1_Measure of Central Tendency

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

Mean, Media, Mode are often used in data cleaning

In [3]:
```
dataset = pd.read_csv("titanic.csv")
```

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

Out[4]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
0	0	3	Mr. Owen Harris Braund	male	22.0	1	0	7.2500
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833
2	1	3	Miss. Laina Heikkinen	female	26.0	0	0	7.9250

In [42]:
```
dataset["Age"]
```

Out[42]:
```
0      22.0
1      38.0
2      26.0
3      35.0
4      35.0
       ... 
882    27.0
883    19.0
884     7.0
885    26.0
886    32.0
Name: Age, Length: 887, dtype: float64
```

Find Median

To remove null values in age column

In [5]:
```
# In order to see how many null entries are present in all columns
dataset.isnull().sum()
```

Out[5]:
```
Survived                   0
Pclass                     0
Name                       0
Sex                        0
Age                        0
Siblings/Spouses Aboard    0
Parents/Children Aboard    0
Fare                       0
dtype: int64
```

In [6]:
```
# There are no null values above, in case there are null values we can remove them
dataset["Age"].fillna(dataset["Age"].mean(), inplace=True)
```

In [7]:
```
np.median(dataset["Age"])
```

Out[7]:
```
28.0
```

Find Mean

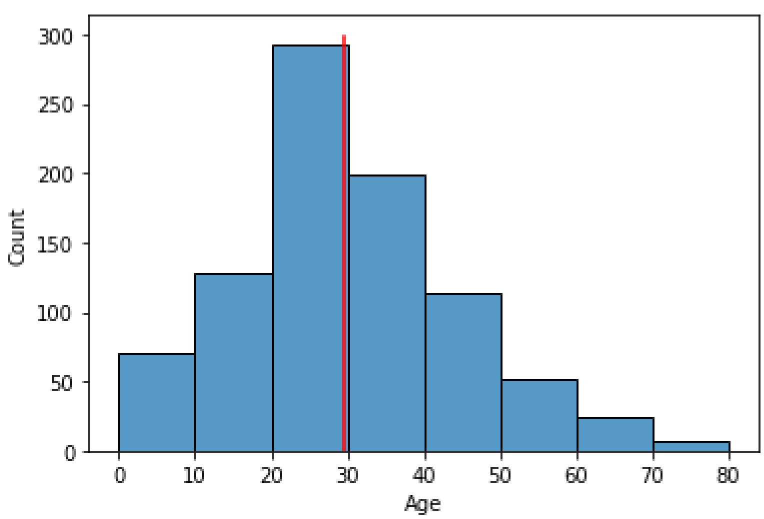
In [8]:
```
dataset["Age"].mean()
```

Out[8]:
```
29.471443066516347
```

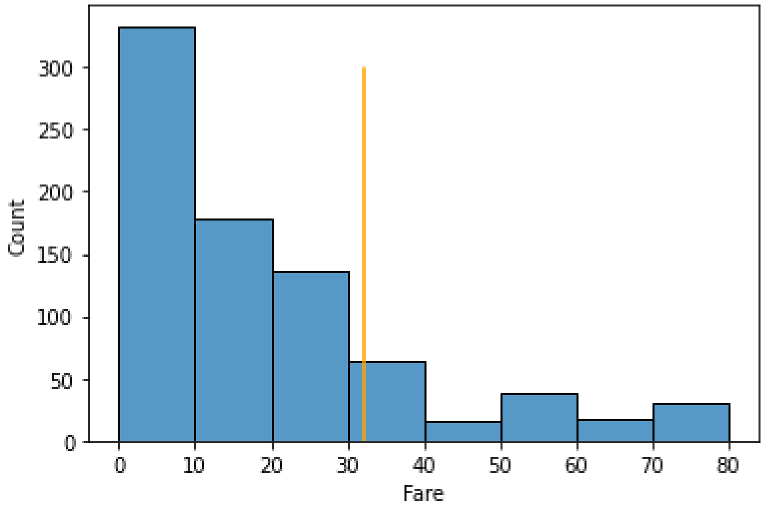
In [13]:
```
mn = np.mean(dataset["Age"])
md = np.mean(dataset["Fare"])
md
```

Out[13]:
```
32.30542018038331
```

In [10]:
```
sns.histplot(x="Age", data=dataset, bins= [i for i in range(0,81,10)])
plt.plot([mn for i in range(0,300)],[i for i in range(0,300)], c="red")
plt.show()
```



In [49]:
```
sns.histplot(x="Fare", data=dataset, bins =[i for i in range(0,81,10)])
plt.plot([md for i in range(0,300)],[i for i in range(0,300)], c="orange")
plt.show()
```



Finding Mode

In [21]:
```
dataset["Fare"].mode()
```

Out[21]:
```
0    8.05
Name: Fare, dtype: float64
```

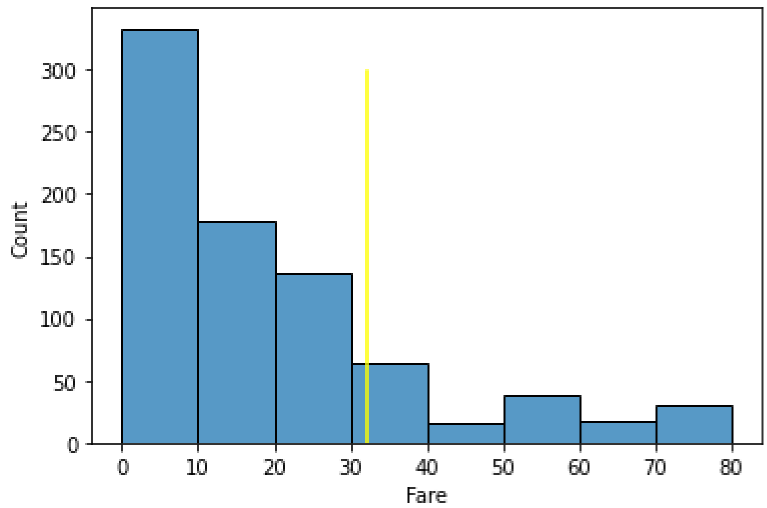
In [27]:
```
mo = dataset["Fare"].mode()[0]
mo
```

Out[27]:
```
8.05
```

In [28]:
```
# To determine the frequency of fare
dataset["Fare"].value_counts()
```

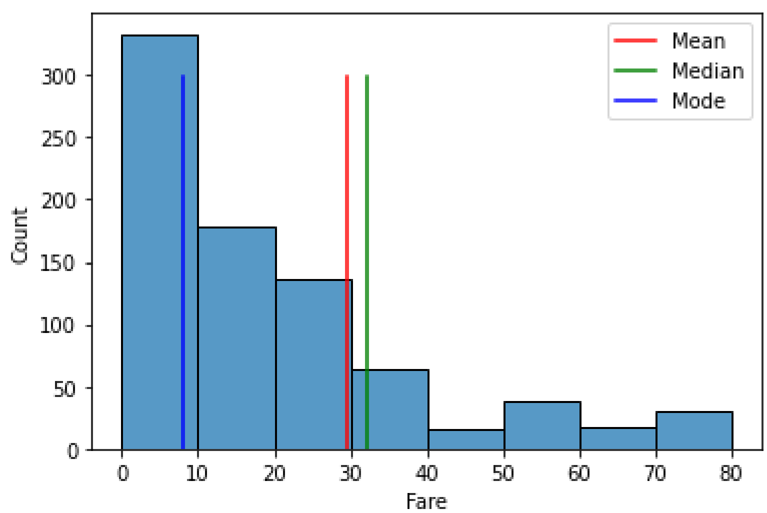
Out[28]:
```
8.0500     43
13.0000    42
7.8958     36
7.7500     33
26.0000    31
           ..
35.0000     1
28.5000     1
6.2375      1
14.0000     1
10.5167     1
Name: Fare, Length: 248, dtype: int64
```

In [48]:
```
# to plot the mode of Fare ind dataset
sns.histplot(x="Fare", data=dataset, bins=[i for i in range(0,81,10)])
plt.plot([md for i in range(0,300)], [i for i in range(0,300)], c="yellow")
plt.show()
```



To show all variables in one plot

In [56]:
```
sns.histplot(x="Fare", data=dataset, bins=[i for i in range(0,81,10)])
plt.plot([mn for i in range(0,300)], [i for i in range(0,300)], c="red", label="Mea
plt.plot([md for i in range(0,300)], [i for i in range(0,300)], c="green", label="M
plt.plot([mo for i in range(0,300)], [i for i in range(0,300)], c="blue", label="Mo
plt.legend()
plt.show()
```



In [ ]:

In [ ]:

2. Measure of Variability

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

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

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

Out[4]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
0	0	3	Mr. Owen Harris Braund	male	22.0	1	0	7.2500
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833
2	1	3	Miss. Laina Heikkinen	female	26.0	0	0	7.9250

2.1 Range

In [8]:
```
min_r = dataset['Age'].min()
max_r = dataset['Age'].max()
```

In [9]:
```
min_r, max_r
```

Out[9]:
```
(0.42, 80.0)
```

In [10]:
```
range = max_r - min_r
```

In [11]:
```
range
```

Out[11]:
```
79.58
```

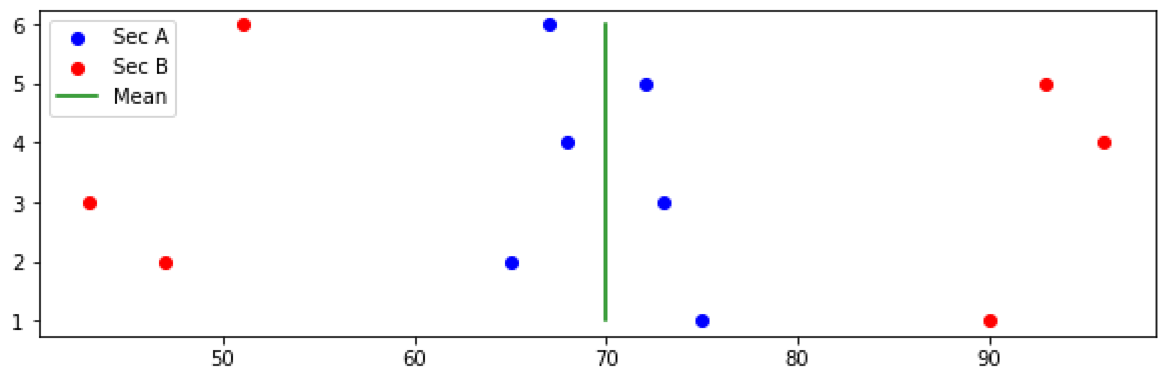
2.2 Mean Absolute Division

To simply print graph

In [23]:
```
sec_a = np.array([75,65,73,68,72,67])
sec_b = np.array([90,47,43,96,93,51])
ne = np.array([1,2,3,4,5,6])
```

In [36]:
```
mean = np.mean(sec_a)
```

In [42]:
```
plt.figure(figsize=(10,3))
plt.scatter(sec_a, ne, color="blue", label="Sec A")
plt.scatter(sec_b, ne, color="red", label="Sec B")
plt.plot([70,70,70,70,70,70], ne, c="green", label="Mean")
#plt.plot([mean for i in range(1,7)], ne, c="green", label="Mean")
plt.legend()
plt.show()
```



To use MAD formula

In [44]:
```
# To calculate xi-x
sec_b - mean
```

Out[44]:
```
array([ 5., -5.,  3., -2.,  2., -3.])
```

In [48]:
```
# To calculate |xi-x|
np.abs(sec_a -mean)
```

Out[48]:
```
array([5., 5., 3., 2., 2., 3.])
```

In [49]:
```
# To calculate sigma|xi-x|
np.sum(np.abs(sec_a - mean))
```

Out[49]:
```
20.0
```

In [51]:
```
# To calculate sigma|xi-x|/n
mad_sec_a = np.sum(np.abs(sec_a - mean))/len(sec_a)
```

In [52]:
```
# Likewise we will calculat mean absolute division of sec_b
mad_sec_b = np.sum(np.abs(sec_b - mean))/len(sec_b)
```

In [53]:
```
mad_sec_a, mad_sec_b
```

Out[53]:
```
(3.3333333333333335, 23.0)
```

So you will take sec_a for machine learning model as it contains low mean absolute division

2.3 Calculate Standard Deviation and Variance

In [55]:
```
# To calculate standard deviation of data of section A and section B
np.std(sec_a), np.std(sec_b)
```

Out[55]: `(3.559026084010437, 23.18045153428495)`

In [56]:
```
# To calculate variance of data of section A and section B
np.var(sec_a), np.var(sec_b)
```

Out[56]: `(12.666666666666666, 537.3333333333334)`

So We will take data of section A becuase it has low variance as well as less standard deviation

To calculate std and var on real world data

In [58]:
```
dataset = pd.read_csv('titanic.csv')
```

In [60]:
```
dataset.head(3)
```

Out[60]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
0	0	3	Mr. Owen Harris Braund	male	22.0	1	0	7.2500
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833
2	1	3	Miss. Laina Heikkinen	female	26.0	0	0	7.9250

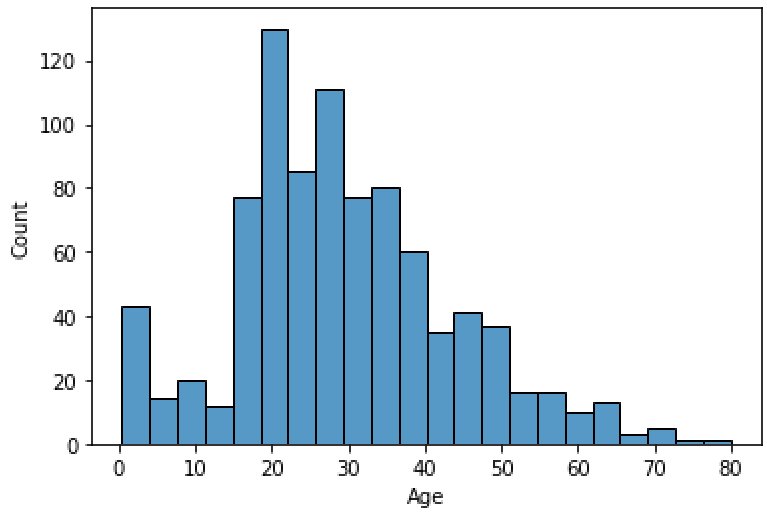
In [61]:
```
dataset['Age'].var()
```

Out[61]: `199.42829701227413`

In [64]:
```
dataset['Age'].std()
```

Out[64]: `14.12190840546256`

In [63]:
```
sns.histplot(x='Age', data=dataset)
plt.show()
```



In [65]:
```
dataset.describe()
```

Out[65]:

	Survived	Pclass	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
count	887.000000	887.000000	887.000000	887.000000	887.000000	887.00000
mean	0.385569	2.305524	29.471443	0.525366	0.383315	32.30542
std	0.487004	0.836662	14.121908	1.104669	0.807466	49.78204
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.00000
25%	0.000000	2.000000	20.250000	0.000000	0.000000	7.92500
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.45420
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.13750
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.32920

In [ ]:

3_Percentage, Percentile and Quartile

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

In [2]:
```
dataset = pd.read_csv('titanic.CSV')
```

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

Out[3]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
0	0	3	Mr. Owen Harris Braund	male	22.0	1	0	7.2500
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833
2	1	3	Miss. Laina Heikkinen	female	26.0	0	0	7.9250

In [5]:
```
dataset.isnull().sum()
```

Out[5]:
```
Survived                   0
Pclass                     0
Name                       0
Sex                        0
Age                        0
Siblings/Spouses Aboard    0
Parents/Children Aboard    0
Fare                       0
dtype: int64
```

In [ ]:
```
# So no null value is present in above data
```

In [7]:
```
np.percentile(dataset['Age'], 25), np.percentile(dataset['Age'], 75)
```

Out[7]:
```
(20.25, 38.0)
```

In [13]:
```
np.percentile(dataset['Age'], 0), np.percentile(dataset['Age'], 100), np.percentile
```

Out[13]:
```
(0.42, 80.0, 28.0)
```

In [14]:
```
dataset['Age'].min(), dataset['Age'].max(), dataset['Age'].median()
```

Out[14]:
```
(0.42, 80.0, 28.0)
```

In [16]:
```
# So in above 2 rows, min. age account for 0% percentile and max. age accounts for
# and median age is 50% percentile of age
```

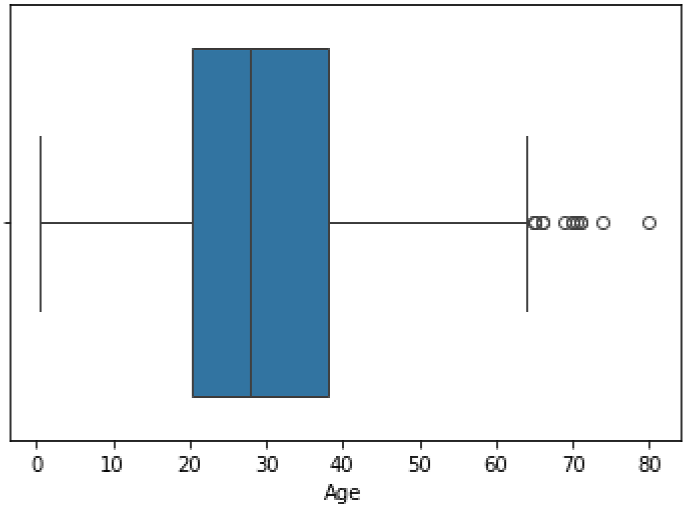
In [17]:
```
dataset.describe()
```

Out[17]:

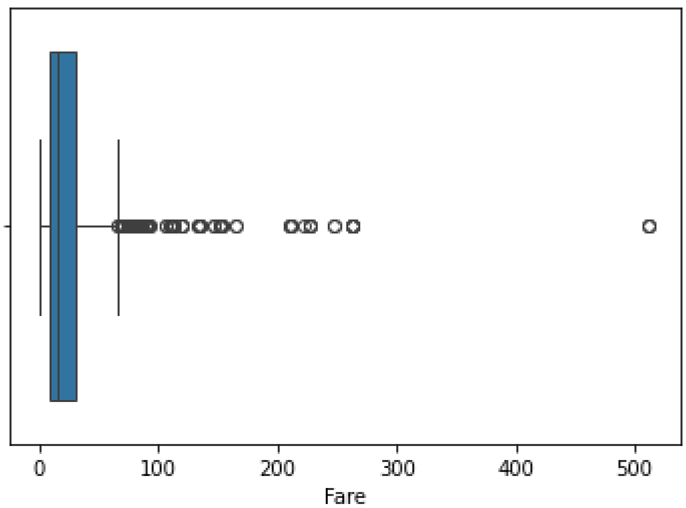
	Survived	Pclass	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
count	887.000000	887.000000	887.000000	887.000000	887.000000	887.00000
mean	0.385569	2.305524	29.471443	0.525366	0.383315	32.30542
std	0.487004	0.836662	14.121908	1.104669	0.807466	49.78204
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.00000
25%	0.000000	2.000000	20.250000	0.000000	0.000000	7.92500
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.45420
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.13750
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.32920

In [20]:
```
#If you see closely on age you can see that
# min(0%)       : 0.42
# Q1 : 25%      : 20.25
# Q2 : 50%      : 28.00
# Q3 : 75%      : 38.00
# Q4 : max(80%): 80.00
# So you can see the huge difference between Q3 and Q4. So it is clear that outlier
# Also difference between min (0%) and Q1 is significant larger, so there is also c
# median (Q2) is 28, so it is evident that the median is inclined towards left side
# So this whole analysis tell that there is definitely outlier present in this data
```

In [23]:
```
# To show it in the boxplot
sns.boxplot(x='Age', data=dataset)
plt.show()
```



In [25]:
```
# To show it in the boxplot
sns.boxplot(x='Fare', data=dataset)
plt.show()
```



In [ ]:

In [ ]:

4_Measures of Shape - Skewness

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

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

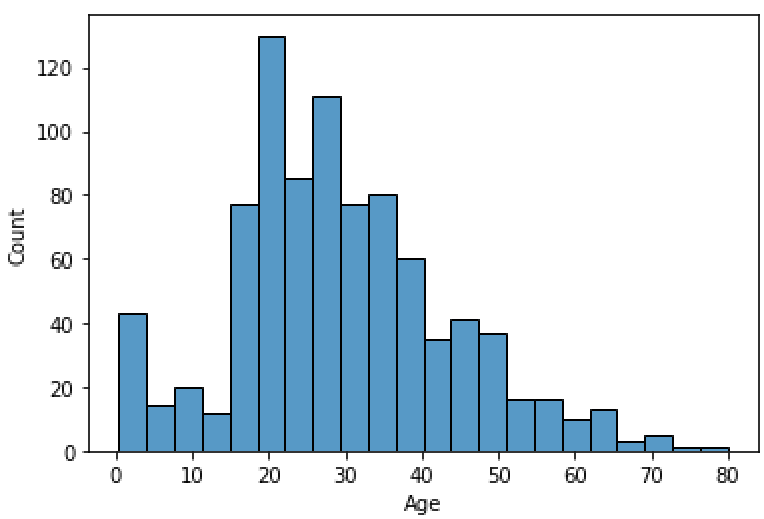
In [4]:
```
dataset.head(3)
```

Out[4]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
0	0	3	Mr. Owen Harris Braund	male	22.0	1	0	7.2500
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833
2	1	3	Miss. Laina Heikkinen	female	26.0	0	0	7.9250

To see if Age has skewness or no skewness

In [6]:
```
sns.histplot(x='Age', data=dataset)
plt.show()
```



This is right skewed chart (Positive skewness)

In [8]:
```
# If skew is greater than zero - Positive skewness and vice versa
dataset['Age'].skew()
```

Out[8]:
```
0.44718857190799916
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

5_Probability - Correlation

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

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

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

Out[3]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3

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

Out[4]:
```
total_bill    0
tip           0
sex           0
smoker        0
day           0
time          0
size          0
dtype: int64
```

In [5]:
```
# To check datatypes in dataset
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   total_bill  244 non-null    float64
 1   tip         244 non-null    float64
 2   sex         244 non-null    object 
 3   smoker      244 non-null    object 
 4   day         244 non-null    object 
 5   time        244 non-null    object 
 6   size        244 non-null    int64  
dtypes: float64(2), int64(1), object(4)
memory usage: 13.5+ KB
```

In [6]:
```
dataset.select_dtypes("float64" ,"int64")
```

Out[6]:

	total_bill	tip
0	16.99	1.01
1	10.34	1.66
2	21.01	3.50
3	23.68	3.31
4	24.59	3.61
...	...	...
239	29.03	5.92
240	27.18	2.00
241	22.67	2.00
242	17.82	1.75
243	18.78	3.00

244 rows × 2 columns

In [10]:
```
data_cor = dataset.select_dtypes("float64" ,"int64").corr()
data_cor
```

Out[10]:

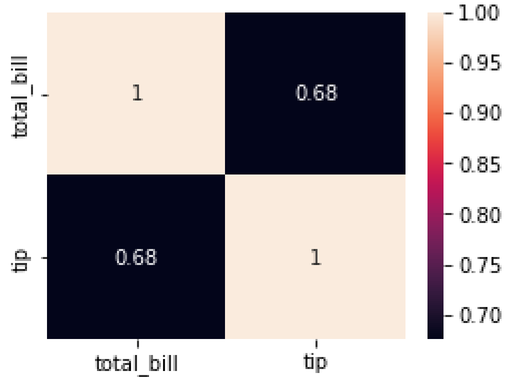
	total_bill	tip
total_bill	1.000000	0.675734
tip	0.675734	1.000000

In [11]:
```
data_cov = dataset.select_dtypes("float64" ,"int64").cov()
data_cov
```

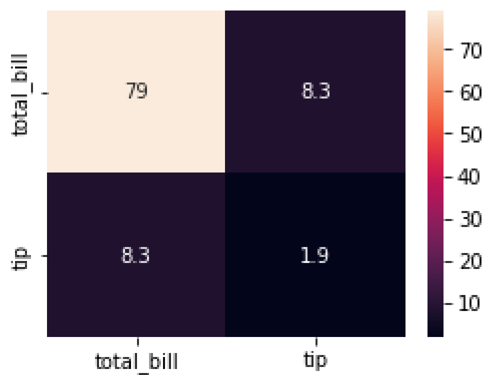
Out[11]:

	total_bill	tip
total_bill	79.252939	8.323502
tip	8.323502	1.914455

In [16]:
```
plt.figure(figsize=(4,3))
sns.heatmap(data_cor, annot=True)
plt.show()
```



In [17]:
```
plt.figure(figsize=(4,3))
sns.heatmap(data_cov, annot=True)
plt.show()
```



In [ ]:

6_Central Limit Theorem

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

In [23]:
```
# Generate random data by using list comprehension

pop_data = [np.random.randint(10,100) for i in range(10000)]
pop_data
```

```
Out[23]: [85,
 56,
 80,
 74,
 62,
 54,
 20,
 70,
 51,
 13,
 80,
 15,
 92,
 23,
 34,
 67,
 98,
 64,
 48,
 97,
 10,
 72,
 47,
 48,
 80,
 56,
 27,
 28,
 85,
 39,
 11,
 97,
 21,
 28,
 64,
 65,
 80,
 70,
 16,
 78,
 41,
 29,
 60,
 23,
 82,
 93,
 51,
 59,
 60,
 60,
 43,
 12,
 35,
 49,
 10,
 19,
```

37,
52,
42,
68,
22,
61,
30,
23,
52,
98,
40,
19,
49,
38,
48,
67,
71,
22,
41,
30,
78,
14,
27,
78,
97,
29,
59,
82,
73,
97,
87,
45,
93,
46,
21,
63,
59,
79,
27,
36,
49,
23,
91,
96,
95,
79,
43,
89,
26,
75,
73,
15,
61,
72,
99,
64,

```
38,
20,
72,
11,
85,
67,
88,
86,
70,
35,
22,
38,
68,
48,
69,
20,
12,
48,
54,
30,
73,
79,
73,
31,
46,
40,
58,
86,
36,
12,
12,
52,
50,
28,
32,
68,
74,
88,
80,
51,
55,
42,
35,
81,
78,
49,
10,
16,
31,
22,
41,
87,
58,
63,
66,
80,
```

82,
55,
47,
58,
93,
12,
82,
52,
51,
21,
58,
45,
87,
60,
49,
81,
82,
21,
15,
49,
67,
61,
16,
69,
85,
40,
12,
57,
98,
13,
46,
14,
62,
13,
83,
31,
75,
64,
82,
35,
51,
63,
47,
90,
90,
14,
41,
10,
27,
93,
20,
10,
67,
46,
40,
67,

```
42,
90,
86,
81,
22,
95,
65,
42,
74,
89,
81,
48,
80,
17,
98,
94,
91,
14,
68,
17,
63,
34,
27,
32,
91,
40,
76,
81,
52,
61,
94,
46,
23,
87,
18,
87,
60,
93,
11,
39,
27,
30,
30,
22,
77,
47,
88,
17,
33,
27,
57,
33,
20,
10,
19,
40,
```

```
15,
95,
72,
85,
36,
34,
64,
71,
29,
46,
27,
43,
58,
69,
85,
87,
54,
96,
89,
81,
64,
64,
30,
93,
60,
17,
51,
79,
39,
87,
72,
67,
30,
84,
53,
59,
54,
25,
75,
71,
82,
21,
12,
17,
29,
46,
33,
73,
99,
22,
54,
16,
39,
58,
46,
69,
```

17,
32,
25,
44,
96,
75,
97,
50,
11,
49,
48,
25,
40,
21,
54,
78,
61,
58,
12,
67,
62,
86,
49,
74,
18,
35,
56,
62,
68,
15,
40,
29,
95,
39,
72,
98,
69,
66,
56,
41,
90,
84,
26,
63,
33,
16,
72,
22,
99,
40,
81,
64,
37,
32,
31,
38,

```
40,
47,
16,
75,
17,
60,
68,
90,
85,
25,
25,
83,
78,
33,
14,
23,
51,
54,
56,
11,
47,
90,
17,
43,
33,
87,
13,
65,
84,
54,
29,
20,
89,
45,
81,
65,
34,
34,
80,
64,
69,
79,
36,
15,
49,
56,
38,
81,
61,
27,
51,
89,
34,
84,
23,
16,
```

46,
59,
51,
27,
17,
18,
29,
81,
67,
81,
83,
73,
46,
14,
82,
31,
80,
74,
24,
93,
12,
98,
93,
53,
35,
50,
15,
71,
61,
33,
54,
71,
67,
29,
39,
44,
61,
88,
30,
53,
48,
32,
92,
22,
65,
67,
42,
66,
73,
45,
99,
57,
73,
29,
28,
54,

67,
47,
14,
10,
70,
16,
32,
93,
12,
53,
52,
63,
13,
91,
65,
52,
94,
28,
36,
33,
49,
83,
48,
15,
93,
25,
55,
24,
32,
67,
95,
61,
53,
45,
43,
47,
38,
28,
37,
98,
21,
92,
84,
74,
58,
69,
90,
68,
12,
98,
34,
21,
91,
89,
36,
23,

98,
73,
93,
86,
22,
57,
34,
36,
81,
17,
58,
47,
96,
12,
38,
95,
63,
65,
41,
51,
71,
65,
23,
49,
65,
43,
75,
91,
38,
10,
83,
11,
21,
46,
99,
75,
66,
52,
88,
82,
16,
20,
56,
92,
32,
87,
37,
32,
68,
39,
16,
32,
90,
72,
81,
56,

```
10,
77,
70,
62,
64,
61,
29,
22,
16,
40,
24,
40,
71,
49,
21,
93,
60,
57,
12,
58,
59,
73,
64,
25,
70,
37,
70,
29,
93,
68,
32,
11,
85,
79,
51,
92,
62,
78,
54,
55,
67,
49,
66,
18,
56,
25,
66,
44,
38,
68,
92,
61,
95,
21,
86,
22,
```

94,
16,
85,
34,
19,
21,
22,
22,
78,
89,
18,
57,
86,
44,
75,
31,
99,
13,
80,
47,
26,
22,
89,
69,
52,
94,
86,
80,
18,
25,
28,
32,
52,
14,
37,
40,
57,
33,
53,
73,
25,
77,
77,
53,
76,
68,
10,
25,
11,
41,
65,
21,
43,
92,
62,
79,

```
95,
29,
88,
79,
78,
66,
82,
95,
91,
17,
51,
65,
54,
23,
10,
23,
90,
48,
18,
44,
47,
63,
74,
84,
58,
77,
62,
90,
39,
41,
54,
91,
97,
79,
42,
66,
35,
25,
32,
94,
18,
32,
28,
63,
95,
24,
20,
57,
73,
89,
86,
69,
86,
35,
39,
96,
```

82,
92,
75,
91,
38,
83,
84,
27,
25,
62,
12,
40,
90,
97,
59,
47,
30,
66,
87,
10,
12,
78,
77,
38,
54,
60,
24,
29,
12,
69,
48,
51,
39,
97,
68,
59,
98,
46,
15,
50,
60,
89,
48,
81,
79,
54,
15,
57,
96,
35,
25,
14,
38,
68,
82,
50,

58,
19,
67,
34,
18,
85,
26,
35,
59,
90,
60,
68,
77,
80,
78,
76,
95,
92,
39,
24,
50,
33,
32,
64,
12,
40,
15,
79,
84,
43,
51,
28,
63,
49,
42,
29,
22,
36,
67,
57,
86,
61,
61,
39,
81,
49,
91,
22,
35,
72,
81,
31,
17,
57,
81,
80,

73,
55,
32,
81,
76,
71,
60,
11,
88,
65,
77,
64,
28,
11,
75,
23,
71,
30,
48,
56,
26,
82,
45,
44,
65,
39,
45,
77,
69,
58,
35,
92,
59,
45,
78,
25,
95,
71,
75,
26,
13,
63,
10,
81,
77,
27,
63,
92,
23,
37,
84,
48,
57,
67,
82,
57,

```
 83,
 75,
 22,
 79,
 24,
 36,
 97,
 83,
 18,
 51,
 59,
 19,
 94,
 99,
 15,
 39,
 97,
 52,
 55,
 97,
 94,
 57,
 66,
 76,
 95,
 91,
 93,
 39,
 48,
 12,
 64,
 80,
 28,
 28,
 49,
 80,
 32,
 42,
 88,
 88,
 86,
 87,
 51,
 19,
 85,
 72,
 62,
 84,
 ...]
```

In [24]:

```
# the above line of code could be written as:
pop_data = []
for i in range(10000):
    pop_data.append(np.random.randint(10,100))
pop_data
```

```
Out[24]: [71,
 45,
 29,
 38,
 64,
 82,
 64,
 35,
 74,
 76,
 63,
 26,
 10,
 87,
 15,
 57,
 54,
 38,
 52,
 97,
 32,
 87,
 49,
 93,
 39,
 83,
 33,
 11,
 20,
 85,
 94,
 79,
 87,
 76,
 53,
 89,
 25,
 93,
 33,
 29,
 12,
 92,
 91,
 73,
 28,
 21,
 81,
 15,
 24,
 67,
 61,
 65,
 13,
 46,
 92,
 46,
```

35,
10,
50,
55,
16,
50,
93,
19,
98,
83,
60,
84,
16,
59,
98,
38,
65,
78,
27,
43,
80,
76,
67,
70,
91,
57,
50,
42,
88,
95,
35,
93,
25,
65,
32,
39,
79,
37,
20,
58,
71,
61,
75,
96,
60,
39,
98,
79,
32,
68,
12,
24,
47,
54,
38,
97,

```
40,
40,
52,
84,
26,
61,
96,
92,
11,
32,
62,
31,
76,
35,
43,
33,
89,
94,
68,
11,
85,
79,
53,
53,
56,
67,
20,
93,
50,
83,
25,
66,
43,
69,
43,
70,
39,
42,
95,
30,
52,
46,
48,
86,
68,
30,
94,
27,
47,
33,
78,
50,
27,
15,
66,
38,
```

65,
72,
78,
39,
93,
90,
86,
19,
13,
76,
19,
56,
53,
82,
54,
77,
81,
16,
56,
92,
75,
39,
45,
25,
28,
39,
51,
91,
32,
85,
43,
17,
41,
19,
53,
24,
50,
14,
12,
92,
51,
85,
33,
39,
54,
66,
58,
24,
19,
29,
25,
89,
60,
57,
33,
76,

61,
49,
80,
69,
18,
16,
93,
19,
20,
57,
66,
38,
13,
71,
49,
32,
84,
88,
92,
84,
87,
11,
14,
83,
66,
71,
46,
59,
44,
16,
40,
13,
42,
48,
17,
14,
11,
86,
28,
11,
62,
87,
33,
99,
61,
40,
23,
77,
77,
80,
25,
50,
16,
75,
87,
37,

36,
97,
75,
39,
47,
32,
87,
58,
93,
21,
66,
34,
67,
32,
33,
84,
93,
61,
19,
95,
34,
80,
20,
36,
12,
31,
60,
81,
19,
97,
56,
11,
10,
70,
39,
30,
32,
33,
80,
85,
52,
27,
56,
19,
74,
21,
68,
45,
75,
55,
66,
16,
84,
92,
71,
59,

93,
78,
27,
16,
76,
18,
29,
52,
25,
63,
52,
80,
57,
82,
33,
85,
67,
60,
92,
43,
42,
93,
27,
42,
21,
51,
20,
74,
32,
37,
89,
56,
92,
98,
59,
77,
85,
22,
32,
15,
86,
94,
78,
80,
92,
91,
97,
18,
38,
86,
53,
68,
41,
82,
63,
60,

52,
98,
12,
43,
19,
41,
49,
48,
17,
99,
95,
43,
54,
99,
78,
81,
20,
66,
72,
45,
46,
34,
51,
96,
90,
98,
10,
57,
52,
64,
35,
76,
43,
42,
59,
54,
45,
44,
20,
34,
90,
22,
50,
36,
27,
54,
70,
55,
41,
23,
42,
19,
49,
19,
59,
68,

87,
99,
78,
72,
91,
74,
63,
35,
24,
24,
23,
36,
73,
50,
31,
89,
85,
67,
62,
50,
99,
21,
65,
23,
96,
53,
35,
22,
47,
26,
40,
57,
93,
24,
73,
68,
63,
86,
60,
50,
73,
51,
56,
29,
59,
64,
24,
57,
12,
84,
38,
89,
33,
30,
78,
78,

```
41,
23,
40,
89,
90,
17,
42,
61,
13,
77,
41,
29,
25,
75,
50,
35,
36,
92,
92,
88,
57,
12,
76,
40,
80,
50,
24,
87,
36,
80,
98,
20,
13,
82,
18,
90,
43,
57,
92,
28,
37,
89,
56,
26,
18,
84,
55,
34,
60,
88,
68,
47,
20,
67,
27,
78,
```

95,
76,
53,
76,
50,
88,
44,
97,
54,
25,
24,
21,
86,
77,
25,
23,
92,
24,
14,
96,
63,
13,
46,
95,
16,
25,
71,
61,
65,
13,
58,
92,
93,
88,
74,
29,
84,
82,
88,
38,
52,
53,
56,
78,
14,
53,
39,
42,
66,
14,
34,
72,
12,
32,
17,
95,

84,
24,
55,
71,
60,
84,
85,
85,
44,
14,
39,
38,
76,
83,
16,
85,
78,
56,
98,
65,
68,
42,
15,
57,
64,
23,
39,
78,
73,
84,
90,
63,
64,
65,
58,
95,
91,
33,
83,
14,
23,
63,
71,
14,
44,
14,
86,
75,
28,
97,
77,
60,
27,
39,
68,
67,

```
22,
40,
73,
89,
90,
20,
90,
17,
90,
69,
63,
15,
35,
16,
78,
68,
36,
53,
70,
71,
64,
72,
74,
73,
97,
78,
45,
27,
69,
21,
63,
50,
27,
15,
94,
49,
22,
15,
55,
42,
25,
10,
74,
76,
19,
65,
91,
61,
32,
15,
69,
46,
35,
14,
18,
14,
```

```
55,
85,
67,
88,
61,
30,
64,
45,
74,
16,
96,
97,
78,
90,
71,
22,
82,
26,
15,
69,
61,
31,
96,
97,
72,
78,
91,
83,
18,
14,
56,
98,
55,
84,
44,
37,
84,
78,
55,
13,
73,
97,
68,
52,
99,
83,
86,
23,
90,
29,
48,
89,
22,
84,
65,
43,
```

83,
48,
11,
71,
53,
48,
99,
73,
24,
93,
75,
50,
36,
66,
35,
65,
82,
84,
65,
27,
53,
63,
96,
70,
55,
34,
75,
93,
47,
12,
64,
94,
28,
23,
83,
51,
22,
23,
28,
99,
28,
92,
48,
63,
79,
24,
66,
65,
66,
57,
78,
10,
43,
70,
39,
42,

39,
27,
57,
37,
66,
37,
90,
77,
80,
10,
99,
36,
79,
44,
88,
16,
99,
39,
37,
52,
98,
51,
31,
72,
38,
31,
10,
28,
86,
13,
38,
36,
61,
52,
26,
96,
93,
31,
44,
22,
73,
87,
17,
13,
72,
82,
18,
77,
31,
24,
33,
98,
97,
22,
78,
46,

16,
65,
58,
46,
83,
14,
23,
87,
23,
94,
34,
86,
45,
80,
26,
40,
41,
55,
67,
87,
14,
83,
50,
48,
11,
43,
70,
54,
20,
54,
26,
44,
38,
77,
55,
95,
95,
86,
82,
93,
16,
51,
56,
33,
76,
78,
87,
10,
23,
13,
66,
75,
16,
53,
68,
98,

```
 95,
 60,
 18,
 77,
 85,
 62,
 39,
 71,
 11,
 78,
 54,
 31,
 69,
 64,
 76,
 70,
 28,
 79,
 86,
 78,
 12,
 79,
 87,
 38,
 36,
 81,
 18,
 17,
 78,
 17,
 46,
 20,
 57,
 98,
 48,
 66,
 21,
 25,
 84,
 25,
 18,
 84,
 57,
 63,
 17,
 24,
 76,
 70,
 ...]
```

In [25]:
```
len(pop_data)
```

Out[25]:
```
10000
```

In [26]:
```
# TO convert population data into a csv file
pop_table = pd.DataFrame({'pop_data':pop_data})
```

```
pop_table
```

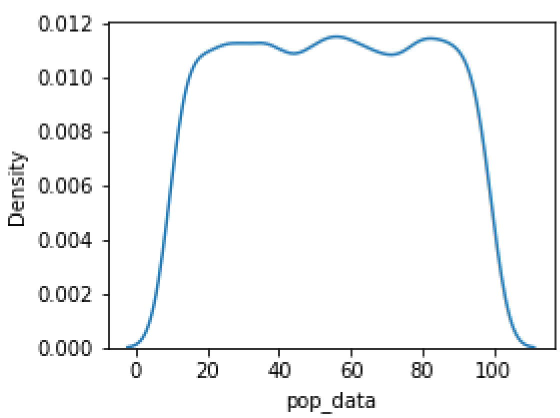
Out[26]:

	pop_data
0	71
1	45
2	29
3	38
4	64
...	...
9995	55
9996	20
9997	43
9998	74
9999	66

10000 rows × 1 columns

In [27]:

```
plt.figure(figsize=(4,3))
sns.kdeplot(x='pop_data', data=pop_table)
plt.show()
```



above graph shows that our data is not normally distributed, so we will apply CLT

In [33]:

```
# First we will pick up random samples from population data
# Pre-req: Sample should not be more than 10% population and more than 30 samples s

# so calculate 10% of 10000 data

10/100 * 10000
```

Out[33]: 1000.0

That means i.e. n>30 and n< 1000, so are taking n= [50,500]

In [32]:
```
# To pick random data from population data
np.random.choice(pop_data)
```

Out[32]:
```
37
```

In [37]:
```
# So will take sample data less than 1000
sample_mean = []
# to take number of sample data 50 (to meet requirement n>30)
for no_of_sample in range(50):
    sample_data = []
    # to take number of sample data less than 1000 (so will take 500 sample)
    for i in range(500):
        sample_data.append(np.random.choice(pop_data))
    # To calculate mean of sample data
    sample_mean.append(np.mean(sample_data))
```

In [41]:
```
len(sample_data), len(sample_mean)
```

Out[41]:
```
(500, 50)
```

In [42]:
```
sample_data
```

```
Out[42]: [78,
 78,
 16,
 32,
 24,
 68,
 74,
 89,
 44,
 21,
 57,
 61,
 27,
 70,
 34,
 84,
 29,
 77,
 77,
 24,
 13,
 88,
 37,
 43,
 17,
 50,
 98,
 36,
 87,
 24,
 15,
 69,
 77,
 52,
 89,
 24,
 93,
 56,
 42,
 42,
 67,
 36,
 20,
 50,
 45,
 43,
 88,
 46,
 13,
 17,
 95,
 68,
 10,
 26,
 95,
 37,
```

79,
69,
35,
97,
57,
86,
77,
76,
97,
23,
63,
60,
12,
54,
37,
27,
33,
89,
70,
58,
65,
55,
91,
35,
12,
86,
64,
59,
59,
64,
50,
36,
98,
64,
19,
22,
77,
32,
69,
95,
73,
12,
82,
42,
37,
90,
72,
43,
83,
32,
80,
85,
59,
38,
82,
50,

```
56,
91,
83,
73,
24,
66,
10,
84,
61,
76,
35,
80,
21,
70,
57,
30,
82,
50,
55,
23,
88,
90,
96,
56,
67,
98,
11,
61,
27,
10,
92,
37,
22,
71,
34,
90,
50,
98,
36,
36,
39,
94,
58,
40,
91,
91,
43,
61,
81,
50,
33,
47,
61,
41,
34,
81,
```

76,
76,
18,
41,
87,
99,
45,
65,
86,
21,
69,
83,
62,
19,
61,
66,
19,
17,
89,
46,
48,
95,
54,
65,
18,
49,
29,
88,
74,
38,
28,
93,
29,
54,
93,
23,
79,
64,
27,
68,
67,
75,
79,
77,
16,
51,
82,
69,
34,
80,
24,
17,
89,
37,
48,
12,

63,
76,
98,
91,
46,
61,
60,
75,
33,
95,
25,
91,
52,
26,
76,
38,
46,
13,
50,
20,
75,
99,
81,
91,
57,
87,
81,
69,
87,
39,
61,
83,
79,
85,
81,
38,
62,
71,
56,
21,
79,
61,
28,
58,
56,
30,
27,
89,
47,
91,
76,
27,
17,
35,
88,
97,

50,
32,
68,
72,
96,
91,
70,
23,
23,
63,
45,
63,
26,
62,
96,
84,
55,
65,
60,
29,
14,
80,
26,
50,
99,
65,
53,
39,
84,
56,
95,
66,
53,
62,
98,
68,
65,
65,
20,
27,
96,
91,
43,
36,
42,
79,
66,
59,
78,
71,
58,
36,
38,
17,
93,
85,

87,
90,
33,
25,
94,
58,
57,
60,
45,
53,
79,
97,
22,
85,
24,
53,
87,
93,
50,
31,
12,
31,
45,
91,
35,
58,
17,
52,
51,
38,
72,
37,
71,
13,
76,
82,
27,
19,
18,
22,
73,
19,
67,
89,
92,
23,
85,
87,
29,
95,
85,
71,
97,
72,
47,
89,

52,
32,
42,
56,
50,
44,
82,
90,
84,
62,
49,
41,
80,
21,
72,
72,
67,
27,
22,
96,
87,
25,
27,
96,
75,
42,
10,
34,
87,
62,
12,
86,
44,
43,
54,
81,
80,
84,
43,
59,
29,
20,
83,
59,
23,
37,
41,
53,
79,
25,
51,
25,
54,
53,
20,
90,

```
 83,
 30,
 46,
 96,
 36,
 16,
 29,
 77,
 87,
 65,
 36,
 14,
 68,
 29,
 54,
 96,
 98,
 85,
 62,
 92,
 41,
 43,
 22,
 89,
 73,
 89,
 97,
 96,
 44,
 44,
 35,
 84,
 74,
 50,
 44,
 18,
 55,
 31,
 94,
 18,
 39,
 72,
 87,
 34,
 11,
 63,
 62,
 99,
 17,
 56,
 60,
 29]
```

In [43]:

```
sample_mean
```

Out[43]:
```
[53.812,
 54.18,
 56.164,
 56.114,
 53.844,
 54.842,
 54.934,
 54.79,
 53.856,
 55.86,
 53.654,
 57.08,
 53.74,
 54.942,
 53.634,
 54.91,
 55.612,
 54.014,
 55.036,
 54.076,
 54.232,
 54.018,
 52.262,
 55.06,
 52.824,
 52.568,
 54.11,
 55.224,
 52.204,
 54.616,
 53.556,
 54.684,
 54.642,
 54.806,
 54.81,
 53.248,
 55.382,
 54.486,
 53.684,
 52.336,
 55.712,
 53.55,
 53.78,
 54.64,
 55.746,
 54.278,
 53.292,
 56.57,
 53.29,
 57.052]
```

In [44]:
```
# To see data in sample_mean is normally distributed or not
sample_mean_DF = pd.DataFrame({"Sample_mean":sample_mean})
```

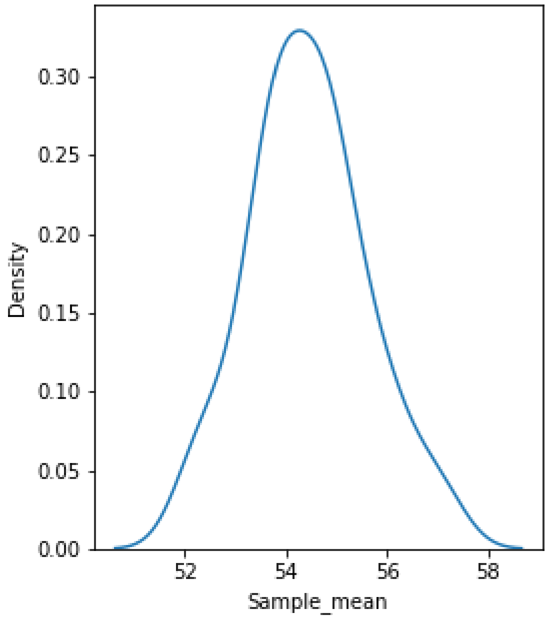
In [45]:
```
sample_mean_DF
```

Out[45]:

	Sample_mean
0	53.812
1	54.180
2	56.164
3	56.114
4	53.844
5	54.842
6	54.934
7	54.790
8	53.856
9	55.860
10	53.654
11	57.080
12	53.740
13	54.942
14	53.634
15	54.910
16	55.612
17	54.014
18	55.036
19	54.076
20	54.232
21	54.018
22	52.262
23	55.060
24	52.824
25	52.568
26	54.110
27	55.224
28	52.204
29	54.616

	Sample_mean
30	53.556
31	54.684
32	54.642
33	54.806
34	54.810
35	53.248
36	55.382
37	54.486
38	53.684
39	52.336
40	55.712
41	53.550
42	53.780
43	54.640
44	55.746
45	54.278
46	53.292
47	56.570
48	53.290
49	57.052

In [50]:
```
plt.figure(figsize=(4,5))
sns.kdeplot(x="Sample_mean", data=sample_mean_DF)
plt.show()
```



So the data is normally distributed

In [53]:
```
# To meat another requirement of CLT that is the mean of population data and the me
# so we will check the both means
np.mean(pop_data), np.mean(sample_mean)
```

Out[53]:
```
(54.3654, 54.43512)
```

In [ ]:

7_Calculating Z-test

In [11]:
```
import scipy.stats as st
import numpy as np
```

In [19]:
```
# To calculate Z-value (from Z-table)
z_table = st.norm.ppf(1-0.05)
z_table
```

Out[19]: 1.6448536269514722

In [20]:
```
s_x = 90
p_u = 82
p_std = 20
n = 81
```

In [21]:
```
z_cal = (s_x - p_u) / (p_std/np.sqrt(n))
z_cal
```

Out[21]: 3.5999999999999996

In [24]:
```
if z_table < z_cal:
    print("Alternate Hypothesis (Ha) is correct")
else:
    print("Null hypothesis (Ho) is correct")
```

Alternate Hypothesis (Ha) is correct

In [ ]:
```

```

8_Calculating T-test

In [17]:
```
import scipy.stats as st
import numpy as np
```

In [18]:
```
Ho = "Weight of bag is 150gm"
Ha = "Weight of bag is less than 150gm"
```

In [19]:
```
t_table = st.t.ppf(0.05,24)
t_table
```

Out[19]:
```
-1.7108820799094282
```

In [20]:
```
u_p = 150
x_s = 148
n_s = 25
std_s = 5
```

In [21]:
```
t_cal = (x_s - u_p)/(std_s/np.sqrt(n_s))
t_cal
```

Out[21]:
```
-2.0
```

In [22]:
```
if t_table > t_cal:
    print(Ha)
else:
    print(Ho)
```

```
Weight of bag is less than 150gm
```

In [ ]:

9_Calculating Chi-Square Test

9.1_To check goodness of data

In [1]:
```
import numpy as np
```

In [4]:
```
ob = np.array([22,17,20,26,22,13])
ex = np.array([20,20,20,20,20,20])
```

In [5]:
```
ob-ex
```

Out[5]:
```
array([ 2, -3,  0,  6,  2, -7])
```

In [9]:
```
np.sum(np.square(ob-ex)/ex)
```

Out[9]:
```
5.1000000000000005
```

9.2_To check dependency of variables

In [22]:
```
row1 = np.array([40,45,25,10])
row2 = np.array([35,30,20,30])
```

In [25]:
```
sum_r1 = np.sum(row1)
sum_r2 = np.sum(row2)
sum_row = np.array([sum_r1, sum_r2])
sum_row
```

Out[25]:
```
array([120, 115])
```

In [26]:
```
sum_col = row1 + row2
sum_col
```

Out[26]:
```
array([75, 75, 45, 40])
```

In [32]:
```
exp = []
for i in sum_row:
    for j in sum_col:
        exp.append(i*j/235)
print(exp)
```

```
[38.297872340425535, 38.297872340425535, 22.97872340425532, 20.425531914893618, 36.7
02127659574465, 36.702127659574465, 22.02127659574468, 19.574468085106382]
```

In [34]:
```
# join both columns for observed values
obj = np.array([40,45,25,10,35,30,20,30])
```

In [36]:
```
np.sum(np.square(obj - exp)/exp)
```

Out[36]: 13.788747987117553

In [ ]:

10_ML - Finding missing value in data.ipynb

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

10.1 What is missing value

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

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

Out[3]:

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

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

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

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

Out[6]:

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

614 rows × 13 columns

In [13]:

Out[13]: 149

In [7]:

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

Out[7]:

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

In [11]:

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

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

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

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

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

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

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

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

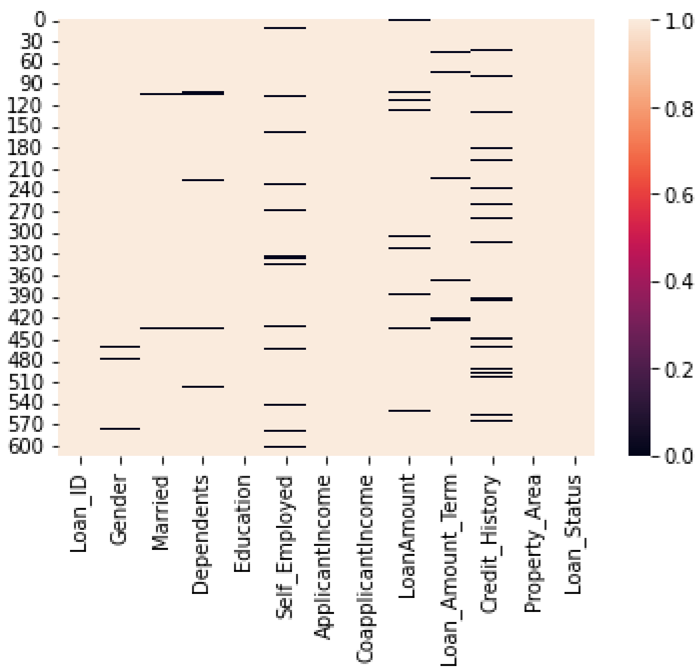
In [20]:
```
# To determine percentage totall null value in the data
# Total number of null data / total number of data * 100
dataset.isnull().sum().sum()/(dataset.shape[0] * dataset.shape[1])*100
```

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

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

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

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



10.2 How to handle missing values (Dropping)

Deleting in 2 ways:

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

Deleting a column from the data

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

```
Out[25]: Loan_ID               0
         Gender               13
         Married               3
         Dependents           15
         Education             0
         Self_Employed        32
         ApplicantIncome       0
         CoapplicantIncome     0
         LoanAmount           22
         Loan_Amount_Term     14
         Credit_History       50
         Property_Area         0
         Loan_Status           0
         dtype: int64
```

We will delete Credit_History column as it contains more missing values

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

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

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

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

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

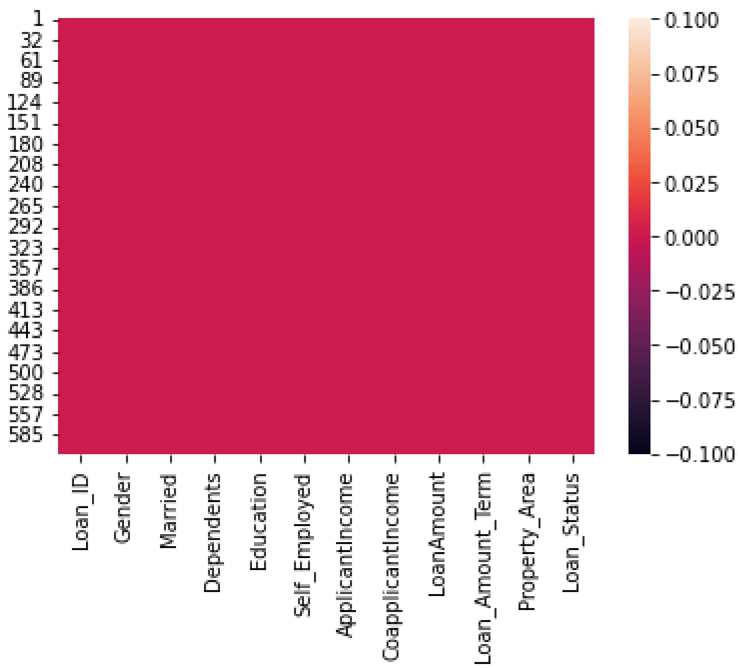
Deleting rows containing null values

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

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

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

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



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

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

To check how much data has been dropped (deleted)

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

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

14% data is lost

10.3 Handling Missing Values (Imputing Category Data)

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

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

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

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

Out[39]:

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

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

Out[40]:
```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

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

Out[42]:

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

614 rows × 13 columns

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

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

In [43]:

```
dataset.info()
```

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

In [47]:
```
# Backward filling - Back data will filled , forexample Loan Amount first row is fi
# nichay wala data oper a k fill ho jaye ga
dataset.fillna(method='bfill')
```

Out[47]:

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

614 rows × 13 columns

In [48]:
```
# Forward filling - oper wala data nicha a k fill ho jaye ga
# by default filling is row wise
dataset.fillna(method='ffill')
```

Out[48]:

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

614 rows × 13 columns

In [50]:

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

Out[50]:

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

614 rows × 13 columns

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

Fill particular column containing missing value by mode method

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

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

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

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

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

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

Out[55]:
```
Loan_ID               0
Gender                0
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

Fill all columns containing missing value by mode method

1. First you will collect all object datatype

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

Out[57]:

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

614 rows × 8 columns

In [58]:
```
dataset.select_dtypes(include='object').isnull()
```

Out[58]:

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

614 rows × 8 columns

In [59]:
```
dataset.select_dtypes(include='object').isnull().sum()
```

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

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

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

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

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

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

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

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

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

10.4 Handling Missing Values (Scikit-learn)

Import fresh datasheet

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

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

Out[66]:

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

In [67]:
```
dataset.info()
```

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

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

Out[69]:

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

614 rows × 4 columns

In [70]:
```
dataset.select_dtypes(include='float64').columns
```

Out[70]:
```
Index(['CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term',
       'Credit_History'],
      dtype='object')
```

Find the missing values using scikit learn

In [71]:
```
from sklearn.impute import SimpleImputer
```

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

In [74]:
```
si = SimpleImputer(strategy='mean')
si.fit_transform(dataset[['CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term',
       'Credit_History']])
```

Out[74]:
```
array([[0.00000000e+00, 1.46412162e+02, 3.60000000e+02, 1.00000000e+00],
       [1.50800000e+03, 1.28000000e+02, 3.60000000e+02, 1.00000000e+00],
       [0.00000000e+00, 6.60000000e+01, 3.60000000e+02, 1.00000000e+00],
       ...,
       [2.40000000e+02, 2.53000000e+02, 3.60000000e+02, 1.00000000e+00],
       [0.00000000e+00, 1.87000000e+02, 3.60000000e+02, 1.00000000e+00],
       [0.00000000e+00, 1.33000000e+02, 3.60000000e+02, 0.00000000e+00]])
```

In [82]:
```
# To convert this data into csv sheet
si = SimpleImputer(strategy='mean')
```

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

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

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

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

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

Out[91]:

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

614 rows × 4 columns

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

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

11_ML - One Hot Encoding and Dummy Variables

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

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

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

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

Out[3]:

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

11.1 Find Missing Values and Handle it

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

Out[4]:
```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

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

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

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

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

Out[12]:
```
Loan_ID               0
Gender                0
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

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

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

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

11.2 Use One Hot Code to handle missing values

First separate Gender and Married data to perform encoding

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

Out[16]:

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

614 rows × 2 columns

In [17]:
```
pd.get_dummies(en_data)
```

Out[17]:

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

614 rows × 4 columns

In [18]:
```
pd.get_dummies(en_data).info()
```

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

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

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

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

sparse matrix contains data in 0 and 1 form

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

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

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

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

Out[27]:

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

614 rows × 4 columns

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

In [28]:

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

Out[28]:

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

In [29]:

```
# Convert the array data into dataframe

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

Out[29]:

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

614 rows × 2 columns

In [ ]:

In [ ]:

12_ML - Label Encoder

12.1 Label encoding on Nominal data

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

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

Out[5]:

	name
0	Rashid
1	Lion
2	Computer
3	Gym
4	Plant

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

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

Out[8]:

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

Now work on real time data

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

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

Out[10]:

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

In [14]:

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

Out[14]:

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

In [12]:

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

Out[12]:

```
▾ LabelEncoder
LabelEncoder()
```

In [13]:

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

Out[13]:

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

In [15]:

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

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

Out[18]:

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

12.1 Label encoding on Ordinal data

12.1.1 Ordincal encoding through Cyclic Line

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

Out[21]:

	size
0	s
1	m
2	l

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

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

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

Out[25]:
```
▾                     OrdinalEncoder
OrdinalEncoder(categories=[['s', 'm', 'l', 'xl', 'xxl']])
```

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

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

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

Out[29]:

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

12.1.2 Ordincal encoding through Map function

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

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

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

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

Out[33]:

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

12.2 Perform Ordinal Encoding on big data

In [35]:
```
dataset = pd.read_csv('loan.csv')
```

In [36]:
```
dataset.head(3)
```

Out[36]:

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

In [37]:
```
dataset['Property_Area'].unique()
```

Out[37]:
```
array(['Urban', 'Rural', 'Semiurban'], dtype=object)
```

In [40]:
```
# if there is nan (missing data)m, you can fill it
# if the data is categorical then you should do mode filling
dataset['Property_Area'].fillna(dataset['Property_Area'].mode()[0], inplace=True)
```

In [42]:
```
en_data_loan = [['Urban', 'Rural', 'Semiurban']]
```

In [45]:
```
oen = OrdinalEncoder(categories=en_data_loan)
oen.fit_transform(dataset[['Property_Area']])
```

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

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

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

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

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

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

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

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

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

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

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

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

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

Out[49]:

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

13_Outlier

13.1 Detecting Outlier

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

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

Out[2]:

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

In [4]:
```
dataset.info()
```

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

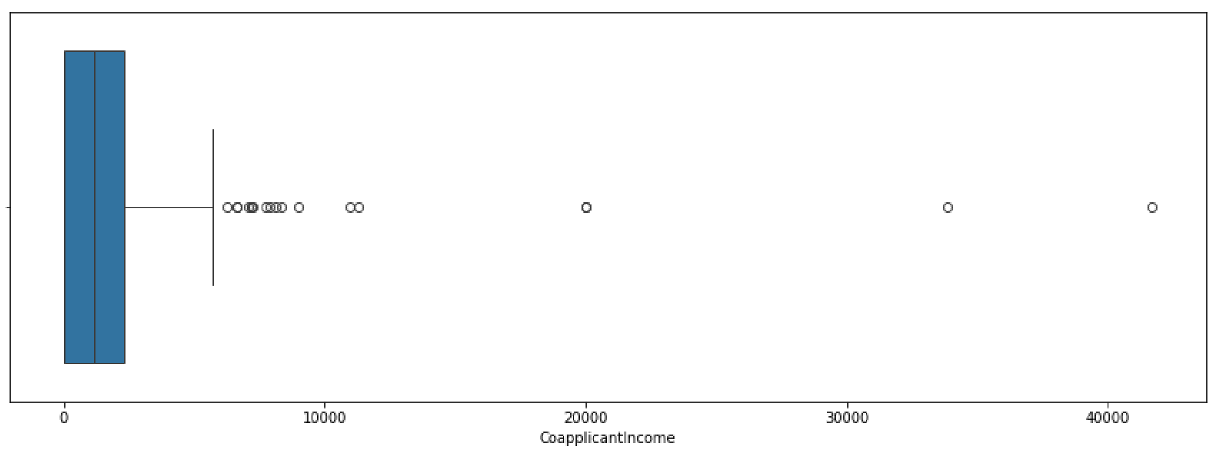
In [6]:
```
dataset.describe()
```

Out[6]:

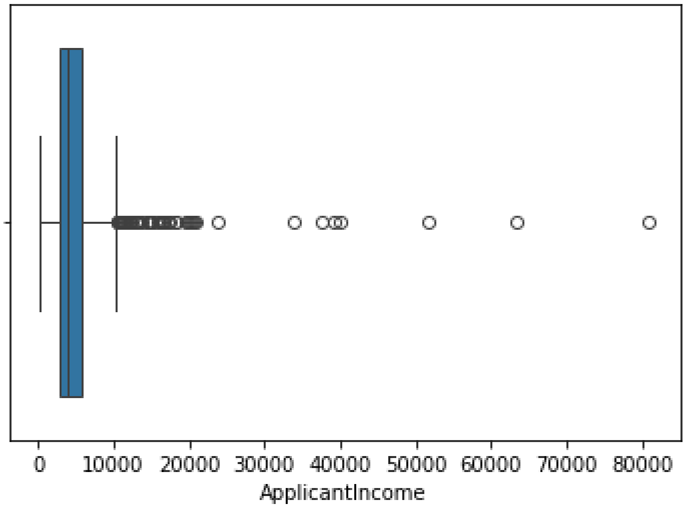
	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_Histo
count	614.000000	614.000000	592.000000	600.00000	564.0000
mean	5403.459283	1621.245798	146.412162	342.00000	0.8421
std	6109.041673	2926.248369	85.587325	65.12041	0.3648
min	150.000000	0.000000	9.000000	12.00000	0.0000
25%	2877.500000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

Detect Outlier through Boxplot

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

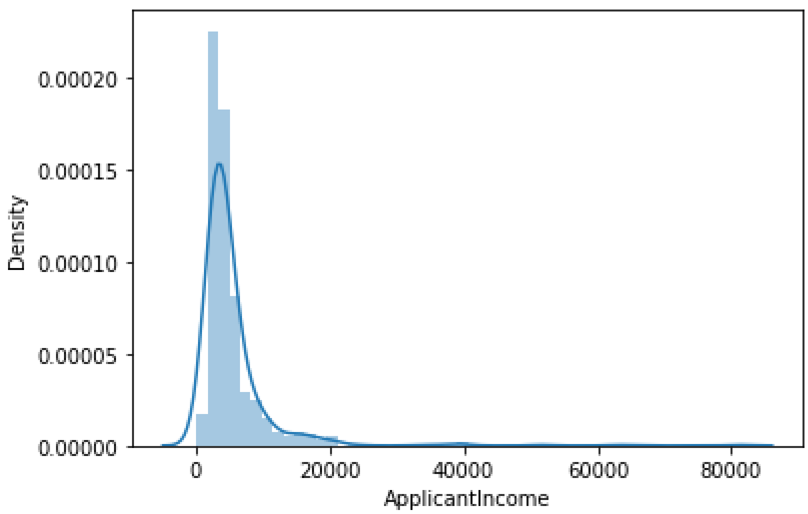


In [8]:
```
sns.boxplot(x='ApplicantIncome', data=dataset)
plt.show()
```



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

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

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

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

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

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



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

13.2 Removing Outlier

There are two methods for removing outlier:

1. IQR (Inter Quartile Range) method

2. Z-Score method

13.2.1 Removing Outlier through IQR Method

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

Out[11]: (614, 13)

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

Out[13]: (0.0, 2297.25)

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

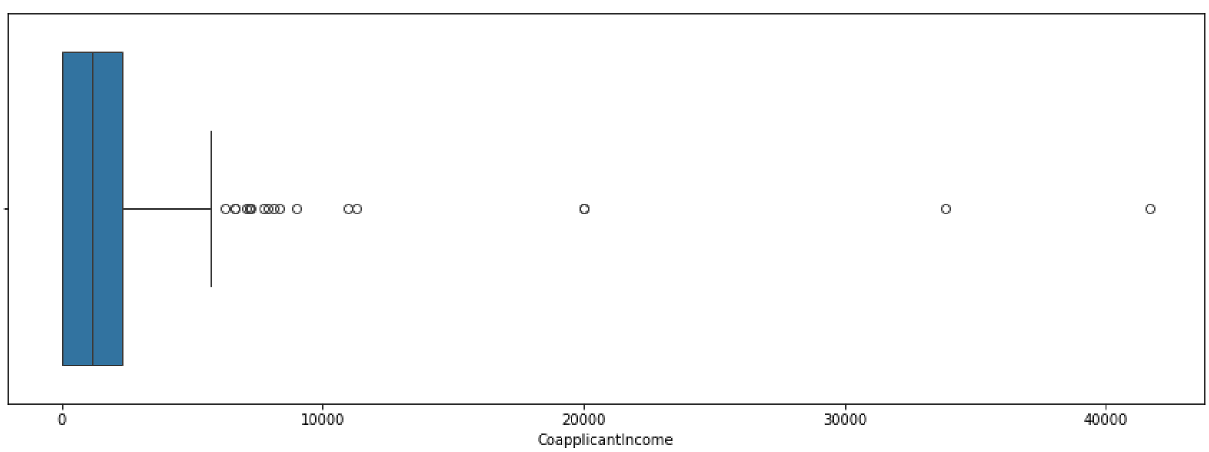
Out[14]: 2297.25

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

Out[15]: (-3445.875, 5743.125)

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

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



So now we will remove the outlier from the data

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

Out[18]:

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

In [19]:

```
dataset['CoapplicantIncome'] < max_range
```

Out[19]:

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

In [23]:

```
dataset.shape
```

Out[23]:

```
(614, 13)
```

In [22]:

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

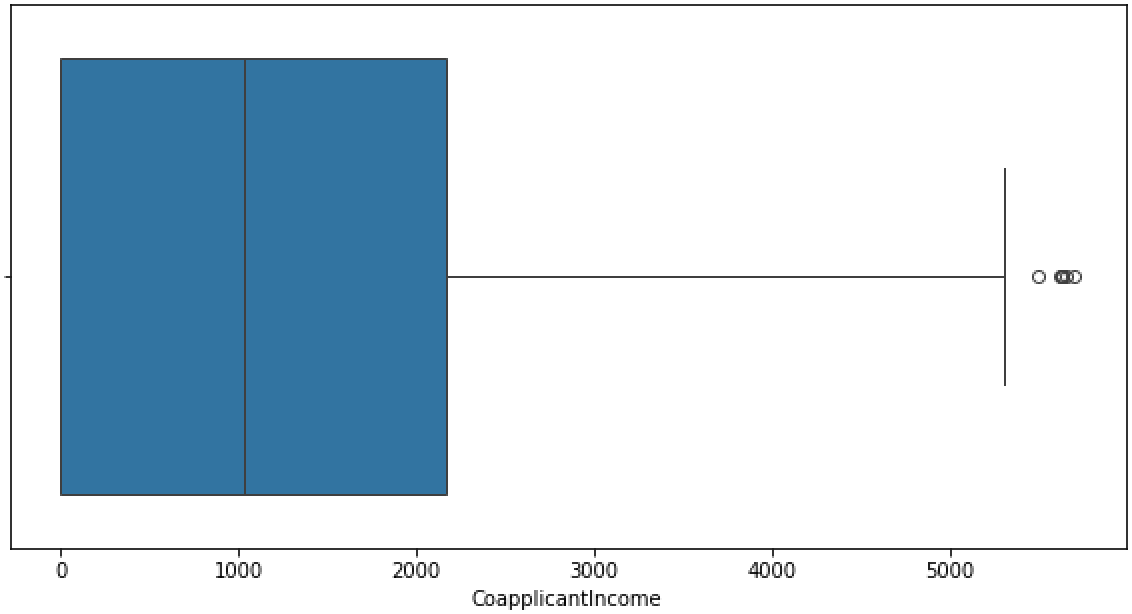
Out[22]:

```
(596, 13)
```

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

In [26]:

```
plt.figure(figsize=(10,5))
sns.boxplot(x='CoapplicantIncome', data=new_dataset)
plt.show()
```



So number of outliers have been decreased significantly

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

13.2.2 Removing Outlier through Z-Score Method 1

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

Out[28]:
```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

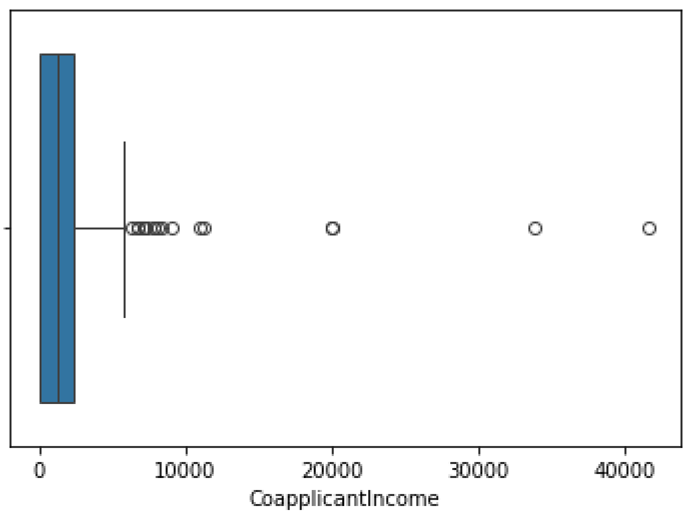
In [29]:
```
dataset.describe()
```

Out[29]:

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

In [30]:

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

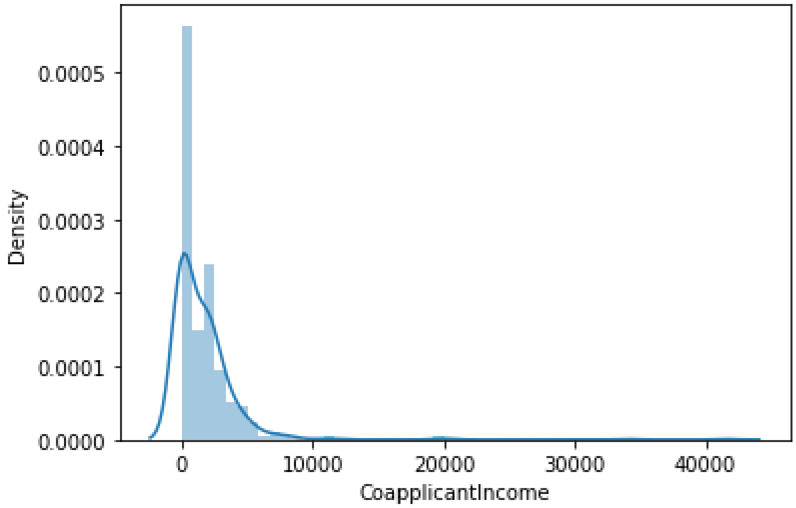


In [31]:

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

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

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

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

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

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

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



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

Out[36]:
```
(-7157.4993096454655, 10399.990905699668)
```

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

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

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

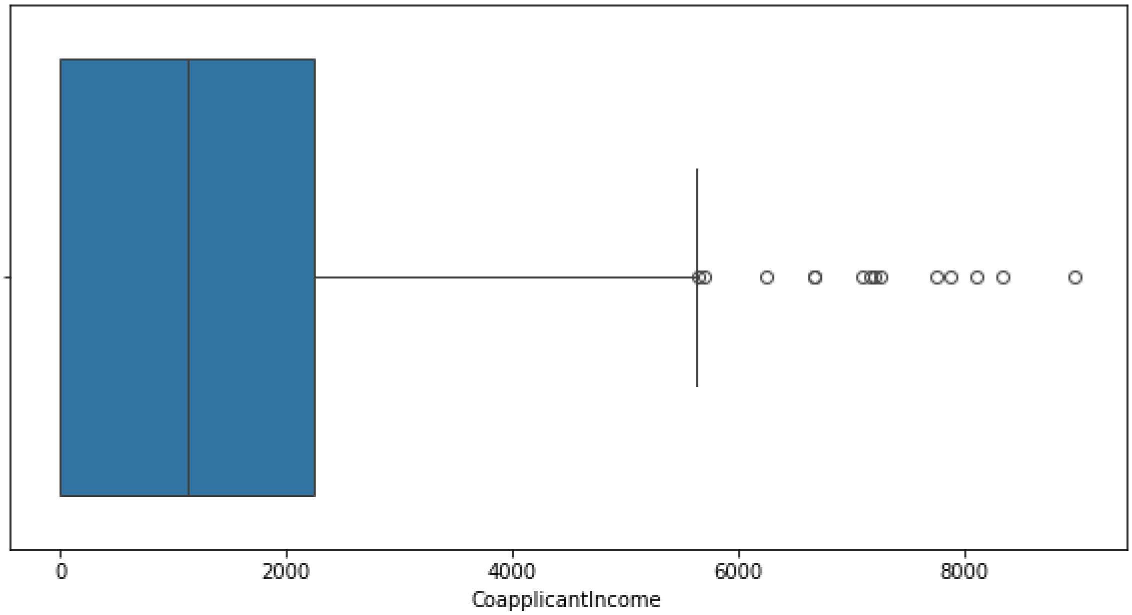
Out[44]:
```
((614, 13), (608, 13))
```

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

Out[45]:
```
6
```

So You can see 6 rows are deleted containing outliers

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



13.2.3 Removing Outlier through Z-Score Method 2

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

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

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

Out[52]:

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

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

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

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

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

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

In [ ]: