Heart Disease Prediction

Group No. 6

Achanta Sai Krishna – CB.EN.U4ELC20004
Kuralanbu S – CB.EN.U4ELC20033
Vimal Dharshan N – CB.EN.U4ELC20081

AMRITA VISHWA VIDYAPEETHAM

CA1 Summary

- We have researched about Knn classifier and Normalization inferred the following:
  - The Minkowski Distance is a metric in a normed vector space that uses both Manhattan and Euclidean distance in a generalized form for calculation.
  - Splitting the dataset into training and testing, where 70% for training and 30% for testing.
  - Cross Validation(cv): Re-sampling procedure used to evaluate a model.
  - For various values of K, the accuracy rates change, and the best case was found by plotting all the values.
  - The confusion matrix is used to determine the performance of the classification models for a given set of test data.
  - Normalization is a scaling technique where all the data in the dataset is scaled between a range that is 0 and 1.

References

1. Dataset:

   https://www.kaggle.com/datasets/aasheesh200/framingham-heart-study-dataset

2. Reference:

   https://www.geeksforgeeks.org/ml-principal-component-analysispca/
   https://www.elastic.co/guide/en/elasticsearch/reference/current/coerce.html

Problem Formulation

Objective: To identify Heart Disease of a patient based on the given features

Dataset details:

- No. of rows: 4239
- No. of columns: 17
- No. of Class: 02
- Method of data collection is unknown

Assumptions:

- From the link mentioned for dataset, “framingham.csv” was considered for solving
  - In features, “Male” is changed to “Sex”
  - In features, “TenYearCHD” is changed to “target”.
  - In features, “Education” is changed to “Chest Pain Type”.

Problem Formulation

Assumptions:

- Missing data were filled with the mean of the rest of the corresponding data.

Link to full code mentioned in slides:

https://colab.research.google.com/drive/1LQE3ZlfJMKcVebWDTYiN2gSTh9R_c3vy#scrollTo=rn2cTWr-j_xu

https://colab.research.google.com/drive/1mq0485r2WlABTFt6FzbX85SF2COhHe4w#scrollTo=Uu61gVSazlNK

https://colab.research.google.com/drive/1APUqgwjLrax2i9cOLqrIHnJM_8NiDZ4h#scrollTo=NHpdMRJbp8Lo

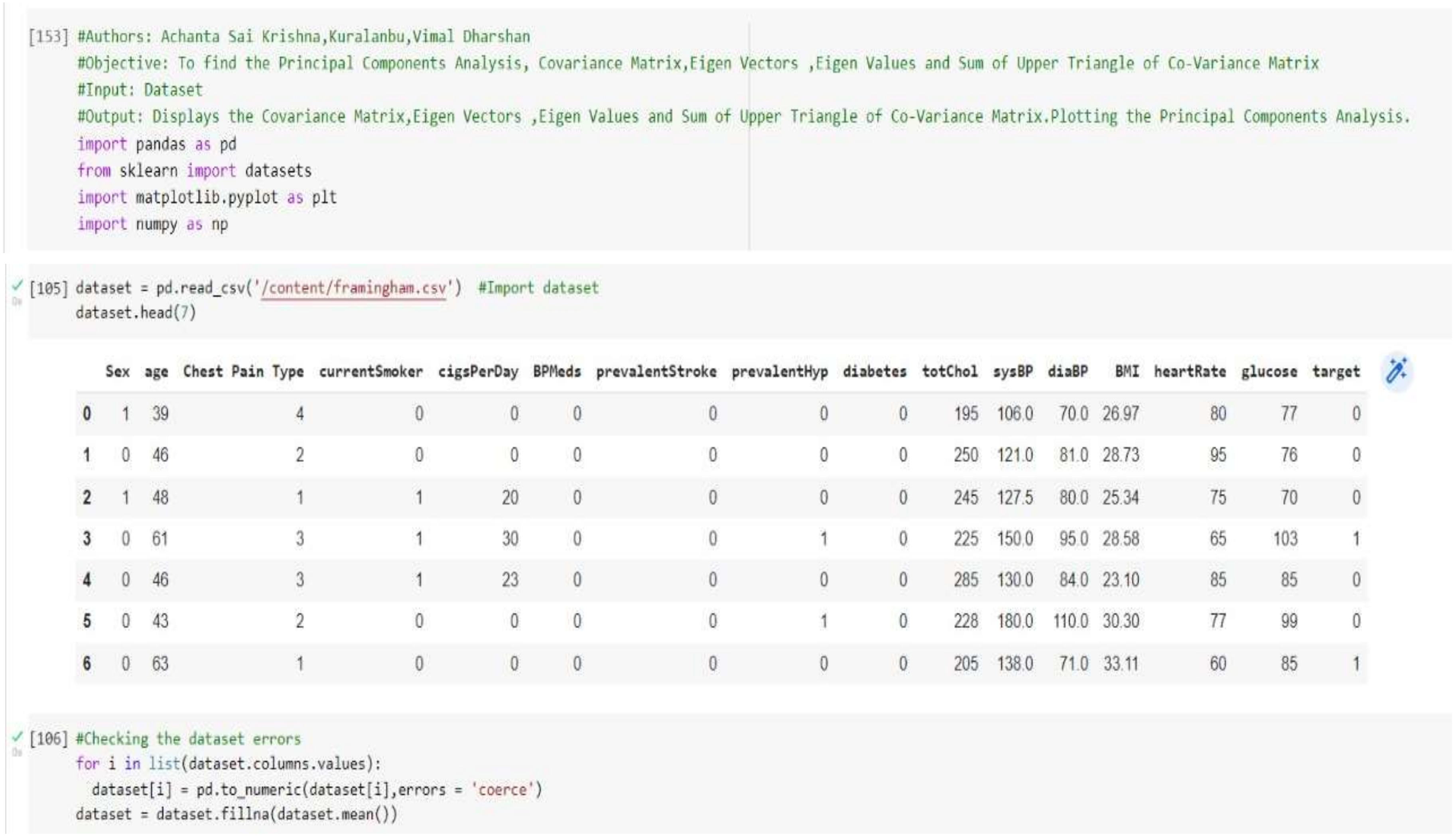
Feature Description

- sex – The person's sex (0 = female; 1 = male)
- age – The person's age in years
- Chest Pain Type – 1: Typical Angina, 2: Atypical Angina, 3:Non-Angina Pain, 4:Asymptomatic
- currentSmoker – The person is currently smoking (0 = false; 1 = true)
- cigsPerDay – Amount of cigarettes smoked per day by a person
- BPMeds – The person is taking medicine for blood pressure (0 = false; 1 = true)
- prevalentStroke – The person has a common stroke (0 = false; 1 = true)
- prevalentHyp – The person has common hypertension (0 = false; 1 = true)
- diabetes – The person has diabetes (0 = false; 1 = true)

Feature Description

- totChol – Total cholesterol of a person (in mg/dl)
- sysBP – Systolic blood pressure of a person (in mm Hg)
- diaBP – Diastolic blood pressure of a person (in mm Hg)
- BP – Total blood pressure of a person (sysBP/diaBP)
- BMI – Body Mass Index of a person (kg/m^2)
- heartRate – Total heart rate achieved by a person (bpm)
- glucose – Fasting blood sugar level of a person (mg/dl)
- target – Heart Disease of a person (0 = false; 1 = true)

PCA

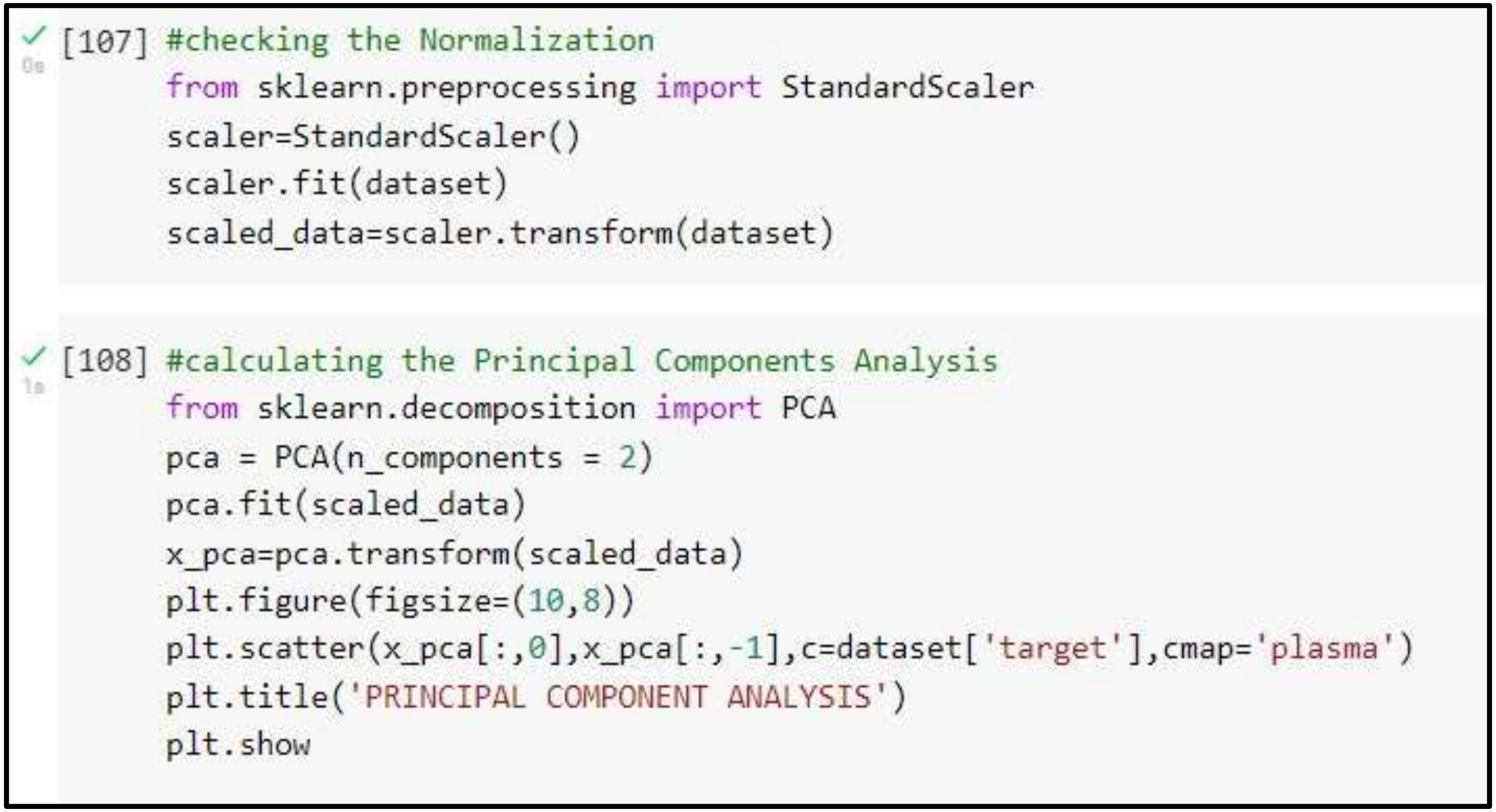


```
[153] #Authors: Achanta Sai Krishna,Kuralanbu,Vimal Dharshan
      #Objective: To find the Principal Components Analysis, Covariance Matrix,Eigen Vectors ,Eigen Values and Sum of Upper Triangle of Co-Variance Matrix
      #Input: Dataset
      #Output: Displays the Covariance Matrix,Eigen Vectors ,Eigen Values and Sum of Upper Triangle of Co-Variance Matrix.Plotting the Principal Components Analysis.
      import pandas as pd
      from sklearn import datasets
      import matplotlib.pyplot as plt
      import numpy as np
```

```
[105] dataset = pd.read_csv('/content/framingham.csv')  #Import dataset
      dataset.head(7)
```

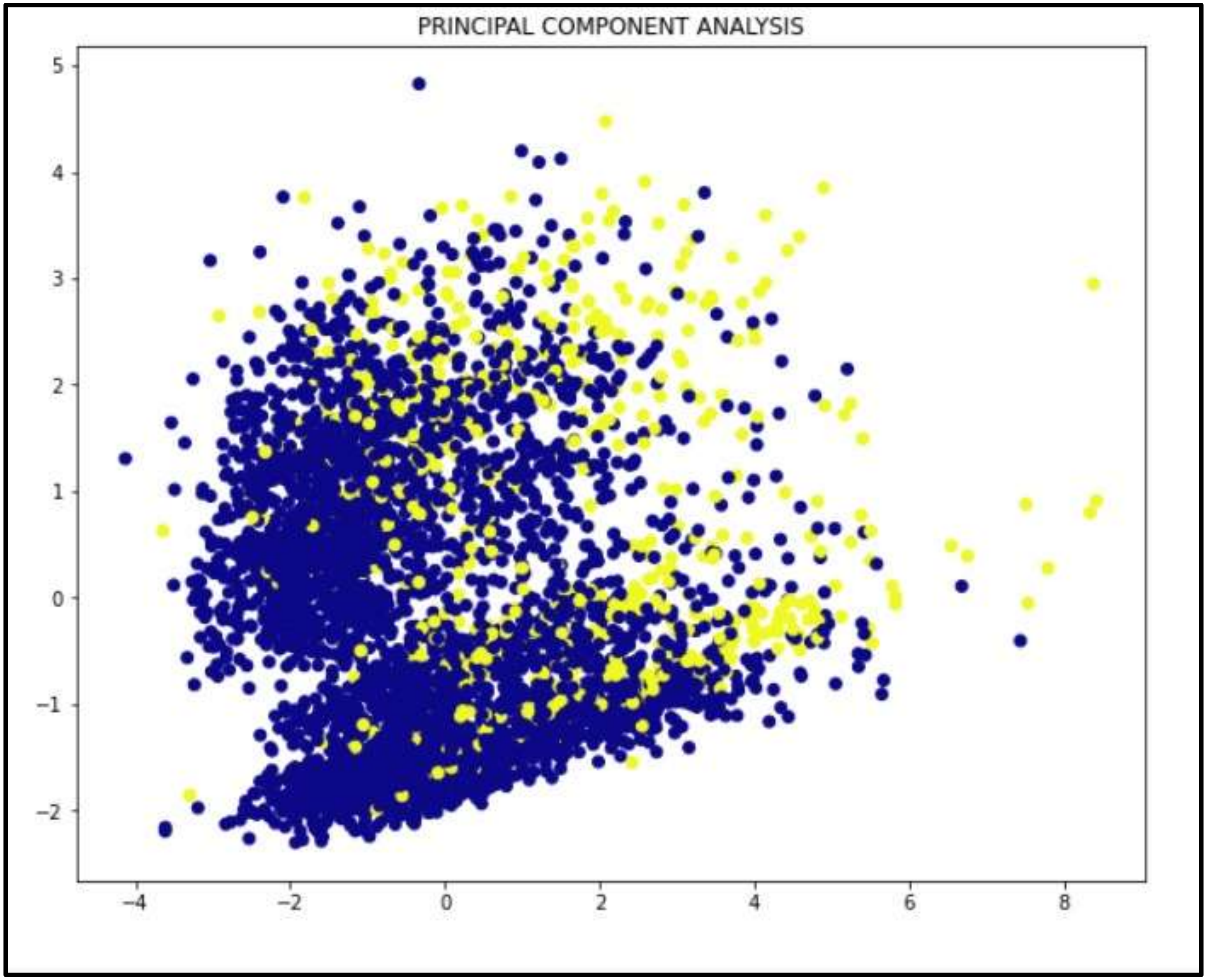
	Sex	age	Chest Pain Type	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	target
0	1	39	4	0	0	0	0	0	0	195	106.0	70.0	26.97	80	77	0
1	0	46	2	0	0	0	0	0	0	250	121.0	81.0	28.73	95	76	0
2	1	48	1	1	20	0	0	0	0	245	127.5	80.0	25.34	75	70	0
3	0	61	3	1	30	0	0	1	0	225	150.0	95.0	28.58	65	103	1
4	0	46	3	1	23	0	0	0	0	285	130.0	84.0	23.10	85	85	0
5	0	43	2	0	0	0	0	1	0	228	180.0	110.0	30.30	77	99	0
6	0	63	1	0	0	0	0	0	0	205	138.0	71.0	33.11	60	85	1

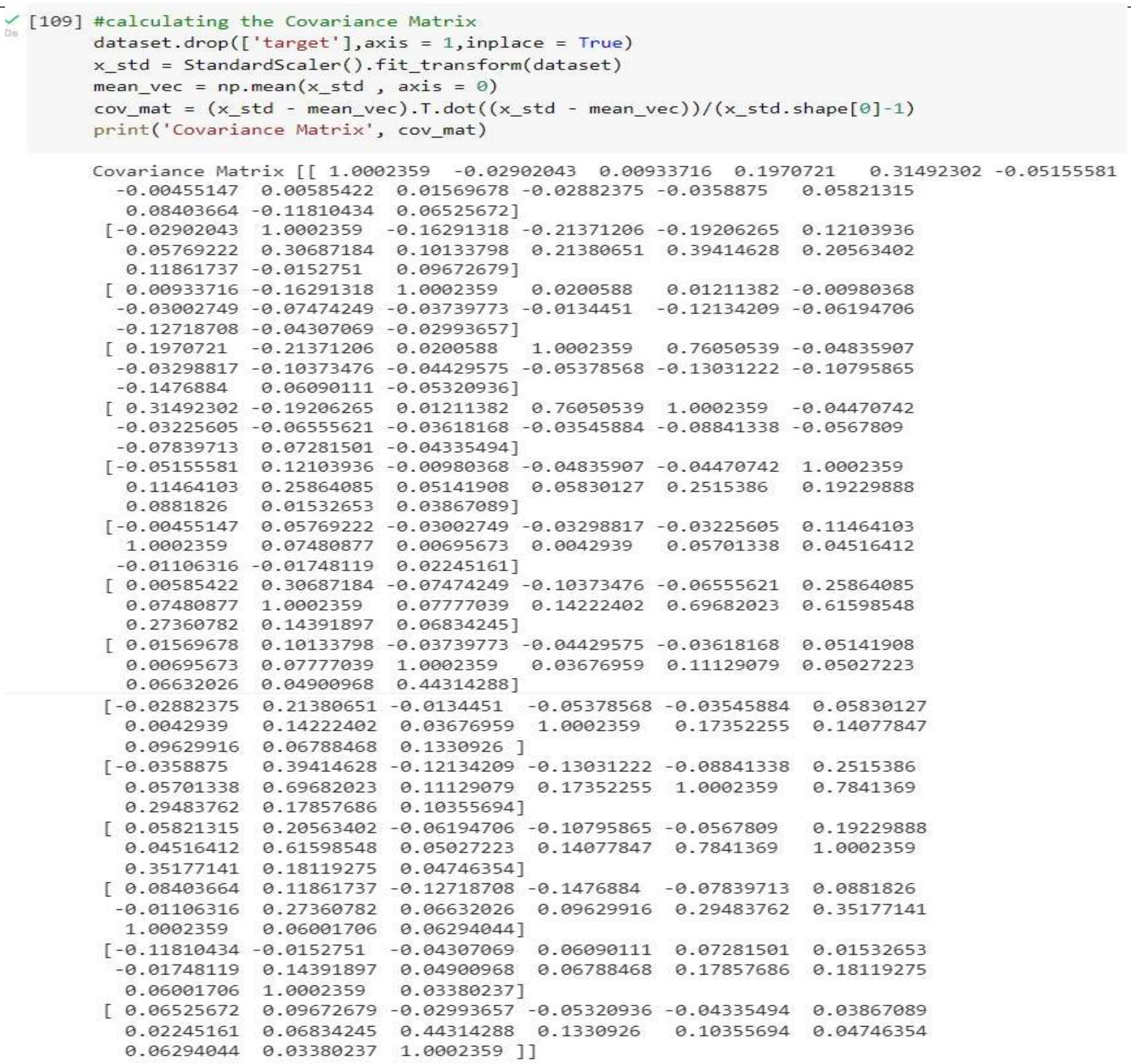
```
[106] #Checking the dataset errors
      for i in list(dataset.columns.values):
        dataset[i] = pd.to_numeric(dataset[i],errors = 'coerce')
      dataset = dataset.fillna(dataset.mean())
```



```
✓ [107] #checking the Normalization
0s      from sklearn.preprocessing import StandardScaler
        scaler=StandardScaler()
        scaler.fit(dataset)
        scaled_data=scaler.transform(dataset)
```

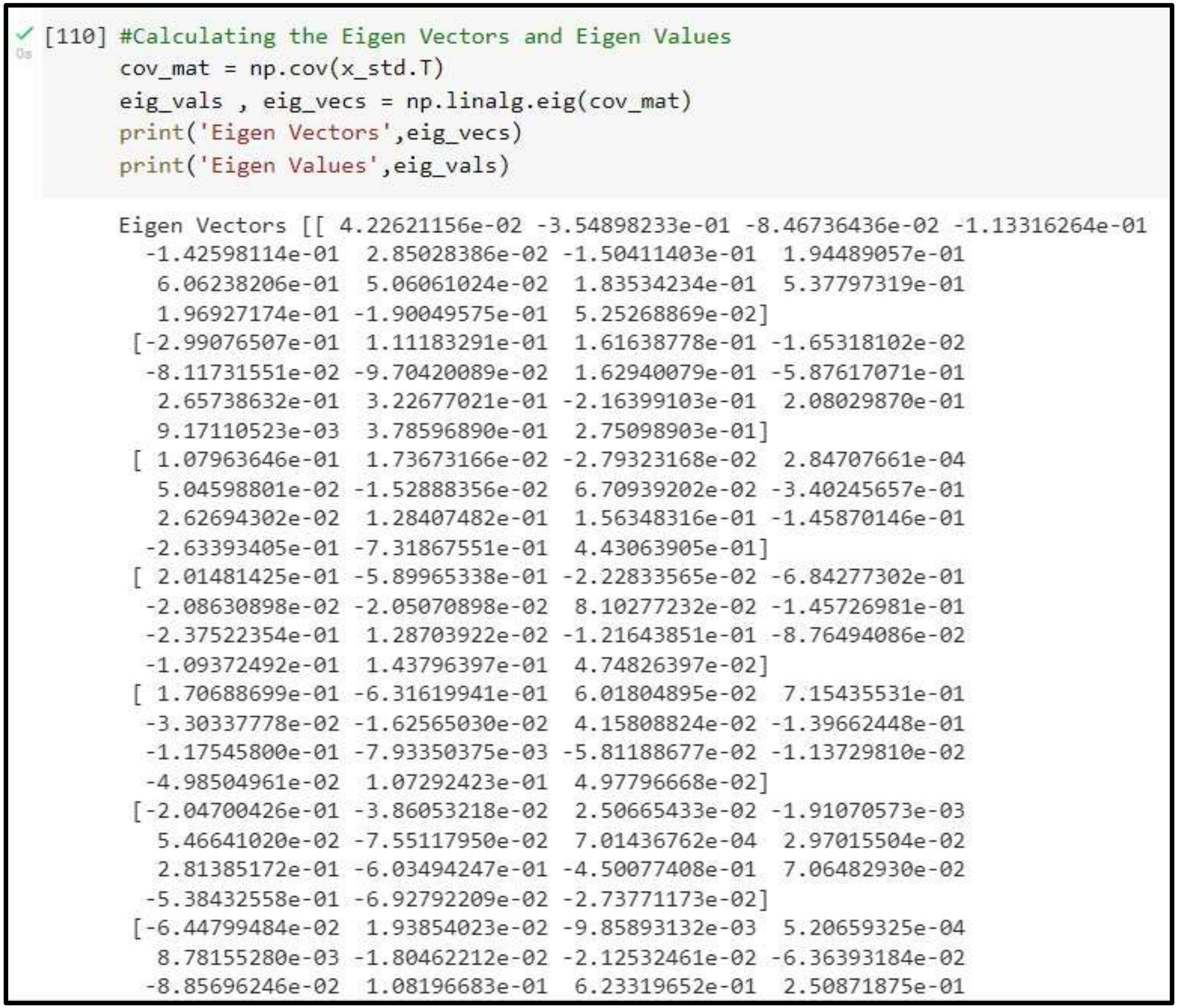
```
✓ [108] #calculating the Principal Components Analysis
1s      from sklearn.decomposition import PCA
        pca = PCA(n_components = 2)
        pca.fit(scaled_data)
        x_pca=pca.transform(scaled_data)
        plt.figure(figsize=(10,8))
        plt.scatter(x_pca[:,0],x_pca[:,-1],c=dataset['target'],cmap='plasma')
        plt.title('PRINCIPAL COMPONENT ANALYSIS')
        plt.show
```


PRINCIPAL COMPONENT ANALYSIS
5
4
3
2
1
0
−1
−2
−4
−2
0
2
4
6
8

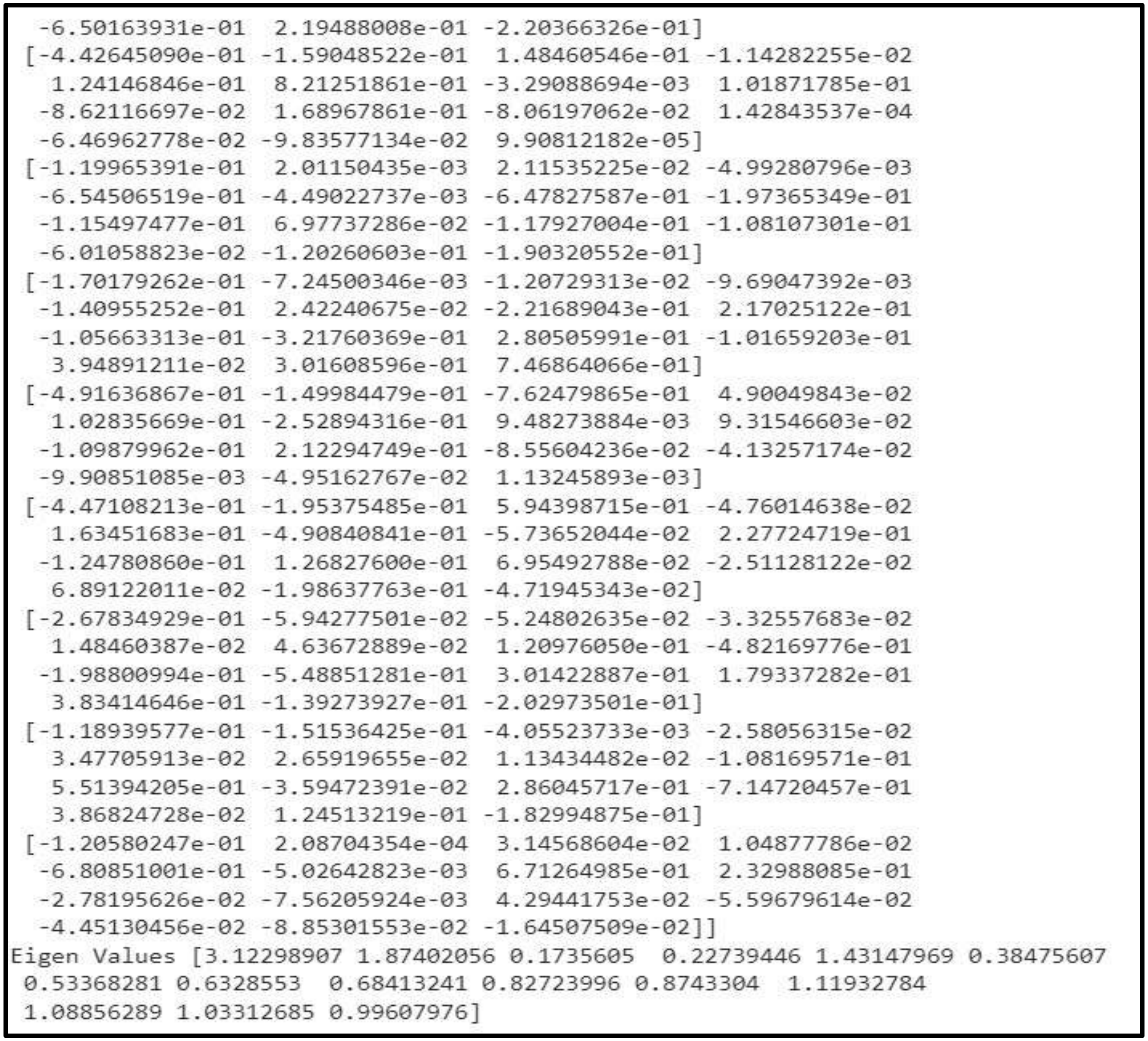


```
[109] #calculating the Covariance Matrix
      dataset.drop(['target'],axis = 1,inplace = True)
      x_std = StandardScaler().fit_transform(dataset)
      mean_vec = np.mean(x_std , axis = 0)
      cov_mat = (x_std - mean_vec).T.dot((x_std - mean_vec))/(x_std.shape[0]-1)
      print('Covariance Matrix', cov_mat)
```

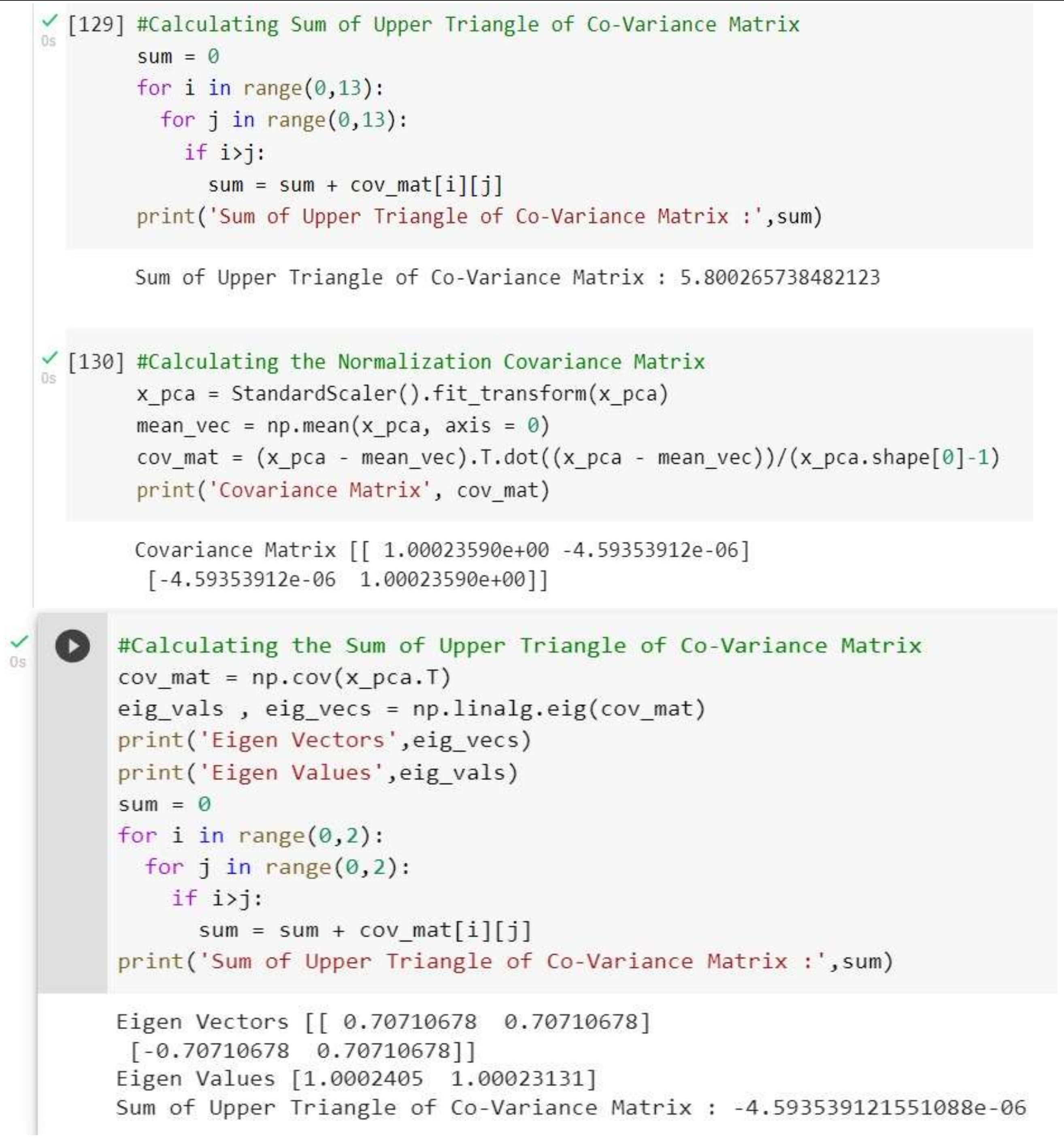
```
Covariance Matrix [[ 1.0002359  -0.02902043  0.00933716  0.1970721   0.31492302 -0.05155581
  -0.00455147  0.00585422  0.01569678 -0.02882375 -0.0358875   0.05821315
   0.08403664 -0.11810434  0.06525672]
 [-0.02902043  1.0002359  -0.16291318 -0.21371206 -0.19206265  0.12103936
   0.05769222  0.30687184  0.10133798  0.21380651  0.39414628  0.20563402
   0.11861737 -0.0152751   0.09672679]
 [ 0.00933716 -0.16291318  1.0002359   0.0200588   0.01211382 -0.00980368
  -0.03002749 -0.07474249 -0.03739773 -0.0134451  -0.12134209 -0.06194706
  -0.12718708 -0.04307069 -0.02993657]
 [ 0.1970721  -0.21371206  0.0200588   1.0002359   0.76050539 -0.04835907
  -0.03298817 -0.10373476 -0.04429575 -0.05378568 -0.13031222 -0.10795865
  -0.1476884   0.06090111 -0.05320936]
 [ 0.31492302 -0.19206265  0.01211382  0.76050539  1.0002359  -0.04470742
  -0.03225605 -0.06555621 -0.03618168 -0.03545884 -0.08841338 -0.0567809
  -0.07839713  0.07281501 -0.04335494]
 [-0.05155581  0.12103936 -0.00980368 -0.04835907 -0.04470742  1.0002359
   0.11464103  0.25864085  0.05141908  0.05830127  0.2515386   0.19229888
   0.0881826   0.01532653  0.03867089]
 [-0.00455147  0.05769222 -0.03002749 -0.03298817 -0.03225605  0.11464103
   1.0002359   0.07480877  0.00695673  0.0042939   0.05701338  0.04516412
  -0.01106316 -0.01748119  0.02245161]
 [ 0.00585422  0.30687184 -0.07474249 -0.10373476 -0.06555621  0.25864085
   0.07480877  1.0002359   0.07777039  0.14222402  0.69682023  0.61598548
   0.27360782  0.14391897  0.06834245]
 [ 0.01569678  0.10133798 -0.03739773 -0.04429575 -0.03618168  0.05141908
   0.00695673  0.07777039  1.0002359   0.03676959  0.11129079  0.05027223
   0.06632026  0.04900968  0.44314288]
 [-0.02882375  0.21380651 -0.0134451  -0.05378568 -0.03545884  0.05830127
   0.0042939   0.14222402  0.03676959  1.0002359   0.17352255  0.14077847
   0.09629916  0.06788468  0.1330926 ]
 [-0.0358875   0.39414628 -0.12134209 -0.13031222 -0.08841338  0.2515386
   0.05701338  0.69682023  0.11129079  0.17352255  1.0002359   0.7841369
   0.29483762  0.17857686  0.10355694]
 [ 0.05821315  0.20563402 -0.06194706 -0.10795865 -0.0567809   0.19229888
   0.04516412  0.61598548  0.05027223  0.14077847  0.7841369   1.0002359
   0.35177141  0.18119275  0.04746354]
 [ 0.08403664  0.11861737 -0.12718708 -0.1476884  -0.07839713  0.0881826
  -0.01106316  0.27360782  0.06632026  0.09629916  0.29483762  0.35177141
   1.0002359   0.06001706  0.06294044]
 [-0.11810434 -0.0152751  -0.04307069  0.06090111  0.07281501  0.01532653
  -0.01748119  0.14391897  0.04900968  0.06788468  0.17857686  0.18119275
   0.06001706  1.0002359   0.03380237]
 [ 0.06525672  0.09672679 -0.02993657 -0.05320936 -0.04335494  0.03867089
   0.02245161  0.06834245  0.44314288  0.1330926   0.10355694  0.04746354
   0.06294044  0.03380237  1.0002359 ]]
```



```
[110] #Calculating the Eigen Vectors and Eigen Values
      cov_mat = np.cov(x_std.T)
      eig_vals , eig_vecs = np.linalg.eig(cov_mat)
      print('Eigen Vectors',eig_vecs)
      print('Eigen Values',eig_vals)

      Eigen Vectors [[ 4.22621156e-02 -3.54898233e-01 -8.46736436e-02 -1.13316264e-01
        -1.42598114e-01  2.85028386e-02 -1.50411403e-01  1.94489057e-01
         6.06238206e-01  5.06061024e-02  1.83534234e-01  5.37797319e-01
         1.96927174e-01 -1.90049575e-01  5.25268869e-02]
       [-2.99076507e-01  1.11183291e-01  1.61638778e-01 -1.65318102e-02
        -8.11731551e-02 -9.70420089e-02  1.62940079e-01 -5.87617071e-01
         2.65738632e-01  3.22677021e-01 -2.16399103e-01  2.08029870e-01
         9.17110523e-03  3.78596890e-01  2.75098903e-01]
       [ 1.07963646e-01  1.73673166e-02 -2.79323168e-02  2.84707661e-04
         5.04598801e-02 -1.52888356e-02  6.70939202e-02 -3.40245657e-01
         2.62694302e-02  1.28407482e-01  1.56348316e-01 -1.45870146e-01
        -2.63393405e-01 -7.31867551e-01  4.43063905e-01]
       [ 2.01481425e-01 -5.89965338e-01 -2.22833565e-02 -6.84277302e-01
        -2.08630898e-02 -2.05070898e-02  8.10277232e-02 -1.45726981e-01
        -2.37522354e-01  1.28703922e-02 -1.21643851e-01 -8.76494086e-02
        -1.09372492e-01  1.43796397e-01  4.74826397e-02]
       [ 1.70688699e-01 -6.31619941e-01  6.01804895e-02  7.15435531e-01
        -3.30337778e-02 -1.62565030e-02  4.15808824e-02 -1.39662448e-01
        -1.17545800e-01 -7.93350375e-03 -5.81188677e-02 -1.13729810e-02
        -4.98504961e-02  1.07292423e-01  4.97796668e-02]
       [-2.04700426e-01 -3.86053218e-02  2.50665433e-02 -1.91070573e-03
         5.46641020e-02 -7.55117950e-02  7.01436762e-04  2.97015504e-02
         2.81385172e-01 -6.03494247e-01 -4.50077408e-01  7.06482930e-02
        -5.38432558e-01 -6.92792209e-02 -2.73771173e-02]
       [-6.44799484e-02  1.93854023e-02 -9.85893132e-03  5.20659325e-04
         8.78155280e-03 -1.80462212e-02 -2.12532461e-02 -6.36393184e-02
        -8.85696246e-02  1.08196683e-01  6.23319652e-01  2.50871875e-01
```



```
  -6.50163931e-01  2.19488008e-01 -2.20366326e-01]
 [-4.42645090e-01 -1.59048522e-01  1.48460546e-01 -1.14282255e-02
   1.24146846e-01  8.21251861e-01 -3.29088694e-03  1.01871785e-01
  -8.62116697e-02  1.68967861e-01 -8.06197062e-02  1.42843537e-04
  -6.46962778e-02 -9.83577134e-02  9.90812182e-05]
 [-1.19965391e-01  2.01150435e-03  2.11535225e-02 -4.99280796e-03
  -6.54506519e-01 -4.49022737e-03 -6.47827587e-01 -1.97365349e-01
  -1.15497477e-01  6.97737286e-02 -1.17927004e-01 -1.08107301e-01
  -6.01058823e-02 -1.20260603e-01 -1.90320552e-01]
 [-1.70179262e-01 -7.24500346e-03 -1.20729313e-02 -9.69047392e-03
  -1.40955252e-01  2.42240675e-02 -2.21689043e-01  2.17025122e-01
  -1.05663313e-01 -3.21760369e-01  2.80505991e-01 -1.01659203e-01
   3.94891211e-02  3.01608596e-01  7.46864066e-01]
 [-4.91636867e-01 -1.49984479e-01 -7.62479865e-01  4.90049843e-02
   1.02835669e-01 -2.52894316e-01  9.48273884e-03  9.31546603e-02
  -1.09879962e-01  2.12294749e-01 -8.55604236e-02 -4.13257174e-02
  -9.90851085e-03 -4.95162767e-02  1.13245893e-03]
 [-4.47108213e-01 -1.95375485e-01  5.94398715e-01 -4.76014638e-02
   1.63451683e-01 -4.90840841e-01 -5.73652044e-02  2.27724719e-01
  -1.24780860e-01  1.26827600e-01  6.95492788e-02 -2.51128122e-02
   6.89122011e-02 -1.98637763e-01 -4.71945343e-02]
 [-2.67834929e-01 -5.94277501e-02 -5.24802635e-02 -3.32557683e-02
   1.48460387e-02  4.63672889e-02  1.20976050e-01 -4.82169776e-01
  -1.98800994e-01 -5.48851281e-01  3.01422887e-01  1.79337282e-01
   3.83414646e-01 -1.39273927e-01 -2.02973501e-01]
 [-1.18939577e-01 -1.51536425e-01 -4.05523733e-03 -2.58056315e-02
   3.47705913e-02  2.65919655e-02  1.13434482e-02 -1.08169571e-01
   5.51394205e-01 -3.59472391e-02  2.86045717e-01 -7.14720457e-01
   3.86824728e-02  1.24513219e-01 -1.82994875e-01]
 [-1.20580247e-01  2.08704354e-04  3.14568604e-02  1.04877786e-02
  -6.80851001e-01 -5.02642823e-03  6.71264985e-01  2.32988085e-01
  -2.78195626e-02 -7.56205924e-03  4.29441753e-02 -5.59679614e-02
  -4.45130456e-02 -8.85301553e-02 -1.64507509e-02]]
Eigen Values [3.12298907 1.87402056 0.1735605  0.22739446 1.43147969 0.38475607
 0.53368281 0.6328553  0.68413241 0.82723996 0.8743304  1.11932784
 1.08856289 1.03312685 0.99607976]
```



```
[129] #Calculating Sum of Upper Triangle of Co-Variance Matrix
      sum = 0
      for i in range(0,13):
        for j in range(0,13):
          if i>j:
            sum = sum + cov_mat[i][j]
      print('Sum of Upper Triangle of Co-Variance Matrix :',sum)
```

```
Sum of Upper Triangle of Co-Variance Matrix : 5.800265738482123
```

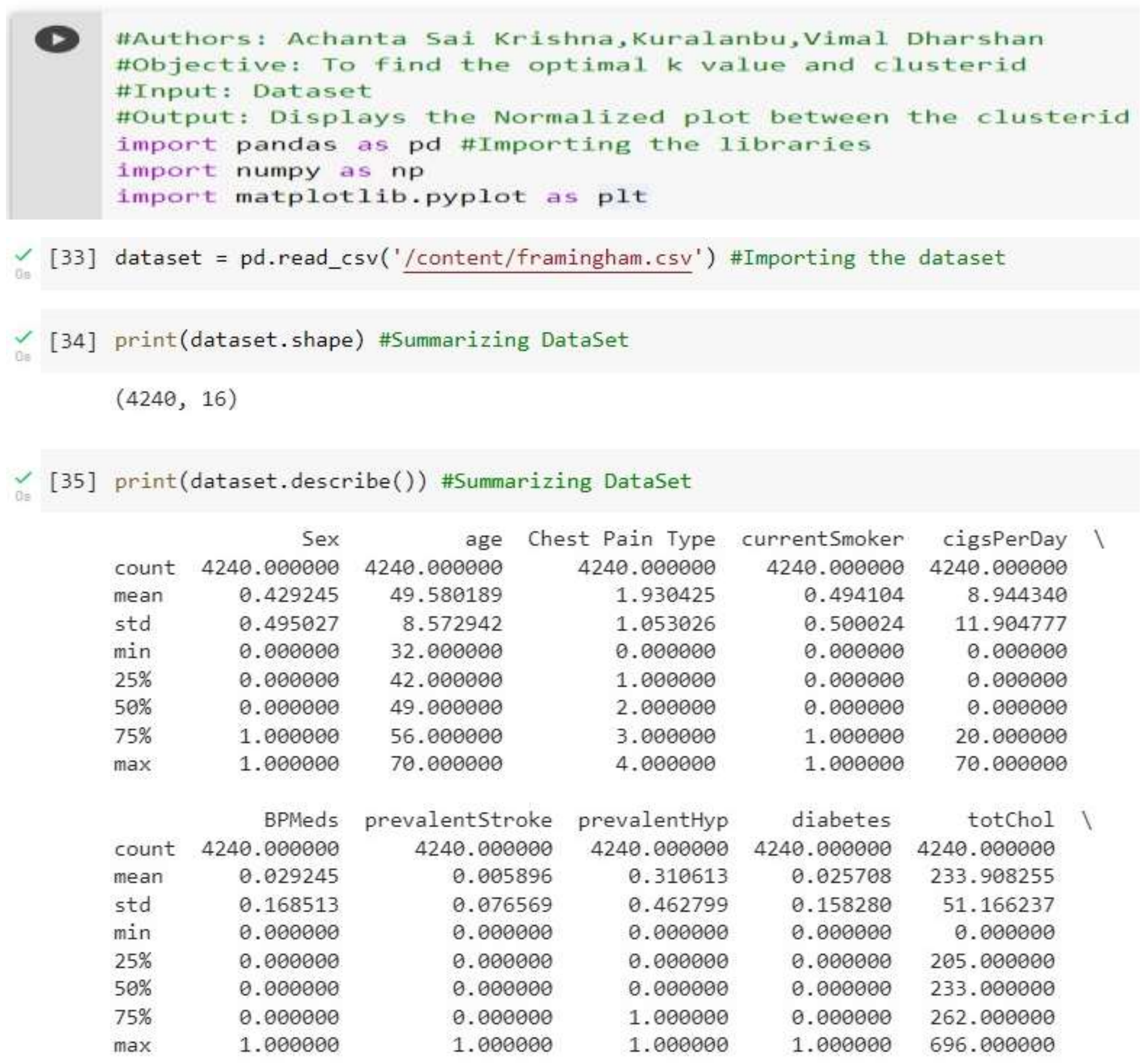
```
[130] #Calculating the Normalization Covariance Matrix
      x_pca = StandardScaler().fit_transform(x_pca)
      mean_vec = np.mean(x_pca, axis = 0)
      cov_mat = (x_pca - mean_vec).T.dot((x_pca - mean_vec))/(x_pca.shape[0]-1)
      print('Covariance Matrix', cov_mat)
```

```
Covariance Matrix [[ 1.00023590e+00 -4.59353912e-06]
 [-4.59353912e-06  1.00023590e+00]]
```

```
#Calculating the Sum of Upper Triangle of Co-Variance Matrix
cov_mat = np.cov(x_pca.T)
eig_vals , eig_vecs = np.linalg.eig(cov_mat)
print('Eigen Vectors',eig_vecs)
print('Eigen Values',eig_vals)
sum = 0
for i in range(0,2):
  for j in range(0,2):
    if i>j:
      sum = sum + cov_mat[i][j]
print('Sum of Upper Triangle of Co-Variance Matrix :',sum)
```

```
Eigen Vectors [[ 0.70710678  0.70710678]
 [-0.70710678  0.70710678]]
Eigen Values [1.0002405  1.00023131]
Sum of Upper Triangle of Co-Variance Matrix : -4.593539121551088e-06
```

Clustering



```
#Authors: Achanta Sai Krishna,Kuralanbu,Vimal Dharshan
#Objective: To find the optimal k value and clusterid
#Input: Dataset
#Output: Displays the Normalized plot between the clusterid
import pandas as pd #Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
```

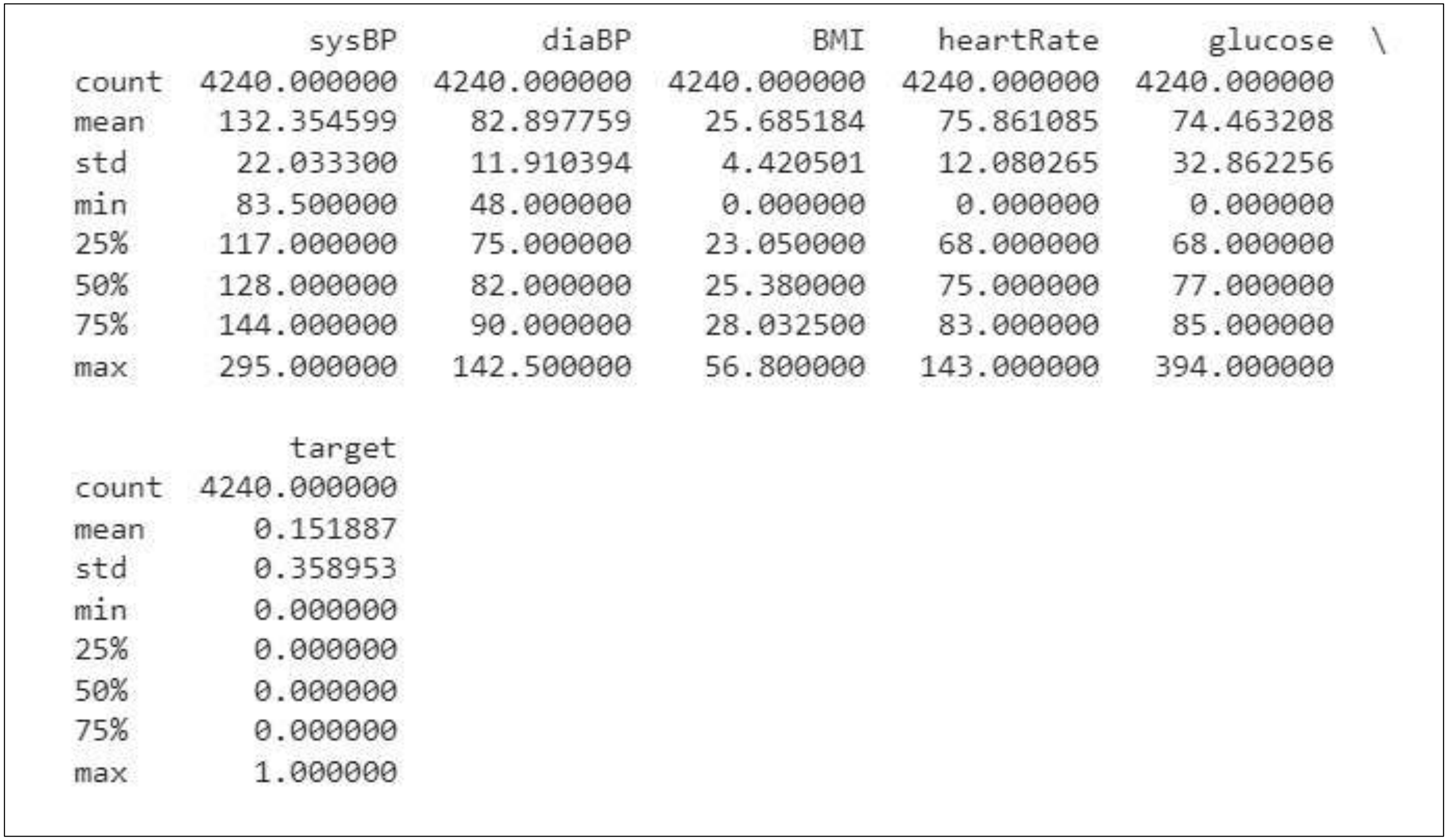
```
[33] dataset = pd.read_csv('/content/framingham.csv') #Importing the dataset
```

```
[34] print(dataset.shape) #Summarizing DataSet
```

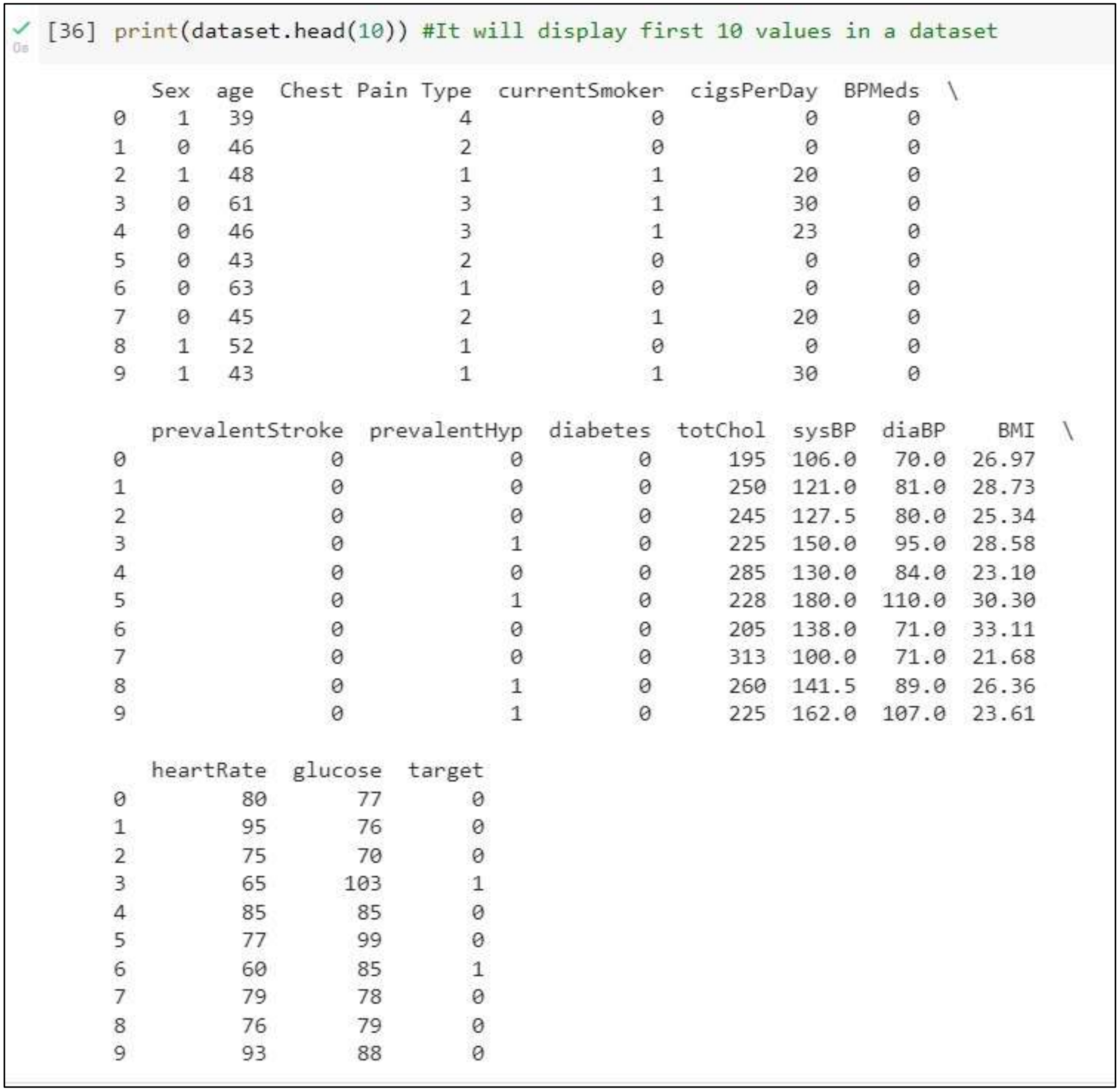
```
(4240, 16)
```

```
[35] print(dataset.describe()) #Summarizing DataSet
```

```
               Sex          age  Chest Pain Type  currentSmoker   cigsPerDay  \
count  4240.000000  4240.000000      4240.000000    4240.000000  4240.000000
mean      0.429245    49.580189         1.930425       0.494104     8.944340
std       0.495027     8.572942         1.053026       0.500024    11.904777
min       0.000000    32.000000         0.000000       0.000000     0.000000
25%       0.000000    42.000000         1.000000       0.000000     0.000000
50%       0.000000    49.000000         2.000000       0.000000     0.000000
75%       1.000000    56.000000         3.000000       1.000000    20.000000
max       1.000000    70.000000         4.000000       1.000000    70.000000

            BPMeds  prevalentStroke  prevalentHyp     diabetes      totChol  \
count  4240.000000      4240.000000   4240.000000  4240.000000  4240.000000
mean      0.029245         0.005896      0.310613     0.025708   233.908255
std       0.168513         0.076569      0.462799     0.158280    51.166237
min       0.000000         0.000000      0.000000     0.000000     0.000000
25%       0.000000         0.000000      0.000000     0.000000   205.000000
50%       0.000000         0.000000      0.000000     0.000000   233.000000
75%       0.000000         0.000000      1.000000     0.000000   262.000000
max       1.000000         1.000000      1.000000     1.000000   696.000000
```

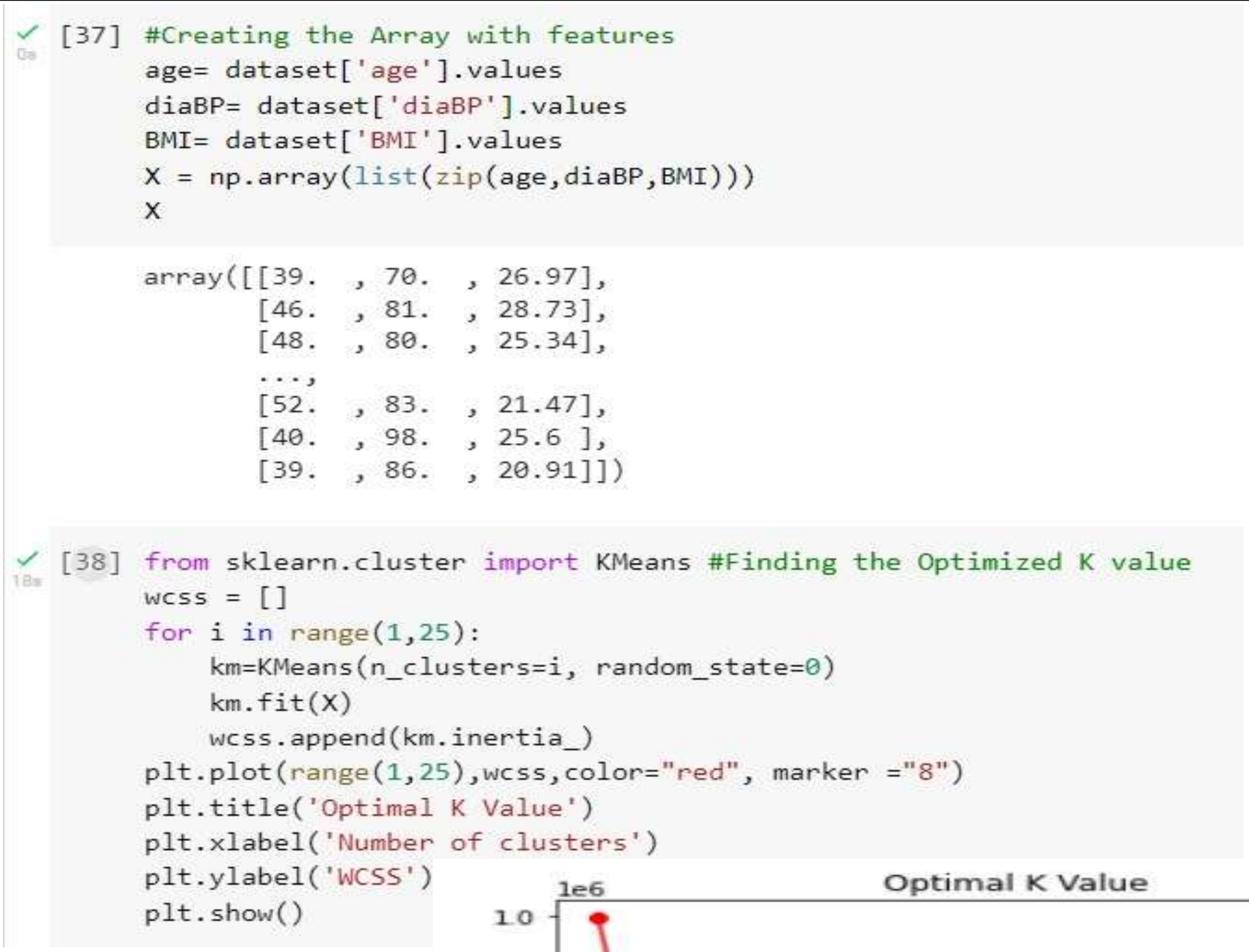


```
            sysBP        diaBP          BMI    heartRate      glucose  \
count 4240.000000  4240.000000  4240.000000  4240.000000  4240.000000
mean   132.354599    82.897759    25.685184    75.861085    74.463208
std     22.033300    11.910394     4.420501    12.080265    32.862256
min     83.500000    48.000000     0.000000     0.000000     0.000000
25%    117.000000    75.000000    23.050000    68.000000    68.000000
50%    128.000000    82.000000    25.380000    75.000000    77.000000
75%    144.000000    90.000000    28.032500    83.000000    85.000000
max    295.000000   142.500000    56.800000   143.000000   394.000000

           target
count 4240.000000
mean     0.151887
std      0.358953
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max      1.000000
```



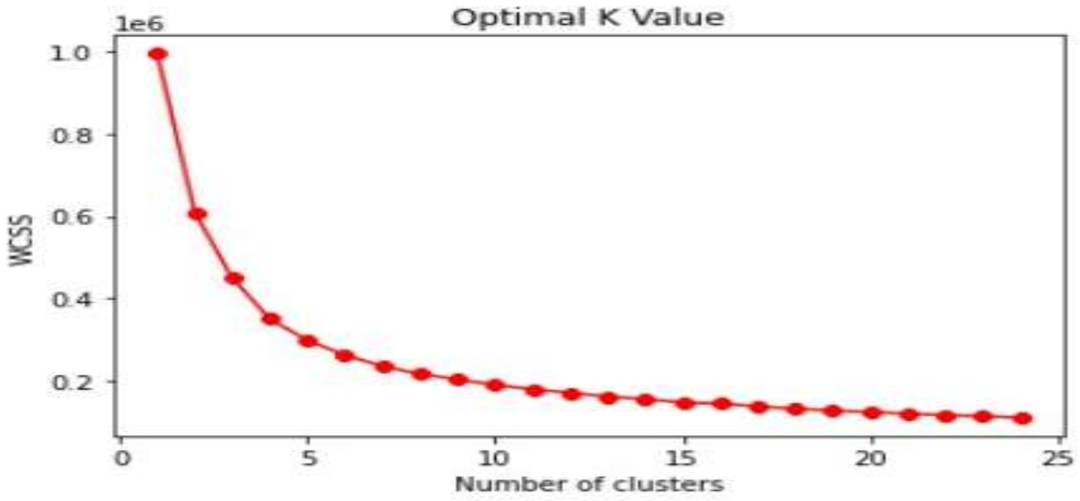
```
[36] print(dataset.head(10)) #It will display first 10 values in a dataset
```

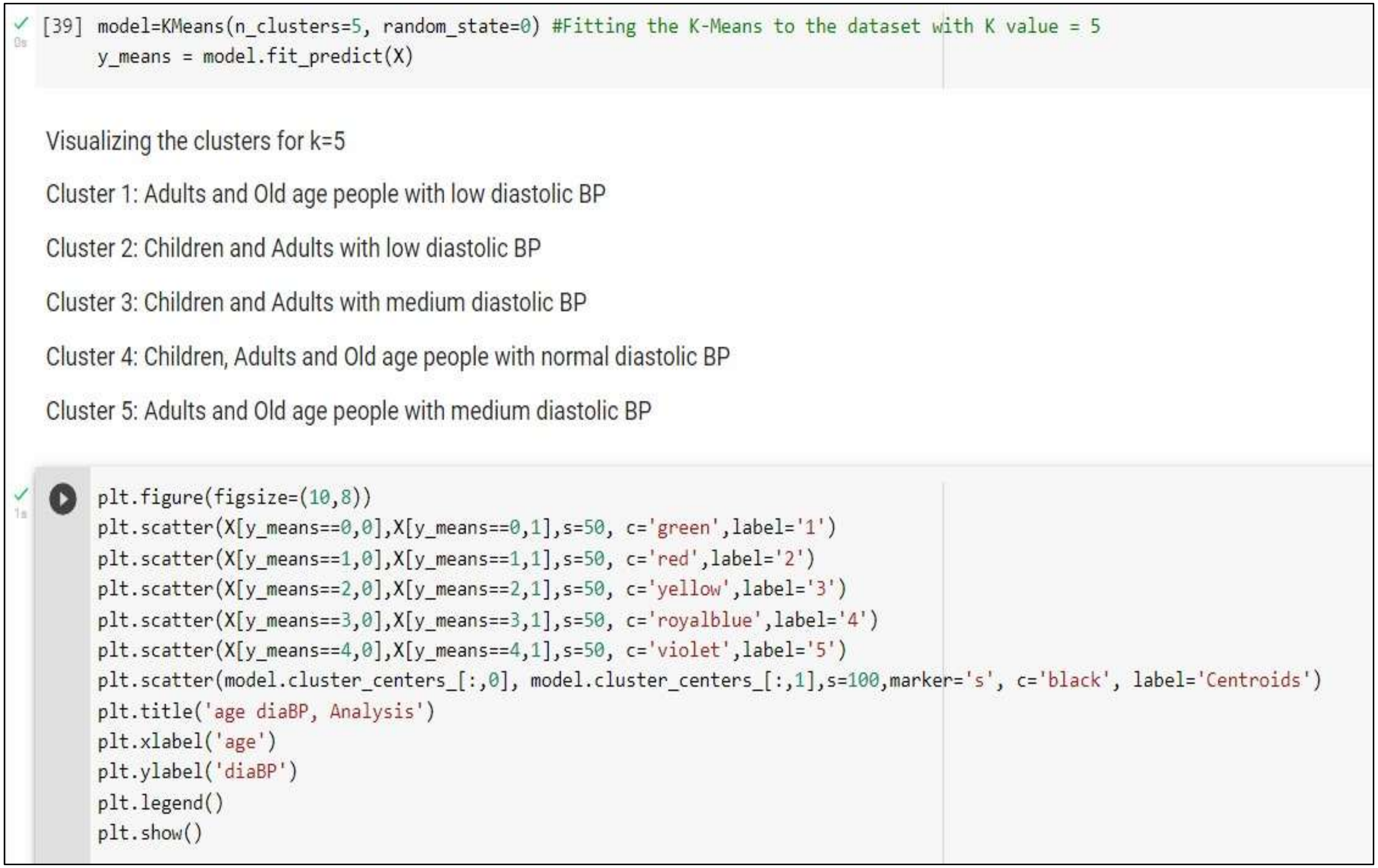
```
   Sex  age  Chest Pain Type  currentSmoker  cigsPerDay  BPMeds  \
0    1   39                4              0           0       0
1    0   46                2              0           0       0
2    1   48                1              1          20       0
3    0   61                3              1          30       0
4    0   46                3              1          23       0
5    0   43                2              0           0       0
6    0   63                1              0           0       0
7    0   45                2              1          20       0
8    1   52                1              0           0       0
9    1   43                1              1          30       0

   prevalentStroke  prevalentHyp  diabetes  totChol  sysBP  diaBP    BMI  \
0                0             0         0      195  106.0   70.0  26.97
1                0             0         0      250  121.0   81.0  28.73
2                0             0         0      245  127.5   80.0  25.34
3                0             1         0      225  150.0   95.0  28.58
4                0             0         0      285  130.0   84.0  23.10
5                0             1         0      228  180.0  110.0  30.30
6                0             0         0      205  138.0   71.0  33.11
7                0             0         0      313  100.0   71.0  21.68
8                0             1         0      260  141.5   89.0  26.36
9                0             1         0      225  162.0  107.0  23.61

   heartRate  glucose  target
0         80       77       0
1         95       76       0
2         75       70       0
3         65      103       1
4         85       85       0
5         77       99       0
6         60       85       1
7         79       78       0
8         76       79       0
9         93       88       0
```



```
[37] #Creating the Array with features
     age= dataset['age'].values
     diaBP= dataset['diaBP'].values
     BMI= dataset['BMI'].values
     X = np.array(list(zip(age,diaBP,BMI)))
     X

     array([[39.  , 70.  , 26.97],
            [46.  , 81.  , 28.73],
            [48.  , 80.  , 25.34],
            ...,
            [52.  , 83.  , 21.47],
            [40.  , 98.  , 25.6 ],
            [39.  , 86.  , 20.91]])
```

```
[38] from sklearn.cluster import KMeans #Finding the Optimized K value
     wcss = []
     for i in range(1,25):
         km=KMeans(n_clusters=i, random_state=0)
         km.fit(X)
         wcss.append(km.inertia_)
     plt.plot(range(1,25),wcss,color="red", marker ="8")
     plt.title('Optimal K Value')
     plt.xlabel('Number of clusters')
     plt.ylabel('WCSS')
     plt.show()
```





```
[39] model=KMeans(n_clusters=5, random_state=0) #Fitting the K-Means to the dataset with K value = 5
     y_means = model.fit_predict(X)
```

Visualizing the clusters for k=5

Cluster 1: Adults and Old age people with low diastolic BP

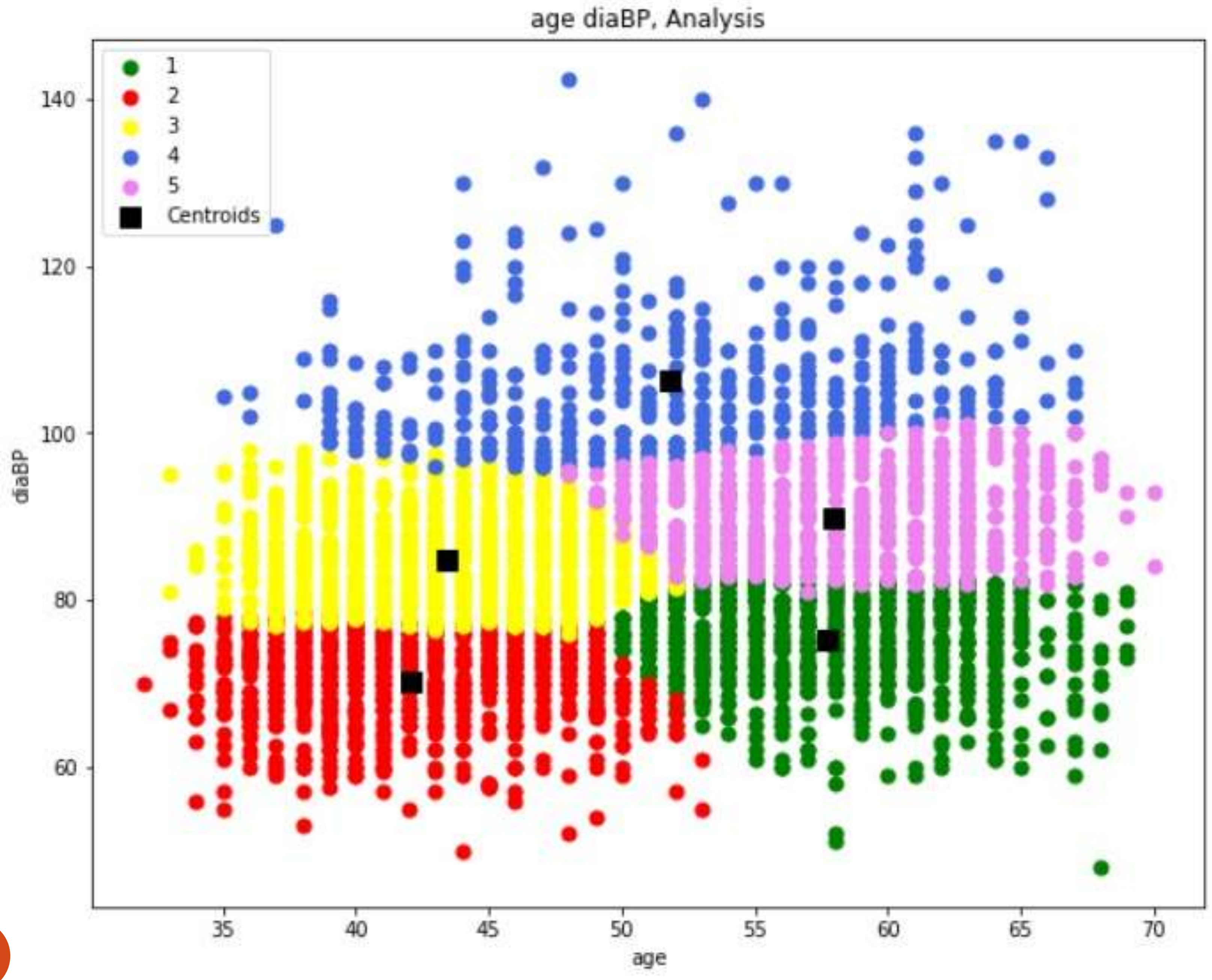
Cluster 2: Children and Adults with low diastolic BP

Cluster 3: Children and Adults with medium diastolic BP

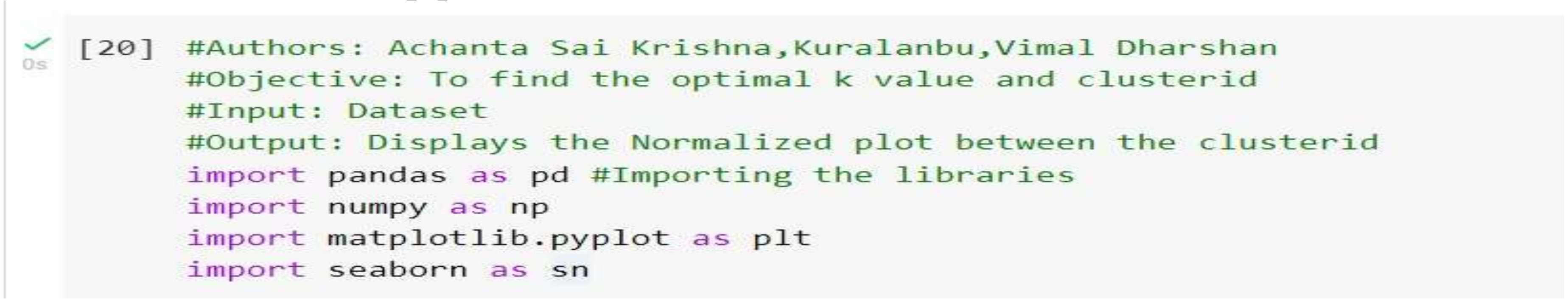
Cluster 4: Children, Adults and Old age people with normal diastolic BP

Cluster 5: Adults and Old age people with medium diastolic BP

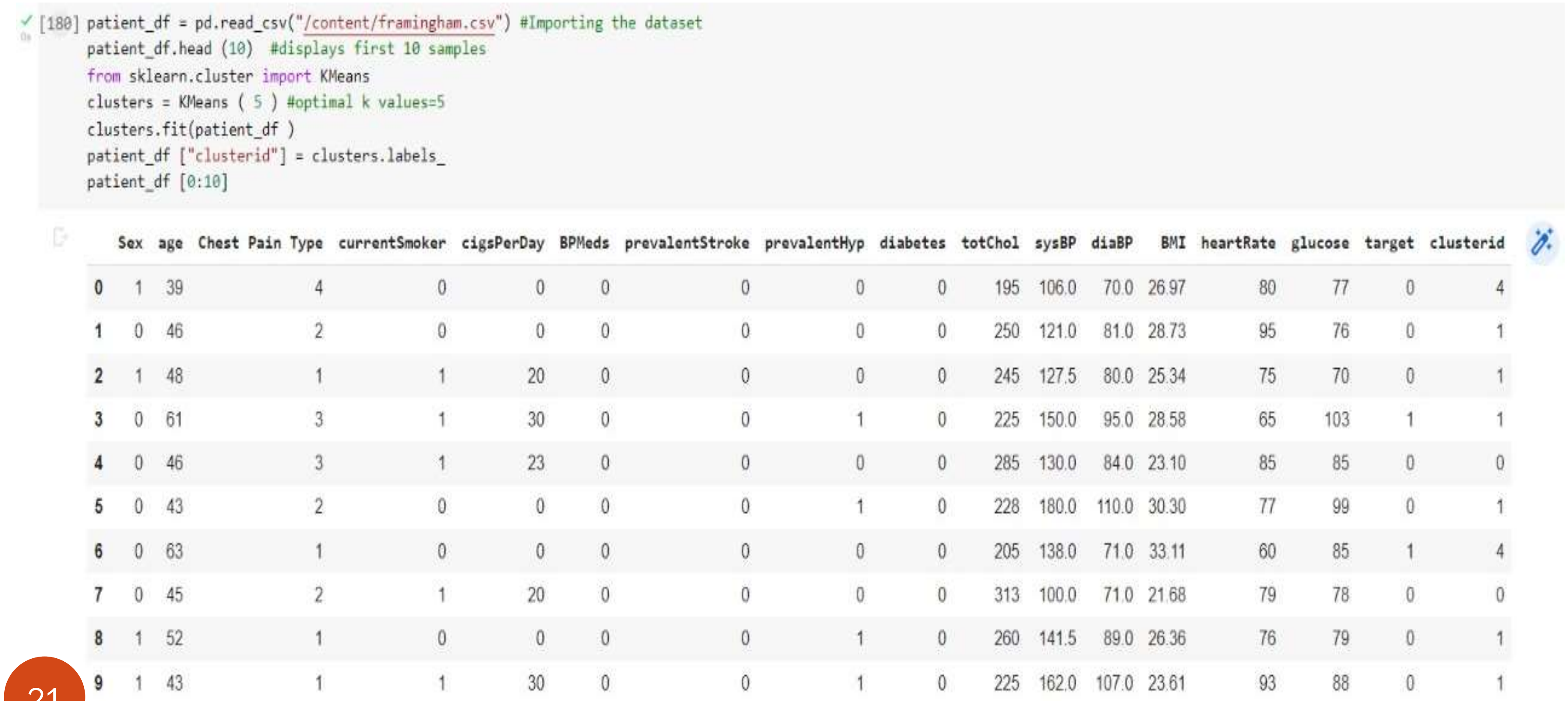
```
plt.figure(figsize=(10,8))
plt.scatter(X[y_means==0,0],X[y_means==0,1],s=50, c='green',label='1')
plt.scatter(X[y_means==1,0],X[y_means==1,1],s=50, c='red',label='2')
plt.scatter(X[y_means==2,0],X[y_means==2,1],s=50, c='yellow',label='3')
plt.scatter(X[y_means==3,0],X[y_means==3,1],s=50, c='royalblue',label='4')
plt.scatter(X[y_means==4,0],X[y_means==4,1],s=50, c='violet',label='5')
plt.scatter(model.cluster_centers_[:,0], model.cluster_centers_[:,1],s=100,marker='s', c='black', label='Centroids')
plt.title('age diaBP, Analysis')
plt.xlabel('age')
plt.ylabel('diaBP')
plt.legend()
plt.show()
```


age diaBP, Analysis
1
2
3
4
5
Centroids
140
120
100
80
60
diaBP
35
40
45
50
55
60
65
70
age

Clustering

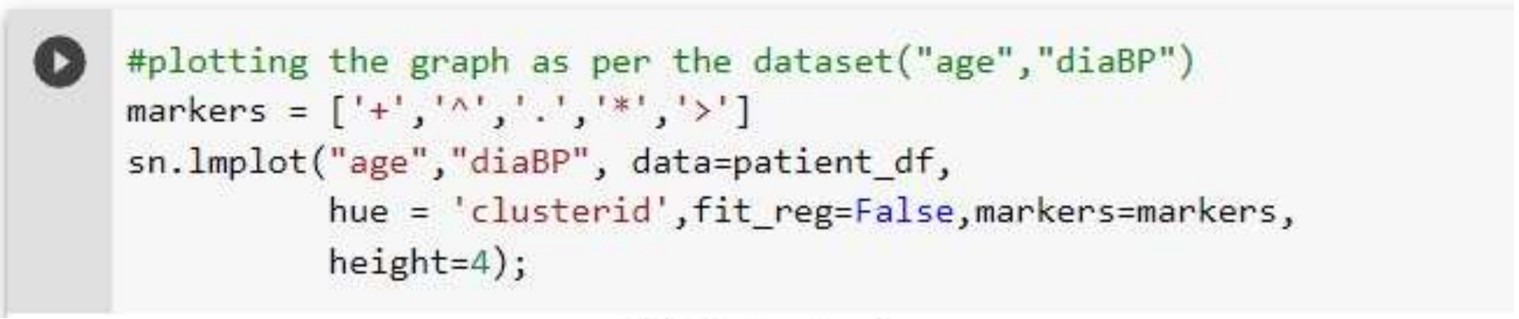


```
[20] #Authors: Achanta Sai Krishna,Kuralanbu,Vimal Dharshan
     #Objective: To find the optimal k value and clusterid
     #Input: Dataset
     #Output: Displays the Normalized plot between the clusterid
     import pandas as pd #Importing the libraries
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sn
```

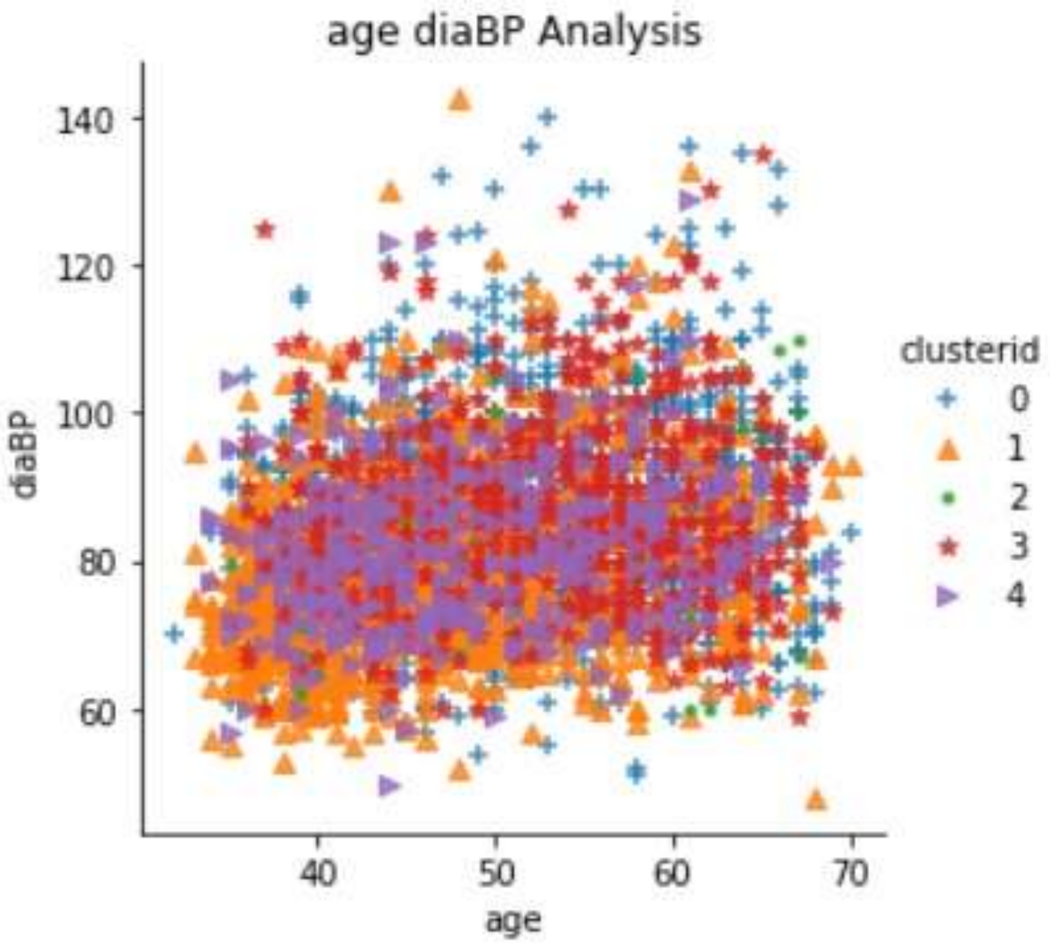


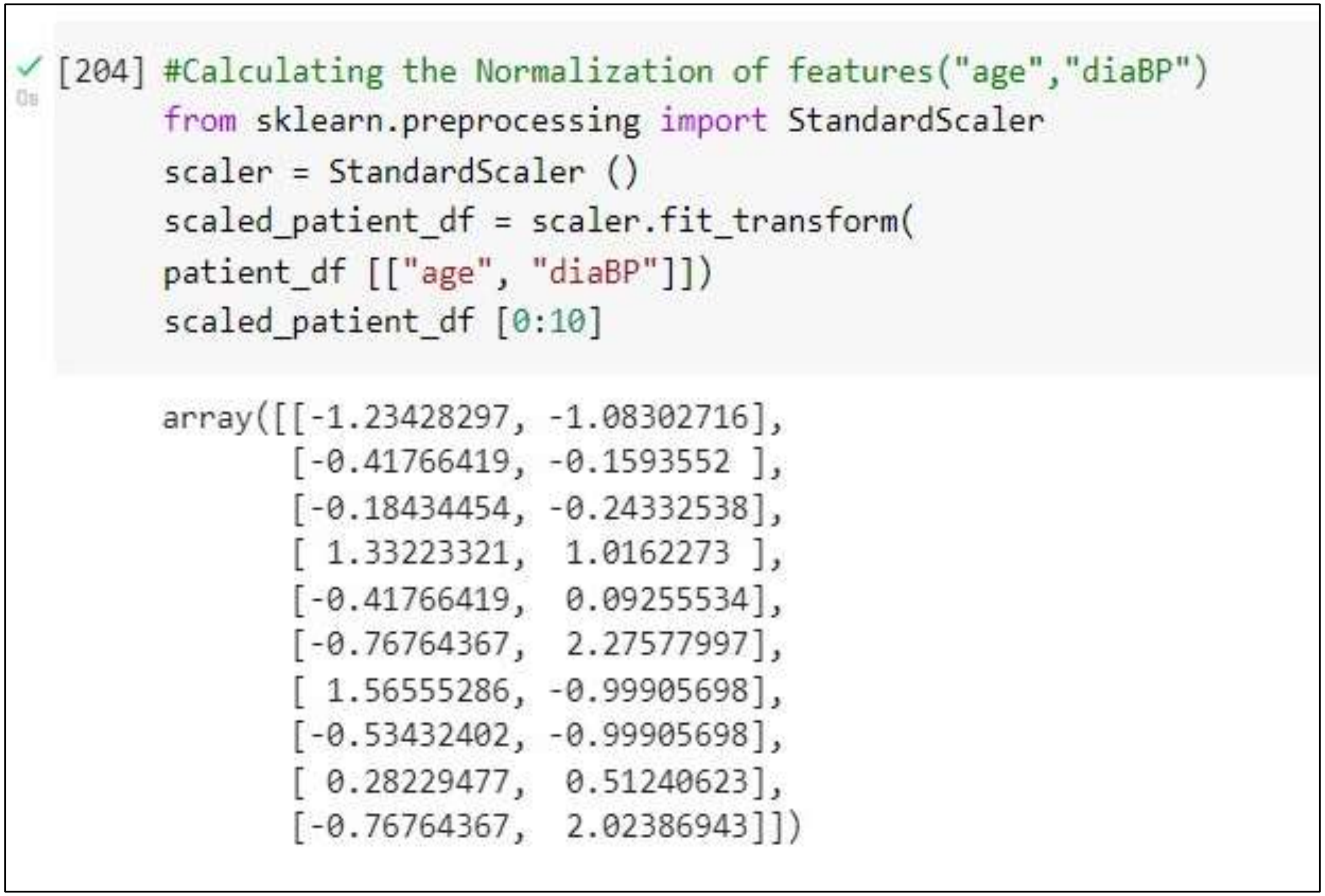
```
[180] patient_df = pd.read_csv("/content/framingham.csv") #Importing the dataset
      patient_df.head (10)  #displays first 10 samples
      from sklearn.cluster import KMeans
      clusters = KMeans ( 5 ) #optimal k values=5
      clusters.fit(patient_df )
      patient_df ["clusterid"] = clusters.labels_
      patient_df [0:10]
```

	Sex	age	Chest Pain Type	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	target	clusterid
0	1	39	4	0	0	0	0	0	0	195	106.0	70.0	26.97	80	77	0	4
1	0	46	2	0	0	0	0	0	0	250	121.0	81.0	28.73	95	76	0	1
2	1	48	1	1	20	0	0	0	0	245	127.5	80.0	25.34	75	70	0	1
3	0	61	3	1	30	0	0	1	0	225	150.0	95.0	28.58	65	103	1	1
4	0	46	3	1	23	0	0	0	0	285	130.0	84.0	23.10	85	85	0	0
5	0	43	2	0	0	0	0	1	0	228	180.0	110.0	30.30	77	99	0	1
6	0	63	1	0	0	0	0	0	0	205	138.0	71.0	33.11	60	85	1	4
7	0	45	2	1	20	0	0	0	0	313	100.0	71.0	21.68	79	78	0	0
8	1	52	1	0	0	0	0	1	0	260	141.5	89.0	26.36	76	79	0	1
9	1	43	1	1	30	0	0	1	0	225	162.0	107.0	23.61	93	88	0	1



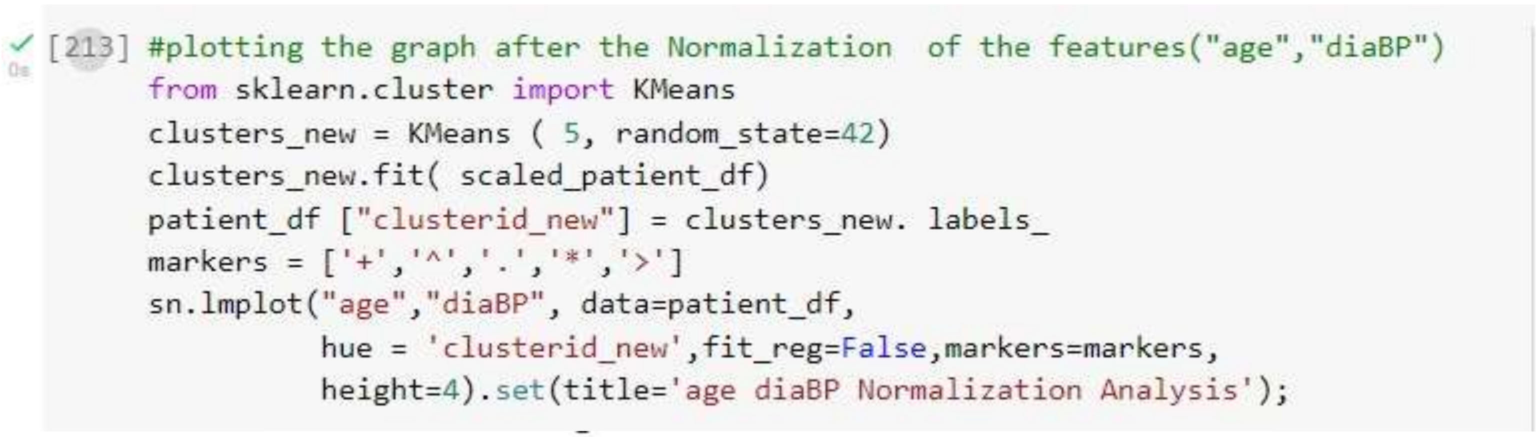
```
#plotting the graph as per the dataset("age","diaBP")
markers = ['+','^','.','*','>']
sn.lmplot("age","diaBP", data=patient_df,
          hue = 'clusterid',fit_reg=False,markers=markers,
          height=4);
```



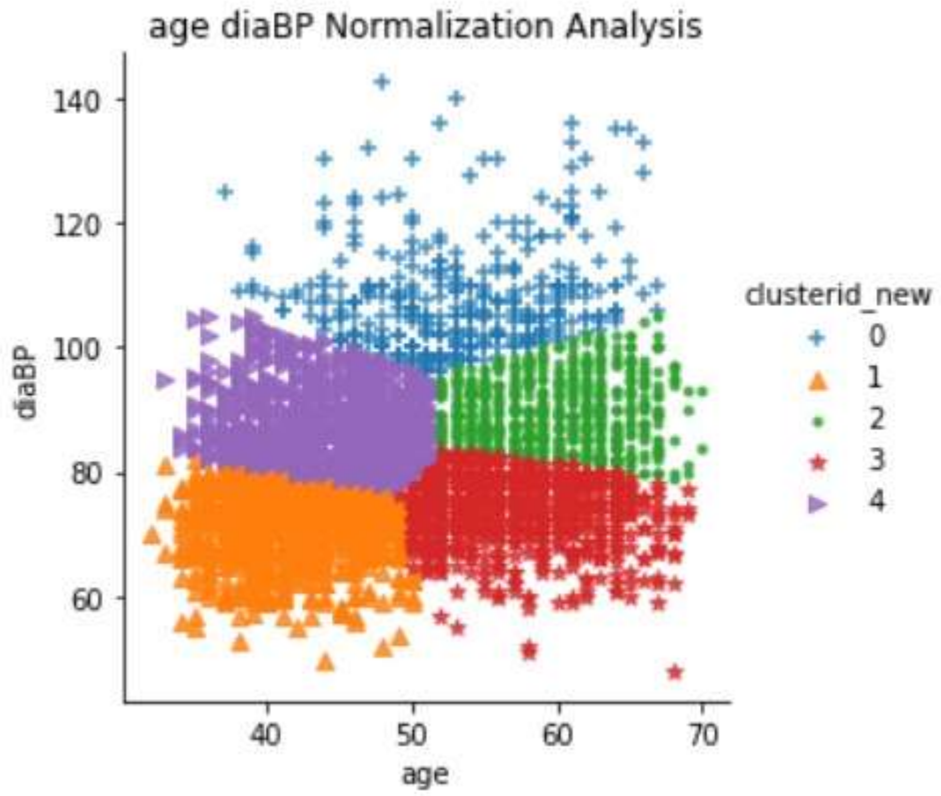


```
[204] #Calculating the Normalization of features("age","diaBP")
      from sklearn.preprocessing import StandardScaler
      scaler = StandardScaler ()
      scaled_patient_df = scaler.fit_transform(
      patient_df [["age", "diaBP"]])
      scaled_patient_df [0:10]
```

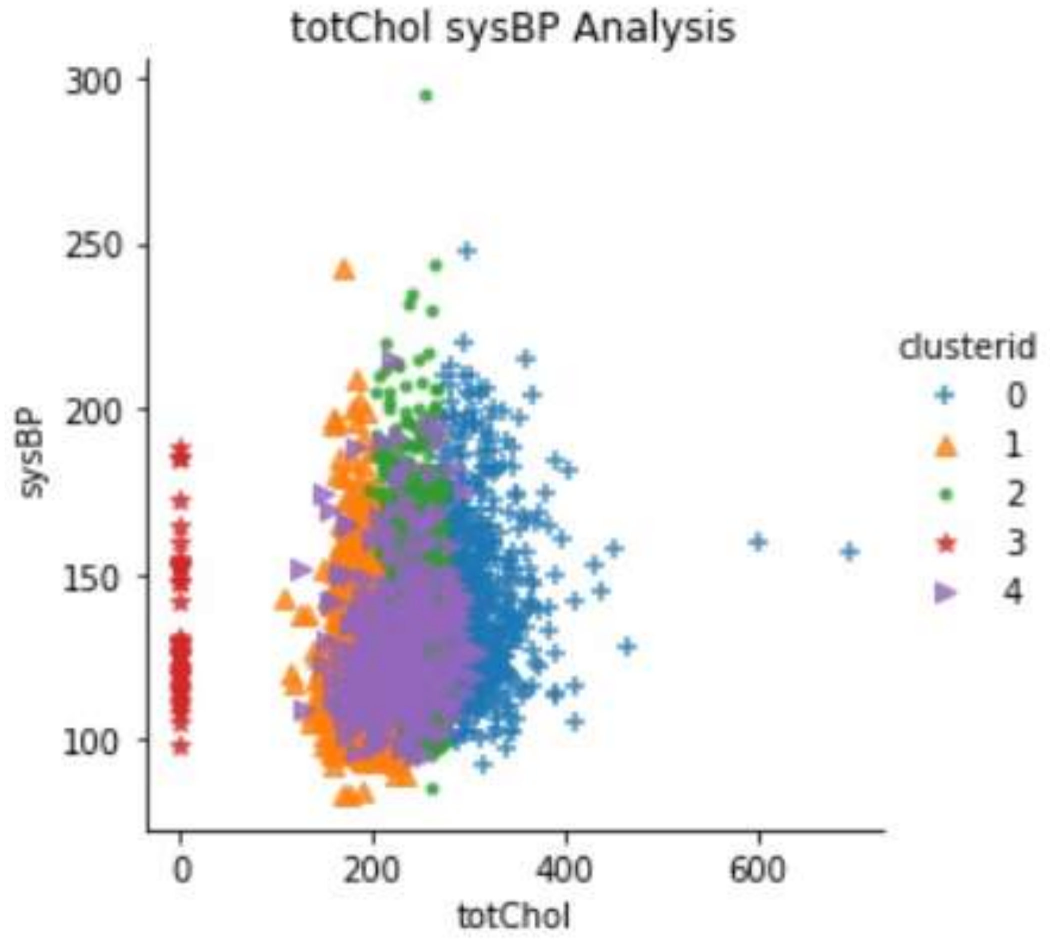
```
array([[-1.23428297, -1.08302716],
       [-0.41766419, -0.1593552 ],
       [-0.18434454, -0.24332538],
       [ 1.33223321,  1.0162273 ],
       [-0.41766419,  0.09255534],
       [-0.76764367,  2.27577997],
       [ 1.56555286, -0.99905698],
       [-0.53432402, -0.99905698],
       [ 0.28229477,  0.51240623],
       [-0.76764367,  2.02386943]])
```

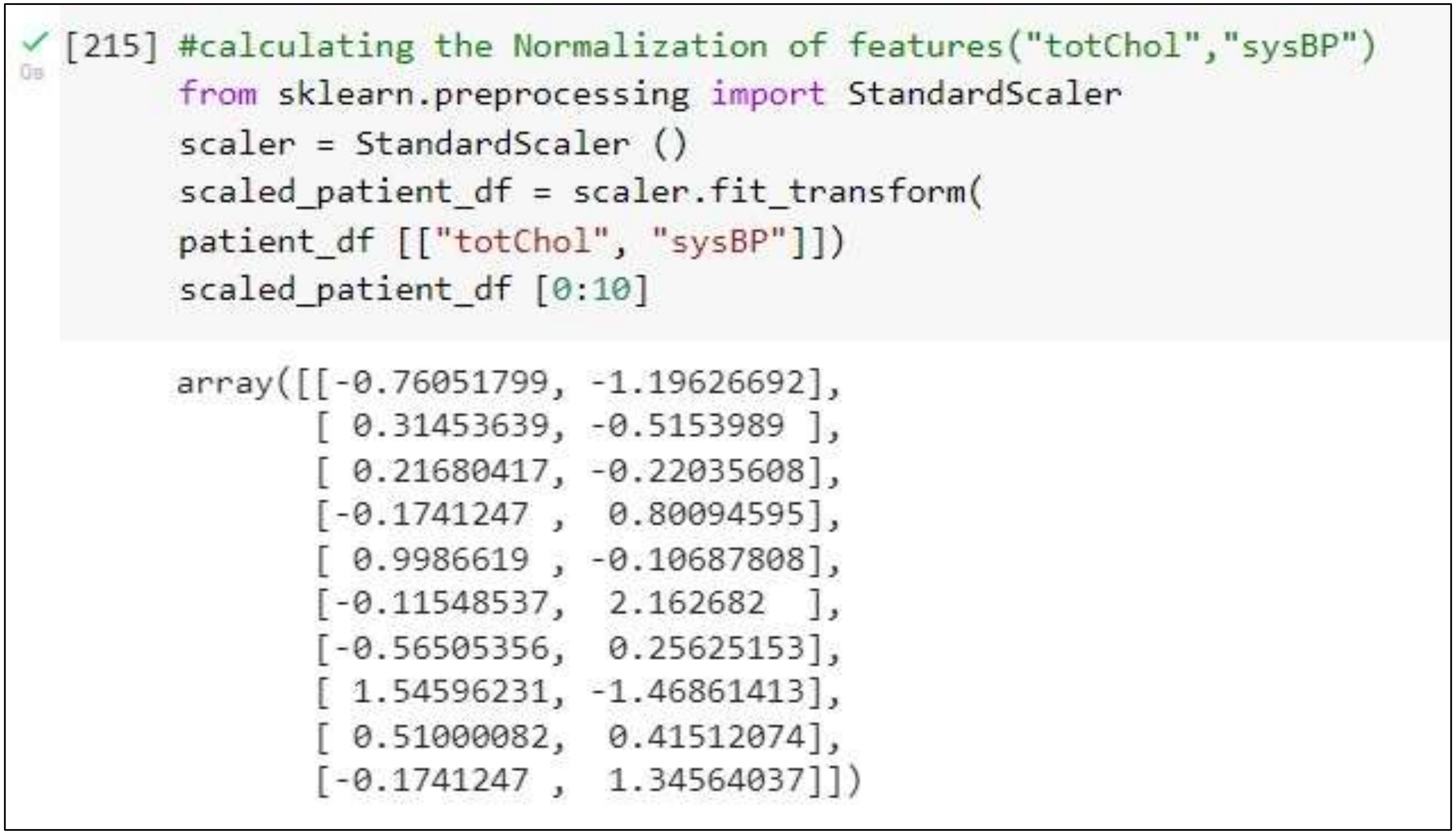


```
[213] #plotting the graph after the Normalization  of the features("age","diaBP")
      from sklearn.cluster import KMeans
      clusters_new = KMeans ( 5, random_state=42)
      clusters_new.fit( scaled_patient_df)
      patient_df ["clusterid_new"] = clusters_new. labels_
      markers = ['+','^','.','*','>']
      sn.lmplot("age","diaBP", data=patient_df,
                hue = 'clusterid_new',fit_reg=False,markers=markers,
                height=4).set(title='age diaBP Normalization Analysis');
```



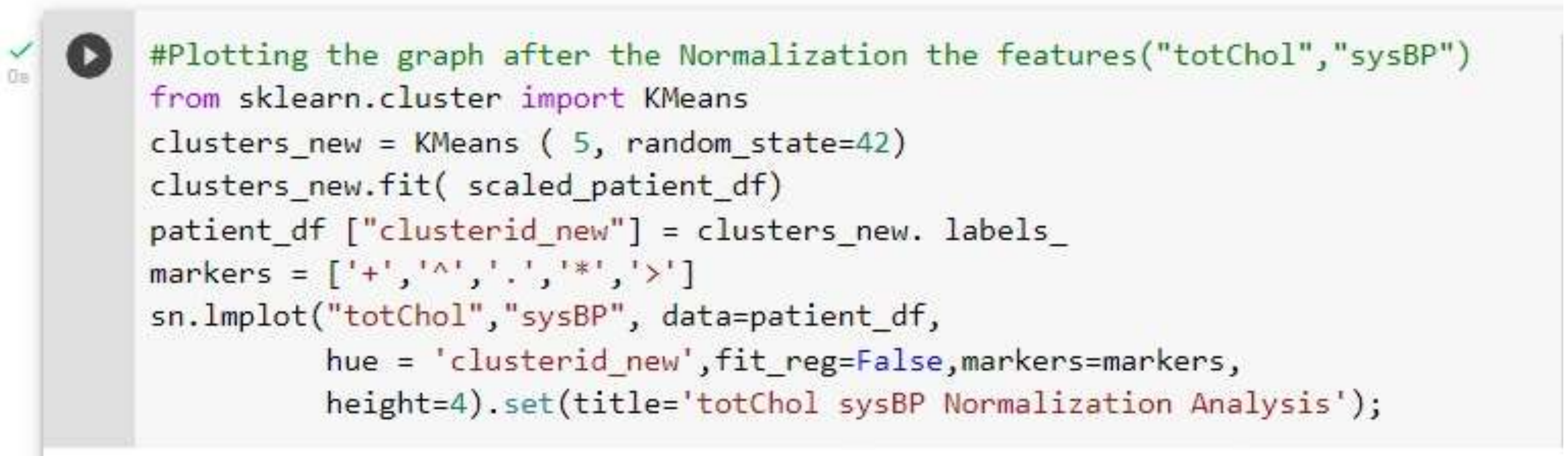
```
#plotting the graph between two features("totChol","sysBP") as per data
markers = ['+','^','.','*','>']
sn.lmplot("totChol","sysBP", data=patient_df,
          hue = 'clusterid',fit_reg=False,markers=markers,
          height=4).set(title='totChol sysBP Analysis');
```



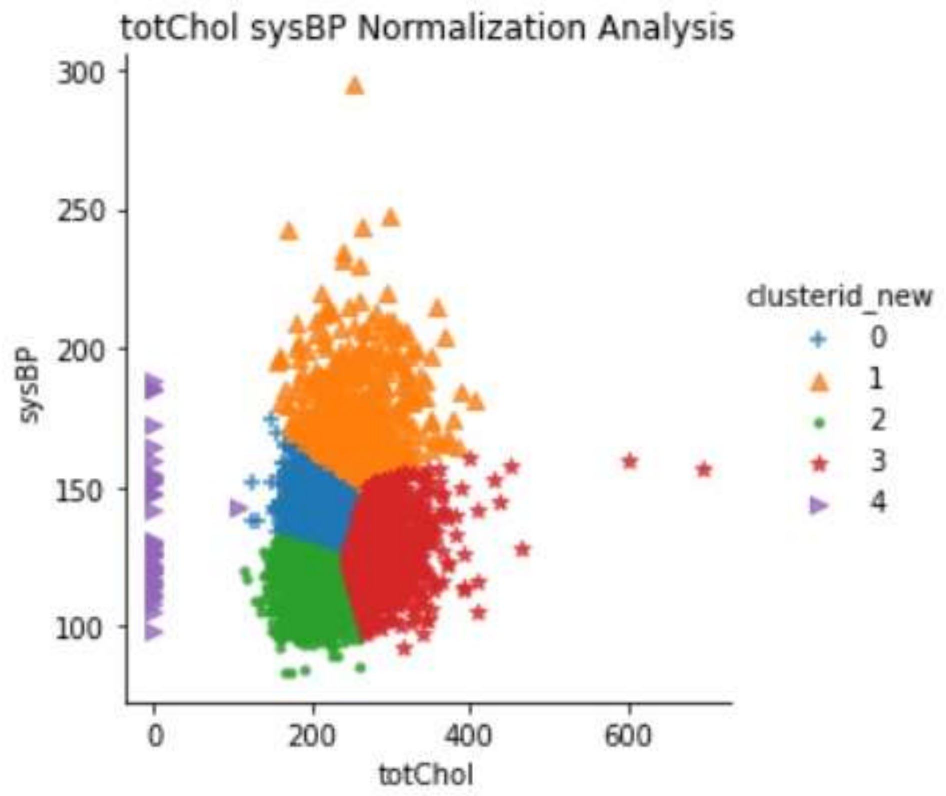


```
[215] #calculating the Normalization of features("totChol","sysBP")
      from sklearn.preprocessing import StandardScaler
      scaler = StandardScaler ()
      scaled_patient_df = scaler.fit_transform(
      patient_df [["totChol", "sysBP"]])
      scaled_patient_df [0:10]
```

```
array([[-0.76051799, -1.19626692],
       [ 0.31453639, -0.5153989 ],
       [ 0.21680417, -0.22035608],
       [-0.1741247 ,  0.80094595],
       [ 0.9986619 , -0.10687808],
       [-0.11548537,  2.162682  ],
       [-0.56505356,  0.25625153],
       [ 1.54596231, -1.46861413],
       [ 0.51000082,  0.41512074],
       [-0.1741247 ,  1.34564037]])
```



```
#Plotting the graph after the Normalization the features("totChol","sysBP")
from sklearn.cluster import KMeans
clusters_new = KMeans ( 5, random_state=42)
clusters_new.fit( scaled_patient_df)
patient_df ["clusterid_new"] = clusters_new. labels_
markers = ['+','^','.','*','>']
sn.lmplot("totChol","sysBP", data=patient_df,
          hue = 'clusterid_new',fit_reg=False,markers=markers,
          height=4).set(title='totChol sysBP Normalization Analysis');
```



PCA: What eigen values did you select and why?

Eigen Vectors:

$$V = [v_1 \quad v_2] = \begin{bmatrix} 0.70710678 & 0.70710678 \\ -0.70710678 & 0.70710678 \end{bmatrix}$$

Because v_1 and v_2 are orthogonal or orthonormal.

Orthogonal matrix:

The matrix is said to be an orthogonal matrix if the product of a matrix and its transpose gives an identity value.
Orthogonal matrix is also known as Orthonormal matrix

$$\boxed{AA^T = I}$$

Eigen Values:

$$\lambda_1 = 1.0002405 \qquad \lambda_2 = 1.00023131$$

By comparing the Eigen values, $\lambda_1 > \lambda_2$, So λ_1 is the first vector and λ_2 is the second vector, finally Eigen vectors matrix is V

PCA: Write inference based on the Sum of upper triangle values of the covariance matrix before and after PCA.

Before the Principal Component Analysis, the sum of the upper triangle values of the covariance matrix is positive.

And after the Principal Component Analysis, the sum of the upper triangle values of the covariance matrix is negative.

PCA: What is the effect of normalization?

Before the effect of normalization, the covariance matrix and eigenvector matrix print all the dataset's features, i.e., 16x16 matrix.

After using the normalization PCA, according to our project, we reduced the features to 2, so the covariance matrix and the eigenvector matrix are printed to a 2x2 matrix.

PCA Inference:

- Except 'totChol' feature in our dataset remaining feature's standard deviation is very low and accepted as good features.
- The covariance matrix ranges from -1 to 1.
- Eigen Vector with highest Eigen value is called Principal Component of the data.
- As the features are increased while calculating the sum of upper triangle of Co-Variances will changes
- After the standardization the covariance matrix is changed and the sum of the upper triangle matrix is changed to negative value.

$$Cov(X,Y) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_x) \times (y_i - \mu_y)$$

$$C = \begin{bmatrix} Cov(X,X) & Cov(X,Y) \\ Cov(Y,X) & Cov(Y,Y) \end{bmatrix}$$

Clustering: Which value of K did you choose. Why?

When we compute and plot the Within-Cluster-Sum-of-Squares (WCSS) against the different values of k (starting from 1). The optimal number of clusters is given by the sharp turn in the Elbow graph.

According to our project, the optimal number (K) is 5.

$$\mathrm{WCSS} = \sum_{C_k}^{C_n} \left(\sum_{d_i\,in\,C_i}^{d_m} distance(d_i, C_k)^2 \right)$$

Where,

C is the cluster centroids and d is the data point in each Cluster.

Clustering: What is the effect of PCA and/or normalization?

After fitting the K-Means value, where K=5, i.e., clusters are divided into five types, and each cluster id is labelled to each sample in the dataset.

Before normalization, the plot between two samples(with relatively the same values) is not able to distinguish the clusters properly.

When we consider two features (with relatively the same values), after standardization, the value changes from -2 to +2.

After normalization, the plot between the two samples distinguishes the clusters properly.

Clustering: Which one is related to number of classes given in the original dataset.

While observing the plotted normalized graph, the number of samples in cluster 1 is greatest compared to the remaining clusters.

After the normalized graph is plotted, the cluster having the least sample is considered as cluster 5.

For each cluster, the centroid has been plotted and calculated by K-Means.

Clustering Inference

- The main objective of clustering is to create heterogeneous subsets(clusters) from the original dataset such that data samples within a cluster are homogeneous.
- Clustering is important as it determines the intrinsic grouping among the present un-labelled data.

K-Means Clustering Algorithm:

- It's the simplest unsupervised algorithm that solves the clustering problem.
- K-Means algorithm partitions (n) samples into (K) clusters where each sample belongs to the cluster with the nearest mean acta as a centroid of the cluster.

Miscellaneous - PCA

- PCA is an unsupervised linear dimensionally reduction algorithm to transform the original variables to a linear combination of these independent variables.
- Number of principal components is always less than or equal to the number of attributes.
- The priority of principal components decreases as their numbers increase.
- Principal components are orthogonal.
- PCA reduces the computational complexity by dropping low variance dimensions, as well as creating better visualization.

Miscellaneous - PCA

- Covariance Matrix is a matrix used to represent the covariance values between pairs of elements given in a random vector.

$$Cov(X,Y) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_x) \times (y_i - \mu_y)$$

- A covariance matrix is usually a square matrix. It is also positive, semi-definite, and symmetric.
- Coercion attempts to clean up dirty values to fit the data type of a field. For instance:

  Strings will be coerced to numbers.

  Floating points will be truncated for integer values.
- Using errors='coerce'. It will replace all non-numeric values with mean.

Miscellaneous - PCA

- Eigenvalues and Eigenvectors:

Eigenvalues and Eigenvectors as these concepts are used in one of the most popular dimensionality reduction techniques – Principal Component Analysis (PCA).

Eigenvalues and Eigenvectors concepts are keys to training computationally efficient and high-performing machine learning models.

The eigenvectors are called principal axes or principal directions of the data. Projections of the data on the principal axes are called principal components.

$$\det(C - \lambda I) = 0$$

Miscellaneous - PCA

- Sum of Upper Triangular of Co-Variance Matrix:

  Formula to find sum of all elements matrix A inside the triangular area is given by -sum = sum + Aij (Where i<j)

- Upper Triangular Matrix:

  A matrix whose all elements below diagonal elements are zeros (i.e. a_ij = 0 for i > j).

Miscellaneous - Libraries

- from sklearn.preprocessing import StandardScaler:

The preprocessing module provides the StandardScaler utility class to perform the operation on an array-like dataset and standardize the dataset's features onto a unit scale (mean = 0 and variance = 1).

- from sklearn.decomposition import PCA:

We need to select the required number of principal components. Usually, n_components is chosen to be 2 for better visualization, but it matters and depends on the data. By the fit and transform method, the attributes are passed.

- from sklearn.cluster import K-Means:

The k-Means algorithm is very fast (one of the fastest clustering algorithms available) but falls in local minima. That's why it can be useful to restart it several times.

Miscellaneous - Clustering

- **Within-Cluster-Sum-of-Squares** (WCSS). WCSS is the sum of squares of the distances of each data point in all clusters to their respective centroids. The idea is to minimize the sum.
- km_inertia: Inertia measures how well a dataset was clustered by K-Means. It is calculated by measuring the distance between each data point and its centroid, squaring this distance, and summing these squares across one cluster.
- A good model is one with low inertia AND a low number of clusters (K). However, this is a tradeoff because as K increases, inertia decreases.
- To find the optimal K for a dataset, use the Elbow method; find where the decrease in inertia begins to slow. K=5 is the "elbow" of this graph.