DETAILS:

Name: Gandi Priyanka Goud

College: St. Peter's Engineering College

Branch: AI&DS (Artificial Intelligence and Data Science)

Year: 2 Year

GitHub Link : https://github.com/gandipriyanka09

Double-click (or enter) to edit

Heart Attack Analysis and Prediction Using Logistic Regression

```
'''

Age : Age of the patient

Sex : Sex of the patient

exang: exercise induced angina (1 = yes; 0 = no)

ca: number of major vessels (0-3)

cp : Chest Pain type chest pain type

Value 1: typical angina
Value 2: atypical angina
Value 3: non-anginal pain
Value 4: asymptomatic
trtbps : resting blood pressure (in mm Hg)

chol : cholestoral in mg/dl fetched via BMI sensor

fbs : (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)

rest_ecg : resting electrocardiographic results

Value 0: normal
Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
thalach : maximum heart rate achieved

target : 0= less chance of heart attack 1= more chance of heart attack

'''
```

```
'\n\nAge : Age of the patient\n\nSex : Sex of the patient\n\nexang: exercise induced angina (1 = yes; 0 = no)\n\nca: number of m
ssels (0-3)\n\ncp : Chest Pain type chest pain type\n\nValue 1: typical angina\nValue 2: atypical angina\nValue 3: non-anginal p
lue 4: asymptomatic\ntrtbps : resting blood pressure (in mm Hg)\n\nchol : cholestoral in mg/dl fetched via BMI sensor\n\nfbs : (
blood sugar > 120 mg/dl) (1 = true; 0 = false)\n\nrest_ecg : resting electrocardiographic results\n\nValue 0: normal\nValue 1: h
T-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)\nValue 2: showing probable or definite l
```

```
#1. DataSet = 'https://raw.githubusercontent.com/gandipriyanka09/DataSets/main/heart.csv'
import pandas as pd
df = pd.read_csv("https://raw.githubusercontent.com/gandipriyanka09/DataSets/main/heart.csv")
df
```

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

303 rows × 14 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
```

```
 3   trtbps    303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalachh  303 non-null    int64
 8   exng      303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slp       303 non-null    int64
 11  caa       303 non-null    int64
 12  thall     303 non-null    int64
 13  output    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

```
df.shape #303 - rows and 14 - columns
```

```
(303, 14)
```

```
df.size
```

```
4242
```

```
df.head(10)
```

```
df.tail(10)
```

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
293	67	1	2	152	212	0	0	150	0	0.8	1	0	3	0
294	44	1	0	120	169	0	1	144	1	2.8	0	0	1	0
295	63	1	0	140	187	0	0	144	1	4.0	2	2	3	0
296	63	0	0	124	197	0	1	136	1	0.0	1	0	2	0
297	59	1	0	164	176	1	0	90	0	1.0	1	2	1	0
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

```
#To find how much people lie under which age
df.value_counts(['age'])
```

```
age
58     19
57     17
54     16
59     14
52     13
51     12
62     11
60     11
44     11
56     11
41     10
64     10
63      9
67      9
65      8
```

55	8
61	8

```
53      8
45      8
43      8
42      8
50      7
66      7
48      7
46      7
49      5
47      5
70      4
39      4
68      4
35      4
69      3
40      3
38      3
71      3
37      2
34      2
76      1
29      1
74      1
77      1
dtype: int64
```

```
#To find how many males and females exist
df.value_counts(['sex'])
```

```
sex
1      207
0       96
dtype: int64
```

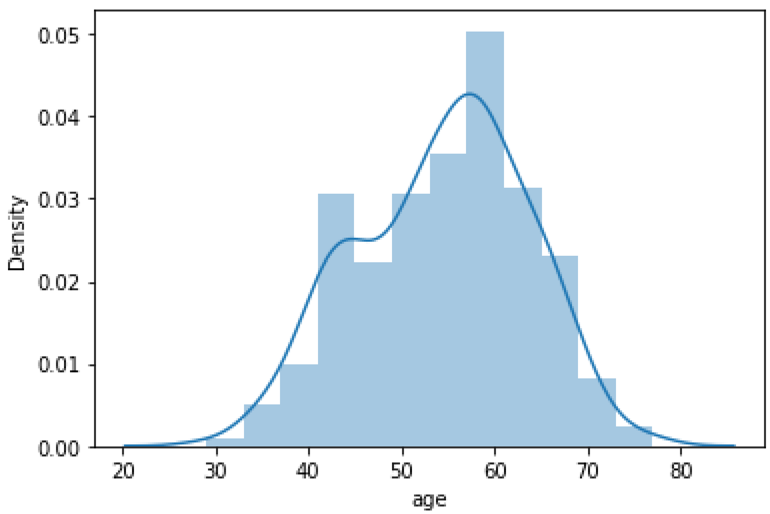
```
#
df.groupby(['sex','age']).size()
```

```
sex  age
0    34     1
     35     1
     37     1
     39     2
     41     4
           ..
```

```
1    67     6
     68     3
     69     2
     70     4
     77     1
Length: 73, dtype: int64
```

```
#Visualization

import seaborn as sns
sns.distplot(df['age'])
```

```
<ipython-input-11-1ef522f2aadf>:4: 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(df['age'])
<Axes: xlabel='age', ylabel='Density'>
```




```
#Chest Pain
ChestPainType = df['cp'].value_counts()
ChestPainType
```

```
0    143
2     87
1     50
3     23
Name: cp, dtype: int64
```

```
#checking how many null values are there in fasting blood sugar(fbs)
df[df['fbs'] == 0]
```

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
5	57	1	0	140	192	0	1	148	0	0.4	1	0	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
296	63	0	0	124	197	0	1	136	1	0.0	1	0	2	0
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

258 rows × 14 columns

```
#divide the data into i/p and o/p
#input - All the columns except the output column
#output - output
x = df.iloc[:,:-1].values
y=df.iloc[:, -1].values
```

```
#TRAIN and TEST VARIABLES
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state = 0)
```

```
print(x.shape)
print(x_train.shape)
print(x_test.shape)
```

```
(303, 13)
(227, 13)
(76, 13)
```

```
print(y.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(303,)
(227,)
(76,)
```

```
#We are applying logistic regression because the predicted output will be either 0 or 1
#0= less chance of heart attack 1= more chance of heart attack
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
```

```
#Fitting the model
model.fit(x_train,y_train)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (stat
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

▾ LogisticRegression

LogisticRegression()

```
#Predict the output
y_pred = model.predict(x_test)
y_pred #PREDCITED VALUES
```

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

```
y_test #actual values
```

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

```
#Accuracy
from sklearn.metrics import accuracy_score
accuracy_score(y_pred,y_test)* 100
```

```
84.21052631578947
```

```
#Individual Prediction
a = scaler.transform([[57,1,0,140,192,0,1,148,0,0.4,1,0,1]])
```

```
model.predict(a)
```

```
b = scaler.transform([[63,0,0,124,197,0,1,136,1,0.0,1,0,2]])
```

```
model.predict(b)
```

MAJOR PROJECT – 1(II)

Home Loan Approval using Multi Linear Regression

```
#HOME LOAN PREDICTION USING MULTI LINEAR REGRESSION
```

Home Loan Approval

```
#Creating the DATASET
import pandas as pd
df =  pd.read_csv("https://raw.githubusercontent.com/gandipriyanka09/DataSets/main/loan_test.csv")
df
```

	Gender	Married	Dependents	Education	Self_Employed	Applicant_Income	Coapplicant_Income	Loan_Amount	Term	Credit_His
0	Male	Yes	0	Graduate	No	572000	0	11000000	360.0	
1	Male	Yes	1	Graduate	No	307600	150000	12600000	360.0	
2	Male	Yes	2	Graduate	No	500000	180000	20800000	360.0	
3	Male	Yes	2	Graduate	No	234000	254600	10000000	360.0	
4	Male	No	0	Not Graduate	No	327600	0	7800000	360.0	
...	...	...	...	...	...	...	...	...	...	
362	Male	Yes	3+	Not Graduate	Yes	400900	177700	11300000	360.0	
363	Male	Yes	0	Graduate	No	415800	70900	11500000	360.0	
364	Male	No	0	Graduate	No	325000	199300	12600000	360.0	
365	Male	Yes	0	Graduate	No	500000	239300	15800000	360.0	
366	Male	No	0	Graduate	Yes	920000	0	9800000	180.0	

367 rows × 11 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 367 entries, 0 to 366
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Gender              356 non-null    object 
 1   Married             367 non-null    object 
 2   Dependents          357 non-null    object 
 3   Education           367 non-null    object 
 4   Self_Employed       344 non-null    object 
 5   Applicant_Income    367 non-null    int64  
 6   Coapplicant_Income  367 non-null    int64  
 7   Loan_Amount         367 non-null    int64  
 8   Term                361 non-null    float64
 9   Credit_History      338 non-null    float64
 10  Area                367 non-null    object 
dtypes: float64(2), int64(3), object(6)
memory usage: 31.7+ KB
```

```
df.shape
```

```
(367, 11)
```

```
df.size
```

```
4037
```

```
#Cleaning The Data / EDA
#dropping the unwanted columns i.e; dropping the dependents, Area and the education column
df = df.drop(["Dependents","Education","Area"], axis = 'columns')
df
```

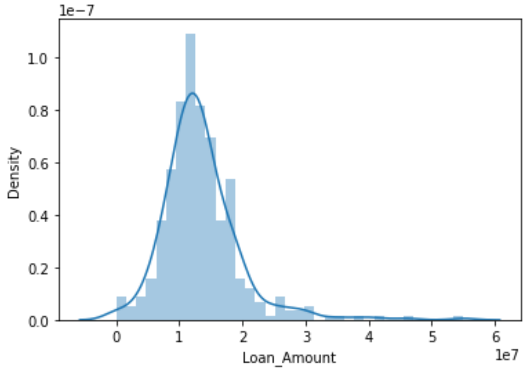
	Gender	Married	Self_Employed	Applicant_Income	Coapplicant_Income	Loan_Amount	Term	Credit_History
0	Male	Yes	No	572000	0	11000000	360.0	1.0
1	Male	Yes	No	307600	150000	12600000	360.0	1.0
2	Male	Yes	No	500000	180000	20800000	360.0	1.0
3	Male	Yes	No	234000	254600	10000000	360.0	NaN
4	Male	No	No	327600	0	7800000	360.0	1.0
...	...	...	...	...	...	...	...	...
362	Male	Yes	Yes	400900	177700	11300000	360.0	1.0

```
#Checking who got more loan approvals
df.value_counts(['Gender'])
#Males have got more loan approvals than females.
```

```
Gender
Male      286
Female     70
dtype: int64
```

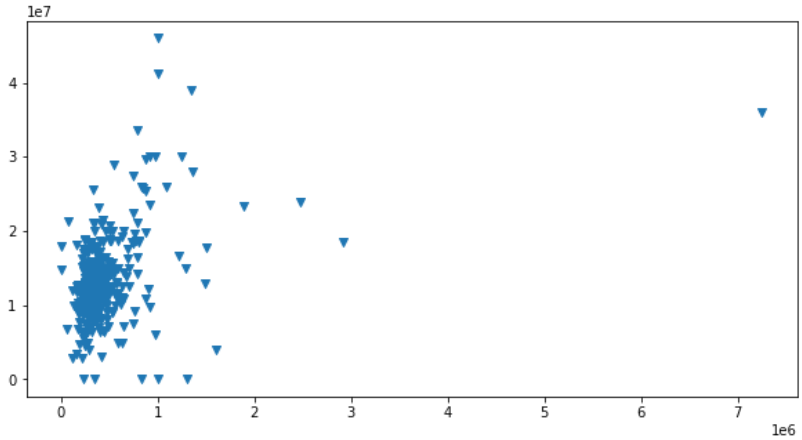
```
#Visualization
import seaborn as sns
sns.distplot(df['Loan_Amount'])
```

```
<ipython-input-8-27242e706083>:3:  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(df['Loan_Amount'])
<Axes: xlabel='Loan_Amount', ylabel='Density'>
```




```
import matplotlib.pyplot as plt
plt.figure(figsize = (10,5))
plt.scatter(df['Applicant_Income'],df['Loan_Amount'],marker  =  'v')
```

```
<matplotlib.collections.PathCollection  at  0x7f3babd1c250>
```

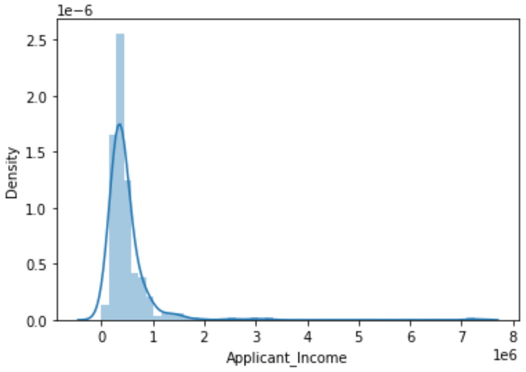


```
#Highest loan amount that has been approved
import numpy as np
np.max(df['Loan_Amount'])
```

```
55000000
```

```
sns.distplot(df['Applicant_Income'])
```

```
<ipython-input-10-8f9e00b3b52f>: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(df['Applicant_Income'])
<Axes: xlabel='Applicant_Income', ylabel='Density'>
```




```
#checking the null values
df.isna().sum()
```

```
Gender                11
Married                0
Self_Employed         23
Applicant_Income       0
Coapplicant_Income     0
Loan_Amount            0
Term                   6
Credit_History        29
dtype: int64
```

```
df.isnull().sum()
```

```
Gender                11
Married                0
Self_Employed         23
Applicant_Income       0
Coapplicant_Income     0
Loan_Amount            0
Term                   6
Credit_History        29
dtype: int64
```

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder ## converts the string to numerical format


le = LabelEncoder()
df['Married'] = le.fit_transform(df['Married']) #No - 0, yes - 1
df['Gender'] = le.fit_transform(df['Gender']) #1 - Male, 0- Female
df['Self_Employed'] = le.fit_transform(df['Self_Employed']) #No - 0, yes -1
df = df.dropna() #dropping the null values
df
```

	Gender	Married	Self_Employed	Applicant_Income	Coapplicant_Income	Loan_Amount	Term	Credit_History
0	1	1	0	572000	0	11000000	360.0	1.0
1	1	1	0	307600	150000	12600000	360.0	1.0
2	1	1	0	500000	180000	20800000	360.0	1.0
4	1	0	0	327600	0	7800000	360.0	1.0
5	1	1	1	216500	342200	15200000	360.0	1.0
...	...	...	...	...	...	...	...	...
361	1	1	0	226900	216700	9900000	360.0	1.0
365	1	1	0	500000	239300	15800000	360.0	1.0

```
#Dividing the data into input and output
x = df.drop('Loan_Amount', axis=1).loc[:,'Gender': ].values #input is all the columns except the Loan Amount so here we are dropping
x                                                          #that column and sliccing.
```

```
array([[1.000e+00, 1.000e+00, 0.000e+00, ..., 0.000e+00, 3.600e+02,
        1.000e+00],
       [1.000e+00, 1.000e+00, 0.000e+00, ..., 1.500e+05, 3.600e+02,
        1.000e+00],
       [1.000e+00, 1.000e+00, 0.000e+00, ..., 1.800e+05, 3.600e+02,
        1.000e+00],
       ...,
       [1.000e+00, 1.000e+00, 0.000e+00, ..., 7.090e+04, 3.600e+02,
        1.000e+00],
       [1.000e+00, 1.000e+00, 0.000e+00, ..., 2.393e+05, 3.600e+02,
        1.000e+00],
       [1.000e+00, 0.000e+00, 1.000e+00, ..., 0.000e+00, 1.800e+02,
        1.000e+00]])
```

```
y= df.iloc[:,5].values
y
```

```
array([11000000, 12600000, 20800000,  7800000, 15200000,  5900000,
       14700000, 28000000, 12300000,  9000000, 16200000, 16600000,
       12400000, 13100000, 20000000, 12600000, 30000000, 10000000,
        4800000,  2800000, 10100000, 12500000, 29000000, 14800000,
       27500000, 12500000,  7500000, 19200000, 15200000, 15800000,
       10100000, 17600000, 18500000,  9000000, 11600000, 13800000,
       10000000, 11000000,  9000000, 20000000,  8400000, 16200000,
       10800000, 18700000, 12400000, 12000000, 16000000,  3000000,
        9200000, 13000000, 13000000, 13400000, 17600000,  9000000,
       11000000, 12500000, 18900000, 10800000, 12500000, 13800000,
       13500000, 13000000, 18700000, 18800000,  9500000,  6500000,
       13900000, 23200000, 14400000, 15500000, 18600000,  5000000,
              0, 18500000, 16300000, 36000000, 14900000, 25700000,
       13100000, 10200000, 13500000,  9500000,  7700000, 20000000,
       39000000, 18500000, 10000000, 12300000, 11000000, 25600000,
       14000000,  6100000, 13100000,        0, 11600000,  5000000,
       20000000, 11900000, 12000000, 14000000, 16500000, 10800000,
        9300000, 10200000, 12200000, 16000000, 18000000, 10400000,
       21300000,  6500000, 14600000, 13500000, 18700000, 30000000,
       12000000,  7100000, 22500000,  7000000, 12400000, 13200000,
       10500000,  9000000,  8300000, 12500000, 14700000, 12000000,
       11000000, 15000000, 10000000, 13900000, 26000000, 15000000,
        9000000, 19900000, 13900000, 15000000, 18000000, 11300000,
       14800000, 11700000,  7200000, 12500000, 21400000, 13300000,
       18700000, 14300000, 20900000,  8400000, 11600000,  6500000,
       17000000, 12000000, 13500000,  9400000,  7900000, 11000000,
       13000000, 14300000, 15900000, 11000000, 16000000, 13100000,
       14300000, 16000000, 16500000, 11000000, 17300000, 15000000,
       23500000,        0, 33600000, 13200000,  9600000, 18000000,
       12800000, 41200000, 11600000, 11400000, 11500000, 10400000,
        8800000, 10800000,  9000000, 10800000,  7800000, 12300000,
       18700000, 14600000,  8000000, 10000000,  5500000, 20000000,
       15000000, 15000000, 15000000, 11800000, 21200000, 21200000,
       12500000, 14900000,  8000000, 15200000, 18700000,  7400000,
       10200000, 10000000, 13000000, 12500000, 13000000, 13800000,
        2800000,  9200000, 10400000, 17600000, 11700000, 10200000,
       10700000,  6600000, 10500000, 10500000, 10500000, 12500000,
        6400000, 15000000, 15000000,        0,  6400000,  4000000,
       14200000,  7000000, 13100000, 12000000, 11400000, 12300000,
        9200000, 16000000, 15100000, 17100000, 11000000, 23400000,
       18400000, 11200000, 11700000,  4900000,  9900000,  9900000,
       21200000, 24000000, 13000000,  9400000, 10800000, 14400000,
       11000000, 17600000, 18500000, 12200000, 12600000, 12200000,
       46000000, 29700000, 10600000, 14100000,        0, 17000000,
       14500000,  9000000,  8800000, 12800000,  8400000, 10800000,
        8300000, 11700000, 12800000,  7500000, 12500000, 17700000,
        6800000,  9600000, 18300000, 12100000, 16200000, 13200000,
       14700000, 15300000, 10400000, 14900000, 13400000, 16500000,
       12000000,  6700000, 12500000, 12000000, 14800000, 18100000,
```

```
         8000000,  4000000,  9000000,  9500000, 12200000, 15000000,
        15000000, 14300000, 30000000, 17100000, 11300000,  3500000,
         4600000, 11900000,  8700000, 16200000, 12200000, 18700000,
         8100000,  8000000, 17600000, 26000000, 13300000,  7000000,
        13500000, 13700000, 25400000, 10900000, 12000000, 15800000,
        19700000,  8500000,  6000000, 15200000,  9900000, 11300000,
        11500000, 15800000,  9800000])
```

```
#Training and Testing variables
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state = 0)
```

```
print(x.shape)
print(x_train.shape)
print(x_test.shape)
```

```
(333, 7)
(249, 7)
(84, 7)
```

```
print(y.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(333,)
(249,)
(84,)
```

```
#RUN a CLASSIFIER/REGRESSOR/CLUSTERER
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

```
#MODEL FITTING
model.fit(x_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
#PREDICTING THE OUTPUT
y_pred = model.predict(x_test)#By taking the input testing data , we are predicting the output
y_pred #PREDICTED VALUES
```

```
array([12149266.85082742,  12559430.74255981,  11610120.11411073,
       12562440.38350965,  13858522.41993775,  10356539.57959394,
       15402520.81716858,  13681040.65084167,  21326553.15988402,
       13432031.28620172,  13807962.55103809,  13181246.23580131,
       12877156.50850145,  16437675.07845588,  12422588.80930341,
       10522032.1522628 ,  10792669.40882197,   9265482.87931693,
       11713191.47224844,  11592939.14675685,  12966515.1846955 ,
       12374072.3502399 ,  13229470.2463038 ,  11164047.04519362,
       13026917.63038287,  13735682.76246626,  14459118.3056553 ,
       11303955.06304936,  10962846.48316486,  12685862.28468577,
        9987488.41333945,  16725067.86145146,  12857194.00566589,
       10850890.81162697,  14745049.82541761,  15013594.588404   ,
       16438987.63344254,  13352654.60274816,  14796942.86010502,
       14591317.94709039,  14522049.70103118,  13004030.88274316,
       11087606.84158628,  17270277.7569616 ,  12363433.84162338,
       11330684.44454458,  14731899.34119296,  15495281.830745   ,
       15193845.73812225,  11410255.60976532,  12261232.94727632,
       11576068.57681214,  11956997.155357  ,   9737233.64442014,
       12753666.08303529,  10421972.60318282,  11082365.62711066,
       12005371.68976955,  11518415.21758034,  10327630.7426217 ,
       12573238.50031338,  11403396.0338809 ,  14002660.63002731,
       10411890.80736789,  10511271.78943535,  14190867.87710631,
       13420373.12860707,  13250227.80137711,  10907575.49414926,
       13216737.23557314,  13464876.60710844,  11058743.27378616,
       11493086.25238179,  13257209.88050828,  13175188.02734144,
       13541512.13840101,  12748580.79545395,  13061433.78014636,
       11470787.02272466,  10786074.61312357,  12505285.64811191,
       13805455.16102991,  12811475.52193237,  12385129.24912806])
```

```
y_test#ACTUAL VALUES
```

```
array([18900000, 11000000,  6800000, 10800000, 13800000, 13000000,
       21200000, 16200000, 10200000, 12500000, 18700000, 14900000,
       18000000, 16000000, 12600000,  9000000, 10500000,  8300000,
       13200000, 13100000, 16300000,  9300000, 15000000, 12200000,
       11700000, 12500000, 12500000, 12500000,  7500000, 13900000,
```

```
       11900000, 41200000, 21300000,  2800000, 17600000, 12000000,
       33600000, 16500000, 23500000, 18800000, 17100000, 14200000,
        6500000, 28000000, 14700000,  9500000, 14400000, 29000000,
              0, 10200000, 11600000, 12500000, 15000000,  9500000,
       13500000,  9500000, 13100000, 12400000, 11000000,  7000000,
       14700000, 10000000, 22500000, 12500000, 10000000, 14300000,
        9900000, 17100000,  5500000,  8400000, 13000000, 15900000,
       14700000, 15000000, 18700000,  8800000, 12600000, 12200000,
       12300000,  6400000,  8000000, 17600000, 12600000, 15200000])
```

```
#Individual Prediction
model.predict([x_train[10]])
```

```
array([12821053.78113022])
```

```
model.predict([[1,1,0,600000,180000,360.0,1.0]])
```

```
array([14312559.16442409])
```

MAJOR PROJECT 2

Applying K-Means Clustering Algorithm for Credit Card Customer Data

```
#Major Project 2
```

Credit Card Customer Data

A Custom Dataset For Customer Segmentation Using Clustering Techniques

```
#Taking the data and creating the DATAFRAME
import pandas as pd
df = pd.read_csv("https://raw.githubusercontent.com/gandipriyanka09/DataSets/main/Credit%20Card%20Customer%20Data.csv")
df
```

	Sl_No	Customer Key	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_made
0	1	87073	100000	2	1	1	0
1	2	38414	50000	3	0	10	9
2	3	17341	50000	7	1	3	4
3	4	40496	30000	5	1	1	4
4	5	47437	100000	6	0	12	3
...	...	...	...	...	...	...	...
655	656	51108	99000	10	1	10	0
656	657	60732	84000	10	1	13	2
657	658	53834	145000	8	1	9	1
658	659	80655	172000	10	1	15	0
659	660	80150	167000	9	0	12	2

660 rows × 7 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 660 entries, 0 to 659
Data columns (total 7 columns):
```

```
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Sl_No                660 non-null    int64
 1   Customer Key         660 non-null    int64
 2   Avg_Credit_Limit     660 non-null    int64
 3   Total_Credit_Cards   660 non-null    int64
 4   Total_visits_bank    660 non-null    int64
 5   Total_visits_online  660 non-null    int64
 6   Total_calls_made     660 non-null    int64
dtypes: int64(7)
memory usage: 36.2 KB
```

```
df.size
```

```
4620
```

```
df.shape
```

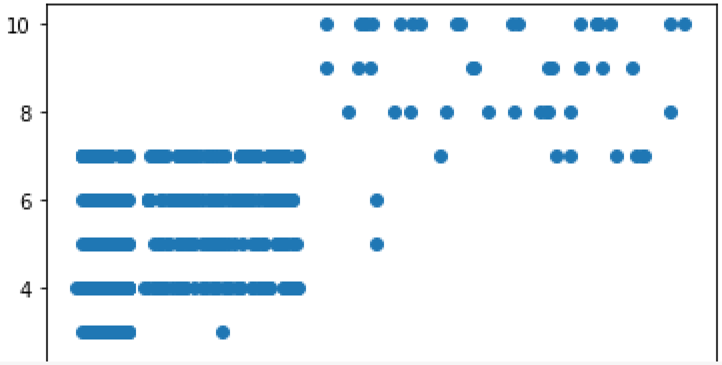
```
(660, 7)
```

```
#Input is Avg_Credit_Limit AND Total_Credit_Cards
#Divide the data into input
x = df.iloc[:,2:4].values
x
```

```
array([[100000,      2],
       [ 50000,      3],
       [ 50000,      7],
       ...,
       [145000,      8],
       [172000,     10],
       [167000,      9]])
```

```
#Visualization
import matplotlib.pyplot as plt
plt.scatter(df['Avg_Credit_Limit'],df['Total_Credit_Cards'])
#Here we got only 1 cluster before we apply any clustering technique because we got only 1 color.
```

```
<matplotlib.collections.PathCollection at 0x7f03309a5df0>
```



```
#Now our main task is to find the number of clusters(k). k = no of clusters
import numpy as np
np.sqrt(660)
#Here 660 is the total no of points here we are finding out the maximum range
#So here k values should be in the range 2 to 25 according to the output
#because only 1 cluster doesnt make any sense so thats where it starts with 2
```

```
25.69046515733026
```

```
#Finding out no of clusters there are 2 methods
#1. Elbow method
#2.Silhouette method
```

```
from sklearn.cluster import KMeans
k = range(2,26) #range is between 2 and 26

sse = [] #blank list later we will be appending the sum of squared error values

#for i in range(2,26)
for i in k:
  model_demo = KMeans(n_clusters = i, random_state = 0)   #algorithm
  model_demo.fit(x)
  sse.append(model_demo.inertia_) #.inertia_ calculates the sum of squared error
plt.scatter(k,sse)
plt.plot(k,sse)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
```

```
#we will consider the point at which the elbow is more prominent.
#Lets guess that the k = 2.
```

```
#2. Silhouette score method - which gives an accurate output
from sklearn.metrics import silhouette_score
k = range(2,26) #range is between 2 and 14

#for i in range(2,26)
for i in k:
  model_demo = KMeans(n_clusters = i, random_state = 0)   #algorithm
  model_demo.fit(x)
  y_pred = model_demo.predict(x)
  print(f"{i} Clusters, Score = {silhouette_score(x,y_pred)}")
  plt.bar(i,silhouette_score(x,y_pred))
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
2 Clusters, Score = 0.7701128291772962
3 Clusters, Score = 0.7166285119635797
4 Clusters, Score = 0.6987364959005717
5 Clusters, Score = 0.7085136674456746
6 Clusters, Score = 0.5994522098611428
7 Clusters, Score = 0.622337018566219
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
8 Clusters, Score = 0.6173961796832858
9 Clusters, Score = 0.6199398713490023
10 Clusters, Score = 0.6091587695264291
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
11 Clusters, Score = 0.62140448654897
12 Clusters, Score = 0.6006973525426734
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
13 Clusters, Score = 0.6025974058765664
14 Clusters, Score = 0.6022981620788393
15 Clusters, Score = 0.6014514817934343
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
16 Clusters, Score = 0.5986791091381775
17 Clusters, Score = 0.5724788560518181
18 Clusters, Score = 0.5924613667097922
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
19 Clusters, Score = 0.5947337148787429
20 Clusters, Score = 0.5891304425312066
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
21 Clusters, Score = 0.5893226367010791
22 Clusters, Score = 0.5750626530529814
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
```

```
#Conclusion: The no of clusters to be considered as 2 as its score is high i.e; 0.7701128291772962
```

```
#Applying clusterer
k = 2
from sklearn.cluster import KMeans
model = KMeans(n_clusters = k, random_state = 0)
model.fit(x)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change
  warnings.warn(
```

```
▾ KMeans
KMeans(n_clusters=2, random_state=0)
```

```
y = model.predict(x) #predicted output
y
```

```
array([1, 0, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
      dtype=int32)
```

```
y.size
```

```
660
```

```
x[y == 1,1]
#The first 1 is cluster no and the other 1 is the index no
```

```
array([ 2,  6,  5,  9,  9,  8,  9, 10,  8,  7,  7, 10,  9, 10, 10,  7,  8,
        9, 10,  9,  8, 10,  8,  7,  7,  8, 10, 10,  9,  8,  7,  8, 10, 10,
        9, 10, 10,  7,  9,  8, 10, 10,  9,  8, 10, 10, 10, 10,  8, 10,  9])
```

```
x[y == 0,1]
#The first 0 is cluster no and the other 1 is the index no
```

```
array([3, 7, 5, 3, 3, 2, 4, 4, 3, 1, 1, 2, 2, 2, 2, 2, 4, 3, 2, 4, 1, 2,
       1, 3, 2, 2, 4, 3, 3, 1, 1, 4, 2, 3, 2, 4, 2, 4, 1, 4, 2, 4, 4, 1,
       1, 4, 3, 1, 2, 2, 3, 2, 1, 1, 2, 1, 4, 4, 1, 1, 3, 1, 1, 2, 4, 1,
       1, 1, 3, 3, 3, 2, 3, 1, 3, 2, 1, 2, 3, 1, 3, 3, 1, 4, 4, 1, 4, 2,
       1, 1, 1, 2, 1, 3, 2, 1, 2, 1, 1, 4, 1, 2, 2, 2, 2, 3, 4, 4, 2, 4,
       4, 4, 2, 1, 4, 3, 3, 4, 3, 2, 1, 2, 2, 4, 3, 4, 3, 3, 3, 4, 3, 4,
```

```
       3, 3, 2, 1, 1, 3, 2, 4, 2, 4, 3, 2, 2, 1, 2, 3, 1, 3, 4, 4, 1, 3,
       1, 3, 1, 1, 3, 2, 3, 4, 1, 4, 4, 1, 3, 2, 2, 2, 2, 4, 1, 2, 3, 1,
       2, 1, 3, 3, 1, 4, 3, 1, 2, 2, 3, 3, 2, 3, 2, 2, 2, 2, 1, 1, 4, 3,
       1, 1, 2, 2, 1, 2, 4, 2, 1, 4, 3, 3, 1, 4, 4, 3, 4, 3, 2, 1, 2, 4,
       2, 4, 4, 1, 2, 7, 7, 4, 6, 5, 7, 6, 7, 7, 6, 5, 5, 4, 4, 4, 6, 7,
       4, 4, 7, 6, 6, 6, 6, 4, 6, 4, 5, 7, 6, 6, 4, 4, 5, 5, 4, 4, 7, 7,
       6, 6, 5, 4, 7, 6, 6, 5, 6, 4, 4, 7, 6, 6, 7, 6, 4, 7, 5, 4, 7, 5,
       6, 6, 7, 4, 5, 6, 6, 4, 7, 6, 6, 6, 6, 4, 4, 4, 6, 6, 5, 4, 6, 4,
       6, 7, 4, 7, 4, 6, 5, 6, 6, 6, 7, 6, 4, 4, 4, 4, 5, 4, 7, 6, 6, 7,
       5, 7, 6, 4, 7, 6, 6, 6, 4, 6, 5, 5, 7, 6, 6, 4, 7, 5, 7, 5, 5, 7,
       5, 6, 4, 5, 4, 6, 6, 5, 5, 6, 7, 7, 4, 4, 6, 5, 5, 6, 4, 6, 6, 5,
       5, 6, 7, 7, 4, 6, 5, 4, 7, 5, 7, 5, 4, 4, 4, 4, 4, 7, 4, 6, 5, 6,
       7, 7, 4, 4, 4, 7, 7, 6, 5, 5, 7, 5, 4, 6, 7, 4, 7, 5, 5, 5, 6, 5,
       5, 6, 5, 5, 6, 7, 7, 4, 4, 6, 6, 6, 6, 4, 4, 4, 6, 7, 7, 5, 6, 4,
       6, 6, 7, 5, 5, 4, 4, 7, 4, 5, 4, 4, 6, 6, 5, 7, 7, 6, 4, 4, 4, 4,
       5, 6, 4, 5, 6, 5, 6, 6, 6, 4, 5, 6, 7, 6, 7, 6, 7, 4, 5, 5, 7, 4,
       7, 6, 7, 4, 4, 6, 6, 4, 7, 5, 4, 4, 6, 5, 6, 6, 6, 6, 5, 6, 6, 7,
       4, 6, 6, 6, 5, 5, 4, 7, 4, 7, 7, 6, 4, 7, 6, 7, 5, 4, 5, 4, 6, 4,
       4, 4, 6, 6, 4, 4, 7, 7, 4, 7, 7, 5, 5, 6, 7, 7, 4, 4, 7, 7, 7, 7,
       7, 5, 4, 4, 7, 6, 7, 6, 7, 7, 5, 7, 6, 6, 7, 6, 6, 4, 7, 7, 7, 4,
       5, 6, 5, 7, 4, 4, 6, 6, 6, 6, 7, 7, 4, 4, 4, 6, 5, 5, 4, 4, 7, 6,
       7, 5, 6, 7, 5, 7, 6, 7, 5, 5, 6, 4, 4, 7, 7])
```

```
np.unique(y,return_counts = True) #we get to know that there are 0 - 609, 1 - 51
```

```
(array([0, 1], dtype=int32), array([609,  51]))
```

```
#Final visualization
plt.figure(figsize = (10,5))
for i in range(k):
  plt.scatter(x[y == i,0], x[y == i,1], label = f'Cluster {i}')
plt.scatter(model.cluster_centers_[:,0], model.cluster_centers_[:,1],s = 300,color = 'lime')
plt.legend()
```

<matplotlib.legend.Legend at 0x7f03308412b0>