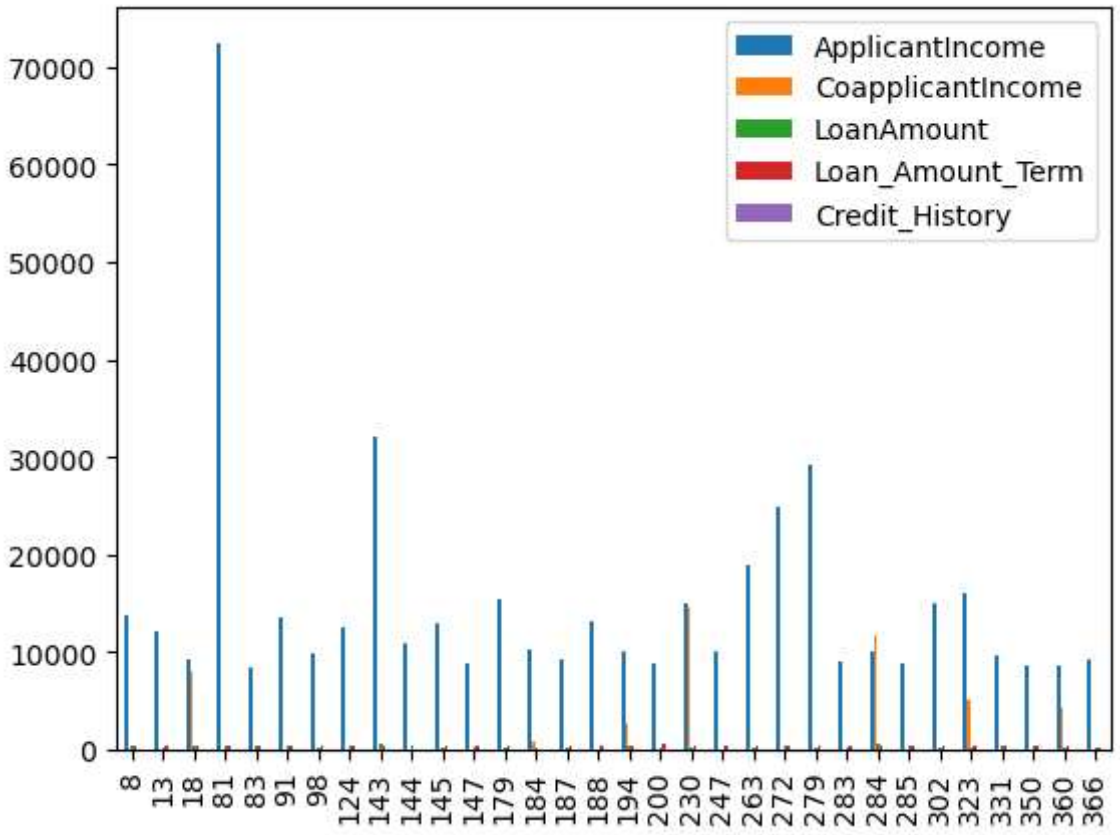
Out[122]: 32

In [123]:
```
outliers_data['ApplicantIncome'].to_list()
```

Out[123]:
```
[13633,
 12173,
 9226,
 72529,
 8449,
 13518,
 9719,
 12500,
 32000,
 10890,
 12941,
 8703,
 15312,
 10166,
 9167,
 13083,
 10000,
 8706,
 14911,
 10000,
 18840,
 24797,
 29167,
 9000,
 10000,
 8750,
 14987,
 16000,
 9699,
 8667,
 8550,
 9200]
```

