An Assessment of Clustering Techniques with Mall Customer Segmentation Data

Will Hinton

College of Business, Engineering, and Technology (COBET)

National University

TIM-8131 v2: Data Mining (8144764577)

Professor Sebhatleab Gezehey

October 26, 2025

An Assessment of Clustering Techniques with Mall Customer Segmentation Data

This research evaluates multiple clustering techniques using the Mall Customer Segmentation dataset from Kaggle, which includes demographic and spending information for 200 customers, such as CustomerID, Gender, Age, Annual Income (k$), and Spending Score (1–100). The dataset is suitable for testing clustering algorithms due to its manageable size, clear features, and relevance to marketing segmentation.

The project notebook outlines an end-to-end workflow that includes exploratory data analysis (EDA), preprocessing, encoding, and the application of six clustering algorithms: K-Means, K-Medoids (PAM), Agglomerative Hierarchical Clustering, DBSCAN, Gaussian Mixture Models (GMM), and Bayesian Clustering. Data preprocessing addressed missing values, scaled features with StandardScaler(), and encoded categorical attributes with LabelEncoder(). Visualizations illustrated the transformation of raw data into a standardized format, which enabled consistent comparison and interpretation of the clustering algorithms (Hastie, Tibshirani, & Friedman, 2009).

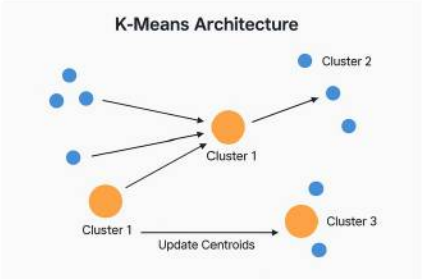
Clustering reveals hidden groupings in unlabeled data, offering insights for segmentation-based marketing. Each algorithm defines similarity and cluster boundaries differently. This study evaluates various algorithms based on performance, scalability, robustness, and interpretability to identify the best approach for mall customer segmentation (Irani, Pise, & Phatak, 2016; Mehta, Bawa, & Singh, 2020).

K-Means Clustering

K-Means is a clustering algorithm that partitions data into k clusters by minimizing within-cluster variance. It works iteratively by randomly initializing centroids, assigning points to the nearest centroid, and updating centroids until convergence. While efficient for large

datasets and easy to interpret, K-Means is sensitive to initial centroid placement and assumes clusters are spherical and equally sized. Using the Elbow and Silhouette analyses, the optimal number of clusters was found to be five, with evaluation metrics of Silhouette = 0.272, Davies–Bouldin = 1.181, and Calinski–Harabasz = 62.13, indicating moderately compact and well-separated groupings. Cluster profiling further revealed interpretable, distinct customer cohorts: for example, one younger, high-income, high-spending segment (mean Age ≈ 28.7 years, Income ≈ $60.9 k, Spending ≈ 70), contrasted with an older, lower-spending group (mean Age ≈ 56.5 years, Income ≈ $46 k, Spending ≈ 39). These findings align with clustering theory that emphasizes high intra-similarity and low inter-similarity among groups (Irani et al., 2016). See Figure 1.

Figure 1. *K-Means Clustering Architecture*.

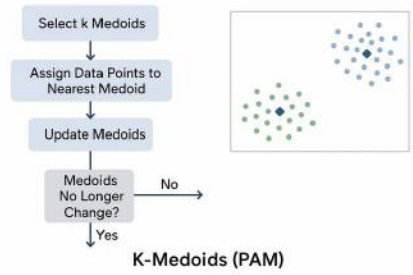


K-Medoids (PAM)

Partitioning Around Medoids (PAM) is similar to K-Means but uses actual data points as cluster centers, enhancing robustness against outliers. Unlike K-Means, which minimizes squared distances, K-Medoids minimizes pairwise dissimilarities, typically measured via Manhattan or Euclidean distance. In the notebook implementation, PAM was applied with k = 5, producing an overall Silhouette = 0.243, Davies–Bouldin = 1.385, and Calinski–Harabasz = 49.94—slightly lower compactness than K-Means but improved stability in the presence of outliers. This method reflects the partition-based category of clustering algorithms discussed by

Mehta, Bawa, and Singh (2020), emphasizing flexibility in distance measures and resilience to noisy data. Cluster profiling revealed several interpretable segments, including a younger, mid-income, high-spending cohort (mean Age ≈ 26.7 years, Income ≈ $49.8 k, Spending ≈ 68.5) and a moderate-age, high-income but low-spending group (mean Age ≈ 38.0 years, Income ≈ $89.1 k, Spending ≈ 23.7) . These distinctions confirm that PAM effectively captures realistic customer typologies using representative data points rather than computed centroids, making it particularly valuable for small to medium datasets where interpretability and robustness outweigh computational cost. See Figure 2.

Figure 2. *K-Medoids (PAM).*

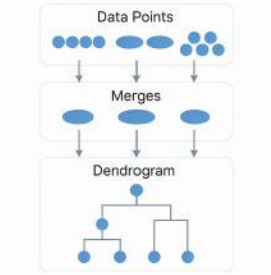


Agglomerative Hierarchical Clustering

Agglomerative Hierarchical Clustering builds clusters in a bottom-up manner, starting with each data point as its own cluster and merging pairs based on linkage criteria—single, complete, or average linkage. In the notebook, the Ward linkage method was applied with k = 5, producing an interpretable dendrogram that clearly illustrated the stepwise merging process and revealed several natural customer groupings . Quantitatively, the model achieved a Silhouette = 0.287, Davies–Bouldin = 1.220, and Calinski–Harabasz = 64.47, indicating moderately cohesive and well-separated clusters with slightly better structure than K-Means and K-Medoids . The resulting cluster profiles highlighted distinct behavioral segments, including a young, high-

income, high-spending group (mean Age ≈ 32.7 years, Income ≈ $86.5 k, Spending ≈ 82) and an older, moderate-income, low-spending group (mean Age ≈ 49.8 years, Income ≈ $44.1 k, Spending ≈ 39.7). The hierarchical approach is consistent with methods outlined by Irani et al. (2016) and Mehta et al. (2020), which emphasize the interpretability of dendrogram structures for exploratory data analysis. The dendrogram visualization effectively exposed hierarchical relationships across these cohorts, confirming the algorithm's value for exploratory structure discovery, though its $O(n^2)$ computational cost and noise sensitivity make it less practical for very large datasets. See Figure 3.

Figure 3. *Agglomerative Hierarchical Clustering.*

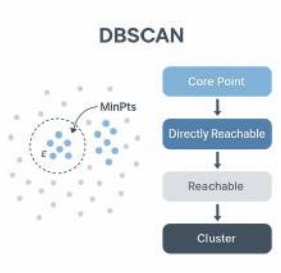


Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

DBSCAN identifies clusters based on data density rather than distance. It defines clusters as regions of high point density separated by sparse areas. The algorithm depends on parameters epsilon (ε) and minimum samples (minPts), which were set to ε = 0.5 and min_samples = 5 in the notebook. Under these parameters, DBSCAN identified nine clusters (excluding noise), with one substantial noise group capturing outlier customers who did not belong to any dense region. Although internal metrics could not be computed due to the irregular cluster count and presence of noise, visualizations confirmed that DBSCAN successfully delineated non-spherical, unevenly distributed clusters while isolating outliers along sparse data regions.

The cluster profile summary showed that typical DBSCAN groupings corresponded to young, high-spending customers (mean Age ≈ 25 years, Spending ≈ 74–75) and older, moderate-income, average-spending customers (mean Age ≈ 49–57 years, Income ≈ $52–62 k, Spending ≈ 48–50) . As highlighted by Mehta et al. (2020), density-based algorithms like DBSCAN excel in detecting arbitrarily shaped clusters and filtering out outliers, a property that strengthens their utility for anomaly detection in customer analytics. See Figure 4.

Figure 4. *Density-Based Spatial Clustering of Applications with Noise (DBSCAN).*

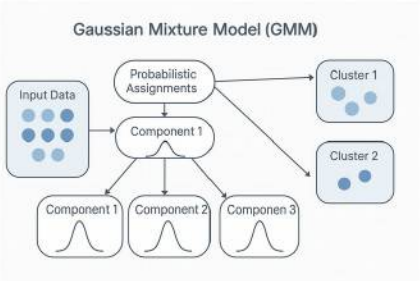


Gaussian Mixture Models (GMM)

GMM is a probabilistic model assuming data originates from a mixture of Gaussian distributions, where each cluster corresponds to a Gaussian component characterized by its mean and covariance matrix. Expectation-Maximization (EM) is used to iteratively estimate parameters until the model converges. In the notebook, a five-component GMM was applied, producing a Silhouette Score of 0.222, Davies–Bouldin Index of 1.211, and Calinski–Harabasz Score of 45.82, indicating moderately cohesive but flexible clusters with smooth transitions between groups. This result reflects the theoretical strengths of model-based clustering as described by Hastie, Tibshirani, and Friedman (2009), who emphasize mixture modeling for probabilistic segmentation. The probabilistic nature of GMM enabled overlapping boundaries between customer segments, reflecting real-world shopping behavior more effectively than hard-partitioning methods. Cluster profiling revealed distinct Gaussian-based groupings, such as a

younger, high-income, high-spending cluster (mean Age ≈ 28 years, Income ≈ $62 k, Spending ≈ 72) and a mid-aged, moderate-income, low-spending group (mean Age ≈ 49 years, Income ≈ $44 k, Spending ≈ 56) . These soft, interpretable memberships illustrate GMM's advantage in capturing nuanced spending patterns among mall customers, even when boundaries between behavioral groups are not sharply defined. See Figure 5.

Figure 5. *Gaussian Mixture Models (GMM)*

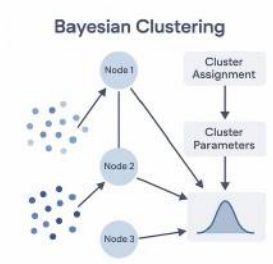


Bayesian Clustering

Bayesian Clustering extends mixture modeling by integrating prior distributions to manage model complexity and uncertainty, automatically adjusting the number of clusters through the use of Dirichlet priors and Bayesian Information Criterion (BIC) for model selection. In the notebook, the Bayesian Gaussian Mixture Model (BGMM) was implemented with a maximum of 10 components, and the model inferred an optimal effective cluster count of approximately 9, reflecting its adaptive nature. The Bayesian model achieved the highest overall internal performance among all algorithms, with a Silhouette Score of 0.415, Davies–Bouldin Index of 0.835, and Calinski–Harabasz Score of 86.12, demonstrating superior cluster separation and compactness. These outcomes align with probabilistic principles outlined by Hastie et al. (2009) and ensemble insights from Zhou (2012), emphasizing adaptive modeling and balance between model simplicity and explanatory power.

Cluster profiling revealed interpretable and realistic consumer segments, including a younger, high-income, high-spending group (mean Age ≈ 32 years, Income ≈ $86 k, Spending ≈ 82) and an older, low-income, moderate-spending group (mean Age ≈ 55.9 years, Income ≈ $46 k, Spending ≈ 40) . These results highlight Bayesian Clustering's strength in automatically inferring meaningful group structures with soft, probabilistic memberships—making it ideal for contexts where the number of customer segments is uncertain or when sample sizes are limited—although its computational demands and sensitivity to prior settings remain practical challenges. See Figure 6.

Figure 6. *Bayesian Clustering*



Evaluation

Evaluating clustering quality requires internal validation metrics since ground truth labels are absent. This project applied three major indices: the Silhouette Score, Davies-Bouldin Index, and Calinski-Harabasz Score. The Silhouette Score measures cohesion and separation, with higher values indicating well-defined clusters. The Davies-Bouldin Index rewards smaller intra-cluster distances and larger inter-cluster distances, where lower scores denote better structure. The Calinski-Harabasz Score quantifies the variance ratio between clusters, favoring high separation. These metrics collectively enabled a quantitative comparison of clustering performance. K-Means and GMM exhibited strong performance consistency, while DBSCAN provided better handling of outliers.

Synthesis and Comparative Assessment

The comparative findings across the six algorithms show trade-offs in efficiency, robustness, and interpretability. K-Means and K-Medoids are efficient and clear but struggle with irregular shapes. Hierarchical and density-based methods reveal complex structures but are computationally intensive. Model-based methods like GMM and Bayesian Clustering provide probabilistic interpretability, with Bayesian Clustering better at adapting complexity. These results align with the taxonomy of clustering approaches by Irani et al. (2016).The following table summarizes the comparative characteristics of each algorithm. See Table 1.

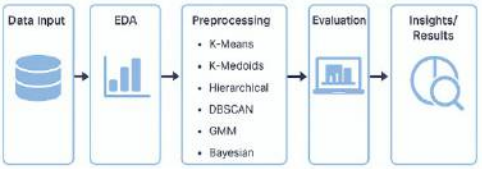
Table 1. *Algorithm Comparative Evaluation.*

Algorithm	Performance (Speed)	Scalability	Robustness	Interpretability
K-Means	High	High	Low–Medium	High (centroids simple)
K-Medoids (PAM)	Medium	Medium–Low	Medium–High (handles outliers)	High (actual medoid exemplars)
Agglomerative Hierarchical (Ward)	Medium	Medium–Low	Medium	Very High (dendrogram visual)
DBSCAN	High	Medium	High (outliers + shape-invariant)	Medium (core/noise distinction)
GMM	Medium	High	Medium (via covariance)	Medium (probabilistic)
Bayesian GMM	Medium–Low	Medium–High	Medium (regularized)	Medium (model complexity control)

Summary

This study analyzed six clustering algorithms on the Mall Customer dataset, highlighting their unique insights into customer segmentation. K-Means and K-Medoids were noted for their speed and interpretability, Hierarchical Clustering revealed structured relationships, DBSCAN identified anomalies, and Bayesian methods offered probabilistic flexibility. The findings showed Bayesian GMM had the highest structural integrity (Silhouette = 0.415), while DBSCAN emphasized density-based analysis. Overall, this research underscores that unsupervised clustering techniques can effectively reveal meaningful customer segments for targeted marketing, aligning with Mehta et al. (2020) that suggests the choice of method is context dependent. The following illustrates the overall analytical workflow. See Figure 7.

Figure 7. *Mall Customer Segmentation Project Workflow*



Conclusion and Reflection

This research highlights that while all six clustering methods have unique strengths, their effectiveness is closely tied to dataset structure and goals. K-Means serves as a solid baseline for segmentation tasks, PAM is robust against outliers, and DBSCAN is effective for outlier detection in irregular distributions. GMM and Bayesian clustering offer advanced probabilistic insights with soft memberships. As suggested by Hastie et al. (2009) and Zhou (2012), hybrid frameworks that combine deterministic and probabilistic approaches might improve performance in the future. This study emphasizes the role of clustering in data-driven marketing decision-making and the need to choose suitable unsupervised learning models.

Future work could focus on anomaly and outlier detection, essential for fraud detection, predictive maintenance, and medical diagnostics. Transitioning from clustering to hybrid unsupervised-supervised pipelines promises enhanced customer segmentation and behavioral modeling applications. See Table 2.

Table 2. *Algorithm Concepts and Comparative Characteristics*

Algorithm	Main Idea	Advantages	Disadvantages	Applications
K-Means	Partition data into k clusters minimizing within-cluster variance.	Fast; simple; reproducible. For this dataset, best-k ≈ 10.	Sensitive to outliers and scale; assumes spherical clusters.	Customer segmentation; market grouping; quick exploratory clustering.
K-Medoids (PAM)	Representative medoids minimize total dissimilarity; robust variant of K-Means.	Robust to outliers; interpretable medoids; can use various distance metrics.	Computationally heavier; slower on large datasets.	Retail/CRM segmentation; mixed-data clustering; anomaly-resilient grouping.
Agglomerative Hierarchical (Ward)	Bottom-up merges using Ward linkage to minimize variance increase.	No need for k upfront; dendrogram reveals multi-scale structure.	O(n^2) memory/time complexity; sensitive to noise with many points.	Persona hierarchies; biological or marketing taxonomies.
DBSCAN	Density-based clustering; discovers arbitrary shapes and noise.	Detects outliers; no k required; non-spherical clusters supported.	Parameter-sensitive (eps, min_samples); struggles with varying densities.	Outlier detection; spatial data; irregular pattern discovery.
Gaussian Mixture Model (GMM)	Probabilistic mixture of Gaussians with soft cluster membership.	Models elliptical clusters; provides probabilities; uses AIC/BIC for model choice.	Assumes Gaussian shapes; may overfit; sensitive to initialization.	Soft segmentation; anomaly scoring; financial behavior modeling.
Bayesian GMM	Extends GMM with priors (Dirichlet) to infer effective number of clusters.	Regularized; handles uncertain k; avoids overfitting redundant components.	Higher compute cost; sensitive to prior selection.	Cohort discovery when k unknown; uncertainty-aware clustering.

References

Irani, J., Pise, N. & Phatak, M. (2016). Clustering techniques and the similarity measures used in clustering: A survey. *International Journal of Computer Applications.* *134*(7). pp. 9-14. https://doi.org/10.5120/ijca2016907841

Mehta, V., Bawa, S. & Singh, J. Analytical review of clustering techniques and proximity measures. *Artif Intell Rev* **53**, 5995–6023 (2020). https://doi.org/10.1007/s10462-020-09840-7

Muggeo, V. M. R. (2003). Segmented regression: Introduction and methodology. Environmetrics, 14(5), 453–463. https://resources.nu.edu/ld.php?content_id=80804856

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Chapters 8. In Deep learning. MIT Press. https://www.deeplearningbook.org

Zhou, Z.-H. (2012). Chapters 2–4: Bagging, Boosting, and Random Forests. In Ensemble methods: Foundations and algorithms. Chapman and Hall/CRC. https://doi.org/10.1201/b12207

Hastie, T., Tibshirani, R., & Friedman, J. (2009). Chapters 14. In The elements of statistical learning: Data mining, inference, and prediction (2nd ed.). Springer. https://hastie.su.domains/ElemStatLearn/

Choudhary, V. J. (n.d.). *Mall Customer Segmentation Data* [Dataset]. Kaggle. https://www.kaggle.com/datasets/vjchoudhary7/customer-segmentation-tutorial-in-python

Appendix

This appendix appends and lists the project notebook in PDF form. The attached shows corresponding Python code and Markdown comments as an illustration of the project to construct, study, and evaluate techniques. See attached.

HintonWTIM8131-6

October 24, 2025

An Assessment of Data Clustering Techniques using Mall Customer Segmentation Data

- Dataset: Mall Customer Segmentation Data (Kaggle)
- URL: https://www.kaggle.com/datasets/vjchoudhary7/customer-segmentation-tutorial-in-python
- Purpose: For comparative clustering analysis using K-Means, PAM, Hierarchical, DBSCAN, GMM, and Bayesian GMM

[45]:
```
__author__ = "Will Hinton"
__email__ = "willhint@gmail.com"
__website__ = "whinton0.github.com/py"
```

0.1 Part 1. Import Libraries and Load Dataset

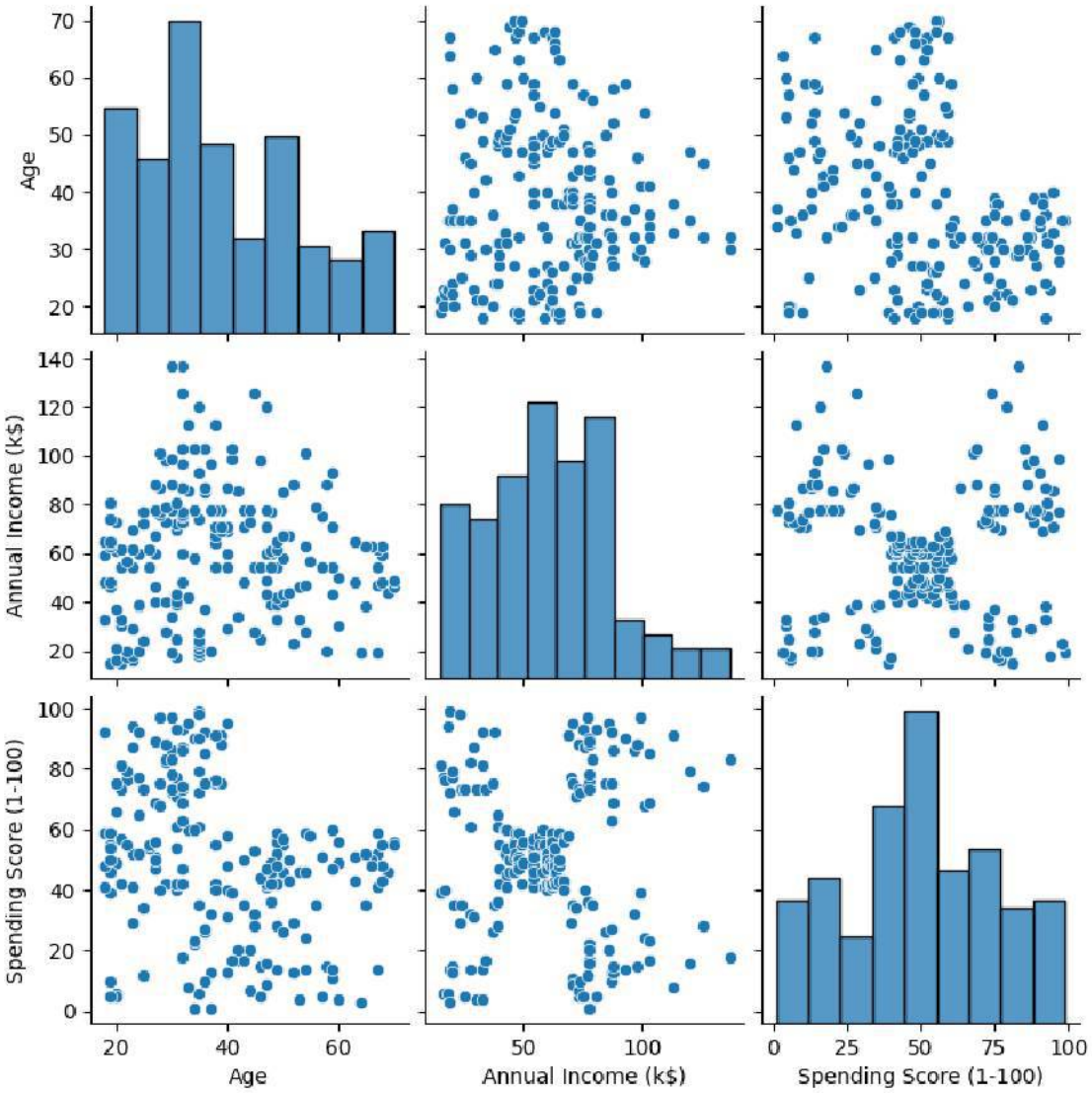
[46]:
```
!pip install -qqq pyclustering
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
from sklearn.mixture import GaussianMixture, BayesianGaussianMixture
from sklearn.metrics import silhouette_score, davies_bouldin_score,␣
 ↪calinski_harabasz_score
from pyclustering.cluster.kmedoids import kmedoids
from scipy.spatial.distance import cdist

# the CSV file is downloaded locally
file_path = './Mall_Customers.csv'
df = pd.read_csv(file_path)
```

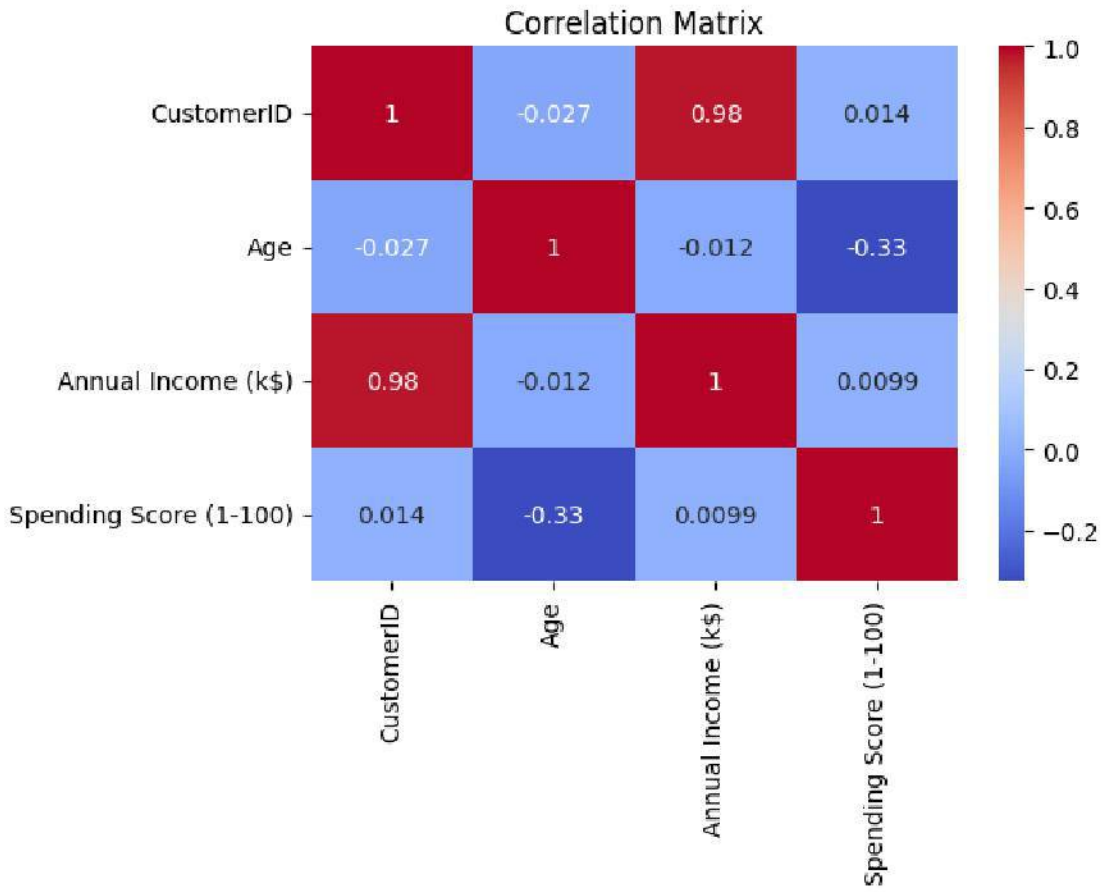
0.2 Part 2. Initial Data Exploration (EDA)

[47]:
```
print("\n--- Basic Info ---")
print(df.info())

print("\n--- Summary Statistics ---")
```

```python
print(df.describe())

print("\n--- Missing Values ---")
print(df.isnull().sum())

# Pairplot or correlation heatmap
sns.pairplot(df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']])
plt.show()

plt.figure(figsize=(6,4))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

```
--- Basic 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
None

--- Summary Statistics ---
       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

--- Missing Values ---
CustomerID                0
Gender                    0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
```

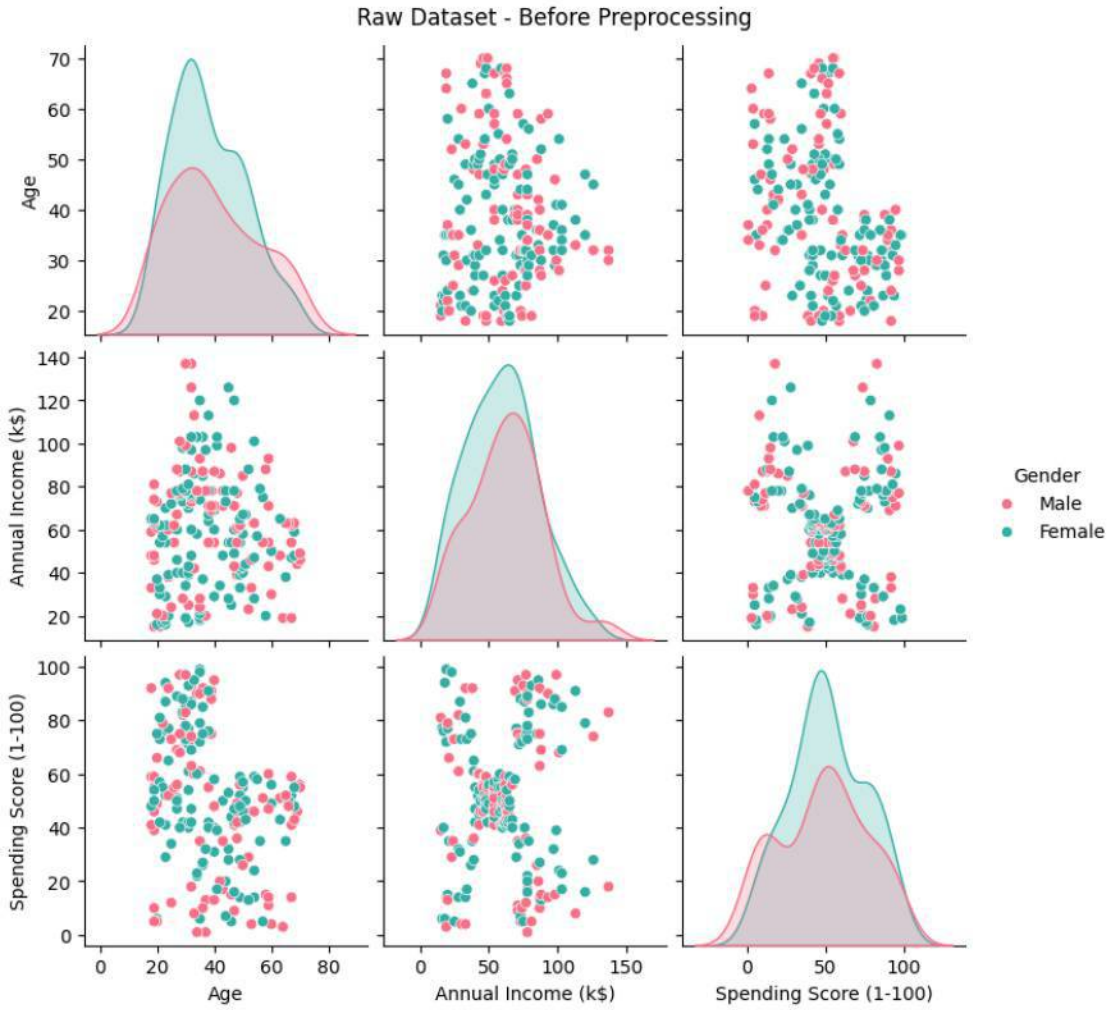

Age
Annual Income (k$)
Spending Score (1-100)
Age
Annual Income (k$)
Spending Score (1-100)

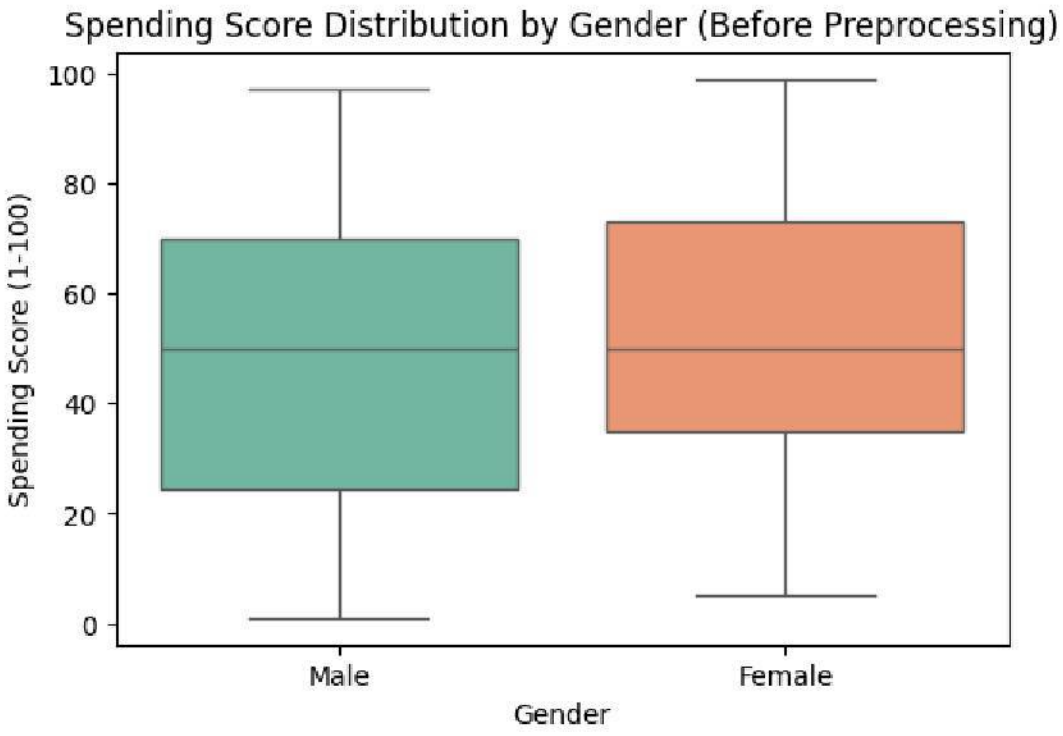


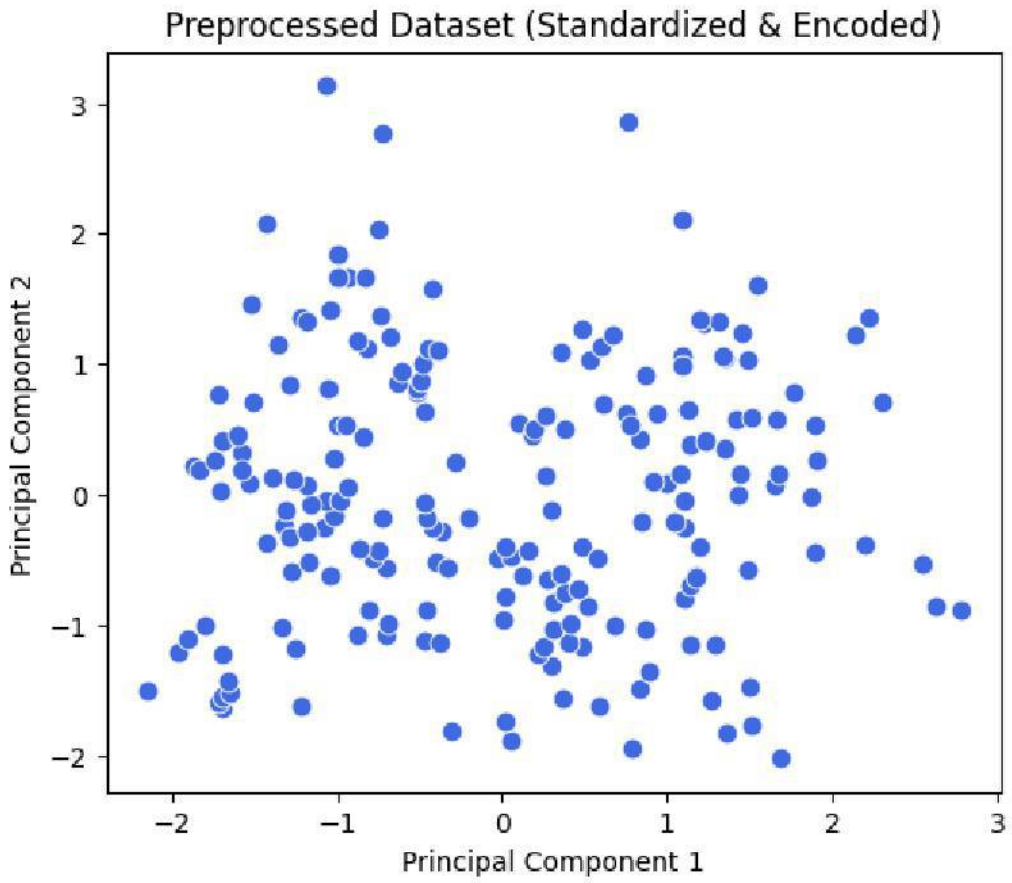
0.3 Part 3. Data Preprocessing

[48]:
```
# 3A. Visualize Raw Dataset (Before Preprocessing)

import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler, LabelEncoder

# Use hue='Gender' to apply palette meaningfully
sns.pairplot(
    df,
    vars=['Age', 'Annual Income (k$)', 'Spending Score (1-100)'],
    hue='Gender',
    diag_kind='kde',
    palette='husl'
)
plt.suptitle("Raw Dataset - Before Preprocessing", y=1.02)
```

```
plt.show()

# Gender-based Spending visualization (explicit hue for palette)
plt.figure(figsize=(6,4))
sns.boxplot(
    data=df,
    x='Gender',
    y='Spending Score (1-100)',
    hue='Gender',
    palette='Set2',
    legend=False
)
plt.title('Spending Score Distribution by Gender (Before Preprocessing)')
plt.show()


# 3B. Visualize After Preprocessing

# Encode Gender and scale features for clustering visualization
le = LabelEncoder()
df_enc = df.copy()
df_enc['Gender'] = le.fit_transform(df_enc['Gender'])
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_enc.drop(columns=['CustomerID'],␣
 ↪errors='ignore'))

# PCA reduction for 2D visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
pca_df = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])

plt.figure(figsize=(6,5))
sns.scatterplot(
    data=pca_df,
    x='PC1',
    y='PC2',
    s=60,
    color='royalblue'
)
plt.title('Preprocessed Dataset (Standardized & Encoded)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```


Raw Dataset - Before Preprocessing
Age
Annual Income (k$)
Spending Score (1-100)
Gender
Male
Female

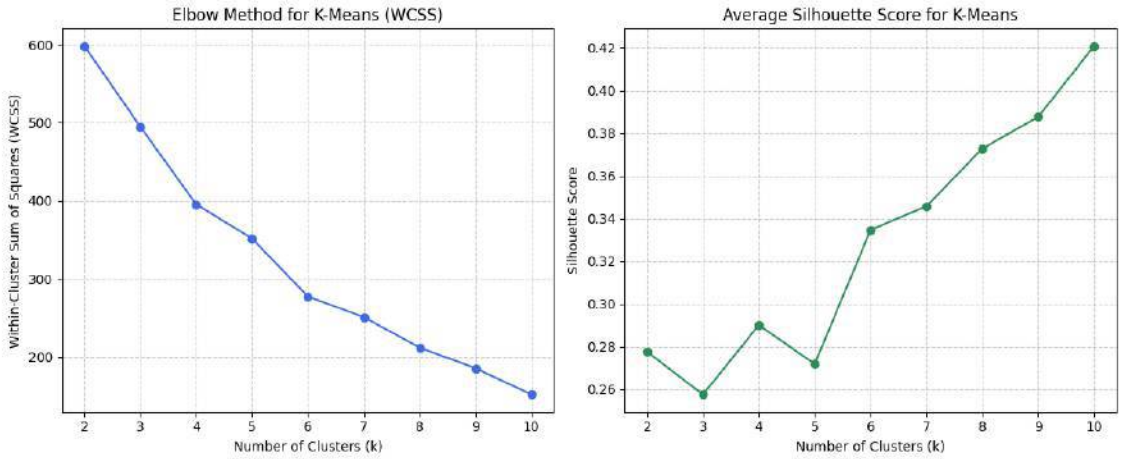

Spending Score Distribution by Gender (Before Preprocessing)
Spending Score (1-100)
100
80
60
40
20
0
Male
Female
Gender

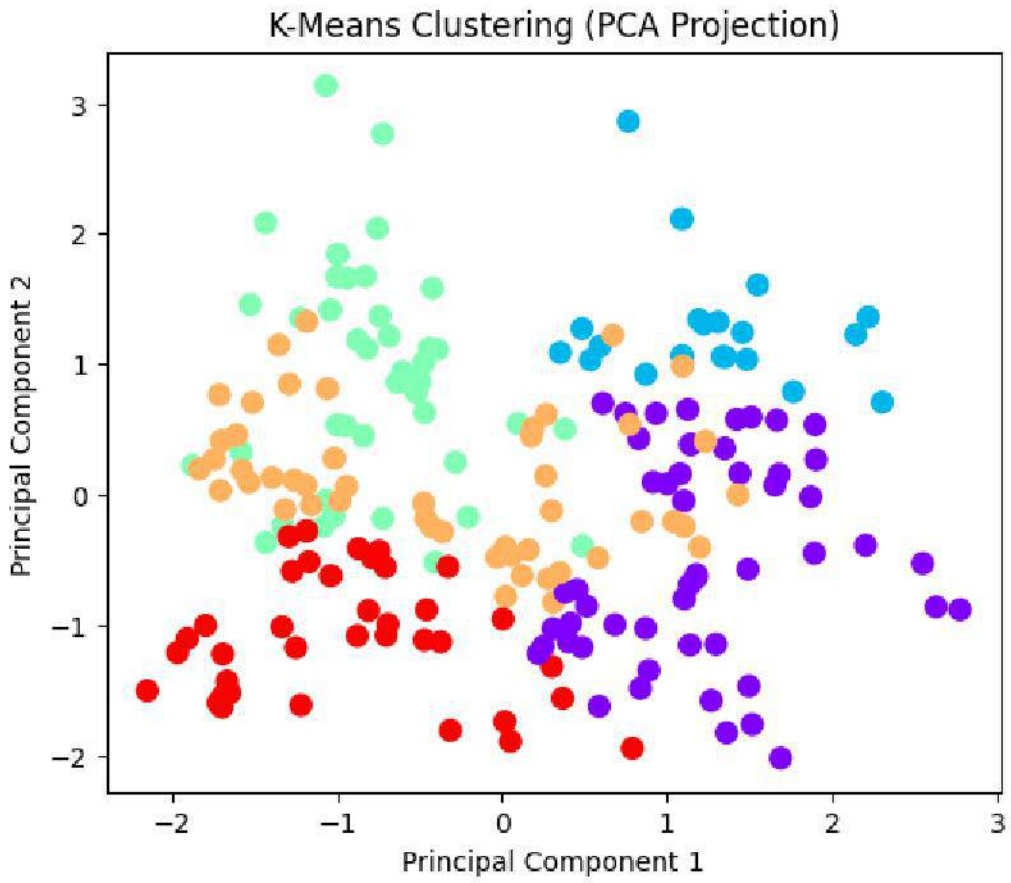


0.4 Part 4. K-Means Clustering

[49]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
import numpy as np

# Elbow method (WCSS) and Silhouette Score side-by-side
wcss = []
silhouette_scores = []
K_range = range(2, 11)

for k in K_range:
    km = KMeans(n_clusters=k, random_state=42)
    labels = km.fit_predict(X_scaled)
    wcss.append(km.inertia_)
    sil_score = silhouette_score(X_scaled, labels)
    silhouette_scores.append(sil_score)
```

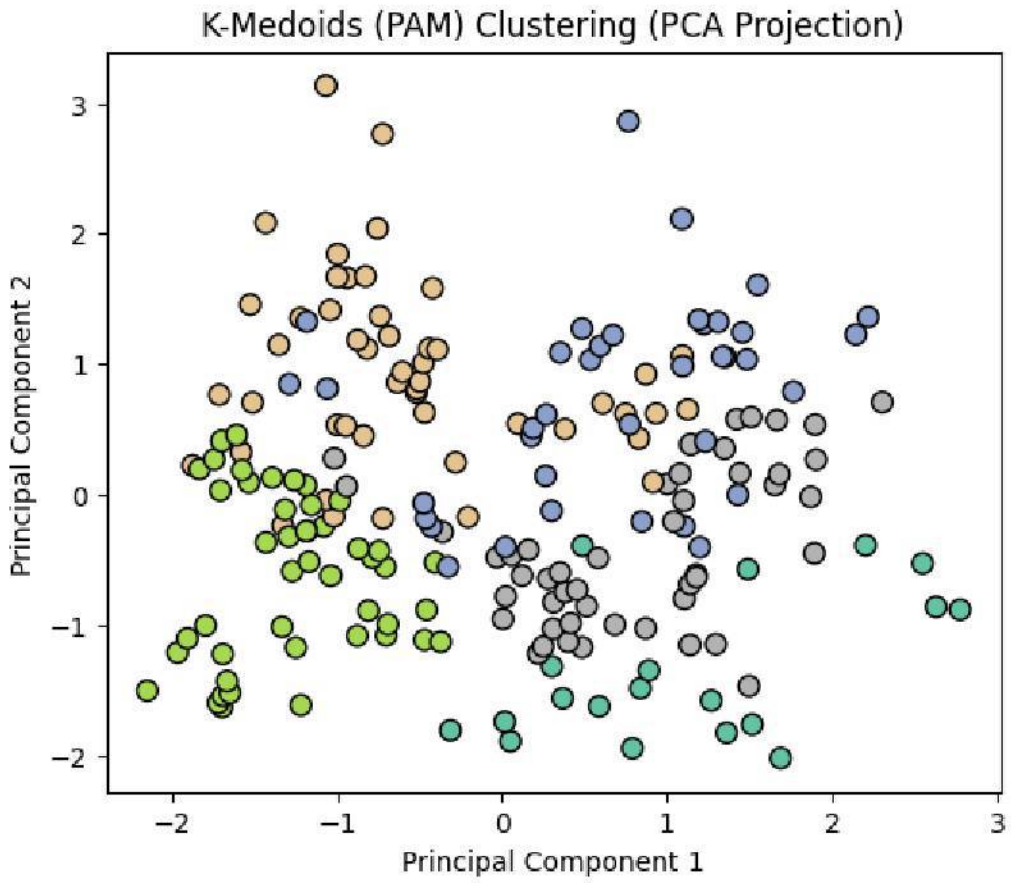
```
# ---- Plot both side-by-side ----
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Elbow Method (WCSS)
axes[0].plot(K_range, wcss, marker='o', color='royalblue')
axes[0].set_title('Elbow Method for K-Means (WCSS)')
axes[0].set_xlabel('Number of Clusters (k)')
axes[0].set_ylabel('Within-Cluster Sum of Squares (WCSS)')
axes[0].grid(True, linestyle='--', alpha=0.6)

# Silhouette Score
axes[1].plot(K_range, silhouette_scores, marker='o', color='seagreen')
axes[1].set_title('Average Silhouette Score for K-Means')
axes[1].set_xlabel('Number of Clusters (k)')
axes[1].set_ylabel('Silhouette Score')
axes[1].grid(True, linestyle='--', alpha=0.6)

plt.tight_layout()
plt.show()

# ---- Apply final K-Means model (example: k=5) ----
kmeans = KMeans(n_clusters=5, random_state=42)
kmeans_labels = kmeans.fit_predict(X_scaled)

# ---- Visualize clusters (PCA projection) ----
plt.figure(figsize=(6,5))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=kmeans_labels, cmap='rainbow', s=60)
plt.title('K-Means Clustering (PCA Projection)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```





0.5 Part 5. K-Medoids (PAM) Clustering

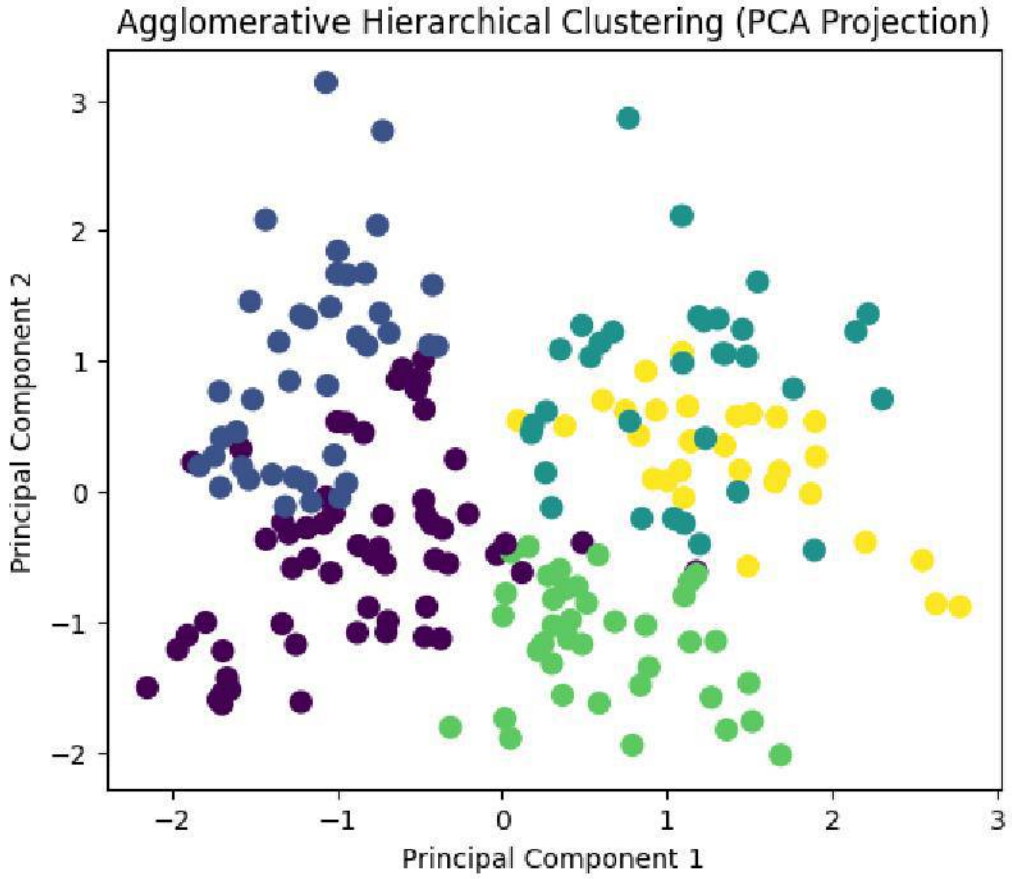
[50]:
```
from sklearn_extra.cluster import KMedoids
import matplotlib.pyplot as plt

# Fit model
kmedoids = KMedoids(n_clusters=5, metric='euclidean', random_state=42)
pam_labels = kmedoids.fit_predict(X_scaled)

# Visualize clusters (PCA projection)
plt.figure(figsize=(6,5))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=pam_labels, cmap='Set2', s=60,␣
↪edgecolor='k')
plt.title('K-Medoids (PAM) Clustering (PCA Projection)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```

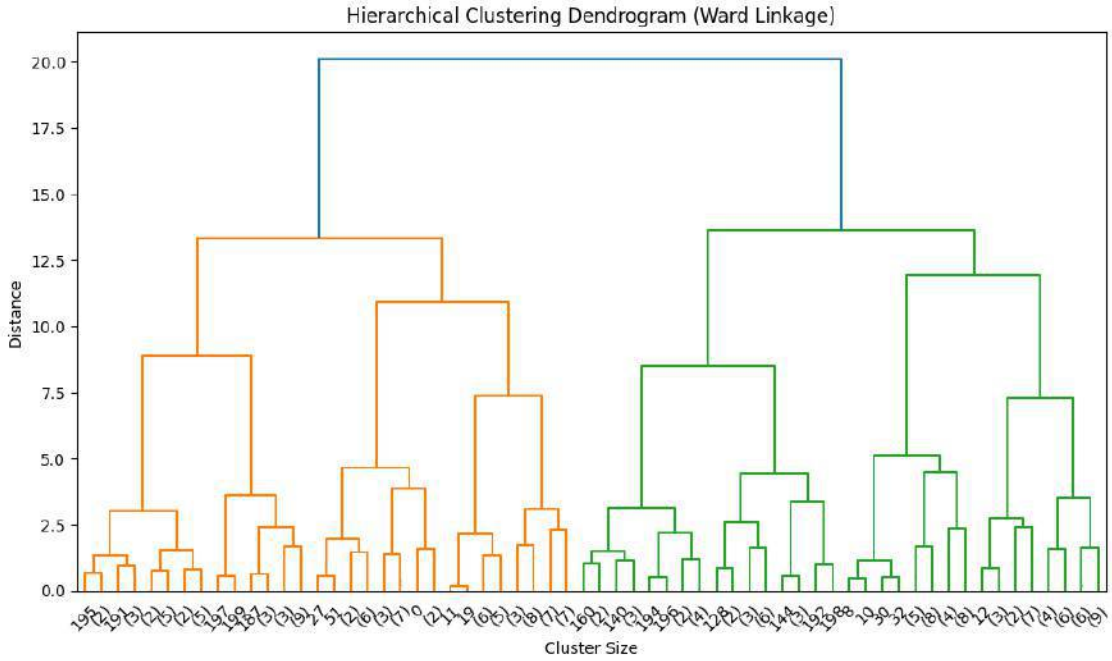


0.6 Part 6. Agglomerative Hierarchical Clustering

[51]:
```
from sklearn.cluster import AgglomerativeClustering
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage

# --- Existing Code ---
agg = AgglomerativeClustering(n_clusters=5, linkage='ward')
agg_labels = agg.fit_predict(X_scaled)

plt.figure(figsize=(6,5))
plt.scatter(X_pca[:,0], X_pca[:,1], c=agg_labels, cmap='viridis', s=60)
plt.title('Agglomerative Hierarchical Clustering (PCA Projection)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()

# --- Additional: Hierarchical Clustering Dendrogram (Ward Linkage) ---
```

```python
# Perform hierarchical clustering on full dataset
# (if necessary, X_scaled[::2])
Z = linkage(X_scaled, method='ward')

plt.figure(figsize=(10, 6))
dendrogram(
    Z,
    truncate_mode='level',  # use 'lastp' or 'level' for compact display
    p=5,                    # number of cluster levels to show
    leaf_rotation=45.,
    leaf_font_size=10.,
    show_contracted=True,
)
plt.title('Hierarchical Clustering Dendrogram (Ward Linkage)')
plt.xlabel('Cluster Size')
plt.ylabel('Distance')
plt.tight_layout()
plt.show()
```





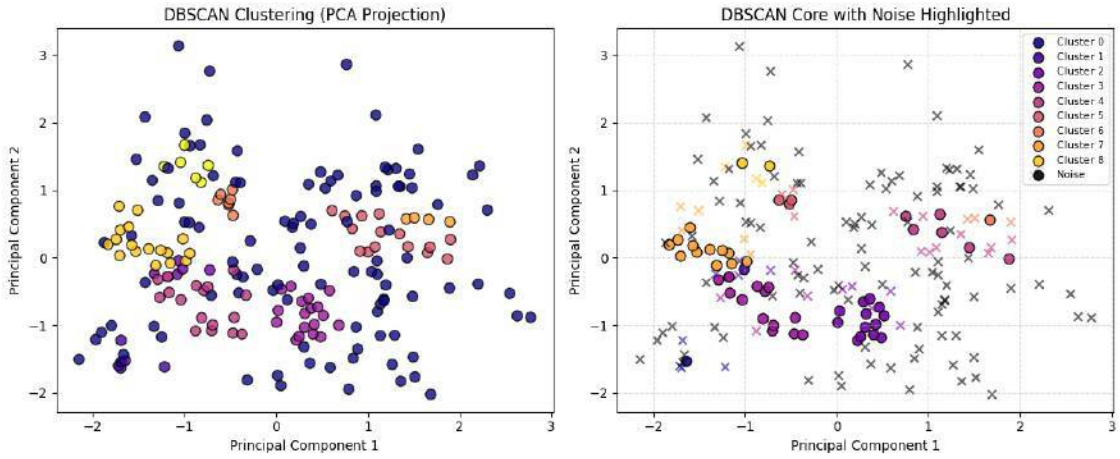
0.7 Part 7. DBSCAN Clustering

[52]:
```
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt
from matplotlib import colormaps
import numpy as np

# Fit DBSCAN model
dbscan = DBSCAN(eps=0.5, min_samples=5)
db_labels = dbscan.fit_predict(X_scaled)

# Identify core samples and noise
core_samples_mask = np.zeros_like(db_labels, dtype=bool)
core_samples_mask[dbscan.core_sample_indices_] = True

# Number of clusters (excluding noise if present)
n_clusters_ = len(set(db_labels)) - (1 if -1 in db_labels else 0)
print(f"Estimated number of clusters (excluding noise): {n_clusters_}")

# ---- Create side-by-side plots ----
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
cmap = colormaps["plasma"]

# --- (Left) Standard DBSCAN Clustering ---
scatter1 = axes[0].scatter(
```

```
    X_pca[:, 0],
    X_pca[:, 1],
    c=db_labels,
    cmap=cmap,
    s=60,
    edgecolor='k',
    alpha=0.8
)
axes[0].set_title("DBSCAN Clustering (PCA Projection)")
axes[0].set_xlabel("Principal Component 1")
axes[0].set_ylabel("Principal Component 2")

# --- (Right) DBSCAN Core vs Noise Highlighted ---
unique_labels = set(db_labels)
colors = [cmap(each / max(1, len(unique_labels)-1)) for each in␣
 ↪range(len(unique_labels))]

for k, col in zip(unique_labels, colors):
    if k == -1:
        # Noise (black or dark gray)
        col = (0, 0, 0, 1)
    class_member_mask = (db_labels == k)

    xy_core = X_pca[class_member_mask & core_samples_mask]
    xy_border = X_pca[class_member_mask & ~core_samples_mask]

    # Core points
    axes[1].scatter(
        xy_core[:, 0],
        xy_core[:, 1],
        s=60,
        c=[col],
        marker='o',
        edgecolor='k',
        label=f'Cluster {k}' if k != -1 else 'Noise',
        alpha=0.9
    )

    # Border or noise points
    axes[1].scatter(
        xy_border[:, 0],
        xy_border[:, 1],
        s=40,
        c=[col],
        marker='x',
        alpha=0.6
    )
```

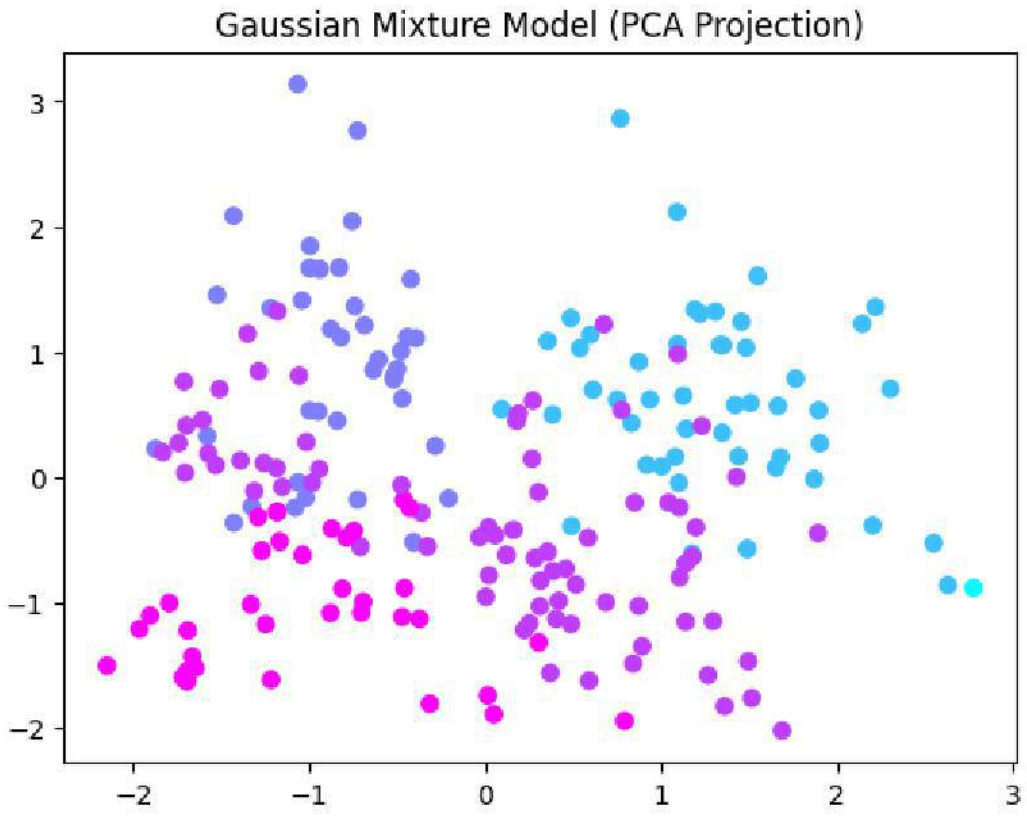
```
axes[1].set_title("DBSCAN Core with Noise Highlighted")
axes[1].set_xlabel("Principal Component 1")
axes[1].set_ylabel("Principal Component 2")
axes[1].legend(loc='best', fontsize=8)
axes[1].grid(True, linestyle='--', alpha=0.4)

plt.tight_layout()
plt.show()
```

```
Estimated number of clusters (excluding noise): 9
```



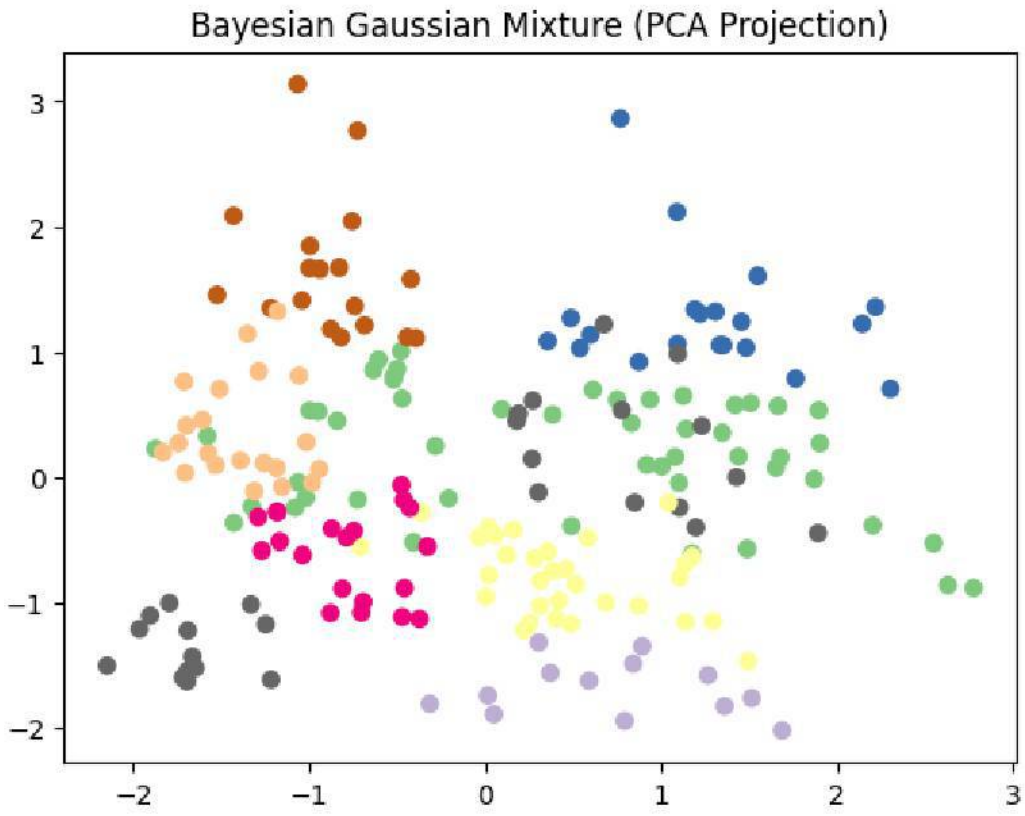
0.8 Part 8. Gaussian Mixture Models (GMM)

[53]:
```
gmm = GaussianMixture(n_components=5, covariance_type='full', random_state=42)
gmm_labels = gmm.fit_predict(X_scaled)

plt.scatter(X_pca[:,0], X_pca[:,1], c=gmm_labels, cmap='cool')
plt.title('Gaussian Mixture Model (PCA Projection)')
plt.show()
```



0.9 Part 9. Bayesian Gaussian Mixture (Bayesian GMM)

[54]:
```
vgmm = BayesianGaussianMixture(n_components=10, covariance_type='full',␣
 ↪random_state=42)
vgmm_labels = vgmm.fit_predict(X_scaled)

plt.scatter(X_pca[:,0], X_pca[:,1], c=vgmm_labels, cmap='Accent')
plt.title('Bayesian Gaussian Mixture (PCA Projection)')
plt.show()
```



0.10 Part 10. Evaluation Metrics

[55]:
```
from sklearn.metrics import silhouette_score, davies_bouldin_score,␣
↪calinski_harabasz_score
import pandas as pd

# Function to compute metrics
def compute_metrics(X, labels):
    """Compute Silhouette, Davies-Bouldin, and Calinski-Harabasz scores if␣
↪valid clusters exist."""
    if len(set(labels)) > 1 and -1 not in set(labels):
        sil = silhouette_score(X, labels)
        db = davies_bouldin_score(X, labels)
        ch = calinski_harabasz_score(X, labels)
        return round(sil, 3), round(db, 3), round(ch, 3)
    else:
        return None, None, None

# Compute metrics for all models
results = {
    'K-Means': compute_metrics(X_scaled, kmeans_labels),
```

```
    'K-Medoids (PAM)': compute_metrics(X_scaled, pam_labels),
    'Agglomerative Hierarchical': compute_metrics(X_scaled, agg_labels),
    'DBSCAN': compute_metrics(X_scaled, db_labels),
    'GMM': compute_metrics(X_scaled, gmm_labels),
    'Bayesian GMM': compute_metrics(X_scaled, vgmm_labels)
}

# Convert to DataFrame
metrics_df = pd.DataFrame(results, index=['Silhouette Score', 'Davies-Bouldin␣
 ↪Score', 'Calinski-Harabasz Score']).T

# Display neatly
display(metrics_df.style.background_gradient(cmap='Blues', subset=['Silhouette␣
 ↪Score'])
                        .format(precision=3)
                        .set_caption("Table 1. Clustering Evaluation Metrics␣
 ↪Across Methods"))
```

```
<pandas.io.formats.style.Styler at 0x13f6a6d90>
```

0.11 Part 11. Cluster Profile Summary

[56]:
```
import pandas as pd
results_df = df.copy()

# Attach cluster labels for each technique
results_df['KMeans_Cluster']        = kmeans_labels
results_df['KMedoids_Cluster']      = pam_labels
results_df['Agglomerative_Cluster'] = agg_labels
results_df['DBSCAN_Cluster']        = db_labels
results_df['GMM_Cluster']           = gmm_labels
results_df['Bayesian_Cluster']      = vgmm_labels   # Dirichlet Process /␣
 ↪Bayesian GMM

# function to compute and print the profile summary
def cluster_profile_summary(df, cluster_col):
    """
    the mean profile of each cluster for numeric variables.
    """
    ## print(f"\n--- Cluster Profile Summary ({cluster_col}) ---")
    summary = df.groupby(cluster_col).mean(numeric_only=True)
    ## print(summary)
    return summary

# Generate and store all summaries
summaries = {}
```

```
summaries['K-Means']        = cluster_profile_summary(results_df,␣
 ↪'KMeans_Cluster')
summaries['K-Medoids (PAM)'] = cluster_profile_summary(results_df,␣
 ↪'KMedoids_Cluster')
summaries['Agglomerative']  = cluster_profile_summary(results_df,␣
 ↪'Agglomerative_Cluster')
summaries['DBSCAN']         = cluster_profile_summary(results_df,␣
 ↪'DBSCAN_Cluster')
summaries['GMM']            = cluster_profile_summary(results_df, 'GMM_Cluster')
summaries['Bayesian']       = cluster_profile_summary(results_df,␣
 ↪'Bayesian_Cluster')

# Ensure summaries dictionary exists
if 'summaries' in locals():
    for method, summary in summaries.items():
        if summary is not None and not summary.empty:
            print(f"\n=== {method} Cluster Profile Summary ===")
            display(
                summary.round(2)
                .style.background_gradient(cmap='Purples', axis=None)
                .set_caption(f"Table - {method} Cluster Profile Summary (Mean␣
 ↪of Numeric Variables)")
                .format(precision=2)
            )
        else:
            print(f"\n=== {method} Cluster Profile Summary ===")
            print("No valid clusters found or insufficient data for summary.")
else:
    print("  No summaries found. Please run the previous cell first.")
```

```
=== K-Means Cluster Profile Summary ===
<pandas.io.formats.style.Styler at 0x13c323bb0>

=== K-Medoids (PAM) Cluster Profile Summary ===
<pandas.io.formats.style.Styler at 0x13c1e0790>

=== Agglomerative Cluster Profile Summary ===
<pandas.io.formats.style.Styler at 0x13e43b0d0>

=== DBSCAN Cluster Profile Summary ===
<pandas.io.formats.style.Styler at 0x13c323700>
```

```
=== GMM Cluster Profile Summary ===

<pandas.io.formats.style.Styler at 0x13c166ca0>


=== Bayesian Cluster Profile Summary ===

<pandas.io.formats.style.Styler at 0x13f67c250>
```

0.12 Part 12 — Synthesis, Method Rationale, and Comparative Assessment

0.12.1 What Was Done and Why

Data & context. Analyze the Kaggle *Mall Customer Segmentation* dataset (200 customers; features: gender, age, annual income, spending score) to compare a suite of clustering techniques on the same low-dimensional business problem. The aim was to understand how choices in modeling affect performance, scalability, robustness to outliers/shape, and interpretability.

EDA (Part 2). The raw data was explored to verify schema and missingness, computed descriptive statistics, and visualized pairwise relationships (scatter/pair plots, box plots). This established basic patterns (e.g., high-spending younger customers) and checked for anomalies.

Preprocessing (Part 3). Conversion was done for the categorical `Gender` to numeric, and all numeric features were standardized so that distance-based methods (K-Means, K-Medoids, Hierarchical, DBSCAN) operate on comparable scales. Principal Component Analysis (PCA) to 2D was used only for visualization; models were fit on the full standardized feature space.

Modeling & evaluation (Parts 4–11). - **K-Means (Part 4):** Baseline centroid method. We used the Elbow (within-cluster sum of squares) and Silhouette curves to choose k and plotted PCA-projected scatterplots to check cluster separation visually. - **K-Medoids / PAM (Part 5):** Like K-Means, but centers are actual data points (medoids), making the method more robust to outliers. Implemented via `scikit-learn-extra` for portability. - **Agglomerative hierarchical clustering (Part 6):** A bottom-up tree of merges using Ward linkage to minimize within-cluster variance. A dendrogram was plotted to visualize the hierarchy and decide on k. - **DBSCAN (Part 7):** Density-based approach that discovers arbitrary-shaped clusters and explicit noise points. Visualizing core vs. noise points helped understand density-based structure and outlier detection. - **Gaussian mixture models (GMM) (Part 8):** Probabilistic mixture model with soft assignments; each component has its own covariance. Model selection via AIC and BIC helped determine the number of components. - **Bayesian GMM (Part 9):** Introduces a Dirichlet prior over mixture weights so that unnecessary components shrink away, yielding an inferred number of clusters and additional regularization. - **Evaluation metrics (Part 10):** For each method, the Silhouette, Davies–Bouldin, and Calinski–Harabasz scores were computed and summarized in a table. These internal metrics quantify cohesion and separation, complementing visual assessments. - **Cluster profiles (Part 11):** The mean of each numeric variable was profiled within each cluster for every method, presenting six tables (one per algorithm). These profiles connect algorithmic output back to interpretable business descriptors (e.g., income and spending patterns by cluster).

0.12.2 Techniques: main ideas, strengths, limits, and uses

K-Means.
Idea: Partition into k spherical clusters by minimizing within-cluster variance around centroids.
Advantages: Fast, scalable; works well when clusters are compact and similar in size.

Disadvantages: Sensitive to initialization and outliers; assumes roughly spherical clusters; must choose k.
Applications: Quick segmentation baselines, market cohort prototypes.

K-Medoids (PAM).
Idea: Use actual observations (medoids) as cluster centers, minimizing average dissimilarity.
Advantages: More robust to outliers; can use non-Euclidean distance metrics.
Disadvantages: More computationally intensive than K-Means on larger datasets.
Applications: Retail/CRM cohorts with outliers or mixed data types.

Agglomerative hierarchical (Ward).
Idea: Start with individual points and iteratively merge clusters; Ward linkage minimizes the increase in within-cluster variance.
Advantages: Produces a dendrogram that reveals multiscale structure; no need to pre-specify k; good interpretability.
Disadvantages: Higher time and memory cost ($O(n^2)$); less scalable to very large datasets.
Applications: Exploratory persona discovery, taxonomy building, understanding cluster hierarchy.

DBSCAN.
Idea: Density-based algorithm where points with enough neighbors within a radius *eps* are core points; others are border or noise.
Advantages: Identifies arbitrarily shaped clusters and explicit noise without specifying k; robust to outliers.
Disadvantages: Sensitive to `eps` and `min_samples`; struggles with varying densities.
Applications: Anomaly detection, spatial or log data with irregular boundaries.

Gaussian mixture models (GMM).
Idea: Fit a mixture of Gaussian distributions with soft (probabilistic) assignments; each component has its own mean and covariance.
Advantages: Flexibility to model elliptical clusters; principled model selection via information criteria (AIC/BIC).
Disadvantages: Assumes Gaussianity; sensitive to initialization; may overfit.
Applications: Soft segmentation, marketing personas with probabilistic membership, anomaly scoring.

Bayesian GMM.
Idea: Place a prior on mixture weights and covariance parameters (e.g., Dirichlet process); unused components shrink, effectively learning the number of clusters.
Advantages: Automatically infers cluster count; regularizes complex models; yields uncertainty estimates.
Disadvantages: More computationally demanding; hyperparameters influence results; can converge slowly.
Applications: Cohort discovery when k is unknown; scenarios requiring parsimonious models and uncertainty quantification.

0.12.3 Comparative analysis on this dataset

Performance (speed): K-Means is fastest; PAM is slower; hierarchical is moderate on 200 observations; DBSCAN is relatively fast; GMM and Bayesian GMM incur extra time for EM/variational inference.

Scalability: K-Means and GMM scale well to large `n`; hierarchical and PAM struggle as `n` grows; DBSCAN scales moderately but is sensitive to the curse of dimensionality.

Robustness: PAM and DBSCAN handle outliers better than K-Means or GMM. DBSCAN also discovers non-spherical patterns and explicit noise. GMM tolerates ellipsoidal clusters via covariance; Bayesian GMM adds regularization to guard against overfitting.

Interpretability:
- **High:** Hierarchical (dendrogram), K-Means (simple centroids), and PAM (representative medoids).
- **Moderate:** DBSCAN (core/noise concept is intuitive but parameter-dependent).
- **Moderate–Low:** GMM/Bayesian GMM (probabilistic assignments and model complexity are harder to explain to non-technical audiences).

Bottom line for mall customers: In this small, low-dimensional dataset, K-Means or GMM produce clear, stable segments for reporting. PAM is a strong alternative when outliers matter. DBSCAN offers outlier and noise discovery, but parameter tuning is needed. Hierarchical clustering is excellent for communicating how clusters form and for exploring different values of k. Bayesian GMM is valuable when you need a principled approach to model complexity and cluster uncertainty.

> **Note:** Use internal metrics (Part 10) together with cluster profiles (Part 11) and business understanding to choose the most appropriate segmentation method.

[57]:
```python
# Summary Table with Automatic Best-k shown

import pandas as pd
import numpy as np

# --- retrieve best k from Silhouette Score list ---
try:
    best_k = K_range[np.argmax(silhouette_scores)]
except Exception:
    best_k = 5   # default fallback
print(f"Detected optimal k (from silhouette): k = {best_k}")


# Table  - Performance, Scalability, Robustness, Interpretability
table_b = pd.DataFrame([
    {"Algorithm": "K-Means",
     "Performance (Speed)": "High",
     "Scalability": "High",
     "Robustness": "Low-Medium",
     "Interpretability": "High (centroids simple)"},
    {"Algorithm": "K-Medoids (PAM)",
     "Performance (Speed)": "Medium",
     "Scalability": "Medium-Low",
     "Robustness": "Medium-High (handles outliers)",
     "Interpretability": "High (actual medoid exemplars)"},
    {"Algorithm": "Agglomerative Hierarchical (Ward)",
```

```
     "Performance (Speed)": "Medium",
     "Scalability": "Medium-Low",
     "Robustness": "Medium",
     "Interpretability": "Very High (dendrogram visual)"},
    {"Algorithm": "DBSCAN",
     "Performance (Speed)": "High",
     "Scalability": "Medium",
     "Robustness": "High (outliers + shape-invariant)",
     "Interpretability": "Medium (core/noise distinction)"},
    {"Algorithm": "GMM",
     "Performance (Speed)": "Medium",
     "Scalability": "High",
     "Robustness": "Medium (via covariance)",
     "Interpretability": "Medium (probabilistic)"},
    {"Algorithm": "Bayesian GMM",
     "Performance (Speed)": "Medium-Low",
     "Scalability": "Medium-High",
     "Robustness": "Medium (regularized)",
     "Interpretability": "Medium (model complexity control)"}
])

display(table_b.style.set_caption(
    "Table B - Comparative Evaluation (Performance, Scalability, Robustness,␣
  ↪Interpretability)"
))
```

```
Detected optimal k (from silhouette): k = 10
```

```
<pandas.io.formats.style.Styler at 0x13e3680a0>
```

[ ]: