Code Analysis:

Step1: Import the libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

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

Step2: Loading the Dataset

```
In [87]: df = pd.read_csv("C:/Users/admin/Desktop/21BCE5746/Assignment 3/Mall_Customers.csv")

In [88]: df
Out[88]:
```

	CustomerID	Gender	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
...	...	...	...	...	...
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

200 rows × 5 columns

Step 3: preprocessing the dataset

df.head()

```
In [89]: df.head()
Out[89]:
```

	CustomerID	Gender	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

The above command gives the top 5 rows of the dataset

df.info()

```
In [90]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Gender                  200 non-null    object
 2   Age                     200 non-null    int64
 3   Annual Income (k$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

df.info() – it includes the data types of columns, non-null counts, and memory usage in one line.

df.isnull().sum()

df.isnull().sum() provides the count of null (missing) values for each column in the DataFrame 'df'.

df.duplicated().sum()

df.duplicated().sum() provides the count of all duplicate values for each column in the DataFrame 'df'.

df.describe()

we use describe method to Provide summary statistics for key features of the data contains count, mean, std, min,1 st quartile,2nd quartile, 3rd quartile and the maximum in each column The above creates subplots for each column in your DataFrame df, displaying boxplots of the data. However, it seems you want to create 9 subplots vertically arranged, each representing a column in the DataFrame.

```
In [91]: df.isnull().sum()

Out[91]: CustomerID                0
         Gender                    0
         Age                       0
         Annual Income (k$)        0
         Spending Score (1-100)    0
         dtype: int64

In [92]: df.duplicated().sum()

Out[92]: 0

In [93]: df.describe()

Out[93]:
```

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

```
import seaborn as sns

#correlation

 corr = df.corr() plt.figure(dpi=140)

sns.heatmap(df.corr(), annot=True, fmt= '.3f')

plt.show()
```

The above code utilizes Seaborn and Matplotlib to generate a heatmap displaying the correlation matrix of a DataFrame df. Each cell in the heatmap represents the correlation coefficient between two variables, with annotations showing the correlation values. Positive correlations are depicted in brighter colors, negative correlations in darker colors, and zero correlations in neutral colors. This visualization aids in identifying relationships and patterns among variables, facilitating data exploration and feature selection processes.

```
In [98]: import seaborn as sns
         #correlation
         corr = df.corr()

         plt.figure(dpi=140)
         sns.heatmap(df.corr(), annot=True, fmt= '.3f')
         plt.show()
```

Spliting dataset into independent variables and dependent variables

X = df.iloc[:,:-1]

y = df.iloc[:,-1]

```
Out[101]:
```

	CustomerID	Gender	Age	Annual Income (k$)
0	1	0	19	15
1	2	0	21	15
2	3	1	20	16
3	4	1	23	16
4	5	1	31	17
...	...	...	...	...
195	196	1	35	120
196	197	1	45	126
197	198	0	32	126
198	199	0	32	137
199	200	0	30	137

200 rows × 4 columns

```
In [102]: y = df.iloc[:,-1]

In [103]: y

Out[103]: 0      39
          1      81
          2       6
          3      77
          4      40
                 ..
          195    79
          196    28
          197    74
          198    18
          199    83
          Name: Spending Score (1-100), Length: 200, dtype: int64
```

from sklearn.cluster import KMeans

wcss = []

for i in range(1, 11):

kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)

kmeans.fit(X)

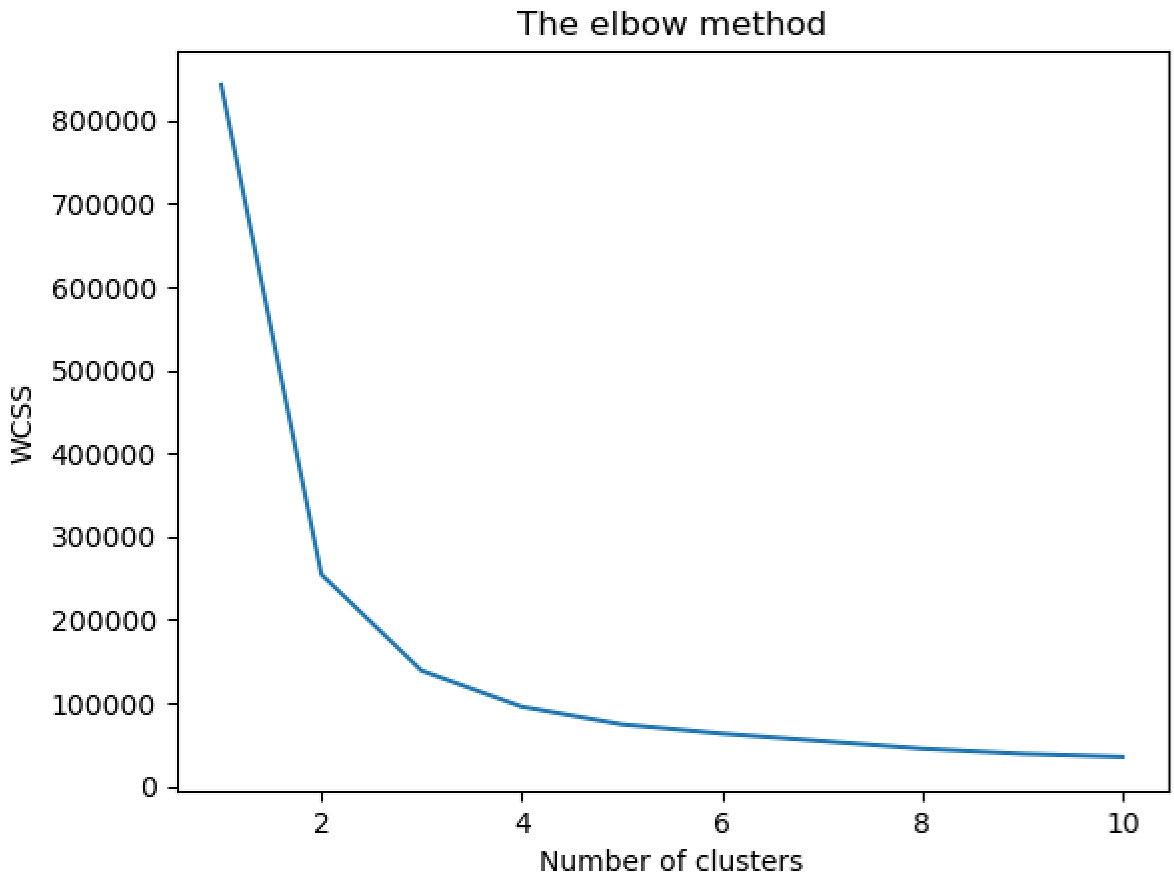
wcss.append(kmeans.inertia_)

wcss

The above code gives the KMeans clustering algorithm from scikit-learn to compute the withincluster sum of squares (WCSS) for different numbers of clusters ranging from 1 to 10. For each number of clusters, KMeans is fitted to the data X, and the corresponding WCSS value is appended to a list. The resulting list contains the WCSS values for each number of clusters, which can be used to determine the optimal number of clusters for the dataset.

```
plt.plot(range(1, 11), wcss)

plt.title('The elbow method')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS') #within cluster sum of squares

plt.show()
```

The code plots the number of clusters against the within-cluster sum of squares (WCSS) values calculated previously. It visualizes the "elbow method," helping to identify the optimal number of clusters. The plot's x-axis represents the number of clusters, while the y-axis represents the corresponding WCSS values. By inspecting the plot, the point where the decrease in WCSS slows down (the "elbow") indicates the optimal number of clusters. This helps in making an informed decision about the appropriate number of clusters for the dataset. Here in our data the no of clusters is 3 for better accuracy.

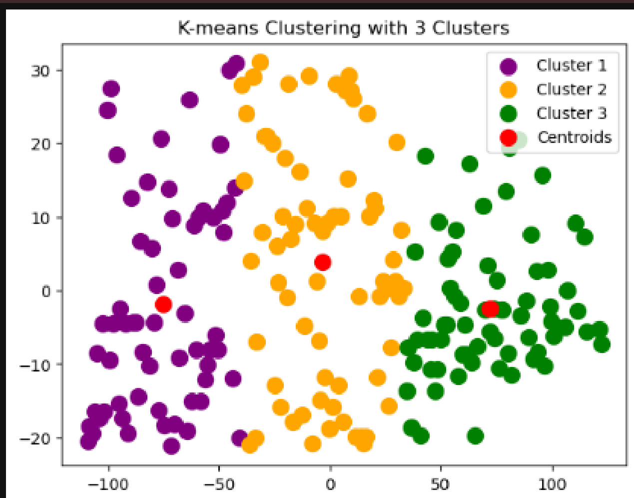


The selection of the number of clusters depends on the data's complexity and structure, and the desired level of granularity in the clustering results. It's essential to consider various factors and validation methods to choose the most suitable number of clusters for a particular dataset. KMeans for the different no of clusters 3,5,7 We need to perform K-means clustering with 3,5,7 clusters on the data X using KMeans from scikitlearn. It then visualizes the clusters and centroids in a scatter plot using principal component analysis (PCA) to reduce the dimensionality of the data. Each cluster is represented by a different color, while centroids are marked in red. The plot provides insight into the distribution and separation of data points among the clusters, aiding in understanding the clustering results.

Performance metrics for clusters = 3:

In [110]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=300, n_init=10, random_state=0)
y_kmeans_3 = kmeans.fit_predict(X)

plt.scatter(X_pca[y_kmeans_3 == 0, 0], X_pca[y_kmeans_3 == 0, 1], s=100, c='purple', label='Cluster 1')
plt.scatter(X_pca[y_kmeans_3 == 1, 0], X_pca[y_kmeans_3 == 1, 1], s=100, c='orange', label='Cluster 2')
plt.scatter(X_pca[y_kmeans_3 == 2, 0], X_pca[y_kmeans_3 == 2, 1], s=100, c='green', label='Cluster 3')

plt.scatter(pca.transform(kmeans.cluster_centers_)[:, 0], pca.transform(kmeans.cluster_centers_)[:, 1], s=100, c='red', label='C
plt.title('K-means Clustering with 3 Clusters')
plt.legend()
plt.show()
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarning: KMeans is known to have a memory leak
on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
```



In [111]:
```
from sklearn.metrics import silhouette_score, calinski_harabasz_score, davies_bouldin_score, homogeneity_completeness_v_measure

davies_bouldin_kmeans = davies_bouldin_score(X, y_kmeans_3)
```

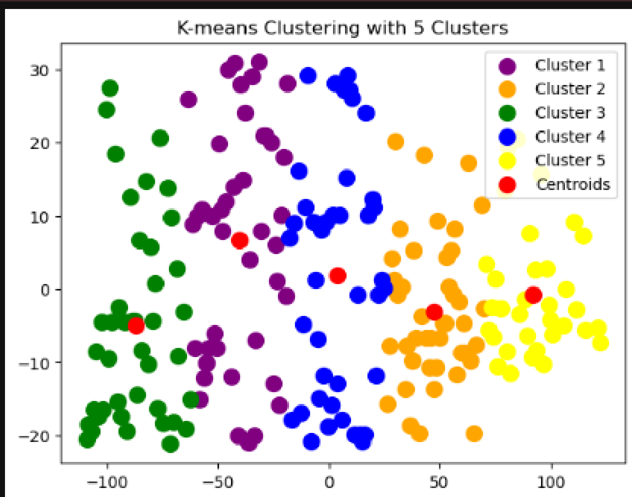
In [112]:
```
# Print metrics for K-means
print("Metrics for K-means for 3 clusters:")
print(f"Davies-Bouldin Index: {davies_bouldin_kmeans}")
print()
```

```
Metrics for K-means for 3 clusters:
Davies-Bouldin Index: 0.6626318945479001
```

Performance metrics for clusters = 5:

In [113]:
```
kmeans = KMeans(n_clusters=5, init='k-means++', max_iter=300, n_init=10, random_state=0)
y_kmeans_5 = kmeans.fit_predict(X)

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

plt.scatter(X_pca[y_kmeans_5 == 0, 0], X_pca[y_kmeans_5 == 0, 1], s=100, c='purple', label='Cluster 1')
plt.scatter(X_pca[y_kmeans_5 == 1, 0], X_pca[y_kmeans_5 == 1, 1], s=100, c='orange', label='Cluster 2')
plt.scatter(X_pca[y_kmeans_5 == 2, 0], X_pca[y_kmeans_5 == 2, 1], s=100, c='green', label='Cluster 3')
plt.scatter(X_pca[y_kmeans_5 == 3, 0], X_pca[y_kmeans_5 == 3, 1], s=100, c='blue', label='Cluster 4')
plt.scatter(X_pca[y_kmeans_5 == 4, 0], X_pca[y_kmeans_5 == 4, 1], s=100, c='yellow', label='Cluster 5')

plt.scatter(pca.transform(kmeans.cluster_centers_)[:, 0], pca.transform(kmeans.cluster_centers_)[:, 1], s=100, c='red', label='C
plt.title('K-means Clustering with 5 Clusters')
plt.legend()
plt.show()
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarning: KMeans is known to have a memory leak
on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
```



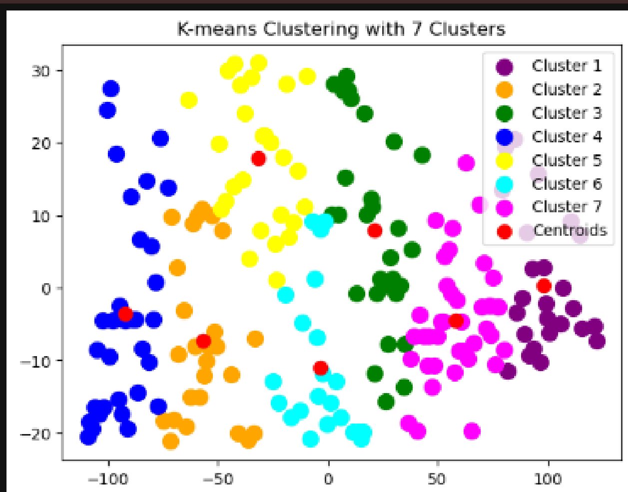
In [116]:
```
davies_bouldin_kmeans5 = davies_bouldin_score(X, y_kmeans_5)
```

In [117]:
```
print("Metrics for K-means for 5 clusters :")
print(f"Davies-Bouldin Index: {davies_bouldin_kmeans5}")
```

```
Metrics for K-means for 5 clusters :
Davies-Bouldin Index: 0.8015524947479726
```

Performance metrics for clusters = 7:

```
In [119]: kmeans = KMeans(n_clusters=7, init='k-means++', max_iter=300, n_init=10, random_state=0)
          y_kmeans_7 = kmeans.fit_predict(X)

          pca = PCA(n_components=2)
          X_pca = pca.fit_transform(X)

          plt.scatter(X_pca[y_kmeans_7 == 0, 0], X_pca[y_kmeans_7 == 0, 1], s=100, c='purple', label='Cluster 1')
          plt.scatter(X_pca[y_kmeans_7 == 1, 0], X_pca[y_kmeans_7 == 1, 1], s=100, c='orange', label='Cluster 2')
          plt.scatter(X_pca[y_kmeans_7 == 2, 0], X_pca[y_kmeans_7 == 2, 1], s=100, c='green', label='Cluster 3')
          plt.scatter(X_pca[y_kmeans_7 == 3, 0], X_pca[y_kmeans_7 == 3, 1], s=100, c='blue', label='Cluster 4')
          plt.scatter(X_pca[y_kmeans_7 == 4, 0], X_pca[y_kmeans_7 == 4, 1], s=100, c='yellow', label='Cluster 5')
          plt.scatter(X_pca[y_kmeans_7 == 5, 0], X_pca[y_kmeans_7 == 5, 1], s=100, c='cyan', label='Cluster 6')
          plt.scatter(X_pca[y_kmeans_7 == 6, 0], X_pca[y_kmeans_7 == 6, 1], s=100, c='magenta', label='Cluster 7')

          plt.scatter(pca.transform(kmeans.cluster_centers_)[:, 0], pca.transform(kmeans.cluster_centers_)[:, 1], s=300, c='red', marker='

          plt.title('K-means Clustering with 7 Clusters')
          plt.legend()
          plt.show()

          C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarning: KMeans is known to have a memory leak
          on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OM
          P_NUM_THREADS=1.
            warnings.warn(
```




```
In [120]: davies_bouldin_kmeans = davies_bouldin_score(X, y_kmeans_7)

In [121]: print("Metrics for K-means for 7 clusters:")
          print(f"Davies-Bouldin Index: {davies_bouldin_kmeans}")

          Metrics for K-means for 7 clusters:
          Davies-Bouldin Index: 0.8331419695296726
```

Davies-Bouldin Index: Lower values indicate better clustering.

From the above metrics:

→ 3 clusters: Davies-Bouldin Index = 0.6626

→ 5 clusters: Davies-Bouldin Index = 0.8015

→ 7 clusters: Davies-Bouldin Index = 0.8331

Based on the overall comparison of metrics, K-meansclustering with 3 clusters consistently outperforms K-medoids clustering with 5 or 7 clusters according to the

Davies-Bouldin index. Therefore, K-meansclustering with 3 clusters seems to be the most suitable choice for this dataset based on these metrics. However, it's essential to consider other factors such as domain knowledge and specific clustering objectives when determining the optimal number of clusters for a given dataset

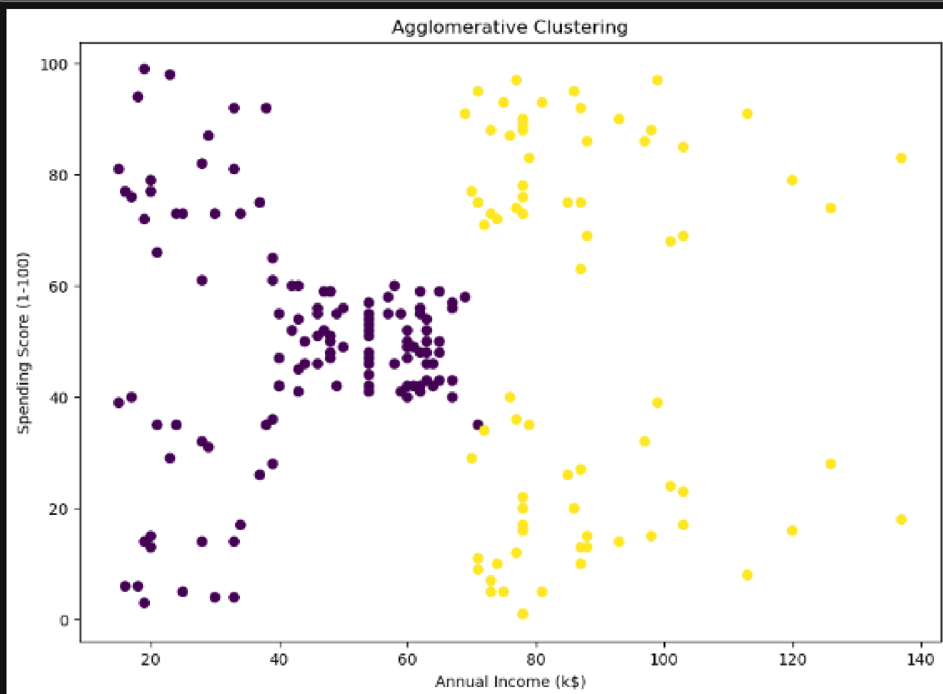
Dendrogram

For the dataset by seeing that Agglomerative Clustering gives the below performance

Davies-Bouldin Index: 0.7848307942899339

Agglomerative clustering:

```
In [125]: from sklearn.cluster import AgglomerativeClustering

cluster = AgglomerativeClustering(n_clusters=2, metric='euclidean', linkage='ward')
cluster_labels = cluster.fit_predict(df)

plt.figure(figsize=(10, 7))
plt.scatter(df['Annual Income (k$)'], df['Spending Score (1-100)'], c=cluster_labels)
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.title('Agglomerative Clustering')
plt.show()
```




```
In [126]: davies_bouldin_index = davies_bouldin_score(df, cluster_labels)
print("Davies-Bouldin Index:", davies_bouldin_index)

Davies-Bouldin Index: 0.7848307942899339
```

Overall Analysis , We get better results with kmeans clusters with n = 3 because the low values of Davies-Bouldin Index gives better values

In [133]:

```python
from sklearn.cluster import DBSCAN

X_train = df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']]

clustering = DBSCAN(eps=12.5, min_samples=4).fit(X_train)
DBSCAN_dataset = X_train.copy()
DBSCAN_dataset.loc[:,'Cluster'] = clustering.labels_

DBSCAN_dataset.Cluster.value_counts().to_frame()

outliers = DBSCAN_dataset[DBSCAN_dataset['Cluster']==-1]

fig2, axes = plt.subplots(1, 2, figsize=(12, 5))

sns.scatterplot(x='Annual Income (k$)', y='Spending Score (1-100)',
                data=DBSCAN_dataset,  # Pass the entire DataFrame here
                hue='Cluster', ax=axes[0], palette='Set2', legend='full', s=200)

sns.scatterplot(x='Age', y='Spending Score (1-100)',
                data=DBSCAN_dataset,  # Pass the entire DataFrame here
                hue='Cluster', palette='Set2', ax=axes[1], legend='full', s=200)

axes[0].scatter(outliers['Annual Income (k$)'], outliers['Spending Score (1-100)'], s=10, label='outliers', c="k")
axes[1].scatter(outliers['Age'], outliers['Spending Score (1-100)'], s=10, label='outliers', c="k")

axes[0].legend()
axes[1].legend()

plt.setp(axes[0].get_legend().get_texts(), fontsize='12')
plt.setp(axes[1].get_legend().get_texts(), fontsize='12')

plt.show()
```