In [124]:
```
outliers_data.plot(kind='bar')
```

Out[124]: <Axes: >



scatter-plot

For scatter plot we need x-axis and y-axis

In [125]:
```
x=[i for i in range(-10,10)]
y=[i*i for i in range(-10,10)]
```

In [126]:
```
x
```

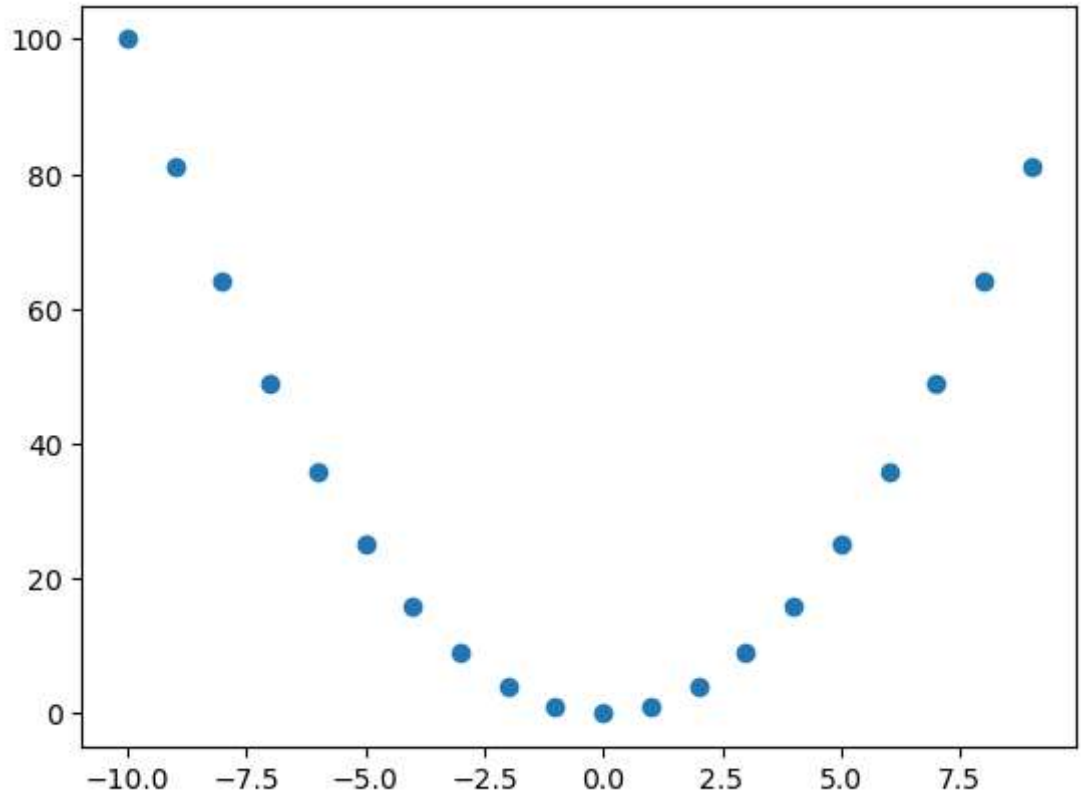
Out[126]: [-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [127]:
```
y
```

Out[127]: [100, 81, 64, 49, 36, 25, 16, 9, 4, 1, 0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

In [128]:
```
plt.scatter(x,y)
```

Out[128]:
```
<matplotlib.collections.PathCollection at 0x2723a7c50f0>
```

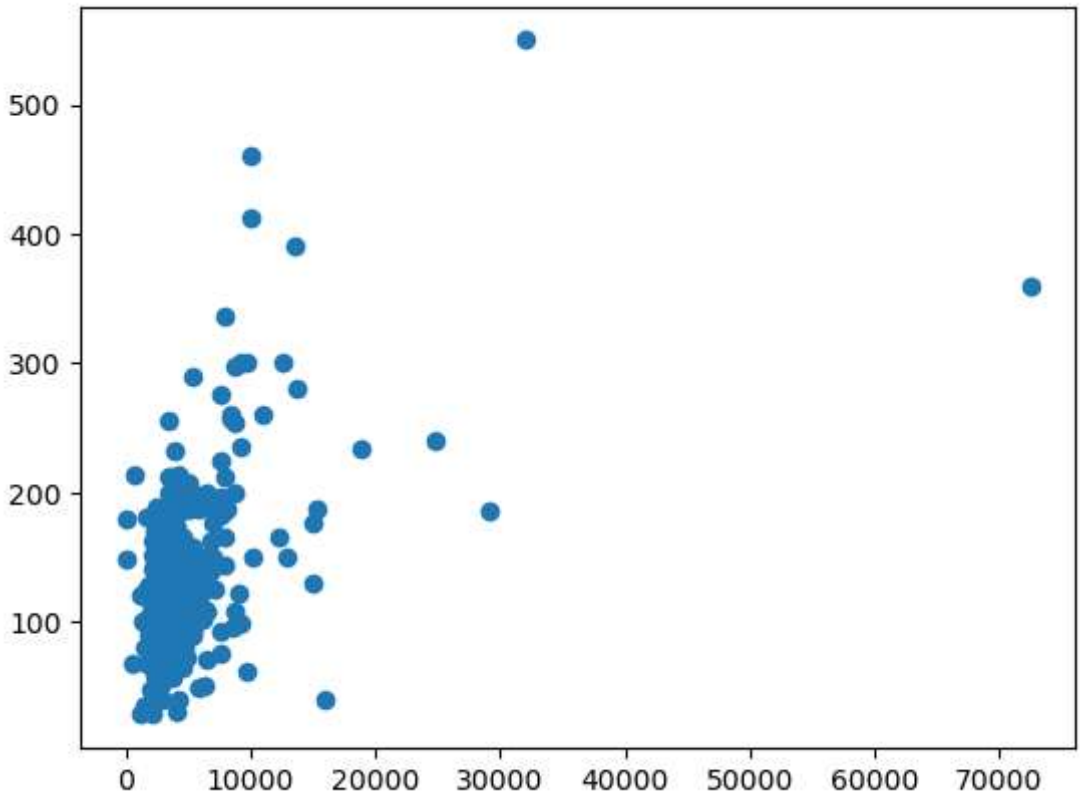


In [130]:
```
num_cols=[]
for key,value in dict(text_sha.dtypes).items():
    if value!='o':
        num_cols.append(key)
num_cols
```

Out[130]:
```
['Loan_ID',
 'Gender',
 'Married',
 'Dependents',
 'Education',
 'Self_Employed',
 'ApplicantIncome',
 'CoapplicantIncome',
 'LoanAmount',
 'Loan_Amount_Term',
 'Credit_History',
 'Property_Area']
```

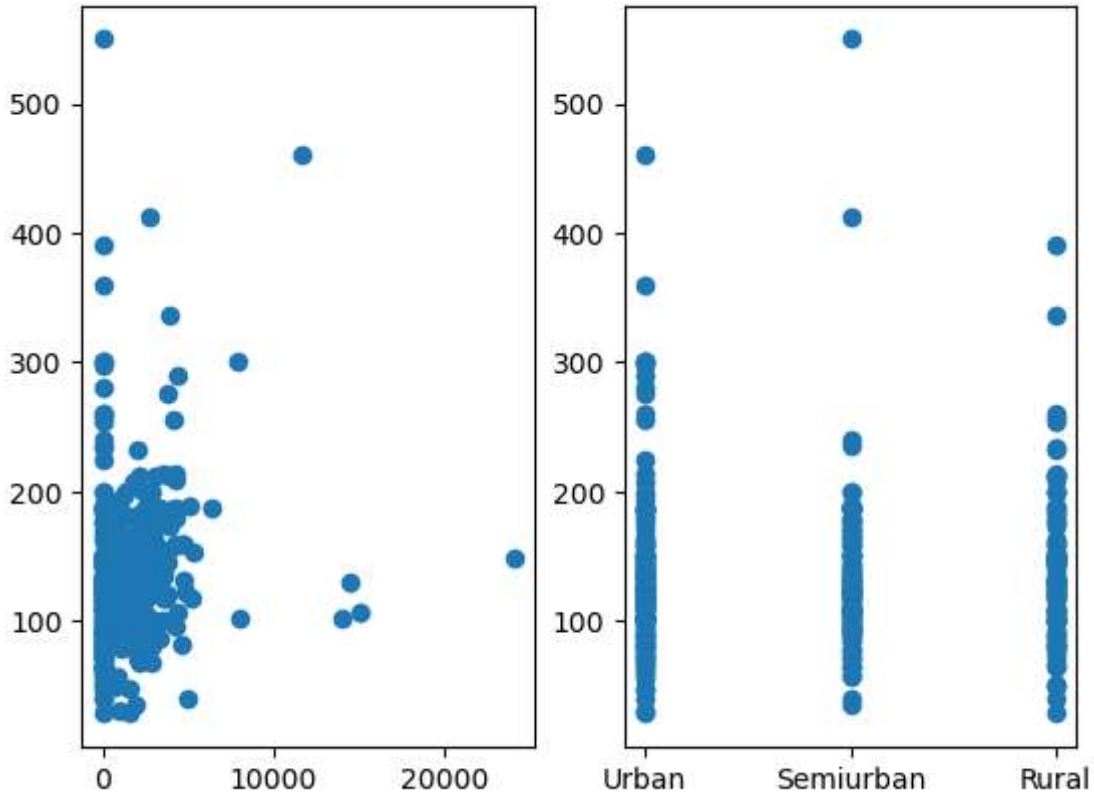
In [146]:
```
col1=text_sha['ApplicantIncome']
col2=text_sha['LoanAmount']
plt.scatter(col1,col2)
```

Out[146]: <matplotlib.collections.PathCollection at 0x27241fe37f0>



In [147]:
```
plt.subplot(1,2,1)
column1=text_sha['CoapplicantIncome']
column2=text_sha['LoanAmount']
plt.scatter(column1,column2)
plt.subplot(1,2,2)
col1=text_sha['Property_Area']
col2=text_sha['LoanAmount']
plt.scatter(col1,col2)
```

Out[147]:
```
<matplotlib.collections.PathCollection at 0x27241fefc70>
```



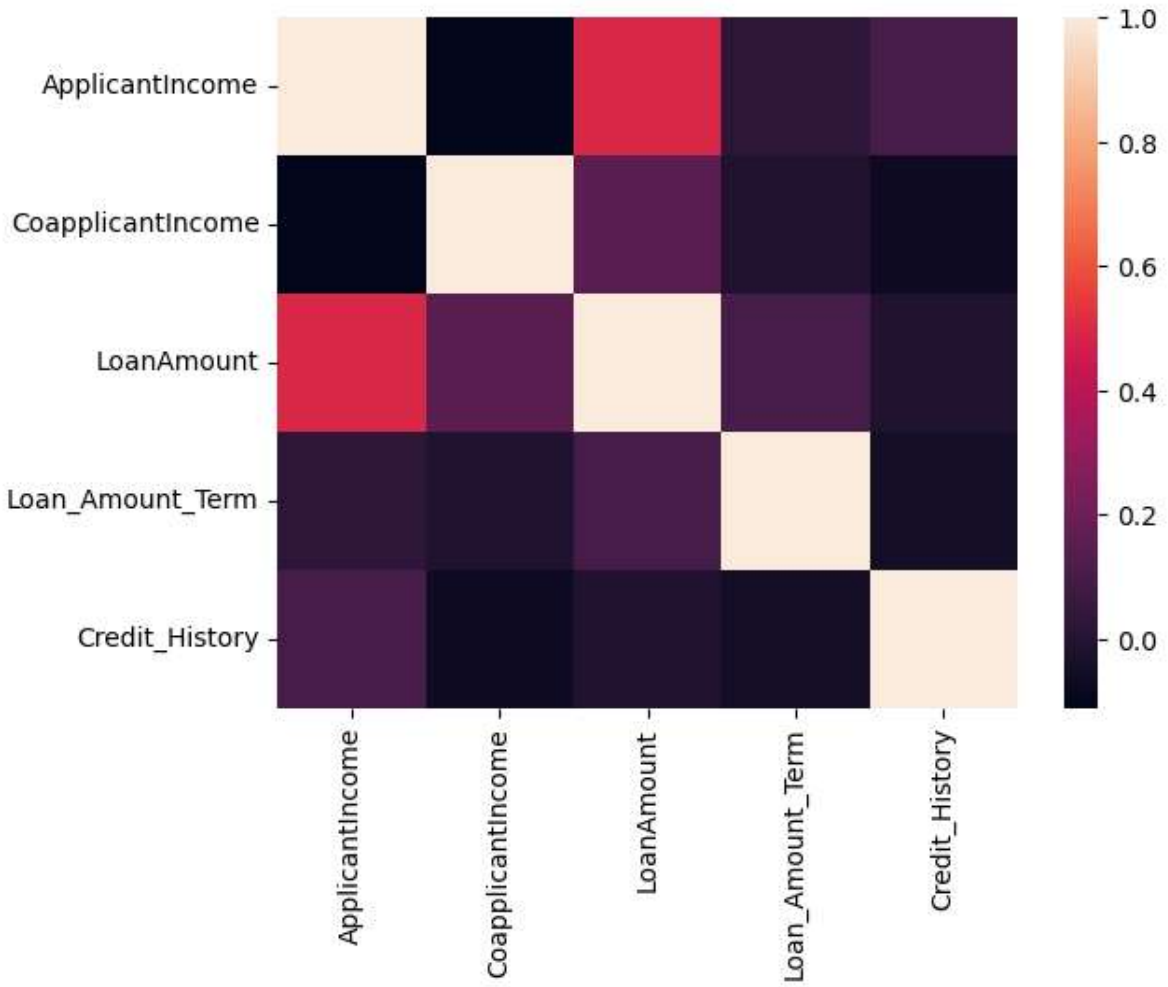
HeatMap

In [148]:
```
corr_values=text_sha.corr()
```

```
C:\Users\91751\AppData\Local\Temp\ipykernel_7364\3405525054.py:1: FutureWarni
ng: The default value of numeric_only in DataFrame.corr is deprecated. In a f
uture version, it will default to False. Select only valid columns or specify
the value of numeric_only to silence this warning.
  corr_values=text_sha.corr()
```

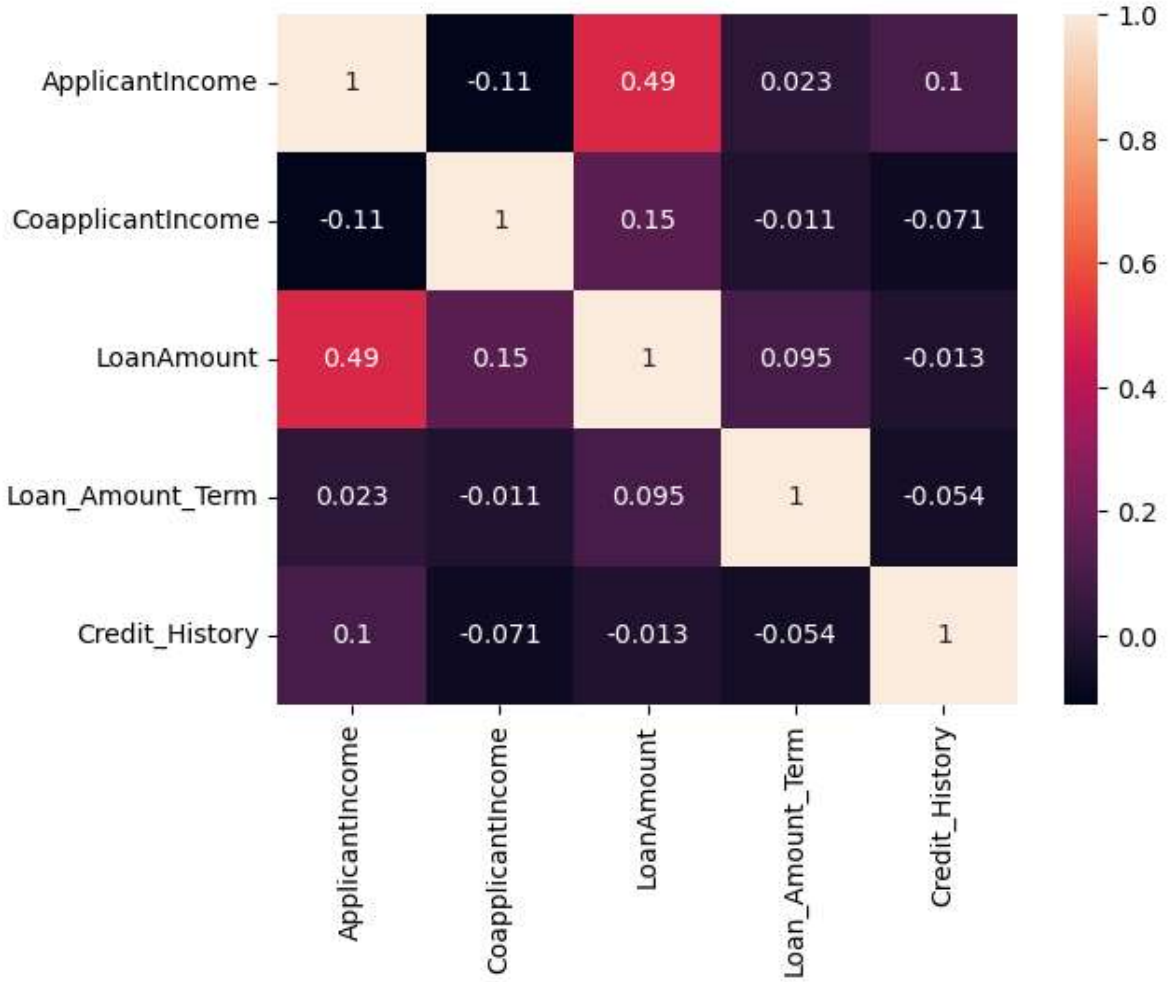
In [149]:
```
sns.heatmap(corr_values)
```

Out[149]: <Axes: >



In [151]:
```
sns.heatmap(corr_values,annot=True)
```

Out[151]: <Axes: >



In [152]:
```
corr_values.values
```

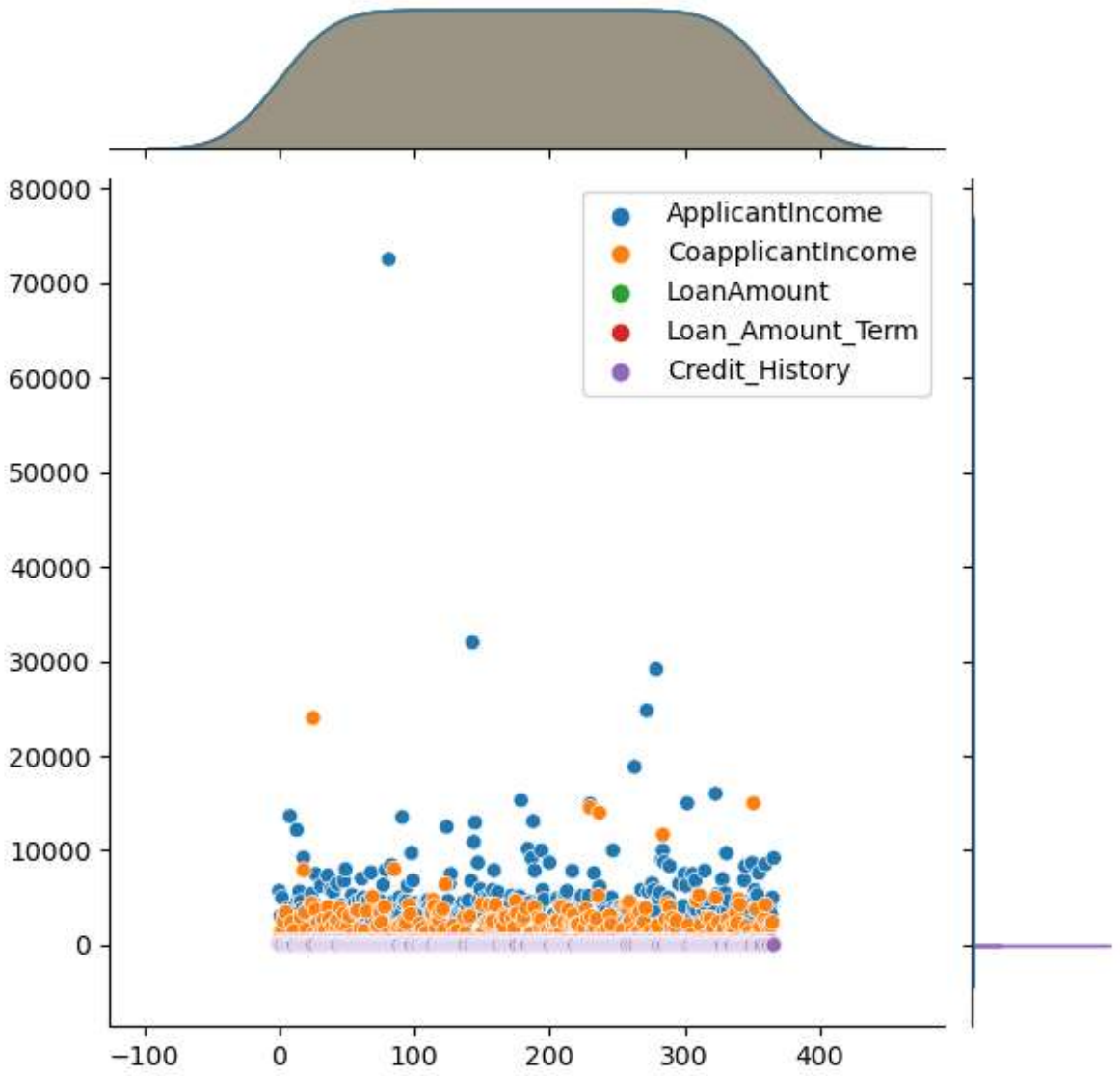
Out[152]:
```
array([[ 1.        , -0.1103348 ,  0.49345058,  0.02324869,  0.09955297],
       [-0.1103348 ,  1.        ,  0.15085   , -0.01098418, -0.07123545],
       [ 0.49345058,  0.15085   ,  1.        ,  0.09495046, -0.01257804],
       [ 0.02324869, -0.01098418,  0.09495046,  1.        , -0.05359287],
       [ 0.09955297, -0.07123545, -0.01257804, -0.05359287,  1.        ]])
```

Joint-Plot

- it is also one of the last type of plotting the data

In [153]:
```
columns1=text_sha['ApplicantIncome']
columns2=text_sha['LoanAmount']
sns.jointplot(text_sha)
```

Out[153]: <seaborn.axisgrid.JointGrid at 0x272420b7f40>



In [ ]:

In [ ]: