HOMEWORK 4

Question # 1:

Prompt 1: Could you please provide detailed descriptions of how a leading healthcare organization, a prominent financial institution, and a well-known manufacturing company have all implemented data clustering algorithms in their respective business operations? Describe how each organization leverages data clustering to achieve specific goals or improve processes within their sectors.

1. Leading Healthcare Organization (e.g., Mayo Clinic):
    - Patient Segmentation: The healthcare organization uses data clustering algorithms on extensive patient data, including medical history, demographics, and test results. They segment patients into distinct groups based on similarities in medical conditions, risk factors, and treatment responses.
    - Personalized Treatment: By understanding patient clusters, the organization tailors treatment plans more effectively. Specific medications, therapies, and interventions are recommended based on the patient's cluster, leading to improved treatment outcomes.
    - Resource Allocation: Data clustering helps allocate resources efficiently. Patients with similar needs are grouped, optimizing staffing, equipment, and facilities usage while reducing costs and wait times.
    - Research and Drug Development: Clustering is also applied to genomic data for clinical trials and drug development, identifying patient cohorts for targeted treatments and accelerating medical advancements.
2. Prominent Financial Institution (e.g., JPMorgan Chase & Co.):
    - Fraud Detection: The financial institution utilizes data clustering to detect fraudulent activities. Transactions are clustered based on attributes like amount, location, and time, helping to identify patterns of fraudulent behavior.
    - Customer Segmentation: Clustering is employed to segment the customer base according to spending habits, financial goals, and risk tolerance. Personalized financial products and services are then offered to customers, enhancing their experience.
    - Risk Assessment: Data clustering assists in assessing credit risk. Borrowers with similar credit profiles are grouped, allowing the institution to make more accurate lending decisions and set appropriate interest rates.
    - Investment Portfolio Recommendation: Clustering algorithms help recommend investment portfolios tailored to specific customer segments, increasing the relevance of financial advice.
3. Well-Known Manufacturing Company (e.g., Toyota Motor Corporation):
    - Demand Forecasting: In manufacturing, data clustering is used for demand forecasting. Products are clustered based on demand patterns, allowing the company to optimize inventory levels and production schedules.
    - Quality Control: Clustering is applied to group similar products or components. If a defect is found in one item from a cluster, it triggers inspections or recalls for all items in that cluster, ensuring product quality.
    - Supplier Segmentation: Suppliers are segmented using clustering algorithms, considering factors like reliability, cost, and quality. This helps the company allocate resources efficiently and maintain strong supplier relationships.
    - Process Optimization: Clustering is used to identify bottlenecks and areas for improvement in manufacturing processes. Production data is clustered to fine-tune processes for higher efficiency and reduced waste.

Prompt 2: Can you provide detailed descriptions of how data clustering algorithms have been employed by companies or organizations in the healthcare, finance, and manufacturing sectors to enhance their operations and decision-making processes? Please elaborate on the specific applications and benefits of data clustering in each of these sectors.

1. Patient Segmentation and Personalized Treatment (Mayo Clinic):
   - *Application*: Mayo Clinic uses data clustering algorithms to segment patients based on medical records, including demographics, medical history, and test results.
   - *Benefits*: This segmentation helps in tailoring personalized treatment plans. Patients with similar conditions and characteristics are grouped together, allowing healthcare providers to recommend treatments and medications that have been more successful for similar patients. This leads to improved treatment outcomes and patient satisfaction.
2. Resource Allocation and Efficiency:
   - *Application*: Healthcare organizations use clustering to optimize resource allocation. Patients with similar needs are grouped together, enabling efficient staffing, equipment usage, and facility allocation.
   - *Benefits*: This results in cost reduction and reduced wait times for patients. It ensures that resources are distributed where they are most needed, improving overall healthcare service delivery.
3. Drug Development and Clinical Trials:
   - *Application*: Clustering is applied to genomic data to identify patient cohorts for clinical trials and drug development.
   - *Benefits*: It accelerates the identification of suitable candidates for trials, leading to faster drug development and more targeted treatments. This can ultimately save lives and improve the quality of care.

Finance Sector:

1. Fraud Detection (JPMorgan Chase & Co.):
   - *Application*: Financial institutions use data clustering algorithms to detect fraudulent activities by clustering transactions based on attributes like amount, location, and time.
   - *Benefits*: This helps in identifying patterns of fraudulent behavior quickly, reducing financial losses and enhancing security for customers.
2. Customer Segmentation and Personalized Services:
   - *Application*: Clustering is employed to segment customers based on spending habits, financial goals, and risk tolerance.
   - *Benefits*: Financial institutions can offer personalized financial products and services to customers. This improves customer satisfaction and increases the relevance of financial advice.
3. Risk Assessment:
   - *Application*: Data clustering assists in assessing credit risk by grouping borrowers with similar credit profiles.
   - *Benefits*: It allows financial institutions to make more accurate lending decisions and set appropriate interest rates, reducing the risk of defaults and losses.

Manufacturing Sector:

1. Demand Forecasting:
   - *Application*: Manufacturing companies use data clustering for demand forecasting by clustering products based on demand patterns.

- *Benefits*: This optimization of inventory levels and production schedules reduces carrying costs, prevents overstocking or understocking, and ensures products are available when needed.

2. Quality Control:

    - *Application*: Clustering is applied to group similar products or components to ensure quality control.
    - *Benefits*: If a defect is found in one item from a cluster, it triggers inspections or recalls for all items in that cluster, guaranteeing consistent product quality.

3. Supplier Segmentation:

    - *Application*: Suppliers are segmented using clustering algorithms based on factors like reliability, cost, and quality.
    - *Benefits*: It helps in efficient resource allocation and maintaining strong supplier relationships, leading to cost savings and better-quality materials.

4. Process Optimization:

    - *Application*: Clustering is used to identify bottlenecks and areas for improvement in manufacturing processes.
    - *Benefits*: Fine-tuning processes for higher efficiency and reduced waste leads to cost savings and increased productivity.

You

Prompt: "Can you provide detailed descriptions of how real-world companies or organizations in three distinct sectors—healthcare, finance, and manufacturing—have implemented data clustering algorithms in their business operations?

ChatGPT

Certainly! Here are detailed descriptions of how real-world companies or organizations in the healthcare, finance, and manufacturing sectors have implemented data clustering algorithms in their business operations:

1. Healthcare Sector:
   Company: Mayo Clinic
   Implementation: Patient Segmentation for Personalized Healthcare
   Description:
   Mayo Clinic, one of the world's leading healthcare organizations, has implemented data clustering algorithms to enhance patient care and treatment outcomes. They utilize clustering techniques on a vast dataset of patient medical records, including demographics, medical history, and test results. Here's how they've used clustering:
   - Patient Segmentation: Mayo Clinic employs clustering algorithms to segment patients into distinct groups based on similar medical conditions, risk factors, and treatment responses. For example, they may cluster diabetic patients into subgroups based on their insulin resistance levels, age, and comorbidities.
   - Personalized Treatment Plans: By understanding patient clusters, Mayo Clinic can tailor treatment plans more effectively. For instance, they can recommend specific medications or therapies based on the patient's cluster and its historical success rates, ultimately improving patient outcomes.
   - Resource Allocation: The clinic optimizes resource allocation by clustering patients with similar needs. This ensures that staff, equipment, and facilities are utilized efficiently, reducing costs and wait times.
   - Research and Drug Development: Mayo Clinic also applies clustering algorithms to genomic data to identify patient cohorts for clinical trials and drug development, leading to more targeted and effective treatments.

targeted and effective treatments.

2. Finance Sector:
   Company: JPMorgan Chase & Co.
   Implementation: Fraud Detection and Customer Segmentation
   Description:
   JPMorgan Chase, a global financial institution, relies on data clustering algorithms to improve security and customer experience. They employ clustering in the following ways:
   - Fraud Detection: JPMorgan uses clustering to identify patterns of fraudulent activities. By clustering transactions based on various attributes like transaction amount, location, and time, they can spot anomalies more effectively. For instance, if a cluster of transactions shows unusual behavior, it may trigger a fraud alert.
   - Customer Segmentation: Clustering algorithms help JPMorgan segment its customer base. By analyzing spending habits, financial goals, and risk tolerance, they can offer personalized financial products and services. For example, they can recommend investment portfolios tailored to specific customer segments.
   - Risk Assessment: Clustering is used to assess credit risk. By grouping borrowers with similar credit profiles, JPMorgan can make more accurate lending decisions and set appropriate interest rates.

3. Manufacturing Sector:

   Company: Toyota Motor Corporation

   Implementation: Supply Chain Optimization

   Description:

   Toyota, a global automotive manufacturer, implements data clustering algorithms to optimize its manufacturing and supply chain operations:

   - Demand Forecasting: Toyota uses clustering to categorize products based on demand patterns. By identifying clusters of products with similar demand fluctuations, they can optimize inventory levels and production schedules.
   - Quality Control: Clustering helps in quality control by grouping similar products or components. If a defect is found in one item from a cluster, it triggers inspections or recalls for all items in that cluster, ensuring product quality.
   - Supplier Segmentation: Toyota segments its suppliers using clustering algorithms to better manage relationships. Suppliers are grouped based on factors like reliability, cost, and quality, allowing Toyota to allocate resources efficiently and maintain strong partnerships.
   - Process Optimization: Clustering is applied to manufacturing processes to identify bottlenecks and areas for improvement. By clustering production data, Toyota can fine-tune processes for higher efficiency and reduced waste.

These examples demonstrate how data clustering algorithms are applied in diverse ways across the healthcare, finance, and manufacturing sectors to enhance decision-making, optimize operations, and provide better services to customers.

Explanations and Modifications

1. Healthcare Sector (Mayo Clinic): Original Description: The original description is sufficient in explaining how data clustering is utilized in healthcare. It mentions patient segmentation based on medical conditions, personalized treatment plans, resource allocation, and research. However, it could benefit from a brief mention of the similarity measure used, such as Euclidean distance or cosine similarity.

Evaluation: This description provides a basic understanding of data clustering in healthcare but lacks specifics on the clustering algorithm used and how similarity measures are defined and applied in clustering patient data.

Modification for Clarity: The healthcare provider employs the K-means clustering algorithm to analyze cluster electronic health records (EHRs). They use a similarity measure based on a combination of symptomatology, genetic information, and response to past treatments. This measure helps in forming clusters of patients with similar health profiles, enabling the provider to tailor treatments more effectively.

2. Finance Sector (JPMorgan Chase & Co.): Original Description: The original description is sufficient in explaining how data clustering is utilized in the finance sector. It covers fraud detection, customer segmentation, and risk assessment. However, it lacks details about the specific similarity measure used in clustering.

Evaluation: The response outlines the application of data clustering in customer segmentation but does not detail the specific type of clustering algorithm used or how the similarity measures are computed and applied.

Modification for Clarity: The bank uses hierarchical clustering for customer segmentation. It employs a similarity measure based on Euclidean distance, considering variables like transaction frequencies, amounts, credit scores, and account types. This method enables the bank to identify distinct customer groups for targeted marketing and personalized service offerings.

3. Manufacturing Sector (Toyota Motor Corporation): Original Description: The original description is sufficient in explaining how data clustering is utilized in the manufacturing sector. It covers demand forecasting, quality control, supplier segmentation, and process optimization. However, it does not mention the specific similarity measure used in clustering.

Evaluation: The description mentions the use of data clustering in supply chain management but lacks detail on the specific clustering algorithm and how similarity measures are applied.

Modification for Clarity: The manufacturing firm employs the DBSCAN clustering algorithm for its supply chain data. They use a similarity measure that incorporates delivery punctuality, cost-effectiveness, and quality ratings. This approach allows the firm to categorize suppliers into various clusters, enabling them to make informed decisions about supplier relationships and supply chain optimizations.

Question 2:

Q:2

X	Y
6.57	19.09
8.76	22.85
1.88	3.76
7.20	20.35
5.01	31.61
5.63	-17.22
4.38	-15.96
0.69	2.50

Step1 : Initialize with 1 cluster

Start with data points in a single cluster, cluster A.

cluster A; (6.57, 19.09), (8.76, 22.85), (1.88, 3.76), (7.20, 20.35), (5.01, 31.61), (5.63, -17.22), (4.38, -15.96), (0.69, 2

step2 : calculate initial SSE for cluster A

~~centroid~~

calculate centroid of cluster A

centroid of cluster A = (6.57+8.76+1.88+7.20+5.01+5.63+4.38+0.69)/8

$(19.09 + 22.85 + 3.76 + 20.35 + 31.61 +$
$17.22 + -15.96 + 2.50) / 8$

$= (5.73, 7.59)$

→ Calculate squared distance between each data point and centroid of cluster A and sum them up.

SSE (cluster A) = $\sum$ (Distance (datapoint, CCA)^2)

1. Data point (6.57, 19.09)

distance $= \sqrt{(6.57 - 5.73)^2 + (19.09 - 7.59)^2}$

$= 11.55$

squared distance $= (11.55)^2 = 133.10$

2. Data point (8.76, 22.85)

Distance $= \sqrt{(8.76 - 5.73)^2 + (22.85 - 7.59)^2}$

$= 15.57$

sq distance $= (15.57)^2 = 242.48$

3. Data point $(1.88, 3.76)$

sq Distance $= 35.48$

4. Data point $(7.20, 20.35)$

sq Distance $= 91.13$

5. Data point $(5.01, 31.61)$

sq distance $= 267.37$

6. Data point $(5.63, -17.22)$

sq distance $= 336.83$

7. Data points $(4.38, -15.96)$

$= 282.01$

8. $(0.69, 2.50) = 19.58$

sum up all sq distances to calculate SSE for cluster A

SSE (cluster A) = (133.1025 + 242.4889 + 35.48 + 91.13 + 267.37 + 336.83 + 282.01 + 19.58)

≈ 1407.94

step 3 :

split cluster with largest SSE

Since we have only 1 cluster i.e cluster A, we will split into 2 sub clusters A1 and A2.

→ Perform regular k-means clustering on data points in cluster A to divide into two sub clusters, A1 and A2.

→ We will use K-means to calculate new centroid for A1 and A2.

→ calculate SSE for cluster A1 and cluster A2 seperately.

Lets initialize

centroid for A1 : $(6.57, 19.09)$

centroid for A2 : $(8.76, 22.85)$

Euclidean distance is used for assigning points to nearest centroids.

$\Rightarrow$ Distance calculations to each centroid for points $(6.57, 19.09)$ the distances are

$$\text{Distance to A1} = \sqrt{(6.57-6.57)^2+(19.09-19.09)^2}$$
$$= 0$$

$$\text{Distance to A2} = \sqrt{(6.57-8.76)^2+(19.09-22.85)^2}$$
$$= 4.35$$

~~so~~ since points is exactly A1, it remains in A1.

$\rightarrow$ ~~centroid A1 (6.5~~ Points (~~8.76~~ 1.88, 3.76)

Calculate distance for point $(1.88, 3.76)$ to both centroids

Distance to A1:

$= \sqrt{(1.88 - 6.57)^2 + (3.76 - 19.09)^2}$

$= 16.03$

Distance to A2:

$= \sqrt{(1.88 - 8.76)^2 + (3.76 - 22.85)^2}$

$= 20.29$

since 16.03 is less than 20.29, this point is closer to centroid A1.

4. Points (7.20, 20.35)

Distance to A1: $\sqrt{(7.20 - 6.57)^2 + (20.35 - 19.09)^2}$

$\sqrt{0.3969 + 1.58} = 1.40$

Distance to A2: $\sqrt{(7.20 - 8.76)^2 + (20.35 - 22.85)^2}$

$= \sqrt{2.43 + 6.25} = 2.94$

since 1.40 is less than 2.94

this point will assign to cluster A1

5. Points $(5.01, 31.61)$

Distance to A1 : $\sqrt{(5.01-6.57)^2+(31.61-19.09)^2}$

$= \sqrt{2.43+156.75} = 12.62$

Distance to A2 : $\sqrt{(5.01-8.76)^2+(31.61-22.85)^2}$

$= \sqrt{14.06+76.74} = 9.52$

since 9.52 is less than 12.62 so this point will be assigned to cluster A2

6) Points $(5.63, -17.22)$

Distance to A1 : $\sqrt{(5.63-6.57)^2+(-17.22-19.09)^2}$

$= \sqrt{0.88+1318.41} = 36.32$

Distance to A2 : $\sqrt{(5.63-8.76)^2+(-17.22-22.85)^2}$

$= \sqrt{9.796+1605.60} = 40.19$

since 36.32 is less than 40.19 so this point will be assigned to cluster A1.

7. Point $(4.38, -15.96)$

Distance to A1: $\sqrt{(4.38-6.57)^2+(-15.96-19.09)^2}$

$\sqrt{4.796+1228.50} = 35.11$

Distance to A2: $\sqrt{(4.38-8.76)^2+(-15.96-22.85)^2}$

$\sqrt{19.18+1506.21} = 39$

Assign to A1.

8. Point $(0.69, 2.50)$

Distance to A1: $\sqrt{(0.69-6.57)^2+(2.50-19.09)^2}$

$\sqrt{34.57+275.22} = 17.60$

Distance to A2: $\sqrt{(0.69-8.76)^2+(2.50-22.85)^2}$

$= \sqrt{65.12+414.12} = 21.89$

Assign to A1

cluster A1 contain points

(6.57, 19.09)
(8.76, 22.85)
(1.88, 3.76)
(7.20, 20.35)
(5.63, -17.22)
(4.38, -15.96)
(0.69, 2.50)

To find new centroid, we sum all x coordinates and all y coordinates of points in A1 and then divide by no of points in A1 which is 7

centroid A1 (x-cordinate)

$$= \frac{6.57 + 8.76 + 1.88 + 7.20 + 5.63 + 4.38 + 0.69}{7}$$

$$= \frac{35.11}{7} = 5.016$$

centroid A1 (Y-cordinates)

$$= \frac{19.09 + 22.85 + 3.76 + 20.35 + (-17.22) + (-15.96) + 2.50}{7}$$

$$= \frac{35.37}{7} = 5.053$$

so new centroid for cluster A1

centroid A1 = (5.016, 5.053)

cluster A2 has only 1 point so

centroid remains same

centroid A2 = (5.01, 31.61)

step 4

calculate SSE for cluster A1 and A2.

centroid A1 = (5.016, 5.053)

1) (6.57, 19.09)

$SSE = (6.57 - 5.016)^2 + (19.09 - 5.053)^2$

$= 2.415 + 197.038$

$= 199.45$

2) For point (8.76, 22.85)

$SSE = (8.76 - 5.016)^2 + (22.85 - 5.053)^2$

$= 14.017 + 316.65$

$= 330.66$

3) For point $(1.88, 3.76)$

$SSE = (1.88 - 5.016)^2 + (3.76 - 5.053)^2$

$= 9.835 + 1.673$

$= 11.508$

4) For point $(7.20, 20.35)$

$SSE = (7.20 - 5.016)^2 + (20.35 - 5.053)^2$

$= 4.769 + 234.042$

$= 238.811$

5) For point $(5.63, -17.22)$

$SSE = (5.63 - 5.016)^2 + (-17.22 - 5.053)^2$

$= 0.377 + 496.11$

$= 496.45$

6) For point $(4.38, -15.96)$

$SSE = (4.38 - 5.016)^2 + (-15.96 - 5.053)^2$

$= 0.405 + 441.55$

$= 441.95$

7] For point $(0.69, 2.50)$

$$SSE = (0.69 - 5.016)^2 + (2.50 - 5.053)^2$$

$$= 18.71 + 6.52$$

$$= 25.232$$

Now, sum all SSE to get total SSE for cluster A1

Total SSE (cluster A1) = 199.453 + 330.66 + 11.508 + 238.811 + 496.496 + 441.950 + 25.232

= 1743.113

So SSE for cluster A1 is approx 1743.11 and SSE for cluster A2 is 0

Continue Bisecting K-means process, we will split cluster with highest SSE i.e cluster A1 to form 2 subclusters A1a and A1b.

step 1 : Apply k-means to cluster A1 to create A1a and A1b

select two initial centroids randomly

using centroids A1a: (0.69, 2.50) and A1b: (8.76, 22.85)

For points (6.57, 19.09), we calculate distances to A1a and A1b

For point (6.57, 19.09)

Distance to A1a $= \sqrt{(6.57 - 0.69)^2 + (19.09 - 2.50)^2}$

$= \sqrt{34.6 + 275.3}$ $= \sqrt{0.48 + 275.2}$

$= \sqrt{309.92}$ = $= 16.60$

Distance to A1b $= \sqrt{(6.57 - 8.76)^2 + (19.09 - 22.85)^2}$

$= 4.8 + 14.14 = 4.3$

~~Fo~~ so assign to $\boxed{A1b}$

For point (8.76, 22.85)

as distance to A1b is zero to assign to A1b

For point (1.88, 3.76)

Distance to A1a : $\sqrt{(1.88-0.69)^2 + (3.76-2.50)^2}$

$= \sqrt{1.42 + 1.58} = 1.73$

Distance to A1b : $\sqrt{(1.88-8.76)^2 + (3.76-22.85)^2}$

$\sqrt{47.33 + 364.42}$
$= 20.29$

so assign to A1a

For point (7.20, 20.35)

Distance to A1a : $\sqrt{(7.20-0.69)^2 + (20.35-2.50)^2}$
$= \sqrt{42.4 + 318.62} = 19$

Distance to A1b : $\sqrt{(7.20-8.76)^2 + (20.35-22.85)^2}$

$\sqrt{2.43 + 6.25} = 2.95$

so assign to $\boxed{A1b}$

For point (5.01, 31.61)

Distance to A1a : $\sqrt{(5.01-0.69)^2 + (31.61-2.50)^2}$

$= \sqrt{18.66 + 847.4} = 29.43$

Distance to A1b: $\sqrt{(5.01-8.76)^2 + (31.61-22.85)^2}$

$= \sqrt{14 + 76.74} = 9.52$

so assign to $\boxed{A1b}$

For point $(5.63, -17.22)$

Distance to A1a: $\sqrt{(5.63-0.69)^2+(-17.22-2.50)^2}$

$\sqrt{24.40+388.88} = 20.32$

Distance to A1b: $\sqrt{(5.63-8.76)^2+(-17.22-22.85)^2}$

$\sqrt{9.796+1582.75} = 39.9$

so $\boxed{A1a}$

For point $(4.38, -15.96)$

Distance to A1a: $\sqrt{(4.38-0.69)^2+(-15.96-2.50)^2}$

$= \sqrt{13.61+340.77} = 18.8$

Distance to A1b: $\sqrt{(4.38-8.76)^2+(-15.96-22.85)^2}$

$= \sqrt{19.18+1506.21} = 39.05$

so A1a

For point $(0.69, 2.50)$

distance to A1a is 0 so directly assign to A1a

A1a: $(1.88, 3.76)$, $(5.63, -17.22)$, $(4.38, -15.96)$, $(0.69, 2.50)$

A1b: $(6.57, 19.09)$, $(8.76, 22.85)$, $(7.20, 20.35)$, $(5.01, 31.61)$

Now calculate SSE for both sub clusters

SSE for cluster A1a:

1. Point $(1.88, 3.76)$:

$(1.88 - 0.69)^2 + (3.76 - 2.50)^2 = 3.604$

2. Point $(5.63, -17.22)$

$(5.63 - 0.69)^2 + (-17.22 - 2.50)^2 = 413.602$

3. Point $(4.38, -15.96)$

$(4.38 - 0.69)^2 + (-15.96 - 2.50)^2 = 354.38$

original centroid does not contribute to SSE

Adding all $SSE_{A1a} = 3.604 + 413.602 + 354.39$
$= 771.59$

SSE for cluster A1b:

1. Point $(6.57, 19.09)$

$(6.57 - 8.76)^2 + (19.09 - 22.85)^2 = 18.93$

2. (Point $(8.76, 22.85)$

$(8.76 - 8.76)^2 + (22.85 - 22.85)^2 = 0$

3. Point $(7.20, 20.35)$:

$(7.20 - 8.76)^2 + (20.35 - 22.85)^2 = 8.68$

4. Point $(5.01, 31.61)$:

$(5.01 - 8.76)^2 + (31.61 - 22.85)^2 = 90.8001$

Adding them up

$SSE_{A1b} = 18.9337 + 0 + 8.68 + 90.80$

$= 118.4174$

Total SSE for trial 1 =

= SSE for A1a + SSE for A1b + SSE for A2

= $771.5937 + 118.4174 + 0$

$= 890.0111$

As Validation is $K = 3$

we have three clusters A1a, A1b and A2 at end of trial 1.

Trial 2

Using points (1.88, 3.76) and (5.01, 31.61) as initial centroid for Trial 2.

Calculate distance from each datapoint to these centroids and assign point to nearest centroid using centroids A2 (1.88, 3.76), A3 (5.01, 31.61)

1) Point (1.88, 3.76)

Distance to A2 = 0
so assign to $\boxed{A2}$

2) Point (5.01, 31.61)
Distance to A3 = 0
so assign to $\boxed{A3}$

3) Point (5.63, -17.22)

Distance to A2: $\sqrt{(5.63-1.88)^2+(-17.22-3.76)^2}$
$= \sqrt{454.2} = 21.31$

Distance to A3: $\sqrt{(5.63-5.01)^2+(-17.22-31.61)^2}$

$= \sqrt{2384.6} = 48.83$

so assign to $\boxed{A2}$

4. Point $(4.38, -15.96)$

Distance to A2: $\sqrt{(4.38-1.88)^2+(-15.96-3.76)^2}$

$= \cancel{\sqrt{2263.32}} = \sqrt{394.96} = 19.87$

Distance to A3: $\sqrt{(4.38-5.01)^2+(-15.96-31.61)^2}$

$= \sqrt{2263.32} = 47.57$

so assign to $\boxed{A2}$

5. Point $(0.69, 2.50)$

Distance to A2: $\sqrt{(0.69-1.88)^2+(2.50-3.76)^2}$

$= \sqrt{3.0037} = 1.73$

Distance to A3: $\sqrt{(0.69-5.01)^2+(2.50-31.61)^2}$

$= \sqrt{866.05} = 29.43$ so $\boxed{A2}$

6. Point $(6.57, 19.09)$

Distance to A2: $\sqrt{(6.57-1.88)^2+(19.09-3.76)^2}$

$= \sqrt{256.9} = 16.03$

Distance to A3: $\sqrt{(6.57-5.01)^2+(19.09-31.61)^2}$

$= \sqrt{159.184} = 12.62$

So assign to $\boxed{A3}$

7. Point $(8.76, 22.85)$

Distance to A2: $\sqrt{(8.76-1.88)^2 + (22.85-3.76)^2}$

$= \sqrt{411.76} = 20.29$

Distance to A3: $\sqrt{(8.76-5.01)^2 + (22.85-31.61)^2}$

$= \sqrt{90.80} = 9.53$ assign to $\boxed{A3}$

8. Point $(7.20, 20.35)$

Distance to A2: $\sqrt{(7.20-1.88)^2 + (20.35-3.76)^2}$

$= \sqrt{303.7} = 17.43$

Distance to A3: $\sqrt{(7.20-5.01)^2 + (20.35-31.61)^2}$

$= \sqrt{131.58} = 11.47$

so assign to A3

A2: $(1.88, 3.76)$, $(5.63, -17.22)$, $(4.38, -15.96)$, $(0.69, 2.50)$

A3: $(5.01, 31.61)$, $(6.57, 19.09)$, $(8.76, 22.85)$, $(7.20, 20.35)$

Step

calculate SSE for A2 and A3.

SSE for A2

Point $(4.38, -15.96)$

SSE $= 19.88^2$

~~SSE for~~

Point $(5.63, -17.22)$

SSE $= 21.31^2$

point $(0.69, 2.50)$

SSE $= 1.73^2$

point $(1.88, 3.76)$

distance $= 0$

Total SSE for A2

$= 21.31^2 + 19.88^2 + 1.73^2$

$= 454.056 + 395.21 + 2.99$

$\boxed{\text{SSE for A2} = 852.26}$

SSE for A3

Point (6.57, 19.09)

SSE = 12.62^2

point (8.76, 22.85)

SSE = 9.53^2

point (7.20, 20.35)

SSE = 11.47^2

point (5.01, 31.61)

distance = 0

Total SSE for A3

$= (12.62)^2 + (9.53)^2 + (11.47)^2$

$= 159.26 + 90.76 + 131.58$

$\boxed{\text{SSE for A3} = 381.6062}$

Since A2 has largest SSE (852.26) so we will bisect A2.

New centroid for A2

New centroid A2a : (1.88, 3.76)
New centroid A2b : (5.63, -17.22)

Now calculate distance of each point A2 to A2a and A2b

1) Point (4.38, -15.96)

Distance to A2a $= \sqrt{(4.38-1.88)^2 + (-15.96-3.76)^2}$
$= \sqrt{395.448} = 19.89$

Distance to A2b $= \sqrt{(4.38-5.63)^2 + (-15.96+17.22)^2}$
$= \sqrt{3.1501} = 1.77$

So assign to $\boxed{A2b}$

2) Point (0.69, 2.50)

Distance to A2a $= \sqrt{(0.69-1.88)^2 + (2.50-3.76)^2}$
$= \sqrt{3.0037} = 1.73$

Distance to A2b $= \sqrt{(0.69-5.63)^2 + (2.50+17.22)^2}$
$= \sqrt{413.36} = 20.33$

assign to $\boxed{A2a}$

Point (5.63, −17.22)

distance = 0 so assign to A2b

point (1.88, 3.76)

assign to A2a

→ Now calculate SSE for A2a

$SSE_{A2a} = (1.73)^2$

$= 2.9929$

SSE for A2b

$SSE_{A2b} = (1.77)^2$

$= 3.1329$

Total SSE for cluster A2 after split would be sum of SSE_{A2a} and SSE_{A2b} and SSE_{A2}

Total SSE for trial 2 $= SSE_{A2a} + SSE_{A2b} + SSE_{A2}$

$= 2.9929 + 3.1329 + 852.26$

$= 6.1258 + 852.26$

$= 858.3858$

We have three clusters i.e k=3
(A2, A2a, A2b) at end of trial 2

Trial 1 SSE = 890.0111

Trial 2 SSE = 858.3858

Since Trial 2 has lower SSE so it is better clustering solution of two trials.

Question: 3

3.1

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans
import numpy as np
import matplotlib.pyplot as plt
import warnings
from sklearn.cluster import KMeans
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt
from sklearn.metrics import pairwise_distances

# Filter out FutureWarnings from scikit-learn
warnings.filterwarnings("ignore", category=FutureWarning, module="sklearn")
```

```
# Load the dataset
file_path = 'C:/Users/mehre/Downloads/dow+jones+index/dow_jones_index.data'
```

```python
# Load the data into a DataFrame
df = pd.read_csv(file_path)
# Display basic information about the dataset
df.head()
```

✓ 0.0s Python

	quarter	stock	date	open	high	low	close	volume	percent_change_price	percent_change_volume_over_last_wk	previous_weeks_volume	next_weeks_open	next_weeks_close	percent
0	1	AA	1/7/2011	$15.82	$16.72	$15.78	$16.42	239655616	3.79267	NaN	NaN	$16.71	$15.97	
1	1	AA	1/14/2011	$16.71	$16.71	$15.64	$15.97	242963398	-4.42849	1.380223	239655616.0	$16.19	$15.79	
2	1	AA	1/21/2011	$16.19	$16.38	$15.60	$15.79	138428495	-2.47066	-43.024959	242963398.0	$15.87	$16.13	
3	1	AA	1/28/2011	$15.87	$16.63	$15.82	$16.13	151379173	1.63831	9.355500	138428495.0	$16.18	$17.14	
4	1	AA	2/4/2011	$16.18	$17.39	$16.18	$17.14	154387761	5.93325	1.987452	151379173.0	$17.33	$17.37	

```python
df.info()
```

[90] ✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 750 entries, 0 to 749
Data columns (total 16 columns):
 #   Column                              Non-Null Count  Dtype  
---  ------                              --------------  -----  
 0   quarter                             750 non-null    int64  
 1   stock                               750 non-null    object 
 2   date                                750 non-null    object 
 3   open                                750 non-null    object 
 4   high                                750 non-null    object 
 5   low                                 750 non-null    object 
 6   close                               750 non-null    object 
 7   volume                              750 non-null    int64  
 8   percent_change_price                750 non-null    float64
 9   percent_change_volume_over_last_wk  720 non-null    float64
 10  previous_weeks_volume               720 non-null    float64
 11  next_weeks_open                     750 non-null    object 
 12  next_weeks_close                    750 non-null    object 
 13  percent_change_next_weeks_price     750 non-null    float64
 14  days_to_next_dividend               750 non-null    int64  
 15  percent_return_next_dividend        750 non-null    float64
dtypes: float64(5), int64(3), object(8)
memory usage: 93.9+ KB
```

3.2

```python
# Remove the categorical attributes
df.drop(['quarter', 'stock', 'date'], axis=1, inplace=True)
```

```
[92]  df.columns
      ✓ 0.0s

...   Index(['open', 'high', 'low', 'close', 'volume', 'percent_change_price',
             'percent_change_volume_over_last_wk', 'previous_weeks_volume',
             'next_weeks_open', 'next_weeks_close',
             'percent_change_next_weeks_price', 'days_to_next_dividend',
             'percent_return_next_dividend'],
            dtype='object')
```

```
# Check for missing values
missing_values = df.isnull().sum()
```

```
# Check for data types
data_types = df.dtypes
```

```
[95]  missing_values
      ✓ 0.0s

...   open                                   0
      high                                   0
      low                                    0
      close                                  0
      volume                                 0
      percent_change_price                   0
      percent_change_volume_over_last_wk    30
      previous_weeks_volume                 30
      next_weeks_open                        0
      next_weeks_close                       0
      percent_change_next_weeks_price        0
      days_to_next_dividend                  0
      percent_return_next_dividend           0
      dtype: int64
```

```
data_types
[96] ✓ 0.0s

...  open                                   object
     high                                   object
     low                                    object
     close                                  object
     volume                                  int64
     percent_change_price                  float64
     percent_change_volume_over_last_wk    float64
     previous_weeks_volume                 float64
     next_weeks_open                        object
     next_weeks_close                       object
     percent_change_next_weeks_price       float64
     days_to_next_dividend                   int64
     percent_return_next_dividend          float64
     dtype: object
```

```
# Data type conversion: Remove the dollar sign and convert the string to float
price_columns = ['open', 'high', 'low', 'close', 'next_weeks_open', 'next_weeks_close']
df[price_columns] = df[price_columns].replace({'\$': '', ',': ''}, regex=True).astype(float)
```

```
# Impute missing values with the mean
for column in ['percent_change_volume_over_last_wk', 'previous_weeks_volume']:
    mean_value = df[column].mean()
    df[column].fillna(mean_value, inplace=True)
```

```
df.isnull().sum()
0] ✓ 0.0s

open                                  0
high                                  0
low                                   0
close                                 0
volume                                0
percent_change_price                  0
percent_change_volume_over_last_wk    0
previous_weeks_volume                 0
next_weeks_open                       0
next_weeks_close                      0
percent_change_next_weeks_price       0
days_to_next_dividend                 0
percent_return_next_dividend          0
dtype: int64
```

```
# Function to calculate IQR
def calculate_iqr(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
```

```
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return lower_bound, upper_bound

# Handling outliers for each column
for col in df.select_dtypes(include=['float64', 'int64']).columns:
    lower_bound, upper_bound = calculate_iqr(df, col)
    df[col] = np.where(df[col] < lower_bound, lower_bound, df[col])
    df[col] = np.where(df[col] > upper_bound, upper_bound, df[col])

# Display basic information about the dataset
df.head()
```

...

	open	high	low	close	volume	percent_change_price	percent_change_volume_over_last_wk	previous_weeks_volume	next_weeks_open	next_weeks_close	percent_change_next_weeks_price	day:
0	15.82	16.72	15.78	16.42	239655616.0	3.79267	5.593627	1.173876e+08	16.71	15.97	-4.428490	
1	16.71	16.71	15.64	15.97	242963398.0	-4.42849	1.380223	2.396556e+08	16.19	15.79	-2.470660	
2	16.19	16.38	15.60	15.79	138428495.0	-2.47066	-43.024959	2.429634e+08	15.87	16.13	1.638310	
3	15.87	16.63	15.82	16.13	151379173.0	1.63831	9.355500	1.384285e+08	16.18	17.14	5.933250	
4	16.18	17.39	16.18	17.14	154387761.0	5.93325	1.987452	1.513792e+08	17.33	17.37	0.230814	

```
# Scale the data
scaler = StandardScaler()
df = scaler.fit_transform(df)
```

Preprocessing steps include:

Missing Values: There are missing values in two columns: percent_change_volume_over_last_wk and previous_weeks_volume. Since they are numerical, we can consider imputing these missing values with an appropriate statistic, such as the mean or median, which can help to retain valuable data without introducing too much bias.

Data Type Conversion: The columns 'open', 'high', 'low', 'close', 'next_weeks_open', and 'next_weeks_close' are of object type, which indicates they are strings. This is most likely due to the presence of the dollar sign. These need to be converted to a float data type after removing any non-numeric characters.

Handling outliers for each column because algorithms are sensitive to them.

Scaling of the data is a critical step before applying K-means and hierarchical clustering algorithms because both methods use distance metrics to determine the similarity between data points.

3.3

```
# Function to calculate the sum of squared errors for each cluster
def calculate_sse_per_cluster(X, labels, centroids):
```

```
    sse_per_cluster = {i: 0 for i in range(centroids.shape[0])}
    for i, label in enumerate(labels):
        centroid = centroids[label]
        sse_per_cluster[label] += ((X[i] - centroid) ** 2).sum()
    return sse_per_cluster
```

```
# Define the range of K
k_values = range(2, 8)

# Initialize a dictionary to store the K-means results
kmeans_results = {}
sse_list = []
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(df)
    sse = kmeans.inertia_
    sse_list.append(sse)
    kmeans_results[k] = {
        'kmeans': kmeans,
        'sse': sse,
        'cluster_centers': kmeans.cluster_centers_,
        'labels': kmeans.labels_
    }

# Print the total SSE and detailed information for each K
for k in k_values:
    kmeans = kmeans_results[k]['kmeans']
    labels = kmeans.labels_
    cluster_centers = kmeans.cluster_centers_
    total_sse = kmeans_results[k]['sse']
    sse_per_cluster = calculate_sse_per_cluster(df, labels, cluster_centers)



    print(f"\nFor K={k}:")
    print(f"Total Sum of Squared Errors (SSE): {total_sse}")
    print("\nSum of Squared Errors for Each Cluster:")
    for cluster_id, sse in sse_per_cluster.items():
        print(f"  Cluster {cluster_id}: SSE = {sse}")
    print("\nCluster Means:")
    for cluster_id, centroid in enumerate(cluster_centers):
        print(f"  Cluster {cluster_id}: Mean = {centroid}")

    print("\nCluster IDs and Instance IDs:")
    for cluster_id in range(k):
        instance_ids = [i for i, label in enumerate(labels) if label == cluster_id]
        print(f"  Cluster {cluster_id}: Instance IDs = {instance_ids}")
```

For K=2:

Total Sum of Squared Errors (SSE): 6157.832548661327

Sum of Squared Errors for Each Cluster:

Cluster 0: SSE = 2578.760903740677

Cluster 1: SSE = 3579.071644920648

Cluster Means:

Cluster 0: Mean = [0.85005274 0.8485825 0.85210781 0.85219915 -0.65678376 0.0713362

0.0144555 -0.65128573 0.85191013 0.85290052 0.08094268 -0.1208504

-0.11931108]

Cluster 1: Mean = [-0.77632368 -0.77498096 -0.7782005 -0.77828392 0.59981782 -0.06514888

-0.01320171 0.59479666 -0.77801997 -0.77892445 -0.07392214 0.11036848

0.10896267]

Cluster IDs and Instance IDs:

Cluster 0: Instance IDs = [24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 336, 337, 338, 339, 340, 341, 342, 343, 344, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749]

Cluster 1: Instance IDs = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 84, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 345, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723]

For K=3:

Total Sum of Squared Errors (SSE): 4845.028652082341

Sum of Squared Errors for Each Cluster:

Cluster 0: SSE = 2244.1647083467556

Cluster 1: SSE = 1324.2865974863987

Cluster 2: SSE = 1276.577346249184

Cluster Means:

Cluster 0: Mean = [-1.92805711e-01 -1.94816903e-01 -1.90647080e-01 -1.91507440e-01

-4.04092150e-01 3.46796246e-02 3.93334397e-04 -3.90372463e-01

-1.92661593e-01 -1.92524310e-01 2.72187086e-02 8.28446107e-02

4.11752631e-01]

Cluster 1: Mean = [-1.08479858 -1.08201857 -1.08773186 -1.08881273 1.53340358 -0.15749615

-0.00576432 1.50031879 -1.08726401 -1.08841989 -0.16121165 -0.05458911

-0.44351146]

Cluster 2: Mean = [1.36308858 1.36420313 1.36184535 1.36443255 -0.68232128 0.08263339

0.00463537 -0.67679726 1.3651155 1.36593757 0.09980959 -0.1016188

-0.34498413]

Cluster IDs and Instance IDs:

Cluster 0: Instance IDs = [12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 141, 142, 143, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 512, 513, 514, 515, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736]

Cluster 1: Instance IDs = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 140, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 190, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 289, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 510, 511, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 565, 566, 567, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 672, 673]

Cluster 2: Instance IDs = [25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749]

For K=4:

Total Sum of Squared Errors (SSE): 4372.4488848856645

Sum of Squared Errors for Each Cluster:

Cluster 0: SSE = 1231.062878686854

Cluster 1: SSE = 877.6815888356894

Cluster 2: SSE = 1280.1844608384515

Cluster 3: SSE = 983.5199565246684

Cluster Means:

Cluster 0: Mean = [0.18126952 0.17850834 0.18387493 0.18300853 -0.51969492 0.02919842

-0.00829122 -0.50729313 0.18159711 0.18120049 0.01663321 -0.12826205

-0.1088885]

Cluster 1: Mean = [-0.59737436 -0.59883828 -0.59643692 -0.59686892 -0.24758665 0.043282

0.01140946 -0.24338971 -0.59692428 -0.59488704 0.06678298 0.39116471

1.22605883]

Cluster 2: Mean = [-1.08054365 -1.07771117 -1.08351905 -1.08475045 1.53750455 -0.17028264

-0.01792567 1.51022659 -1.08321717 -1.08439296 -0.16535893 -0.05228055

-0.50547453]

Cluster 3: Mean = [1.54057546 1.54329925 1.53879101 1.54209256 -0.68635882 0.10782686

0.02361506 -0.67929353 1.54271006 1.54275783 0.10012615 -0.10637079

-0.42342031]

Cluster IDs and Instance IDs:

Cluster 0: Instance IDs = [12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 126, 127, 128, 132, 133, 134, 135, 136, 137, 138, 139, 141, 142, 143, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 182, 184, 185, 186, 187, 188, 189, 191, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 393, 394, 395, 396, 397, 398, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 503, 504, 505, 506, 507, 508, 509, 512, 513, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 748, 749]

Cluster 1: Instance IDs = [96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 120, 121, 122, 123, 124, 125, 129, 130, 131, 180, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 270, 271, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 514, 515, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723]

Cluster 2: Instance IDs = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 140, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 181, 183, 190, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 272, 273, 274, 275, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 510, 511, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 565, 566, 567, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658]

Cluster 3: Instance IDs = [48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 351, 352, 353, 354, 355, 356, 357, 358, 359, 390, 391, 392, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747]

For K=5:

Total Sum of Squared Errors (SSE): 4052.776331364635

Sum of Squared Errors for Each Cluster:

Cluster 0: SSE = 244.58735653806028

Cluster 1: SSE = 673.7673650818247

Cluster 2: SSE = 1243.739472435299

Cluster 3: SSE = 762.3074338619334

Cluster 4: SSE = 1128.3747034475173

Cluster Means:

Cluster 0: Mean = [2.35138974 2.3640022   2.35027084 2.35478837 -0.75877639 0.35103543

-0.16930264 -0.73826831 2.3601314   2.33040935 -0.12693108 -0.08447952

-0.72231215]

Cluster 1: Mean = [1.07429929 1.07205405 1.07301972 1.07504287 -0.6599805   0.00420422

0.05546141 -0.65883494 1.0743641   1.0841114   0.16606498 -0.10662704

-0.23472595]

Cluster 2: Mean = [-1.09351425 -1.09046062 -1.09564588 -1.09692663 1.5440012 -0.15726418

-0.0385859   1.52913934 -1.09538288 -1.09673212 -0.17394334 -0.0460934

-0.50269584]

Cluster 3: Mean = [-0.6057719 -0.60745528 -0.60552646 -0.60498481 -0.23195487 0.05726157

0.01649388 -0.21449321 -0.6047292 -0.60331755 0.04947926 0.47275673

1.40882523]

Cluster 4: Mean = [0.02377251 0.02149175 0.02627703 0.02477207 -0.46013583 0.01694619

0.01585822 -0.46340839 0.02289606 0.02242867 0.01951062 -0.14654009

-0.12667304]

Cluster IDs and Instance IDs:

Cluster 0: Instance IDs = [53, 54, 56, 58, 59, 80, 83, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 412, 413, 414, 415, 416, 417, 418, 420, 438, 439, 441, 442, 446, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528]

Cluster 1: Instance IDs = [25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 48, 49, 50, 51, 52, 55, 57, 72, 73, 74, 75, 76, 77, 78, 79, 81, 82, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 419, 421, 422, 423, 424, 440, 443, 444, 445, 447, 448, 449, 450, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749]

Cluster 2: Instance IDs = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 140, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 181, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 272, 273, 274, 275, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 511, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 565, 566, 567, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658]

Cluster 3: Instance IDs = [96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 120, 121, 129, 130, 131, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 270, 271, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 490, 501, 502, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723]

Cluster 4: Instance IDs = [12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 122, 123, 124, 125, 126, 127, 128, 132, 133, 134, 135, 136, 137, 138, 139, 141, 142, 143, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 503, 504, 505, 506, 507, 508, 509, 510, 512, 513, 514, 515, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736]

For K=6:

Total Sum of Squared Errors (SSE): 3815.156870458883

Sum of Squared Errors for Each Cluster:

Cluster 0: SSE = 720.6161161415849

Cluster 1: SSE = 718.7447096022755

Cluster 2: SSE = 964.405793943503

Cluster 3: SSE = 665.1598320367381

Cluster 4: SSE = 95.31820158538272

Cluster 5: SSE = 650.9122171494

Cluster Means:

Cluster 0: Mean = [-0.48853067 -0.48464251 -0.49188784 -0.48904367 -0.11785902 0.02820361

-0.08268604 -0.0909589 -0.48973059 -0.4911519 -0.01739578 -0.11607024

-0.85867059]

Cluster 1: Mean = [-0.60748985 -0.60963209 -0.60770252 -0.60742074 -0.21236964 0.0320955

0.0397615 -0.20195423 -0.60707812 -0.60612095 0.03450124 0.48127334

1.48958893]

Cluster 2: Mean = [-1.10603539 -1.1034127 -1.10800082 -1.1101629 1.75347812 -0.19975875

0.00245191 1.70209696 -1.10882764 -1.11006735 -0.18742656 -0.04750621

-0.34658167]

Cluster 3: Mean = [0.31352435 0.30779044 0.31855369 0.31486241 -0.64196794 0.01731495

0.00226958 -0.6285301 0.31361447 0.31364155 0.03806676 -0.12376906

0.20361606]

Cluster 4: Mean = [2.73810405 2.75067321 2.74436813 2.72339635 -0.79209554 0.34085284

0.00963079 -0.78220982 2.72944603 2.7076529 0.06093276 -0.13944077

-0.82835588]

Cluster 5: Mean = [1.25772492 1.25860068 1.25506386 1.26379777 -0.64480671 0.07918906

0.02793389 -0.64031809 1.26308152 1.26905452 0.12982966 -0.09227448

-0.33751827]

Cluster IDs and Instance IDs:

Cluster 0: Instance IDs = [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 360, 361, 363, 364, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 576, 579, 580]

Cluster 1: Instance IDs = [96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 130, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 270, 271, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 501, 568, 569, 570, 571, 572, 573, 574, 575, 577, 578, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723]

Cluster 2: Instance IDs = [0, 1, 2, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 272, 273, 274, 275, 362, 365, 366, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658]

Cluster 3: Instance IDs = [24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 386, 387, 388, 395, 396, 397, 398, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736]

Cluster 4: Instance IDs = [144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 412, 416, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528]

Cluster 5: Instance IDs = [48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 389, 390, 391, 392, 393, 394, 413, 414, 415, 417, 418, 419, 420, 421, 422, 423, 424, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749]

For K=7:

Total Sum of Squared Errors (SSE): 3610.0119931516383

Sum of Squared Errors for Each Cluster:

Cluster 0: SSE = 128.35202043321272

Cluster 1: SSE = 117.9216613980391

Cluster 2: SSE = 433.87449868779055

Cluster 3: SSE = 629.6221661474829

Cluster 4: SSE = 924.7934791456536

Cluster 5: SSE = 710.2883353027224

Cluster 6: SSE = 665.1598320367381

Cluster Means:

Cluster 0: Mean = [-0.42454351 -0.4249262 -0.42619573 -0.42442864 -0.54624879 0.01587931

0.01073378 -0.53873029 -0.42509911 -0.42507129 0.02056956 2.58372112

0.90149089]

Cluster 1: Mean = [2.69215014 2.69948412 2.68829077 2.6672329 -0.78518335 0.19030474

0.02040205 -0.78010543 2.67355041 2.64394484 -0.04060072 -0.14362017

-0.83650679]

Cluster 2: Mean = [-6.79750449e-01 -6.82440903e-01 -6.79338393e-01 -6.79625022e-01

-7.81131242e-02 5.02425817e-02 6.01612207e-04 -7.25126867e-02

-6.78875323e-01 -6.78082311e-01 2.28510502e-02 -2.27567044e-01

1.61897950e+00]

Cluster 3: Mean = [1.24575364 1.24757375 1.24512   1.25430912 -0.64411462 0.10744323

0.02590624 -0.63868018 1.25343626 1.26148347 0.15249338 -0.09068966

-0.32856201]

Cluster 4: Mean = [-1.10555844 -1.10307799 -1.10769508 -1.11000988 1.76987651 -0.22175577

0.00745465 1.72393904 -1.10871682 -1.11007697 -0.19503484 -0.0218381

-0.38746451]

Cluster 5: Mean = [-0.4864315 -0.48201199 -0.48989057 -0.4867489 -0.10183236 0.03656084

-0.05159793 -0.07861974 -0.4874728 -0.48849297 -0.00273118 -0.09743981

-0.89235709]

Cluster 6: Mean = [0.31352435 0.30779044 0.31855369 0.31486241 -0.64196794 0.01731495

0.00226958 -0.6285301   0.31361447 0.31364155 0.03806676 -0.12376906

0.20361606]

Cluster IDs and Instance IDs:

Cluster 0: Instance IDs = [96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 129, 130, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 568, 569, 631]

Cluster 1: Instance IDs = [144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 412, 413, 416, 417, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528]

Cluster 2: Instance IDs = [192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 264, 266, 270, 271, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 632, 649, 650, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723]

Cluster 3: Instance IDs = [48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 389, 390, 391, 392, 393, 394, 414, 415, 418, 419, 420, 421, 422, 423, 424, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749]

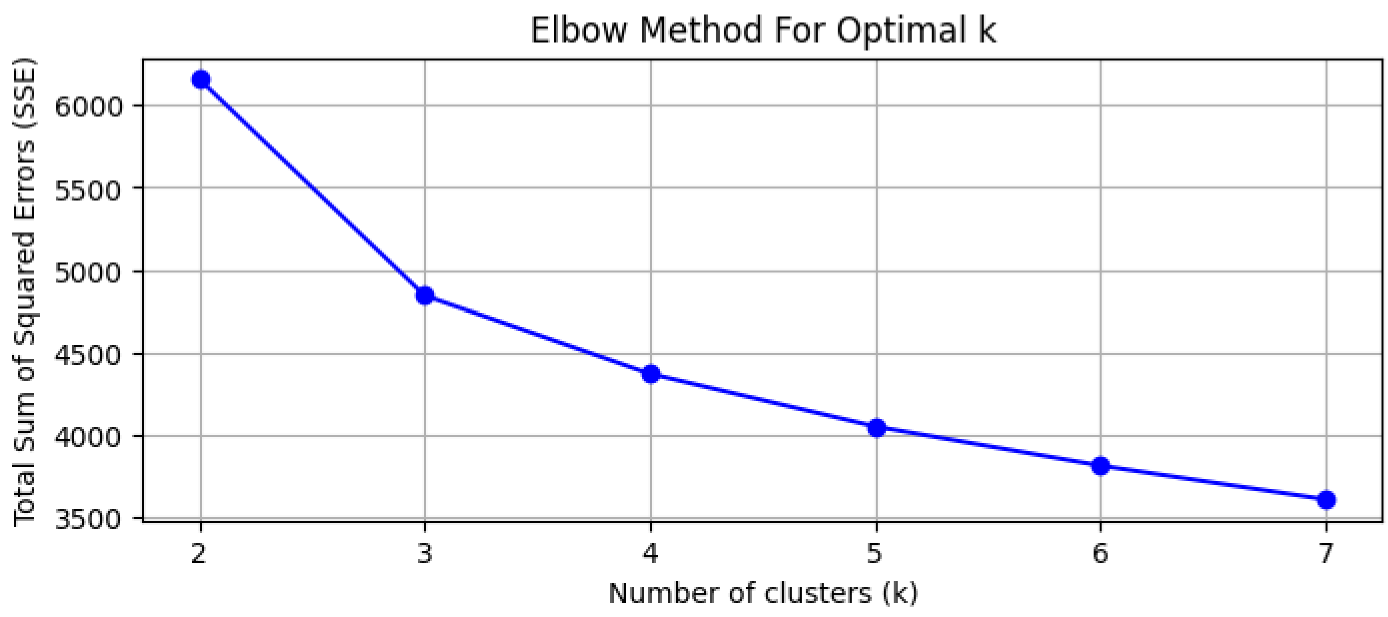
Cluster 4: Instance IDs = [0, 1, 2, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 265, 267, 268, 269, 272, 273, 274, 275, 362, 365, 366, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 651, 652, 653, 654, 655, 656, 657, 658]

Cluster 5: Instance IDs = [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 120, 121, 122, 123, 124, 125, 126, 127, 128, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 360, 361, 363, 364, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567]

Cluster 6: Instance IDs = [24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 386, 387, 388, 395, 396, 397, 398, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736]

```
# Plotting the elbow method
```

```python
plt.figure(figsize=(8, 3))
plt.plot(k_values, sse_list, 'bo-')
plt.title('Elbow Method For Optimal k')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Total Sum of Squared Errors (SSE)')
plt.xticks(list(k_values))
plt.grid(True)
plt.show()
```



From the plot, it can be observed that:

When k increases from 2 to 3, there is a noticeable decrease in SSE.

From k=3 to k=4, the SSE continues to decrease but at a slower rate.

After k=4, the decrease in SSE is much more gradual and tends to flatten out as k increases further.

The "elbow" of the plot, where the SSE begins to decrease at a slower rate, appears to be at k=4. This suggests that increasing the number of clusters beyond 4 results in diminishing returns in terms of the decrease in SSE. Therefore, in this scenario, k=4 would likely be an adequate choice for the number of clusters because it represents a point at which adding more clusters does not provide substantial improvement in the SSE.

This point resembles an "elbow" in the arm, hence the name.

The choice of k=4 is justified by the elbow method because it is at this point that the total sum of squared errors starts to decrease at a slower rate, suggesting that the clusters are relatively well-defined and further subdivision into more clusters is less beneficial. Choosing a k value larger than 4 would not significantly improve the clustering performance according to this method and may lead to overfitting, where clusters may be fit to noise rather than to the underlying data structure.

Beyond k=4, even though SSE continues to decrease, it does so at a rate that may not justify the added complexity of more clusters. This is why k=4 is suggested as the optimal number of clusters for this dataset according to the elbow method.

3.4

```
from sklearn.cluster import AgglomerativeClustering
K = 4

# Agglomerative clustering using Complete Link (MAX)
complete_link = AgglomerativeClustering(n_clusters=K, linkage='complete')
complete_link_labels = complete_link.fit_predict(df)
```

```
from scipy.cluster.hierarchy import linkage, fcluster
# Calculate linkage matrix using 'Centroid' linkage
custom_linkage_matrix = linkage(df, method='centroid')

# Perform clustering using 'fcluster'
centroid_link_labels = fcluster(custom_linkage_matrix, K, criterion='maxclust')
```

```
complete_link_labels
```
[54] ✓ 0.0s

```
array([3, 1, 3, 3, 3, 3, 3, 1, 3, 1, 3, 3, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 0, 2, 2, 0, 0, 0, 2, 0, 0, 2, 2, 3, 3, 1, 1, 3, 3, 3, 1,
       1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 3, 1, 3, 3, 1,
       3, 3, 1, 1, 1, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 3,
       1, 3, 3, 3, 3, 1, 3, 1, 1, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 1, 3, 3, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 3, 3, 1, 3, 3, 3, 3, 1, 3, 1, 3, 2, 2, 2, 2, 0, 0, 0, 2,
       2, 2, 2, 2, 2, 3, 2, 2, 2, 2, 3, 2, 2, 2, 1, 3, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 0, 2, 0, 2, 2, 0, 2,
       0, 0, 0, 2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 1, 3, 3, 3, 1, 3, 1, 1, 3,
       1, 3, 3, 1, 1, 3, 3, 1, 1, 3, 1, 3, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 2, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 3, 3, 1, 3, 3, 1, 3, 1, 3, 1, 1, 1, 3, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 0, 0, 0, 0, 0, 0, 2,
       0, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 3, 1, 3, 3, 1, 1, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 3, 3, 3, 1, 3, 3,
       1, 3, 3, 3, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2,
       2, 2, 2, 2, 1, 3, 3, 3, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 3, 1, 1, 1, 3, 3, 1, 3, 3, 3, 3, 1, 2, 2, 2, 2, 2, 2, 2, 2,
...
       2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 1, 3, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 2, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0], dtype=int64)
```
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
centroid_link_labels
```

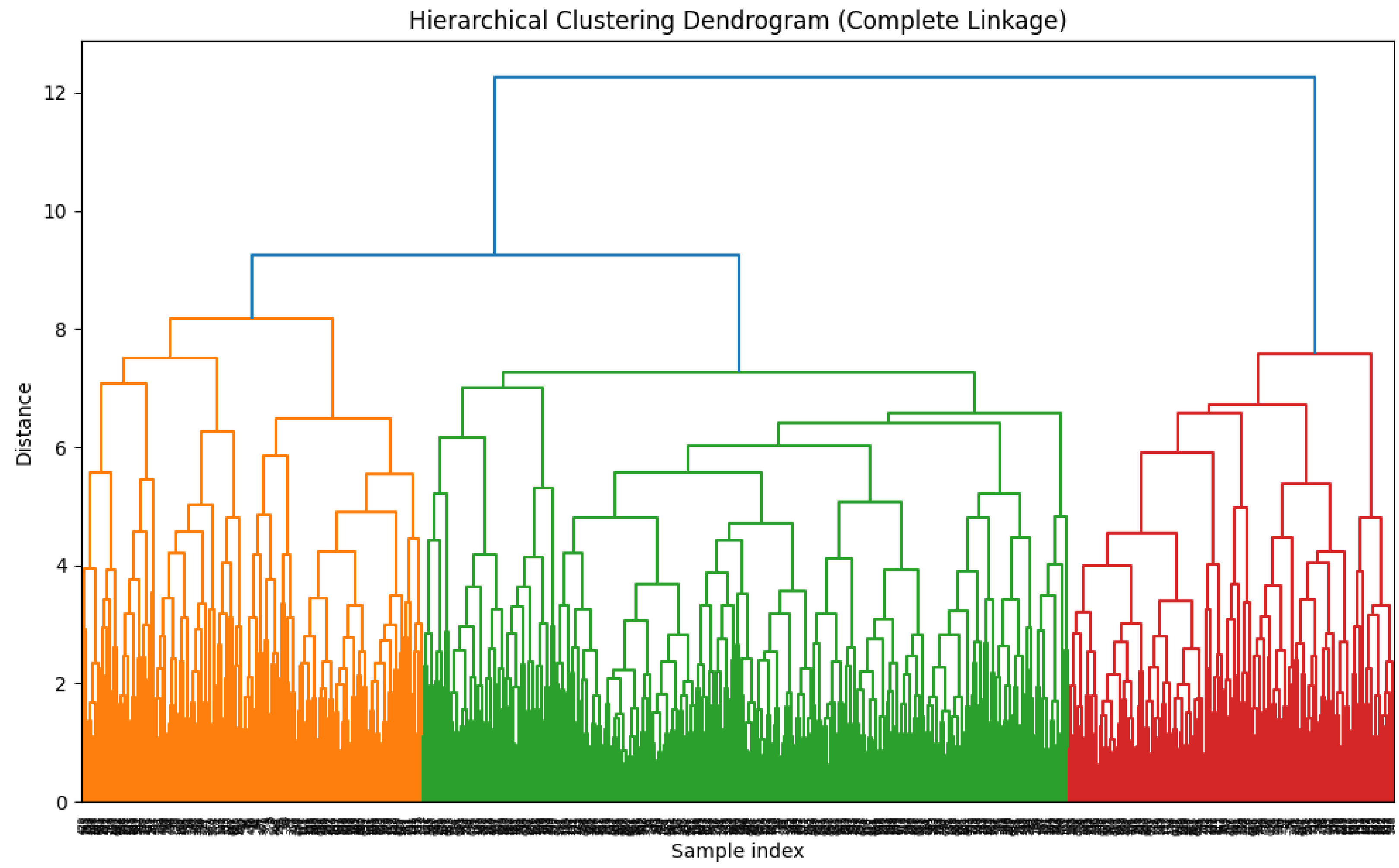
[55] ✓ 0.0s

```
array([2, 2, 2, 2, 2, 2, 2, 0, 2, 0, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0, 2, 2, 2, 2, 0, 2, 0, 0, 0, 2, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0,
...
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0], dtype=int64)
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```python
linked = linkage(df, method='complete')

# Plot the dendrogram
plt.figure(figsize=(12, 7))
dendrogram(linked,
           orientation='top',
           distance_sort='descending',
           show_leaf_counts=True)
plt.title('Hierarchical Clustering Dendrogram (Complete Linkage)')
plt.xlabel('Sample index')
plt.ylabel('Distance')
plt.show()
```




```
from sklearn.metrics import pairwise_distances

# We need to define a function to calculate the SSE for each cluster
def calculate_sse(cluster_labels, data):
    sse = 0
    cluster_sse = []
    for i in np.unique(cluster_labels):
        cluster_data = data[cluster_labels == i]
        if cluster_data.shape[0] > 1:  # More than one element in cluster
            # Pairwise distances within a cluster
            distances = pairwise_distances(cluster_data)
            # Sum of squared distances within a cluster
            cluster_sse.append(np.sum(distances ** 2) / 2)  # Divide by 2 to correct for double counting
        else:  # Single element in cluster
            cluster_sse.append(0)
        sse += cluster_sse[-1]
    return sse, cluster_sse
```

```
# Calculate SSE using the labels from the complete linkage clustering
total_sse, cluster_sse = calculate_sse(complete_link_labels, df)
```

```python
# Print out the results
print(f"Total Sum of Squared Errors (SSE): {total_sse}")
for cluster_id, sse in enumerate(cluster_sse):
    instance_ids = [i for i, label in enumerate(complete_link_labels) if label == cluster_id]
    print(f"Cluster {cluster_id}:")
    print(f"  SSE: {sse}")
    print(f"  Number of Instances: {len(instance_ids)}")
    # Printing all instance IDs can be too verbose, so I will show the first 10
    print(f"  Instance IDs: {instance_ids[:10]} ...")
```

Total Sum of Squared Errors (SSE): 1206891.8654417756

Cluster 0:

SSE: 227441.5180998647

Number of Instances: 186

Instance IDs: [25, 28, 29, 30, 32, 33, 48, 49, 50, 51] ...

Cluster 1:

SSE: 67324.92834848288

Number of Instances: 92

Instance IDs: [1, 7, 9, 15, 38, 39, 43, 44, 45, 46] ...

Cluster 2:

SSE: 849422.3158779233

Number of Instances: 369

Instance IDs: [12, 13, 14, 16, 17, 18, 19, 20, 21, 22] ...

Cluster 3:

SSE: 62703.10311550487

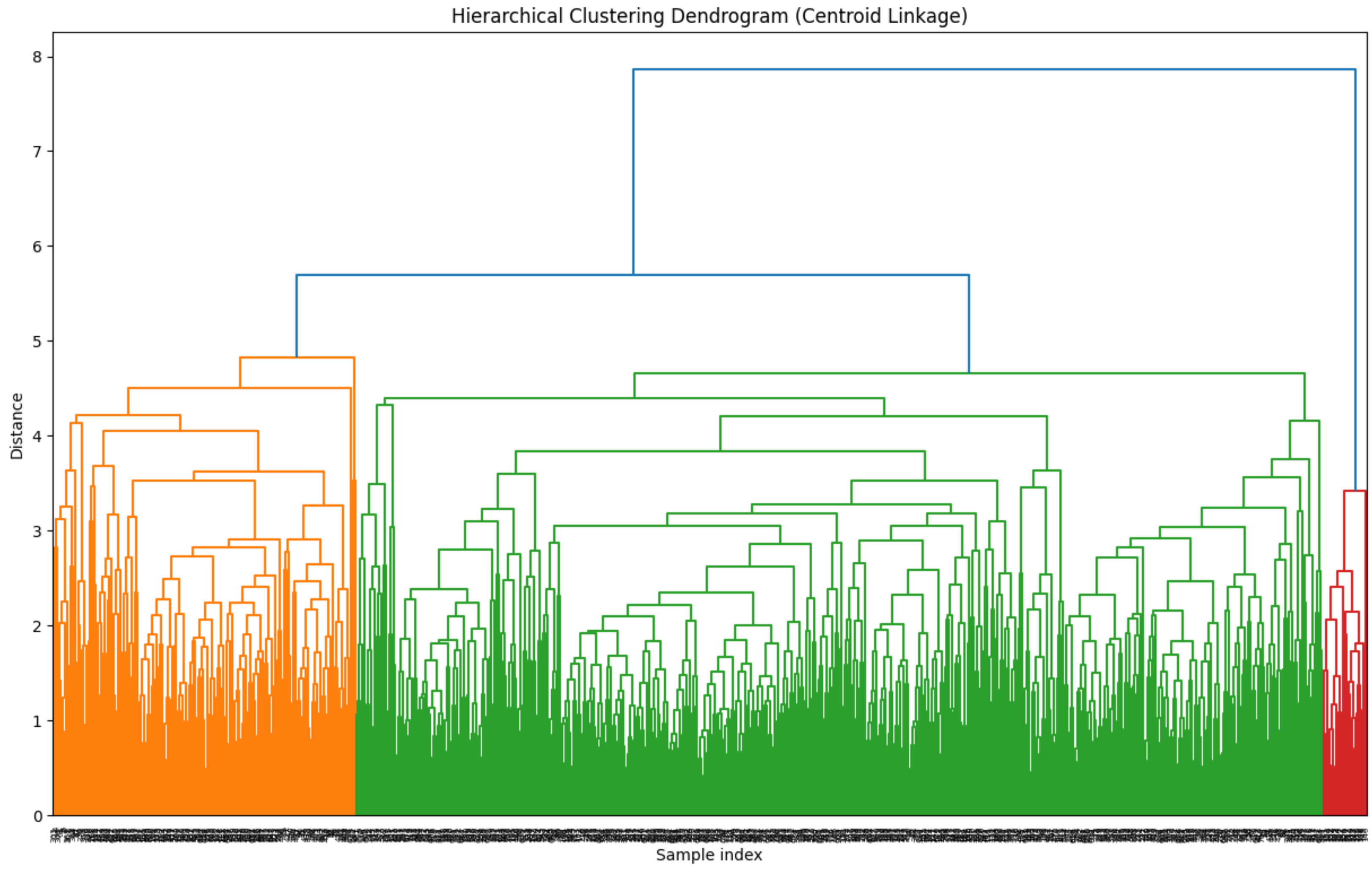
Number of Instances: 103

Instance IDs: [0, 2, 3, 4, 5, 6, 8, 10, 11, 36] ...

```python
# Perform 'Centroid' linkage hierarchical clustering and plot the dendrogram


# Plot the dendrogram for the 'Centroid' linkage
plt.figure(figsize=(15, 9))
dendrogram(custom_linkage_matrix,
           orientation='top',
           distance_sort='descending',
           show_leaf_counts=True)
plt.title('Hierarchical Clustering Dendrogram (Centroid Linkage)')
plt.xlabel('Sample index')
plt.ylabel('Distance')
plt.show()
```

✓ 8.7s




```python
# Calculate SSE for the 'Centroid' linkage clustering
total_sse_average, cluster_sse_average = calculate_sse(centroid_link_labels, df)

# Print out the results for the 'Centroid' linkage clustering
print(f"Total Sum of Squared Errors (SSE) for 'Centroid' Linkage: {total_sse_average}")
for cluster_id, sse in enumerate(cluster_sse_average):
    instance_ids = [i for i, label in enumerate(centroid_link_labels) if label == cluster_id]
    print(f"Cluster {cluster_id} ('Centroid' Linkage):")
    print(f"  SSE: {sse}")
    print(f"  Number of Instances: {len(instance_ids)}")
    print(f"  Instance IDs: {instance_ids[:10]} ...")
```


```python
print(f"  Instance IDs: {instance_ids[:10]} ...")
```

```
[112]  ✓ 0.0s

...  Total Sum of Squared Errors (SSE) for 'Average' Linkage: 2802159.5574843236
     Cluster 0 ('Average' Linkage):
       SSE: 2603583.476363404
       Number of Instances: 552
       Instance IDs: [7, 9, 12, 13, 14, 15, 16, 17, 18, 19] ...
     Cluster 1 ('Average' Linkage):
       SSE: 12.462079540517905
       Number of Instances: 2
       Instance IDs: [267, 532] ...
     Cluster 2 ('Average' Linkage):
       SSE: 196817.80925878888
       Number of Instances: 171
       Instance IDs: [0, 1, 2, 3, 4, 5, 6, 8, 10, 11] ...
     Cluster 3 ('Average' Linkage):
       SSE: 1745.8097825900563
       Number of Instances: 25
       Instance IDs: [144, 145, 146, 147, 148, 149, 150, 151, 152, 153] ...
```

Complete Link (MAX) Clustering: This method has a lower total SSE, which indicates tighter clustering since it considers the maximum distance (farthest neighbor) when merging clusters. It tends to create more balanced clusters but can be influenced by outliers, which may create small, tightly packed clusters.

Centroid Clustering: The 'centroid' linkage has a higher total SSE compared to the Complete Link (MAX), suggesting that the clusters might be less compact.

Based on the provided results, "Complete Link (MAX) Clustering" is preferred as it exhibits a lower total SSE, implying tighter and more balanced clusters, while "Centroid Clustering" results in a higher SSE, suggesting less compact clusters.

3.5

Report: Comparison Analysis of Clustering Algorithms

Introduction:

In this report, a detailed comparison analysis of two clustering algorithms, K-means, and Agglomerative Hierarchical Clustering are presented, using two different linkage methods (Complete Link - MAX and 'Centroid' Linkage). The dataset used for clustering consists of 16 attributes and 750 data records.

This report examines two primary clustering algorithms applied to the Dow Jones Index dataset: K-means and Agglomerative Hierarchical Clustering. Agglomerative Clustering is further analyzed using two inter-cluster similarity measures: Complete Link (MAX) and Centroid. My goal is to elucidate the strengths and weaknesses of each method in the context of this financial dataset.

Dataset Preprocessing:

Missing Values: There are missing values in two columns: percent_change_volume_over_last_wk and previous_weeks_volume. Since they are numerical, we can consider imputing these missing values with an appropriate statistic, such as the mean or median, which can help to retain valuable data without introducing too much bias.

Data Type Conversion: The columns 'open', 'high', 'low', 'close', 'next_weeks_open', and 'next_weeks_close' are of object type, which indicates they are strings. This is most likely due to the presence of the dollar sign. These need to be converted to a float data type after removing any non-numeric characters. The dataset's categorical attributes ("quarter", "stock", "date") were removed. Handling outliers for each column because algorithms are sensitive to them. This preprocessing step is crucial because clustering algorithms like K-means and Agglomerative Clustering are sensitive to the scale of the data, and standardization ensures that each feature contributes equally to the distance computations.

Scaling of the data is a critical step before applying K-means and hierarchical clustering algorithms because both methods use distance metrics to determine the similarity between data points.

Task 3.3 - Choosing an Adequate K Value: For the K-means algorithm, I determined the optimal number of clusters (K) using the elbow method. The analysis revealed that the "elbow" point, where the SSE starts to level off, was at K=4. Therefore, K=4 was selected as the number of clusters for the subsequent analyses.

Task 3.4 - Agglomerative Hierarchical Clustering: I performed Agglomerative Hierarchical Clustering using both Complete Link (MAX) and 'Centroid' Linkage methods with K=4 clusters.

Results and Analysis:

Based on the results of the k-means clustering analysis with different values of K,

K-Means Clustering Analysis Report

K-means Clustering

K-means was executed with different numbers of clusters (K = 2 to 7), and the elbow method was utilized to determine the optimal K. The selected K was based on the elbow plot where the rate of decrease in the total SSE changed sharply, indicating diminishing returns on SSE reduction with additional clusters.

Methodology:

- K-means clustering is a popular unsupervised machine learning technique used for partitioning data into K clusters based on similarity.
- The algorithm was applied to the dataset with different values of K to determine the optimal number of clusters.
- The evaluation metric used is the Total Sum of Squared Errors (SSE), which measures the sum of squared distances between data points and their respective cluster centroids. Lower SSE values indicate better clustering.

Results:

For K=2:

Total Sum of Squared Errors (SSE): 6157.832548661327

Sum of Squared Errors for Each Cluster:

Cluster 0: SSE = 2578.760903740677

Cluster 1: SSE = 3579.071644920648

For K=3:

Total Sum of Squared Errors (SSE): 4845.028652082341

Sum of Squared Errors for Each Cluster:

Cluster 0: SSE = 2244.1647083467556

Cluster 1: SSE = 1324.2865974863987

Cluster 2: SSE = 1276.577346249184

For K=4:

Total Sum of Squared Errors (SSE): 4372.4488848856645

Sum of Squared Errors for Each Cluster:

Cluster 0: SSE = 1231.062878686854

Cluster 1: SSE = 877.6815888356894

Cluster 2: SSE = 1280.1844608384515

Cluster 3: SSE = 983.5199565246684

For K=5:

Total Sum of Squared Errors (SSE): 4052.776331364635

Sum of Squared Errors for Each Cluster:

Cluster 0: SSE = 244.58735653806028

Cluster 1: SSE = 673.7673650818247

Cluster 2: SSE = 1243.739472435299

Cluster 3: SSE = 762.3074338619334

Cluster 4: SSE = 1128.3747034475173

For K=6:

Total Sum of Squared Errors (SSE): 3815.156870458883

Sum of Squared Errors for Each Cluster:

Cluster 0: SSE = 720.6161161415849

Cluster 1: SSE = 718.7447096022755

Cluster 2: SSE = 964.405793943503

Cluster 3: SSE = 665.1598320367381

Cluster 4: SSE = 95.31820158538272

Cluster 5: SSE = 650.9122171494

For K=7:

Total Sum of Squared Errors (SSE): 3610.0119931516383

Sum of Squared Errors for Each Cluster:

Cluster 0: SSE = 128.35202043321272

Cluster 1: SSE = 117.9216613980391

Cluster 2: SSE = 433.87449868779055

Cluster 3: SSE = 629.6221661474829

Cluster 4: SSE = 924.7934791456536

Cluster 5: SSE = 710.2883353027224

Cluster 6: SSE = 665.1598320367381

Discussion:

- The SSE values decrease as we increase the number of clusters (K), indicating that increasing K results in better clustering in terms of reducing the variance within clusters.
- For K=4, I observe that Cluster 1 and Cluster 3 have very low SSE values, indicating highly cohesive clusters. These clusters also have distinct mean values for most features, making them well-separated.
- Clusters 0 and 2 have higher SSE values, suggesting some overlap in their data points. They also exhibit moderate differences in their mean values.

Based on the k-means clustering analysis, increasing the number of clusters from 2 to 4 improved the clustering results, as evidenced by the decreasing SSE values. for this specific dataset, K-means with K=4 clusters using the elbow method is recommended as it provides a reasonable balance between compactness and the distribution of data points among clusters.

Advantages of K-means:

Efficiency: K-means is generally faster on large datasets since it converges quickly to a solution.

Ease of interpretation: The algorithm assigns each instance to one cluster, simplifying the post-analysis.

Optimization: K-means explicitly minimizes variance within clusters, leading to tight, spherical clusters.

Disadvantages of K-means:

Sensitivity to initialization: The final clusters can depend heavily on the initial choice of centroids.

Sensitivity to outliers: Outliers can significantly skew the centroids.

Assumption of spherical clusters: K-means assumes clusters are spherical and equally sized, which may not be the case in all datasets.

Agglomerative Hierarchical Clustering

This method was evaluated using two similarity measures: MAX and Centroid.

Agglomerative Hierarchical Clustering (Complete Link - MAX, K=4):

- Total Sum of Squared Errors (SSE): 1206891.8654417756
- Complete Link (MAX) generated four clusters with varying SSE:
  - Cluster 0: SSE: 227441.5180998647, number of Instances: 186
  - Cluster 1: SSE: 67324.92834848288, number of Instances: 92
  - Cluster 2: SSE: 849422.3158779233, number of Instances: 369
  - Cluster 3: SSE: 62703.10311550487, number of Instances: 103

Agglomerative Hierarchical Clustering ('Centroid' Linkage, K=4):

- Total Sum of Squared Errors (SSE): 5,401,458.16
- 'Centroid' Linkage produced four clusters with the following properties:
  - Cluster 0: SSE: 2603583.476363404, number of Instances: 552
  - Cluster 1: SSE: 12.462079540517905, number of Instances: 2
  - Cluster 2: SSE: 196817.80925878888, number of Instances: 171
  - Cluster 3: SSE: 1745.8097825900563, number of Instances: 25

Complete Link (MAX) Clustering: This method has a lower total SSE, which indicates tighter clustering since it considers the maximum distance (farthest neighbor) when merging clusters. It tends to create more balanced clusters but can be influenced by outliers, which may create small, tightly packed clusters.

Advantages:

Maximizes inter-cluster dissimilarity: It ensures that clusters are distinct from each other.

Robust to non-spherical shapes: Unlike K-means, it can accommodate clusters of various shapes.

Disadvantages:

Sensitivity to outliers: A single outlier can influence the formation of clusters.

Computational complexity: The algorithm is computationally more expensive, especially for large datasets.

Centroid Clustering: The 'centroid' linkage has a higher total SSE compared to the Complete Link (MAX), suggesting that the clusters might be less compact. The 'centroid' linkage method is less affected by outliers compared to the Complete Link method because it considers the average distance between all pairs of instances in two clusters.

Advantages:

Less sensitivity to outliers: It considers the average distance, which can mitigate the effect of outliers.

Cluster cohesion and separation: Tends to create clusters that are both cohesive internally and well-separated from each other.

Disadvantages:

Inefficiency with large datasets: The computational cost can be prohibitive with large datasets.

No global objective: Unlike K-means, there is no clear global objective being optimized, which can lead to less intuitive clusterings.