Academic Year: 2022-2023

Name:	Prerna Sunil Jadhav
Sap Id:	60004220127
Class:	T. Y. B.Tech (Computer Engineering)
Course:	Data Mining and Warehouse Laboratory
Course Code:	DJ19CEL501
Experiment No.:	05

AIM: Implementation of K Means and Hierarchical Clustering algorithm

PART A (Using Inbuilt function)

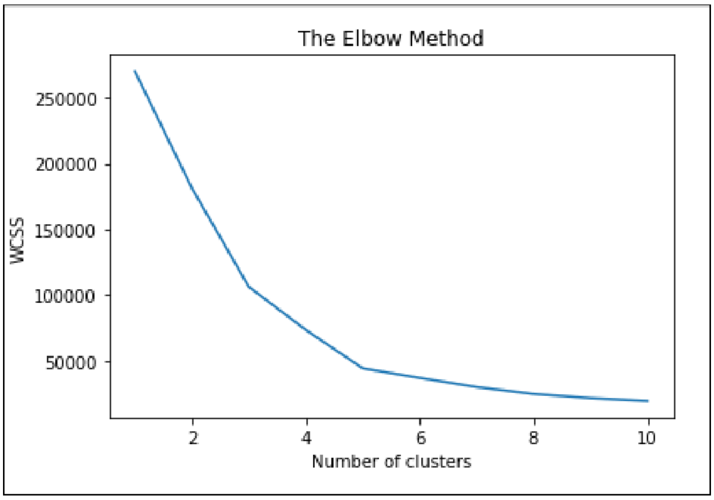
K-Means

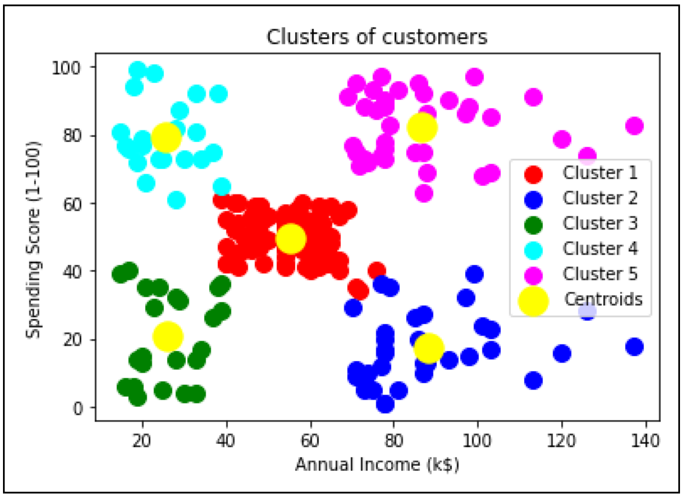
CODE:

```
import numpy as np
import matplotlib.pyplot as plt import pandas as pd
dataset = pd.read_csv('Mall_Customers.csv') dataset.head()
X = dataset.iloc[:, [3, 4]].values
wcss = []
for i in range(1, 11):
        kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
        kmeans.fit(X)
        wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters') plt.ylabel('WCSS')
plt.show()
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X) print(y_kmeans)
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
plt.scatter(kmeans.cluster_centers_[:, 0],
kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroids')
plt.title('Clusters of customers') plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)') plt.legend()
plt.show()
```

OUTPUT:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40



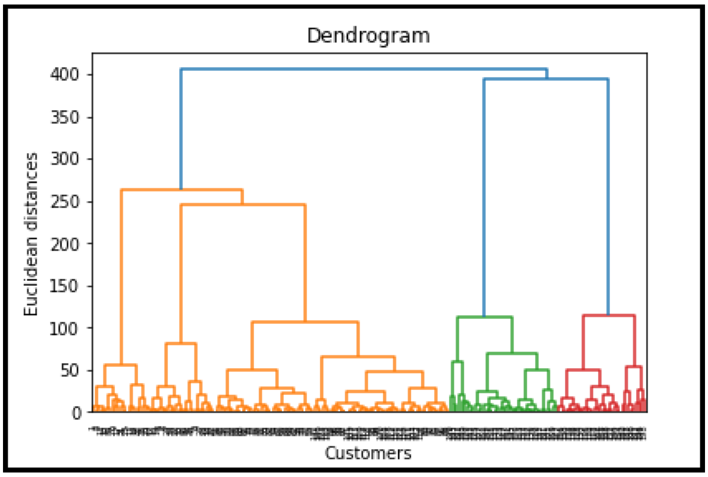


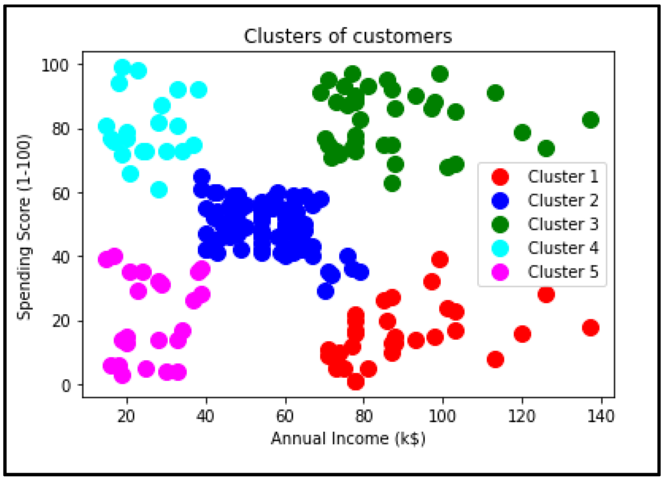
<u>Hierarchical Clustering</u>

CODE:

```
# Importing the libraries import numpy as np
import matplotlib.pyplot as plt import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values len(X)

# Using the dendrogram to find the optimal number of clusters import scipy.cluster.hierarchy
as sch
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward')) plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distances') plt.show()

# Training the Hierarchical Clustering model on the dataset from sklearn.cluster import
AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(X)

print(y_hc)

# Visualising the clusters
plt.scatter(X[y_hc == 0, 0], X[y_hc==0,1],s=100,c     ='red', label = 'Cluster 1')
plt.scatter(X[y_hc == 1, 0], X[y_hc==1,1],s=100,c     =blue,  label = 'Cluster 2')
plt.scatter(X[y_hc == 2, 0], X[y_hc==2,1],s=100,c     ='green', label = 'Cluster 3')
plt.scatter(X[y_hc == 3, 0], X[y_hc==3,1],s=100,c     =cyan,  label = 'Cluster 4')
plt.scatter(X[y_hc == 4, 0], X[y_hc==4,1],s=100,c     ='magenta', label = 'Cluster 5')
plt.title('Clusters of customers') plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)') plt.legend()
plt.show()
```

OUTPUT:






```
[4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4
 3 4 3 4 3 4 1 4 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 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 2 1 2 1 2 0 2 0 2 1 2 0 2 0 2 0 2 0 2 1 2 0 2 1 2
 0 2 0 2 0 2 0 2 0 2 0 2 1 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0
 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2]
```

PART B

<u>K-Means</u>

CODE:

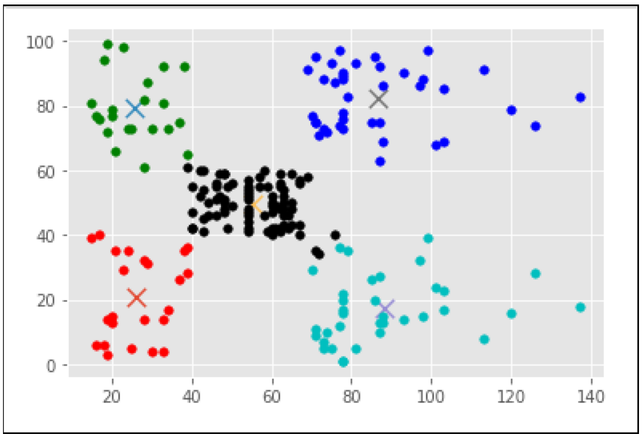
```
import pandas as pd
data = pd.read_csv("driver-data.csv", index_col="id") data.head()
from sklearn.cluster import KMeans kmeans = KMeans(n_clusters=4)
kmeans.fit(data)
kmeans.cluster_centers_ kmeans.labels_
import numpy as np
unique, counts = np.unique(kmeans.labels_, return_counts=True) dict_data = dict(zip(unique, counts))
dict_data

import seaborn as sns
data["cluster"] = kmeans.labels_ sns.pairplot(data)
kmeans.inertia_ kmeans.score
data
from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt from matplotlib import style
import pandas as pd
style.use('ggplot') class K_Means:
def      init      (self, k =3, tolerance = 0.0001, max_iterations = 500):
self.k = k
```

```
self.tolerance = tolerance
self.max_iterations = max_iterations def fit(self, data):
self.centroids = {}

#initialize the centroids, the first 'k' elements in the dataset will be our initial centroids
for i in range(self.k):
self.centroids[i] = data[i]
#begin iterations
for i in range(self.max_iterations): self.classes = {}
for i in range(self.k):
self.classes[i] = []

#find the distance between the point and cluster; choose the nearest centroid
for features in data:
distances = [np.linalg.norm(features - self.centroids[centroid]) for centroid in self.centroids]
classification = distances.index(min(distances))
self.classes[classification].append(features)
previous = dict(self.centroids)
#average the cluster datapoints to re-calculate
for classification in self.classes:
self.centroids[classification] =
np.average(self.classes[classification], axis = 0) isOptimal = True
for centroid in self.centroids:
original_centroid = previous[centroid] curr = self.centroids[centroid]
if np.sum((curr -
original_centroid)/original_centroid * 100.0) > self.tolerance:
isOptimal = False
#break out of the main loop if the results are
optimal, ie. the centroids don't change their positions much(more than our tolerance)
if isOptimal:
break

def pred(self, data):
distances = [np.linalg.norm(data -
self.centroids[centroid]) for centroid in self.centroids] classification =
distances.index(min(distances)) return classification

def main():
df = pd.read_csv("Mall_Customers.csv") df = X = df.iloc[:, [3, 4]]
dataset = df.astype(float).values.tolist()
X = df.values #returns a numpy array km = K_Means(5)
km.fit(X)

# Plotting starts here
colors = 10*["r", "g", "c", "b", "k"]
for centroid in km.centroids:
plt.scatter(km.centroids[centroid][0], km.centroids[centroid][1], s = 130, marker = "x")
for classification in km.classes:
```

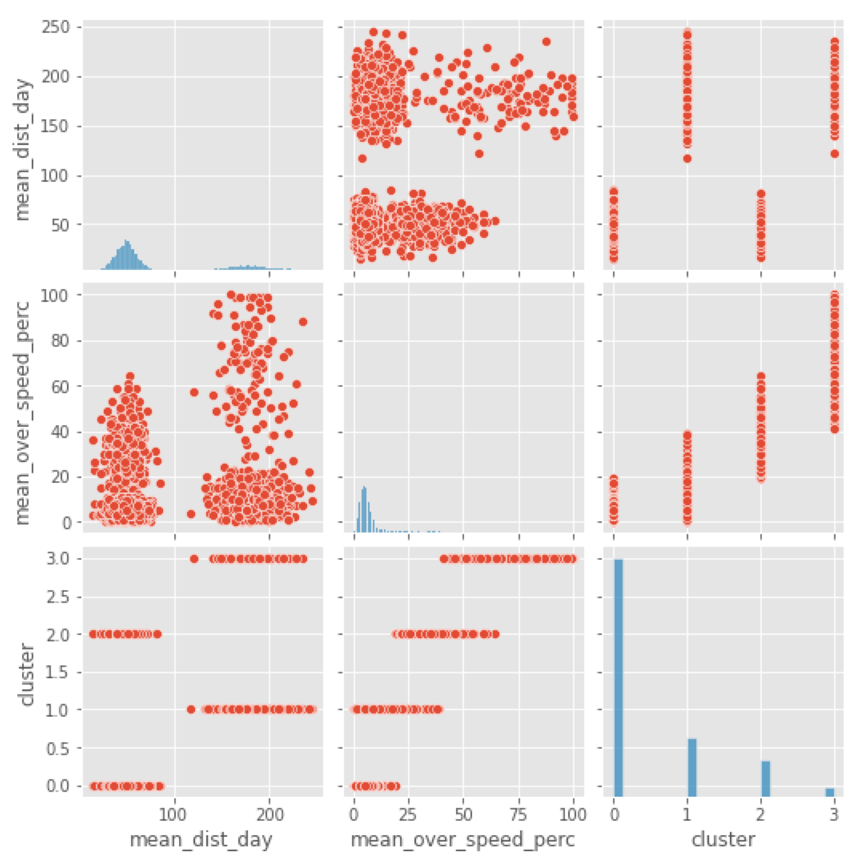
```
color = colors[classification]
for features in km.classes[classification]:
plt.scatter(features[0], features[1], color =
color,s = 30)
plt.show()
if        name   == "     main    ":
          main()
```

OUTPUT:

	mean_dist_day	mean_over_speed_perc
id		
3423311935	71.24	28
3423313212	52.53	25
3423313724	64.54	27
3423311373	55.69	22
3423310999	54.58	25



<seaborn.axisgrid.PairGrid at 0x222b0fc5eb0>



HIERARCHICAL CLUSTERING

CODE:

```
# Importing the libraries import numpy as np
import matplotlib.pyplot as plt import pandas as pd
import seaborn as sns

# Importing the dataset
dataset = pd.read_csv('Mall_Customers.csv') X = dataset.iloc[:, [3, 4]].values
X

new_data = dataset
new_data = new_data.drop('CustomerID', axis=1) new_data
sns.pairplot(dataset)
from sklearn.preprocessing import LabelEncoder
new_data = new_data.apply(LabelEncoder().fit_transform) X = new_data.to_numpy()
class Distance_computation_grid(object):
'''class to enable the Computation of distance matrix'''
def          init      (self): pass
def compute_distance(self,samples):
'''Creates a matrix of distances between individual samples and clusters attained at a
particular step'''
Distance_mat = np.zeros((len(samples),len(samples))) for i in range(Distance_mat.shape[0]):
for j in range(Distance_mat.shape[0]): if i!=j:

Distance_mat[i,j] =
float(self.distance_calculate(samples[i],samples[j]))
else:
Distance_mat[i,j] = 10**4 return Distance_mat


def distance_calculate(self,sample1,sample2):
dist = []
for i in range(len(sample1)):
for j in range(len(sample2)): try:
dist.append(np.linalg.norm(np.array(sample1[i])-np.array(sample2[j])))
except:
dist.append(self.intersampledist(sample1[i],sample2[j])) return min(dist)

def intersampledist(self,s1,s2):
if str(type(s2[0]))!='<class \'list\'>': s2=[s2]
if str(type(s1[0]))!='<class \'list\'>':

s1=[s1]
m = len(s1) n = len(s2) dist = []
if n>=m:
for i in range(n):
for j in range(m):
if (len(s2[i])>=len(s1[j])) and
```

```
str(type(s2[i][0])!='<class \'list\'>'):

dist.append(self.interclusterdist(s2[i],s1[j]))
else:
dist.append(np.linalg.norm(np.array(s2[i])-np.array(s1[j]))) else:
for i in range(m):
for j in range(n):
if (len(s1[i])>=len(s2[j])) and
str(type(s1[i][0])!='<class \'list\'>'):
dist.append(self.interclusterdist(s1[i],s2[j]))
else:
dist.append(np.linalg.norm(np.array(s1[i])-np.array(s2[j]))) return min(dist)

def interclusterdist(self,cl,sample): if sample[0]!='<class \'list\'>':
sample = [sample] dist      = []
for i in range(len(cl)):
for j in range(len(sample)):
dist.append(np.linalg.norm(np.array(cl[i])-np.array(sample[j]))) return min(dist)

progression = [[i] for i in range(X.shape[0])]
samples     = [[list(X[i])] for i in range(X.shape[0])][:10] m = len(samples)
distcal  = Distance_computation_grid() while m>2:
print('Sample size before clustering    :- ',m)
Distance_mat  = distcal.compute_distance(samples) sample_ind_needed =
np.where(Distance_mat==Distance_mat.min())[0]
value_to_add  = samples.pop(sample_ind_needed[1])
samples[sample_ind_needed[0]].append(value_to_add)
print('Cluster Node 1
:-',progression[sample_ind_needed[0]]) print('Cluster Node 2
:-',progression[sample_ind_needed[1]])

progression[sample_ind_needed[0]].append(progression[sample_ind_ne eded[1]])
progression[sample_ind_needed[0]] = [progression[sample_ind_needed[0]]]
v = progression.pop(sample_ind_needed[1]) m = len(samples)

print('Progression(Current Sample)    :-',progression) print('Cluster attained
:-',progression[sample_ind_needed[0]])
print('Sample size after clustering     :-',m) print('\n')

from scipy.cluster.hierarchy import dendrogram, linkage from matplotlib import pyplot as plt
Z = linkage(X, 'single')
fig = plt.figure(figsize=(8, 8)) plt.title('Dendrogram')

dn = dendrogram(Z)
plt.scatter(X[:,2], X[:,3], cmap="rainbow")
from sklearn.cluster import AgglomerativeClustering aggclus =
AgglomerativeClustering().fit(X)
aggclus.labels_
```

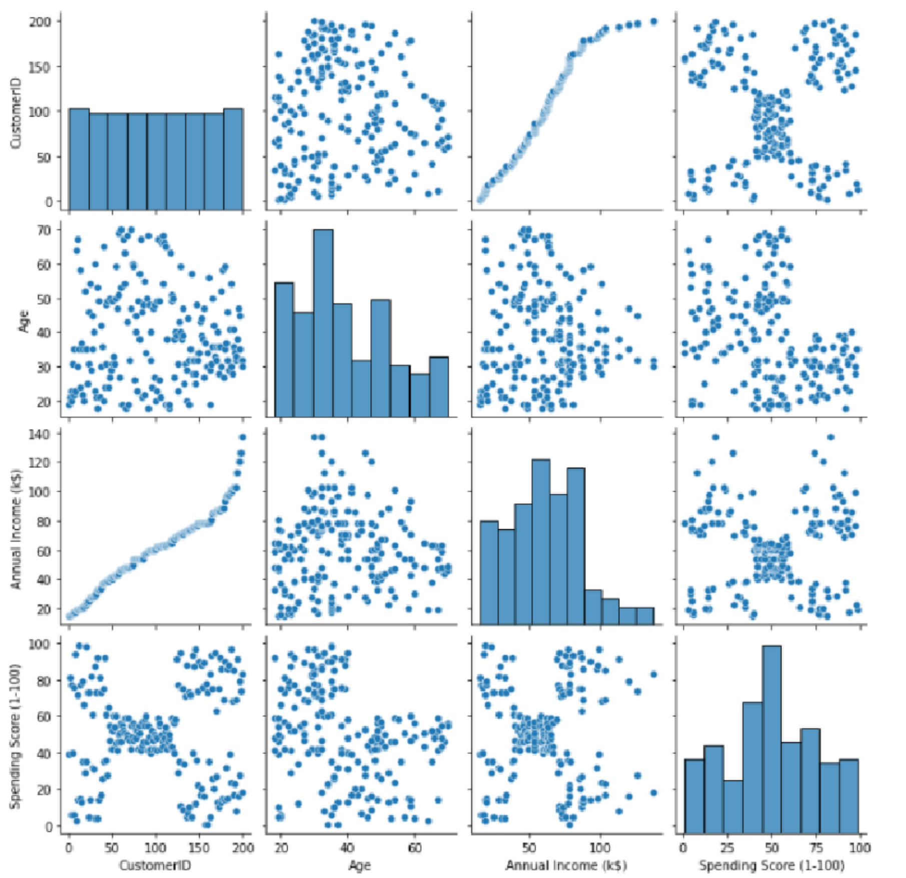
OUTPUT:

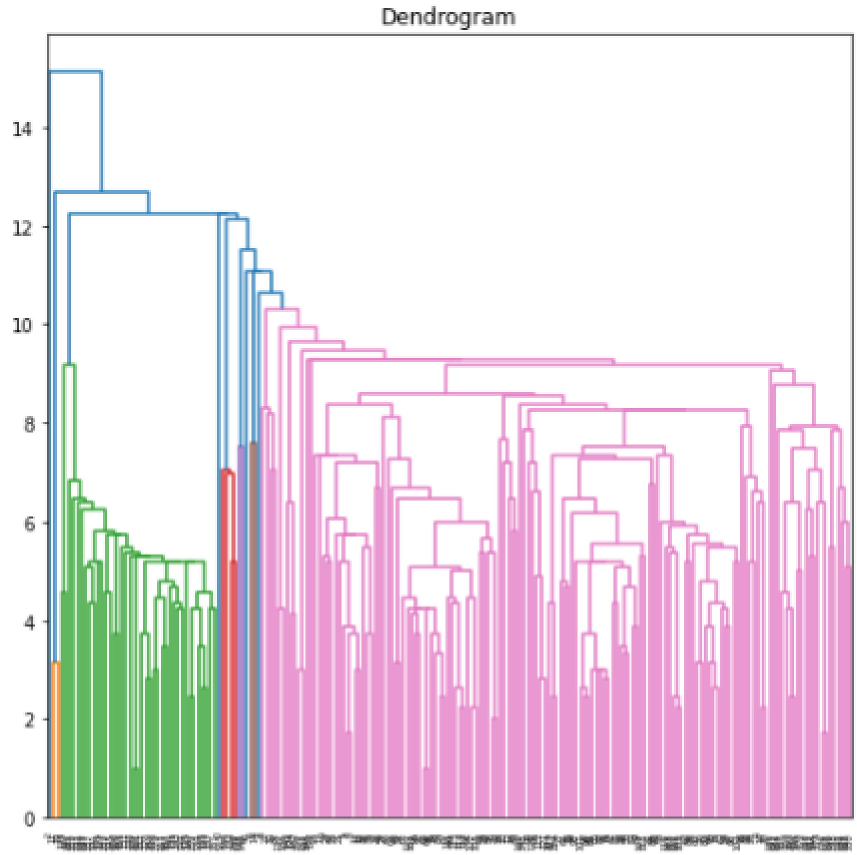
```
array([[ 15,  39],
       [ 15,  81],
       [ 16,   6],
       [ 16,  77],
       [ 17,  40],
       [ 17,  76],
       [ 18,   6],
       [ 18,  94],
       [ 19,   3],
       [ 19,  72],
       [ 19,  14],
       [ 19,  99],
       [ 20,  15],
       [ 20,  77],
       [ 20,  13],
       [ 20,  79],
       [ 21,  35],
       [ 21,  66],
       [ 23,  29],
```

	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	Male	19	15	39
1	Male	21	15	81
2	Female	20	16	6
3	Female	23	16	77
4	Female	31	17	40
...	...	...	...	...
195	Female	35	120	79
196	Female	45	126	28
197	Male	32	126	74
198	Male	32	137	18
199	Male	30	137	83

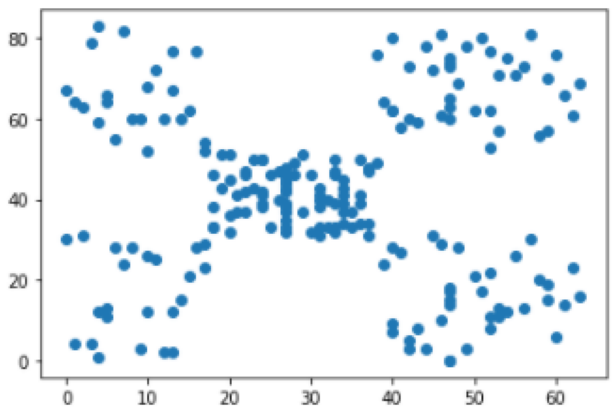
200 rows × 4 columns

```
<seaborn.axisgrid.PairGrid at 0x1eb36cc4e20>
```






```
<matplotlib.collections.PathCollection at 0x1eb365528e0>
```




```
array([0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1,
       1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       0, 1], dtype=int64)
```

PART C

CODE:

```
from sklearn import datasets, preprocessing
from sklearn.preprocessing import LabelEncoder from sklearn.cluster import KMeans

df=pd.read_csv('Mall_Customers.csv')
df = df.apply(LabelEncoder().fit_transform)

scaler = preprocessing.StandardScaler() scaled_df = scaler.fit_transform(df)
pd.DataFrame(scaled_df).describe() clusters = range(1, 11)
sse=[]
for cluster in clusters:
model = KMeans(n_clusters=cluster, init='k-means++', max_iter=300, tol=0.0001,
verbose=0,random_state=0)
model.fit(scaled_df)
sse.append(model.inertia_)
sse_df = pd.DataFrame(np.column_stack((clusters, sse)), columns=['cluster', 'SSE'])
fig, ax = plt.subplots(figsize=(13, 5))
ax.plot(sse_df['cluster'], sse_df['SSE'], marker='o') ax.set_xlabel('Number of clusters')
```

OUTPUT: