Healthcare Scenario - Healthy Living and Wellness Clustering

Abstract

This study focuses on the application of unsupervised machine learning algorithms to identify wellness profiles from a simulated medical dataset. The dataset consisted of 200 patients details including their five key lifestyle variables such as exercise time, healthy meals per day, hours of sleep, stress level and body mass index. Unsupervised machine learning techniques like K-Means, Agglomerative Hierarchical Clustering, Gaussian Mixture Models (GMM), and DBSCAN were performed for clustering and identifying patterns. Dimensionality reduction was done using Principal Component Analysis (PCA), which provided more insight into the data. Evaluation metrics which were used as Silhouette Score, Within-cluster Sum of Squares (WCSS), Davies Bouldin Index and Calinski Harabasz Score to determine a number of well-formed clusters. K-Means was identified as the best technique, achieving the highest silhouette score of 0.398 after PCA transformation, and showing well-formed, distinct clusters. The findings of this study provide support for the use of clustering models to enable the segmentation of patient populations in support of wellness intervention strategies. The study illustrates how unsupervised learning may be leveraged to accelerate data-driven approaches to preventive healthcare.

Introduction

Health is Wealth! It is the the most valuable aspect of human life. It plays a significant and critical role not only in individual well being, but it also influences the productivity and resillience of entire communities. In the last couple of years, the focus on health and preventive health has become more prominent as awareness of chronic diseases, mental health challenges, and unhealthy behaviours has grown. Health care professionals need to not only assess the original state of health of the patient, but they need to understand the complex picture of health, to develop meaningful wellness programs and prescribe appropriate health interventions. The amount of health data is growing in both size and complexity. Therefore, the need for intelligent methods to analyze and interpret actual health data meaningfully is critical now and going to be an essential element in the future.

Unsupervised machine learning has become a revolutionary methodology for analysis in health-related systems and healthcare. Unsupervised techniques explore and cluster observations regardless of labels or categories, highlighting hidden structures in complex datasets (Alanazi, 2022; Jayatilake & Ganegoda, 2021). Clustering methods, such as K-Means and Hierarchical Clustering, have been used to explore lifestyle patterns while patients are grouped together according to similar behaviors. Other methods of dimensionality reduction, such as Principal Component Analysis (PCA) combine measures to help simplify data in a higher dimensionality and for ease of interpretation. These methods are necessary for capturing the complexity of health and wellness.

This study focuses on supporting a healthcare organization in understanding a simulated dataset that includes amount of daily exercise, diet quality, sleep duration, stress level scores, and Body Mass Index (BMI). Through clustering, patients are divided into wellness categories for targeted initiatives. Dimensional reduction technique like PCA has been used to how it impacts clustering, and the clarity of the patient groupings. The insights from this project could support a more individualized or personalized wellness approaches rather than delivering general wellness recommendations (Trezzza et al., 2024; Allenbrand, 2024).

This study is significant as it contributes to healthcare by an approach of using population level data to develop personalized, tailored treatment strategy based on an individual's unique pattern of data. By identifying wellness clusters from a single population group, a healthcare system can more effectively allocate resources knowing the risk levels. Also, a healthcare system can deliver programs that are specific to the lifestyle components of the individual. Prior research has also indicated that the methods of this current research have strong outcomes in the implementation of drugs, fraud prevention and prediction of diseases (Lu & Uddin, 2024; massi et al, 2020). Integrating clustering and PCA is an important advancement towards a more intelligent, smarter, and flexible responsive healthcare system.

Literature Review

The implementation of machine learning in healthcare has increased rapidly, as healthcare organizations have large amount of patient data, both structured and unstructured data. This has changed healthcare industries from being reactive being proactive by identifying data driven opportunities. Also, a wide body of research has been done, looking at the importance of machine learning in clinical decision-making, disease diagnosis, and individualized treatment in both supervised and unsupervised studies, (Jayatilake & Ganegoda, 2021). Although many publications focused on disease prediction using labeled datasets, recent literature has proposed that unsupervised approaches can be helpful in finding patterns in unlabeled health data to support healthy behavior initiatives.

Unsupervised learning, including clustering methods, is becoming a viable option for patient segmentation. Trezza et al. (2024) suggest that unsupervised learning is significant for precision medicine because it can reveal hidden subgroups of patients without the requirement of labels already established. The authors state that unsupervised learning is valuable for personalizing care pathways, mainly in lifestyle and behavioral health pathways. Allenbrand (2024) utilized both the machine learning approaches, supervised and unsupervised to analyze data from health and wellness around pharma and medications. The diversity in results indicates that unsupervised methods can be extremely significant and show incredible results across various domains. These studies show the ways clustering algorithms, such as K-Means or Hierarchical Clustering, may be applied in segmentation of the many behaviors and attitudes in health.

Pu and Uddin (2024) contributes by doing a comparative study of unsupervised methods using multiple databases in health care provides another contribution. Their study illustrates many clustering methods, differences identified based on the data context and dimensionality, and the importance of feature reduction intervention like Principal Component Analysis (PCA), which assists with both computational reduction and interpretation of clusters, particularly in domains of wellness data containing many inter-related variables such as sleep, stress, nutrition. However, Massi et al. (2020) used a two-step clustering process to identify outliers in administrative databases for the purposes of fraud analysis. Their example shows that clustering provides a value beyond just clinical diagnostic, such as clustering in the health and wellness domain.

Alanazi (2022) provides a comprehensive overview of machine learning in health care addressing health issues with a point to future work indicating the number of wellness studies either in association with real-world data or simulated data, as most of the literature in this area is related to disease prediction and improving forms of treatment approaches. However, some literature focusses explicitly on clustering patients, relating specifically to wellness data, and that could inform potential preventive wellness programming, but on a large dataset. While this study is inspired by other literature, it addresses the important gap by applying unsupervised learning techniques to a narrow data set made up of core wellness variables which are, exercise time, eating behavior, sleep, stress levels, and BMI. Instead of analyzing a broad-spectrum patient attribute study, this analysis focuses on everyday conduct factors that are changeable, directly related to general wellness.

To sum up, while prior research supports the utility of unsupervised learning in healthcare, there is still opportunity to investigate the applicability to lifestyle segmentation in wellness-type datasets. This study expands on the work of Trezza et al. (2024) and Lu and Uddin (2024) but does so with a more focused dataset on actionable health behaviors. This offers a new insight into how health organizations can utilize clustering and dimensionality reduction techniques to tailor wellness interventions and promote healthier populations.

Methodology

Dataset Overview

The dataset for this study has 200 records of wellness profiles, based on five central lifestyle measures. These measures are denoted as daily exercise time in minutes, the number of healthy meals consumed per day, hours of sleep each night, and participants self-reported their overall stress level on a scale from 0 to 10 and body mass index (BMI). This concentrated dataset provides key modifiable factors related to health, as it incorporates central ways in which patients can take control over their health outcomes. This dataset gives an insight into patients day to day life behaviour which is quite different from clinical dataset that focuses majorly on disease, medical record or disease specific details. The dataset focuses on regular health behaviours, thus can be effectively analysed using unsupervised learning methods to extract additional, useful insights or to create patient profile segments using clustering approaches. It can be further refined and explored successfully at each level of clustering and dimensionality reduction, to assess and describe wellness pathways and help design specific health interventions.

```python
# Import essential libraries
import pandas as pd
import numpy as np

# Load the dataset
df = pd.read_csv('simulated_health_wellness_data.csv')

# Display the first few rows
df.head()
```

```
   Exercise_Time_Min  Healthy_Meals_Per_Day  Sleep_Hours_Per_Night  \
0          34.967142                      5               7.618856   
1          28.617357                      8               4.105473   
2          36.476885                      4               6.024123   
3          45.230299                      1               8.565319   
4          27.658466                      3               8.301648   

   Stress_Level        BMI  
0             2  33.068556  
1             7  27.267672  
2             1  23.779217  
3             8  29.820436  
4             3  30.947352  
```

Handling Missing Values

The first stage was examining the dataset for null values. A column-wise examination confirmed that there were no null entries across all variables, allowing the dataset to be used in its entirety without the use of imputation. Then data types and exploratory analysis was done to assess for inconsistencies or outlier values.

```
# Dataset structure
df.info()

# Basic statistics
df.describe()

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Exercise_Time_Min      200 non-null    float64
 1   Healthy_Meals_Per_Day  200 non-null    int64  
 2   Sleep_Hours_Per_Night  200 non-null    float64
 3   Stress_Level           200 non-null    int64  
 4   BMI                    200 non-null    float64
dtypes: float64(3), int64(2)
memory usage: 7.9 KB

Exercise_Time_Min        0
Healthy_Meals_Per_Day    0
Sleep_Hours_Per_Night    0
Stress_Level             0
BMI                      0
dtype: int64
```

Above table displays the number of null values discovered for each variable of the dataset. As illustrated, no missing data were discovered for any of the five health and wellness indicators, which allowed the complete dataset (N = 200) to be used in the analysis without proceeding with imputation.

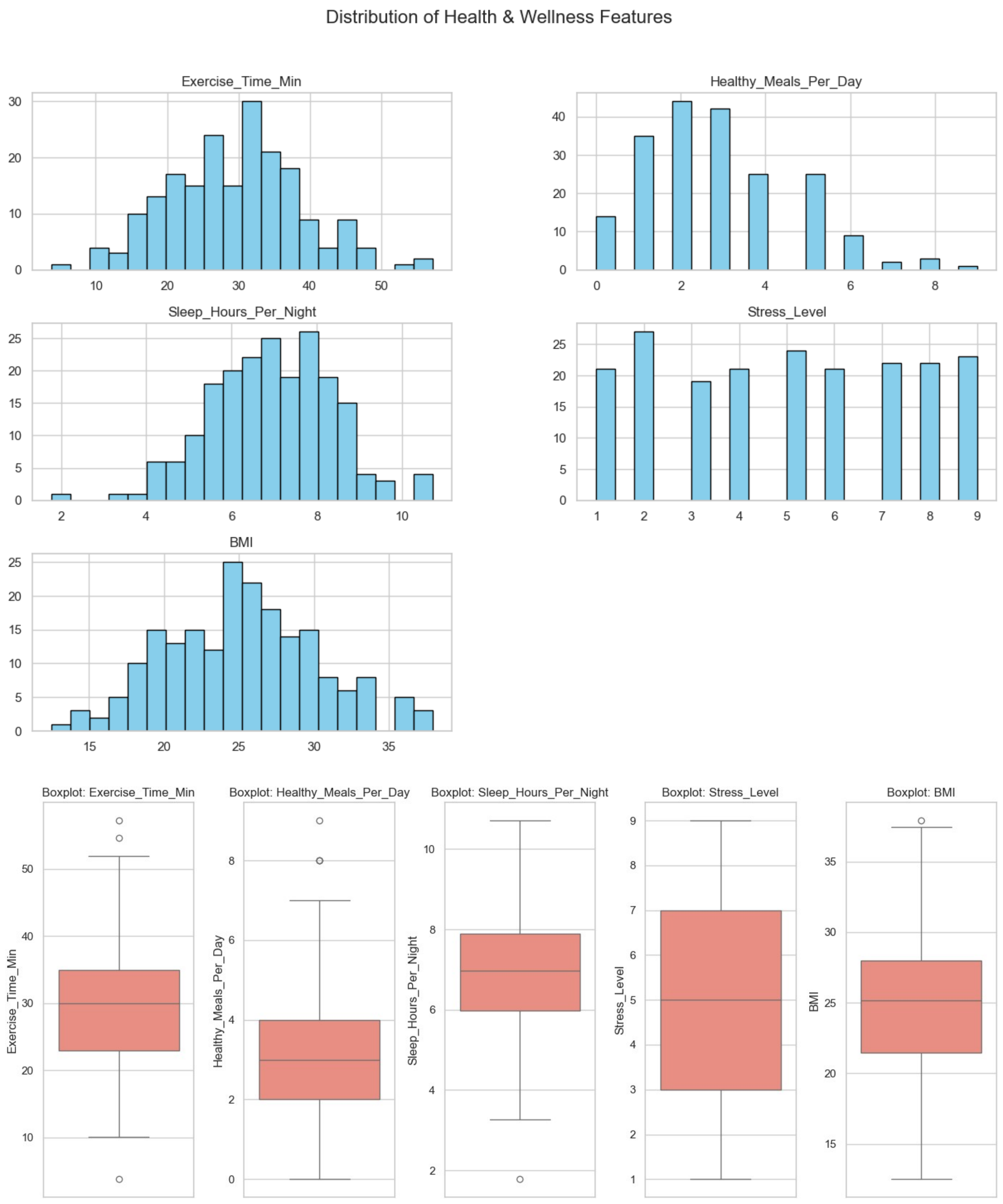
Exploratory Data Analysis

To gain additional insights into the dataset, an exploratory data analysis (EDA) was performed on features such as exercise time, healthy meals consumed each day, sleep hours, stress level, and BMI. All features and data types were analyzed separately and all 5 features began with a histogram for each features to show the distribution of the feature. The distributions for exercise time, sleep hours, and BMI were approximately normal. The distribution for healthy meals consumed per day was positively skewed and therefore suggests that respondents select more healthy meals than average, the distribution for stress level appeared to have no clear distribution pattern. After obtaining histograms, a correlation heatmap was created to view correlations or relationships among the variables. Overall, the features were mostly weakly correlated with each other, which may indicate that these features may contribute in different ways to the latent variable of patient segmentation. The most negative correlation noted was between stress level and BMI (r = –0.13).

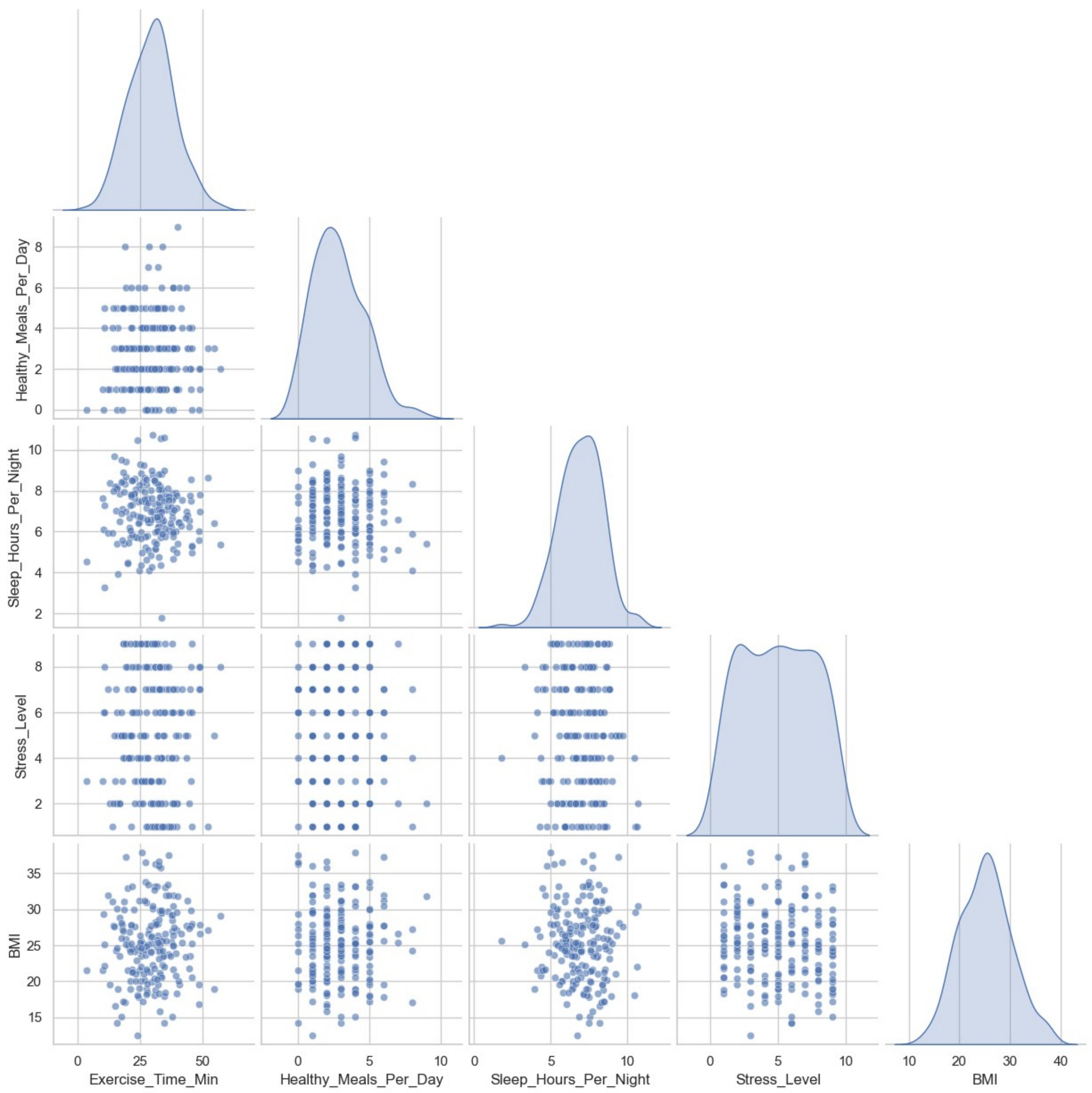
A scatter-KDE pairplot was created to examine the interactions and separability of variables. This offered distributions of the singular variables, along with pairwise relationships, but also provided some evidence of clustering in discrete combinations such as sleep and stress or exercise and BMI. A swarm plot was used next to show the actual distribution of each individual data point, revealing repeat measurements of discrete variables such as healthy meals and stress level, while exposing outliers of BMI and other continuous features. A violin plot, in contrast, provided a smoothed, statistical representation of the distribution of each feature, in which a kernel density estimation was used to provide patterns of spread, skewness, and central tendency with boxplots imbedded within the plot. Together, these two graphs offered complementary visualizations and built a strong analytical foundation for clustering, revealing possible locations of natural behaviors in the patient population.

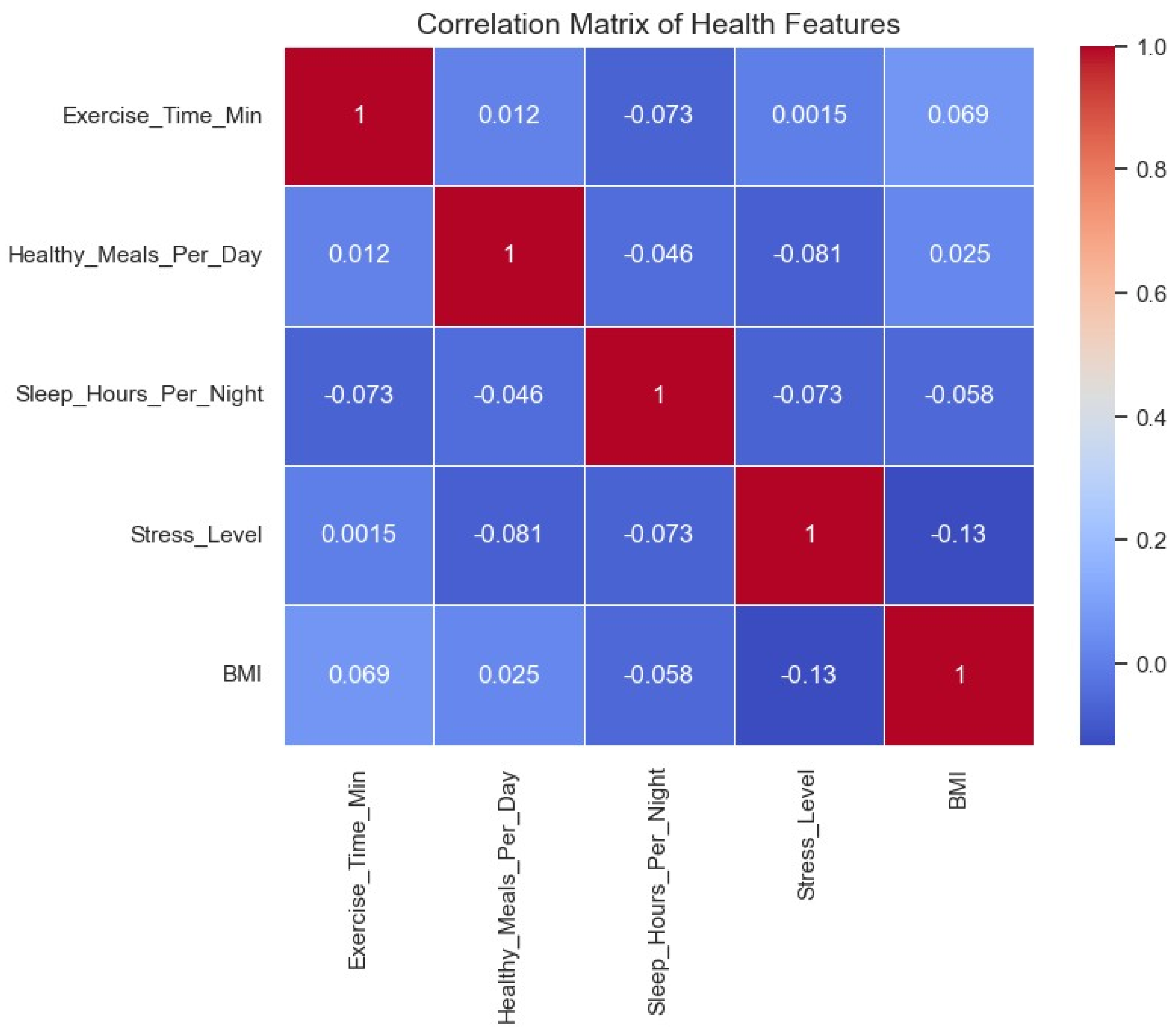
```python
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

# Set up aesthetics
sns.set(style="whitegrid")
plt.rcParams['figure.figsize'] = (12, 6)

# 1. Histograms + KDE Plots
df.hist(bins=20, figsize=(15, 10), color='skyblue', edgecolor='black')
plt.suptitle("Distribution of Health & Wellness Features",
fontsize=16)
plt.show()

# 2. Boxplots (Outlier detection)
plt.figure(figsize=(14, 6))
```

```python
for i, col in enumerate(df.columns):
    plt.subplot(1, 5, i+1)
    sns.boxplot(y=df[col], color='salmon')
    plt.title(f'Boxplot: {col}')
plt.tight_layout()
plt.show()

# 3. Pairplot (Feature relationships)
sns.pairplot(df, diag_kind='kde', corner=True, plot_kws={'alpha':0.6})
plt.suptitle("Pairwise Relationships Between Features", y=1.02,
fontsize=16)
plt.show()

# 4. Correlation heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', linewidths=0.5)
plt.title("Correlation Matrix of Health Features", fontsize=14)
plt.show()

# 5. Violin Plots (Feature distributions with spread)
plt.figure(figsize=(16, 6))
for i, col in enumerate(df.columns):
    plt.subplot(1, 5, i+1)
    sns.violinplot(y=col, data=df, palette="pastel")
    plt.title(f'Violin Plot: {col}')
plt.tight_layout()
plt.show()
```


Distribution of Health & Wellness Features
Exercise_Time_Min
Healthy_Meals_Per_Day
Sleep_Hours_Per_Night
Stress_Level
BMI
Boxplot: Exercise_Time_Min
Boxplot: Healthy_Meals_Per_Day
Boxplot: Sleep_Hours_Per_Night
Boxplot: Stress_Level
Boxplot: BMI
Exercise_Time_Min
Healthy_Meals_Per_Day
Sleep_Hours_Per_Night
Stress_Level
BMI

Pairwise Relationships Between Features

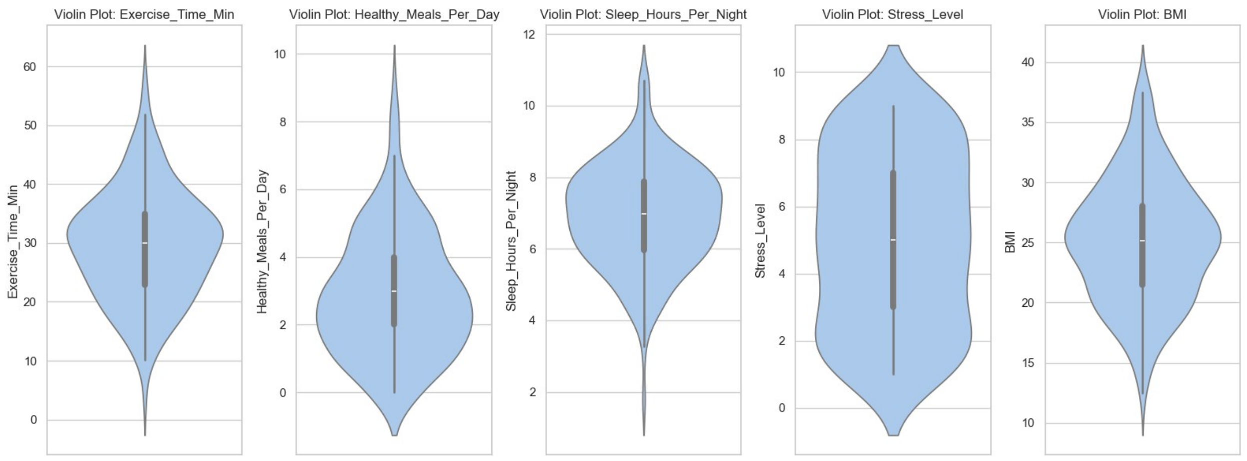




	Exercise_Time_Min	Healthy_Meals_Per_Day	Sleep_Hours_Per_Night	Stress_Level	BMI
Exercise_Time_Min	1	0.012	-0.073	0.0015	0.069
Healthy_Meals_Per_Day	0.012	1	-0.046	-0.081	0.025
Sleep_Hours_Per_Night	-0.073	-0.046	1	-0.073	-0.058
Stress_Level	0.0015	-0.081	-0.073	1	-0.13
BMI	0.069	0.025	-0.058	-0.13	1

```
C:\Users\priya\AppData\Local\Temp\ipykernel_62540\4232543248.py:39:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.violinplot(y=col, data=df, palette="pastel")
C:\Users\priya\AppData\Local\Temp\ipykernel_62540\4232543248.py:39:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.violinplot(y=col, data=df, palette="pastel")
C:\Users\priya\AppData\Local\Temp\ipykernel_62540\4232543248.py:39:
```

```
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.violinplot(y=col, data=df, palette="pastel")
C:\Users\priya\AppData\Local\Temp\ipykernel_62540\4232543248.py:39:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.violinplot(y=col, data=df, palette="pastel")
C:\Users\priya\AppData\Local\Temp\ipykernel_62540\4232543248.py:39:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.violinplot(y=col, data=df, palette="pastel")
```



```
plt.figure(figsize=(14, 6))
for i, col in enumerate(df.columns):
    plt.subplot(1, 5, i+1)
    sns.swarmplot(y=col, data=df, size=4, color='purple')
    plt.title(f'Swarm Plot: {col}')
plt.tight_layout()
plt.show()
```

```
c:\Users\priya\AppData\Local\Programs\Python\Python313\Lib\site-
packages\seaborn\categorical.py:3399: UserWarning: 23.0% of the points
cannot be placed; you may want to decrease the size of the markers or
```

```
use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\priya\AppData\Local\Programs\Python\Python313\Lib\site-
packages\seaborn\categorical.py:3399: UserWarning: 23.0% of the points
cannot be placed; you may want to decrease the size of the markers or
use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\priya\AppData\Local\Programs\Python\Python313\Lib\site-
packages\seaborn\categorical.py:3399: UserWarning: 23.0% of the points
cannot be placed; you may want to decrease the size of the markers or
use stripplot.
  warnings.warn(msg, UserWarning)
c:\Users\priya\AppData\Local\Programs\Python\Python313\Lib\site-
packages\seaborn\categorical.py:3399: UserWarning: 17.0% of the points
cannot be placed; you may want to decrease the size of the markers or
use stripplot.
  warnings.warn(msg, UserWarning)
```

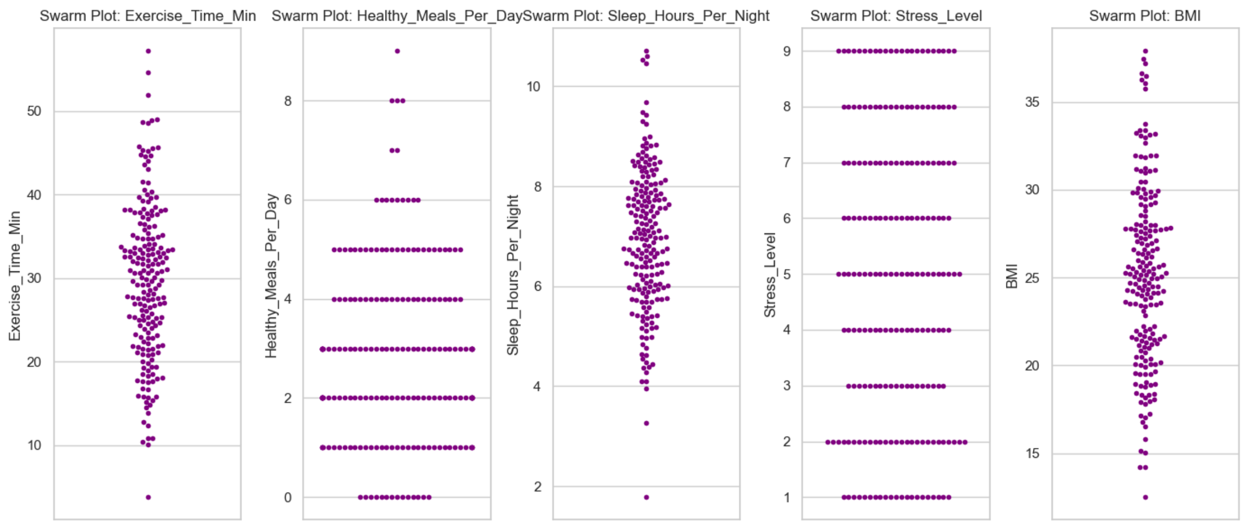


Figure of distribution of all features - five wellness indicators across the 200 patients in the dataset. Exercise time, sleep time, and BMI are roughly normally distributed, whereas healthy meals per day is right-skewed, reflecting that few people eat a high number of healthy meals. Stress level seems to be uniformly distributed, reflecting that the sample population experiences stress differently.

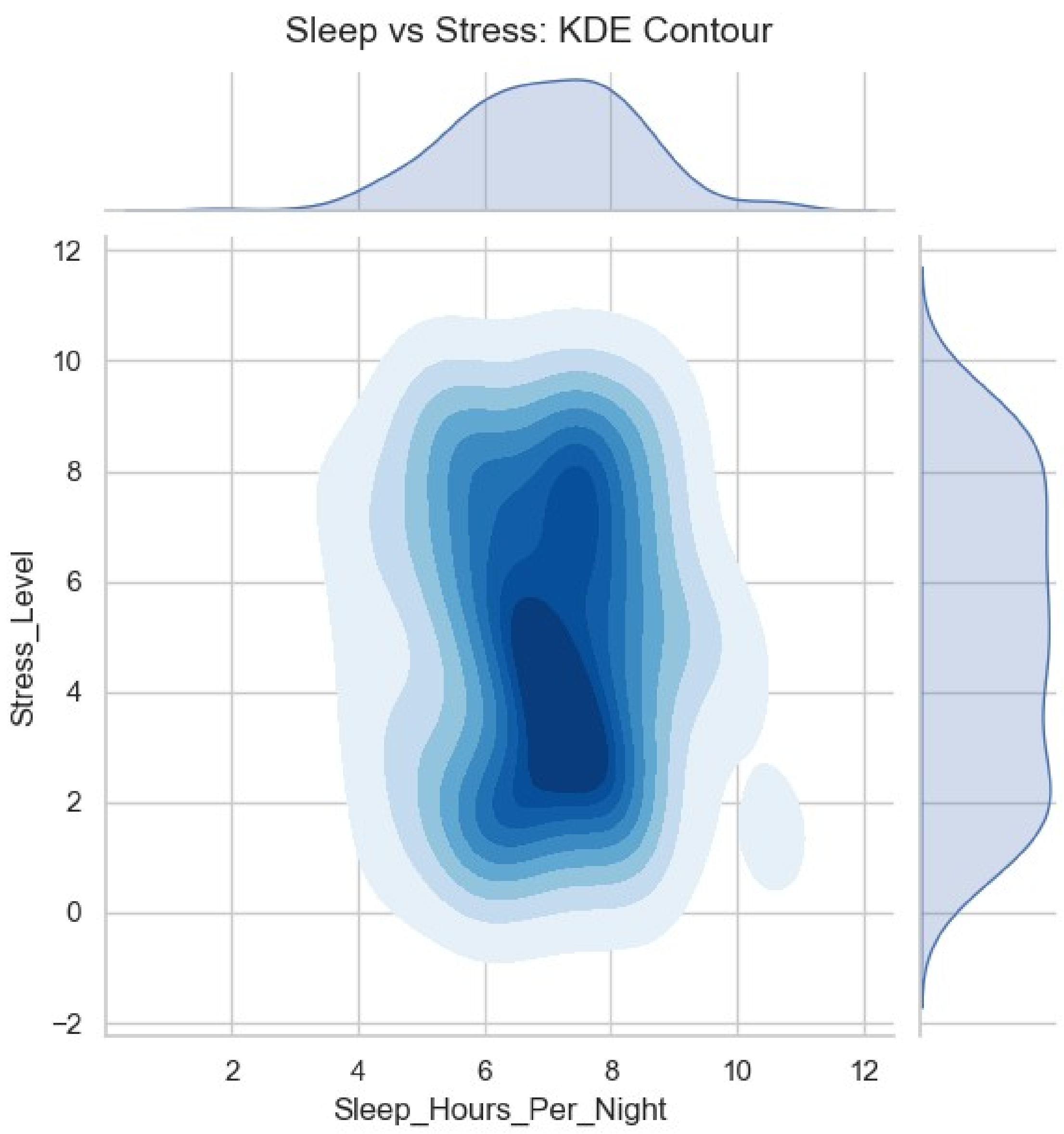
Figure of correlation heat map displays Pearson correlation coefficients between the five wellness variables. There are mostly weak correlations, with the strongest negative relationships being between stress level and BMI (r = -.13). Although exercise time, sleep time, and healthy meals per day have low correlations with each other, this suggests that one or more of these variables may be independent of one another or at least not related through behavior.

Figure of pairplot shows relationships between all health features with scatterplots kernel density estimates, and histograms. Most combinations of features don't have linear relationships, but BMI and exercise time have some density of pattern. The smooth contours indicate the density of observations, which provide evidence for locating potential clustering structures.

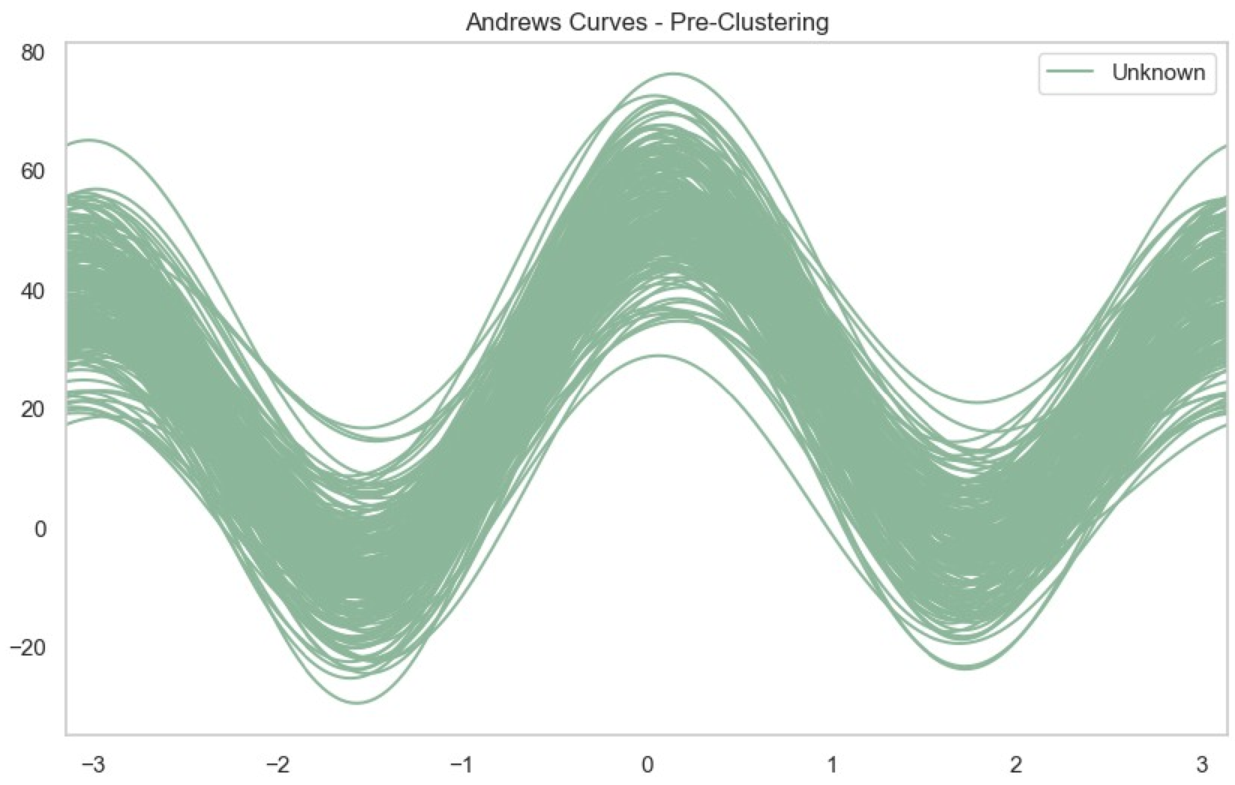
Violin plots show the distribution and density of each feature related to wellness using a boxplot that also incorporates a kernel density estimate. The inner white dot represents the median, while the overall shape portrays the spread of the data and skewness. The distributions related to exercise time and BMI tended to have moderate symmetry, while other features like stress level spread out more uniformly in one direction.

Swarn plot clearly reveals columned values of individual data points for each wellness feature, which highlights the points and the density and repetitions of certain values. The plots containing discrete variables like number of healthy meals per day and stress level also show stacking patterns of values due to how many times the values were repeated across the sample. The plots containing continuous variables like BMI and sleep duration contrast these stacking behaviors of discrete variables demonstrating its natural spread and where there is natural clustering around the value. The graphs also give a chance to examine outliers directly and what the distributions of individual records look like over the sampled measures.

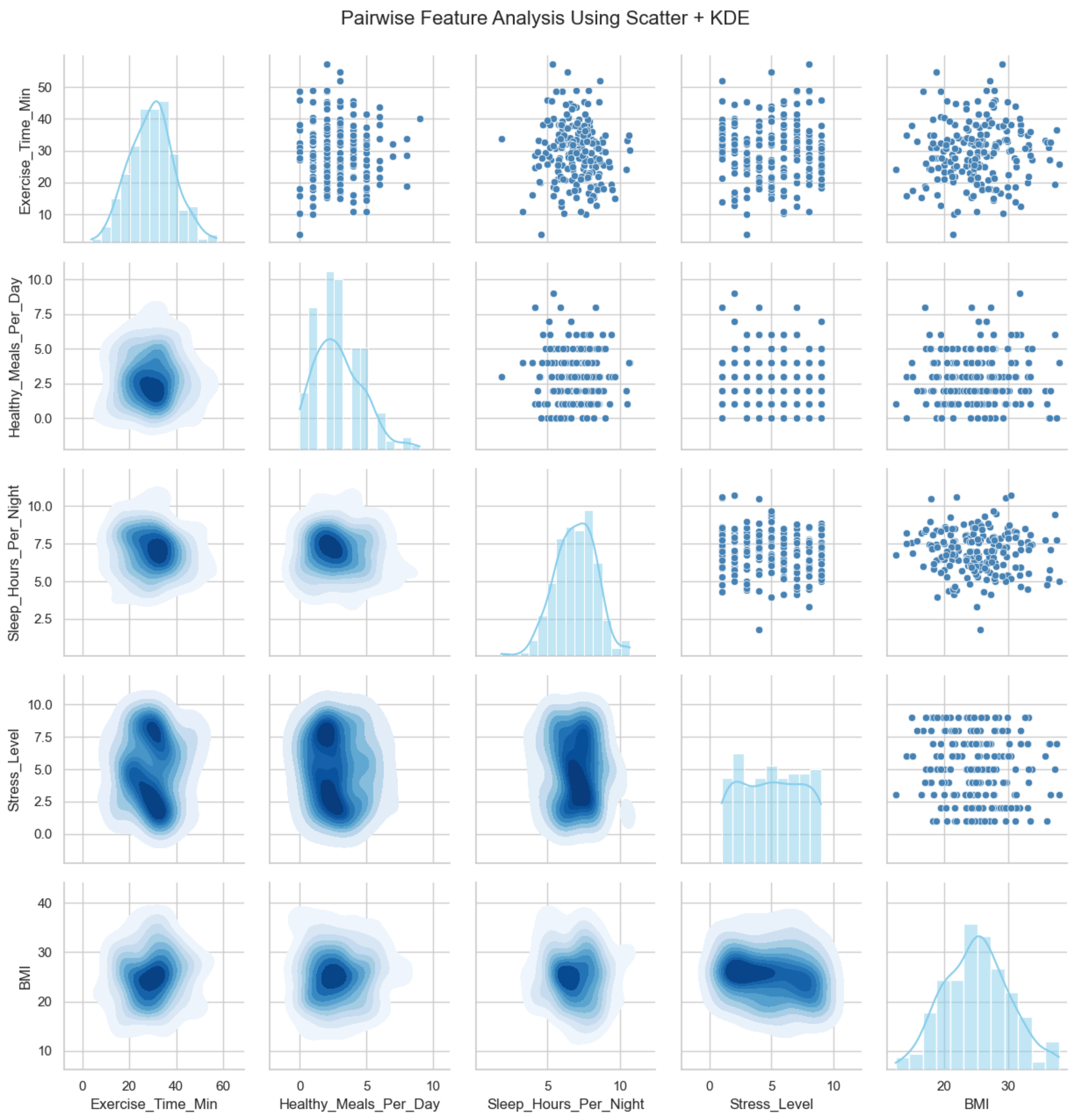
```python
# Example: Sleep vs Stress
sns.jointplot(data=df, x='Sleep_Hours_Per_Night', y='Stress_Level',
kind='kde', fill=True, cmap='Blues')
plt.suptitle("Sleep vs Stress: KDE Contour", y=1.02, fontsize=14)
plt.show()
```



```
from pandas.plotting import andrews_curves
df_copy = df.copy()
df_copy['Cluster'] = 'Unknown'  # Temporarily label before clustering
plt.figure(figsize=(10, 6))
andrews_curves(df_copy, 'Cluster')
plt.title("Andrews Curves - Pre-Clustering")
plt.show()
```



```python
import seaborn as sns
import matplotlib.pyplot as plt

# Create a PairGrid for full pairwise feature relationships
g = sns.PairGrid(df)

# Upper triangle: scatterplots
g.map_upper(sns.scatterplot, color='steelblue')

# Lower triangle: KDE density plots
g.map_lower(sns.kdeplot, cmap="Blues", fill=True)

# Diagonal: histograms with KDE
g.map_diag(sns.histplot, color='skyblue', kde=True)

# Add title
plt.suptitle("Pairwise Feature Analysis Using Scatter + KDE", y=1.02,
fontsize=16)
plt.show()
```



Standardization

Both K-Means clustering, DBSCAN clustering, and Principal Component Analysis (PCA) performs better if there are not any major differences in scale of the features. Thus, every variable was standardized Z-Score normalization using the StandardScaler from scikit-learn. The features, on average, were then rescaled and have a standard deviation of 1 and a mean of 0, meaning that every variable will contribute, equally, as part of the distance calculations used in clustering algorithms.

```
from sklearn.preprocessing import StandardScaler
```

```
# Create scaler instance
scaler = StandardScaler()

# Fit and transform the data
scaled_data = scaler.fit_transform(df)

# Convert back to DataFrame for readability
df_scaled = pd.DataFrame(scaled_data, columns=df.columns)

# Preview the scaled data
df_scaled.head()
```

```
   Exercise_Time_Min  Healthy_Meals_Per_Day  Sleep_Hours_Per_Night  \
0           0.578767               1.173447               0.482957
1          -0.104981               2.830078              -1.993156
2           0.741336               0.621237              -0.640956
3           1.683908              -1.035394               1.149993
4          -0.208235               0.069026               0.964166

   Stress_Level       BMI
0     -1.152351  1.565523
1      0.771441  0.418669
2     -1.537110 -0.271010
3      1.156199  0.923359
4     -0.767593  1.146154
```

Above table displays the first five observations of the dataset after Z-score standardization has been applied. Each variable was normalized to a mean of 0 and standard deviation of 1 using scikit-learn's StandardScaler - this step is important in order to avoid biasing distance-based models including K-Means, DBSCAN, and PCA.

This procedure, as outlined above, provided a solid preprocessing pipeline for ensuring all clustering outcome results were generalized and comparable across models and analytical stages.

Unsupervised Learning Techniques

Four unsupervised clustering algorithms were used to create clusters of the patient population based on the indicator variables describing wellness behaviors, including K-Means, Hierarchical Agglomerative Clustering, Gaussian Mixture Models (GMM) and DBSCAN. Each method was selected to complement inherent characteristics of the data quality and provide differing strengths and appropriateness for healthcare data clustering (Hernandez et al., 2024). K-Means is commonly utilized in health analytics due to its basic functionality and reduced computational load. K-Means is often used to recognize distinguishable behavior. (Alanazi, 2022). Hierarchical clustering was applied to determine possible sub-structures amongst the patients by providing a nested relational framework towards understanding behavior patterns with a dendrogram (Lu & Uddin, 2024). GMM provides probabilistic clustering and was needed in this data, as patient groups may be overlapped considering wellness indicators (Trezza et al., 2024).

Each model was applied to the standardized dataset without any dimensionality reduction to establish a baseline for clustering behavior. K-Means, Hierarchical and GMM were able to distinguish groups in the data though from different perspectives. The different algorithms facilitated crisp (K-Means, Hierarchical) and soft (GMM) clustering which can better inform interpretations in health contexts, where patient profiles are more often along continuum rather than assessed within predefined categories (Jayatilake & Ganegoda, 2021).

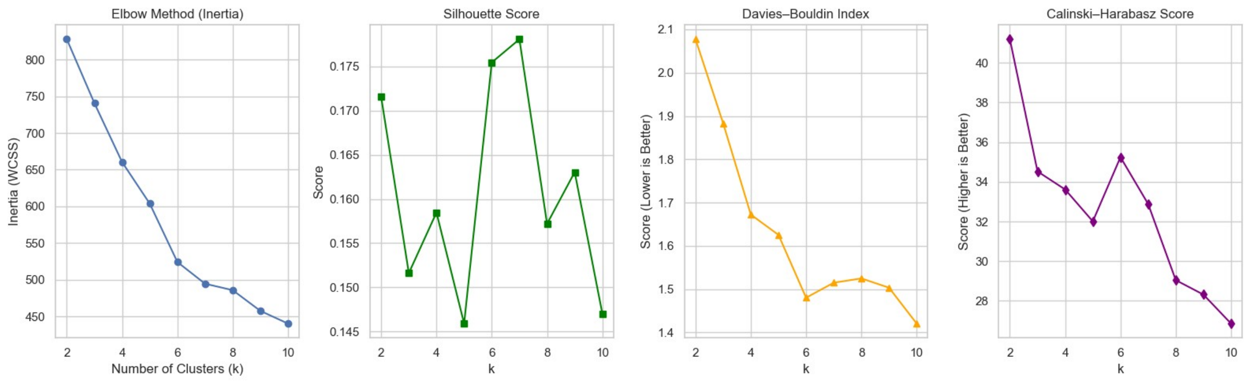
The DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm was evaluated, as it has the capacity to discover clusters of arbitrary shapes and does not require establishing the number of clusters beforehand (Massi et al., 2020). After using standard parameters, it was unsuccessful to identify more than one valid cluster, DBSCAN claimed most of the points were noise and contained one single dense region. Subsequently, it was attempted to tune it to more lenient settings, but still did not identify fewer than two clusters making it to be unusable for this dataset.

Therefore, DBSCAN was removed from further analysis. Ultimately, the lack of density-based clusters indicated that this approach to clustering would not be successful in the wellness dataset, potentially due to a more uniform distribution and a more compact clustering. K-Means, Hierarchical and GMM were retained for downstream comparison and interpretation of the cluster structure.

Results

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, davies_bouldin_score,
calinski_harabasz_score
import matplotlib.pyplot as plt

inertias = []
sil_scores = []
db_scores = []
ch_scores = []
k_range = range(2, 11)

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    labels = kmeans.fit_predict(df_scaled)

    inertias.append(kmeans.inertia_)
    sil_scores.append(silhouette_score(df_scaled, labels))
    db_scores.append(davies_bouldin_score(df_scaled, labels))
    ch_scores.append(calinski_harabasz_score(df_scaled, labels))

# Plot Elbow Curve (Inertia)
plt.figure(figsize=(16, 5))

plt.subplot(1, 4, 1)
plt.plot(k_range, inertias, marker='o')
plt.title('Elbow Method (Inertia)')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia (WCSS)')
```

```
# Plot Silhouette Scores
plt.subplot(1, 4, 2)
plt.plot(k_range, sil_scores, marker='s', color='green')
plt.title('Silhouette Score')
plt.xlabel('k')
plt.ylabel('Score')

# Plot Davies–Bouldin Scores
plt.subplot(1, 4, 3)
plt.plot(k_range, db_scores, marker='^', color='orange')
plt.title('Davies–Bouldin Index')
plt.xlabel('k')
plt.ylabel('Score (Lower is Better)')

# Plot Calinski–Harabasz Scores
plt.subplot(1, 4, 4)
plt.plot(k_range, ch_scores, marker='d', color='purple')
plt.title('Calinski–Harabasz Score')
plt.xlabel('k')
plt.ylabel('Score (Higher is Better)')

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

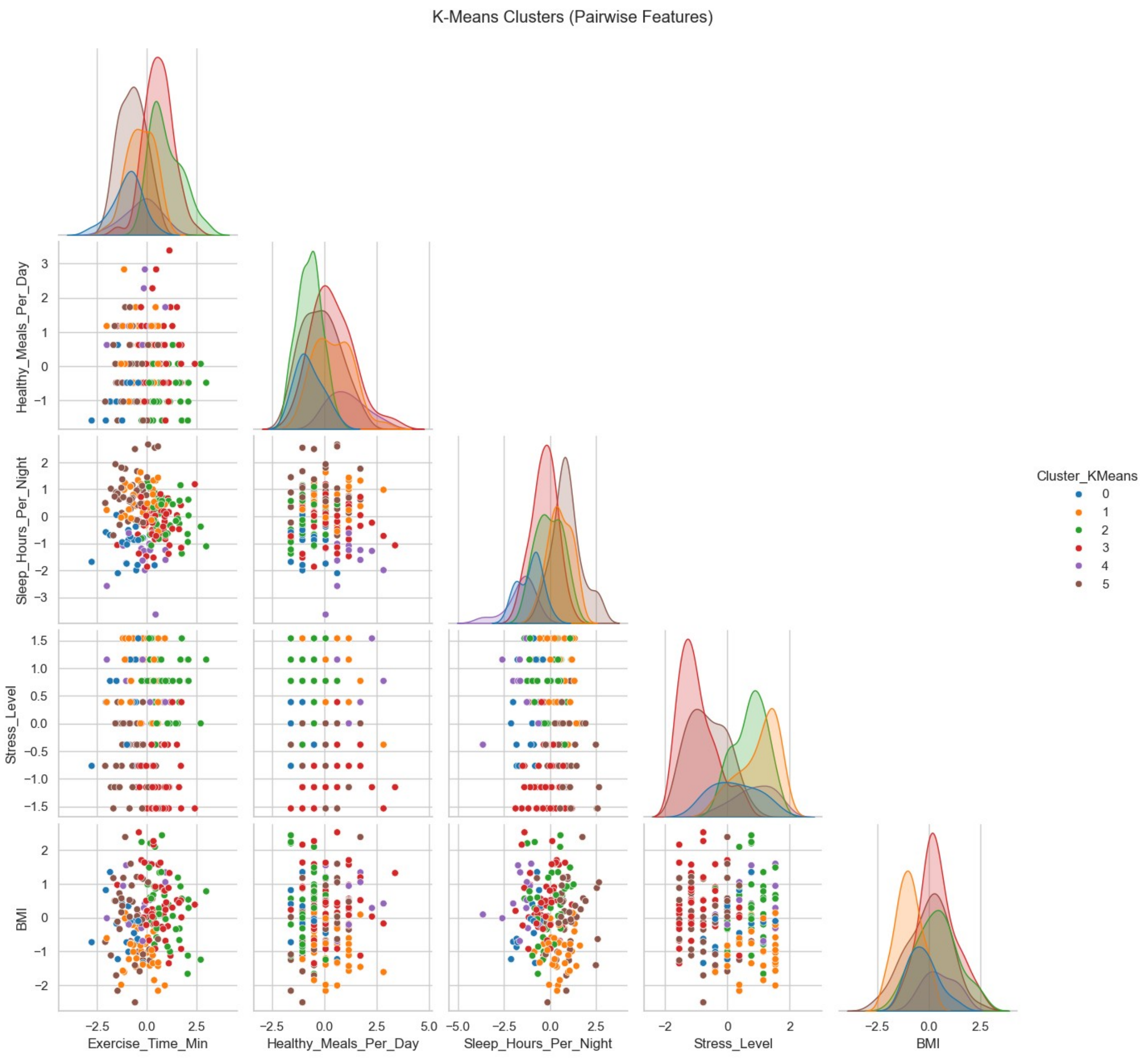


```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.metrics import silhouette_score, davies_bouldin_score,
calinski_harabasz_score

from pandas.plotting import andrews_curves
```

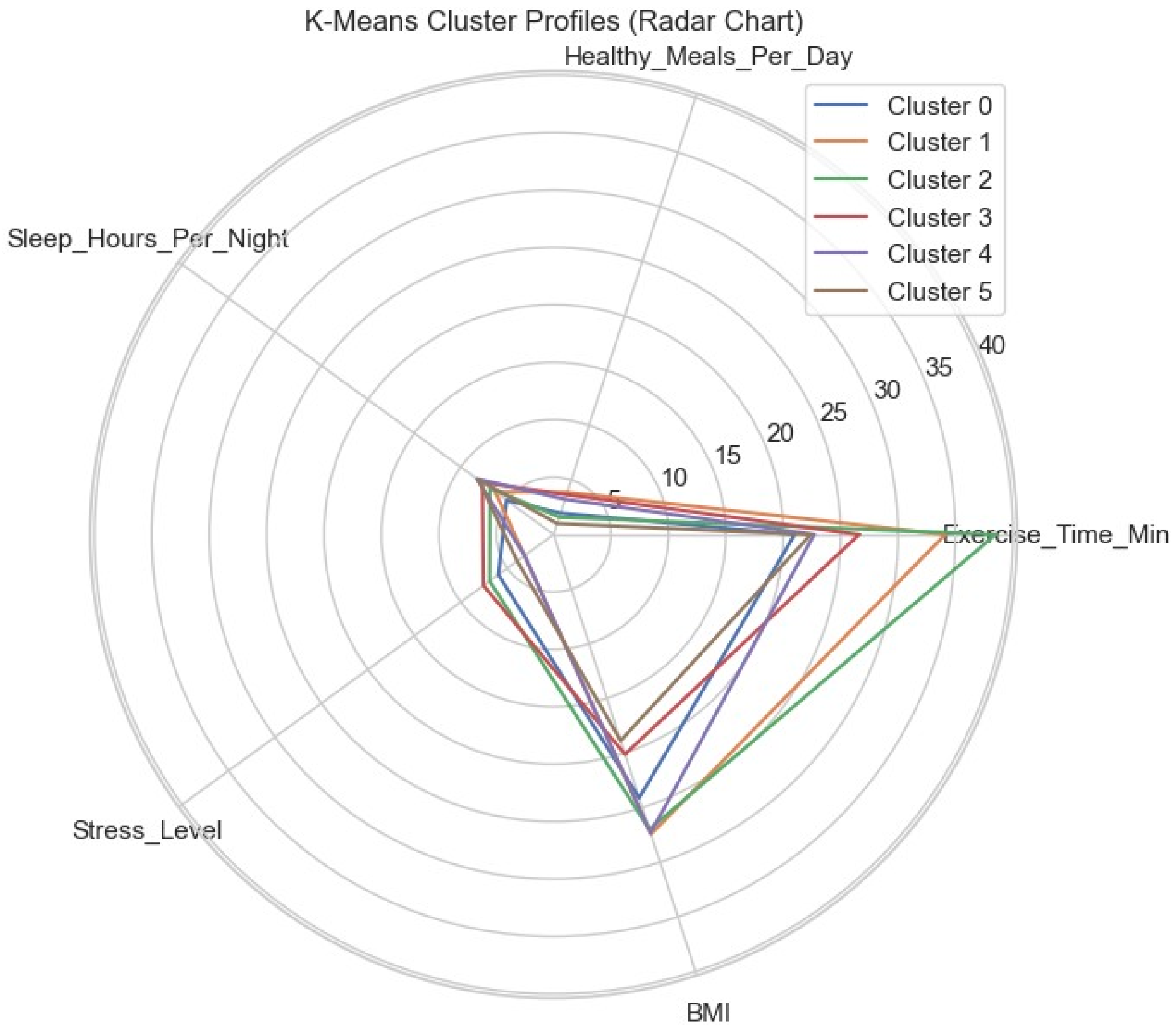
```python
# Fit KMeans
kmeans = KMeans(n_clusters=6, random_state=42)
df_scaled['Cluster_KMeans'] = kmeans.fit_predict(df_scaled)

# Evaluation
print("K-Means Silhouette Score:",
silhouette_score(df_scaled.drop('Cluster_KMeans', axis=1),
df_scaled['Cluster_KMeans']))
print("K-Means Davies–Bouldin Index:",
davies_bouldin_score(df_scaled.drop('Cluster_KMeans', axis=1),
df_scaled['Cluster_KMeans']))
print("K-Means Calinski–Harabasz Score:",
calinski_harabasz_score(df_scaled.drop('Cluster_KMeans', axis=1),
df_scaled['Cluster_KMeans']))
```

```
K-Means Silhouette Score: 0.16404353334102212
K-Means Davies–Bouldin Index: 1.5233403205501836
K-Means Calinski–Harabasz Score: 31.479772066495034
```

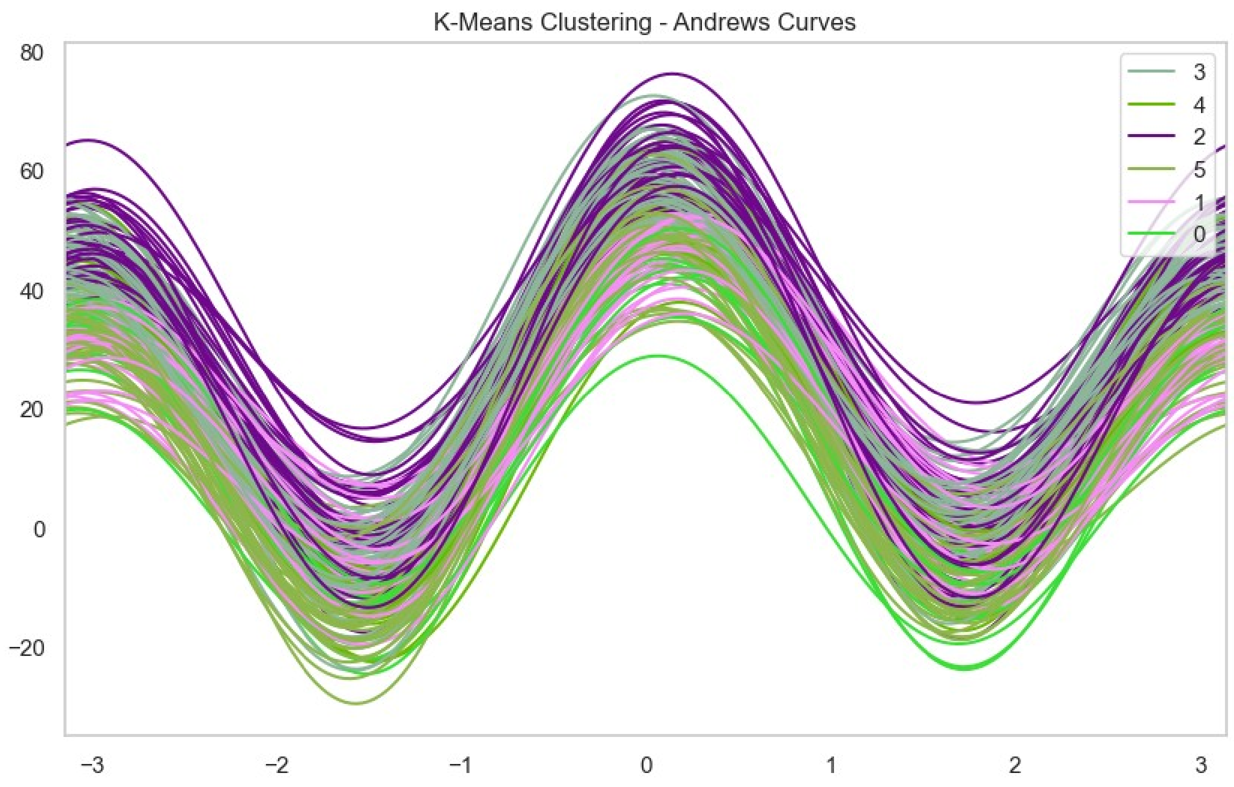
```python
sns.pairplot(df_scaled, hue='Cluster_KMeans', palette='tab10',
corner=True)
plt.suptitle("K-Means Clusters (Pairwise Features)", y=1.02)
plt.show()
```



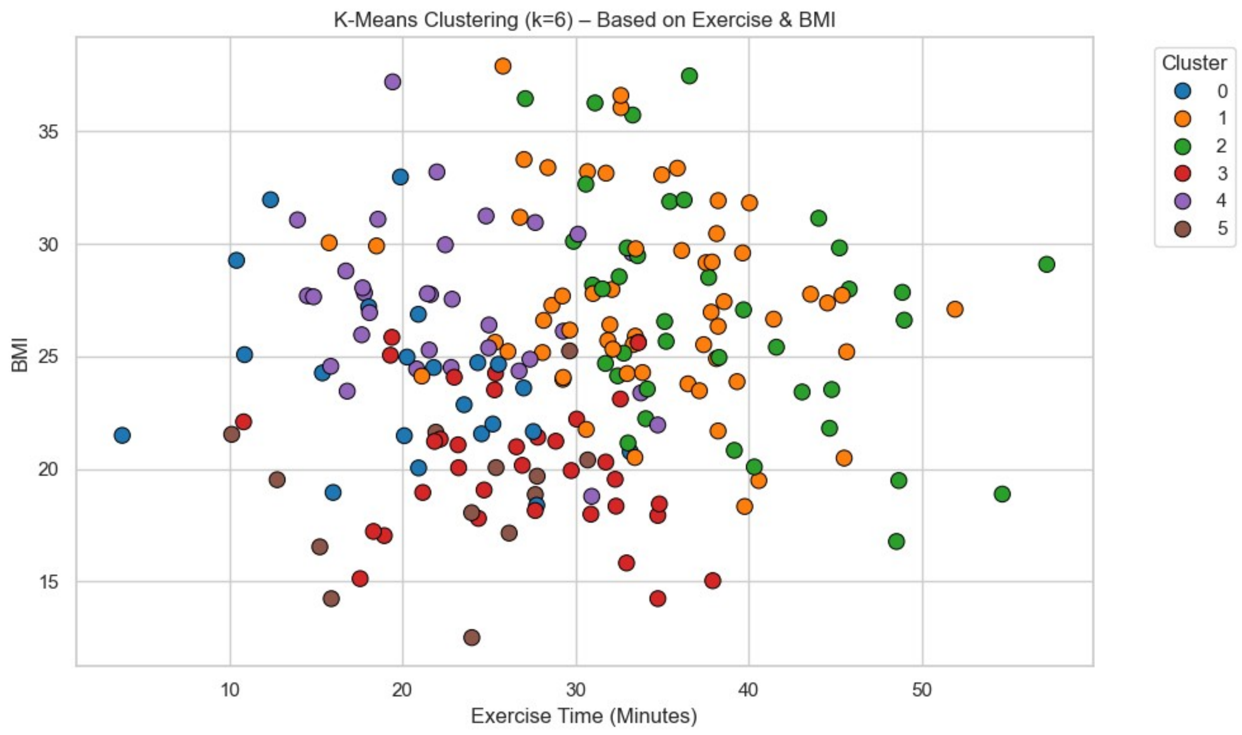
```python
# Calculate cluster means
cluster_profiles = df.groupby('Cluster_KMeans').mean()

# Radar plot setup
features = cluster_profiles.columns.tolist()
N = len(features)

angles = np.linspace(0, 2 * np.pi, N, endpoint=False).tolist()
angles += angles[:1]

plt.figure(figsize=(10, 7))
for idx, row in cluster_profiles.iterrows():
    values = row.tolist()
    values += values[:1]
    plt.polar(angles, values, label=f'Cluster {idx}')
plt.xticks(angles[:-1], features)
plt.title("K-Means Cluster Profiles (Radar Chart)")
plt.legend()
plt.show()
```



```
df_temp = df.copy()
df_temp['Cluster_KMeans'] = df_scaled['Cluster_KMeans']
plt.figure(figsize=(10, 6))
andrews_curves(df_temp, 'Cluster_KMeans')
plt.title("K-Means Clustering - Andrews Curves")
plt.show()
```



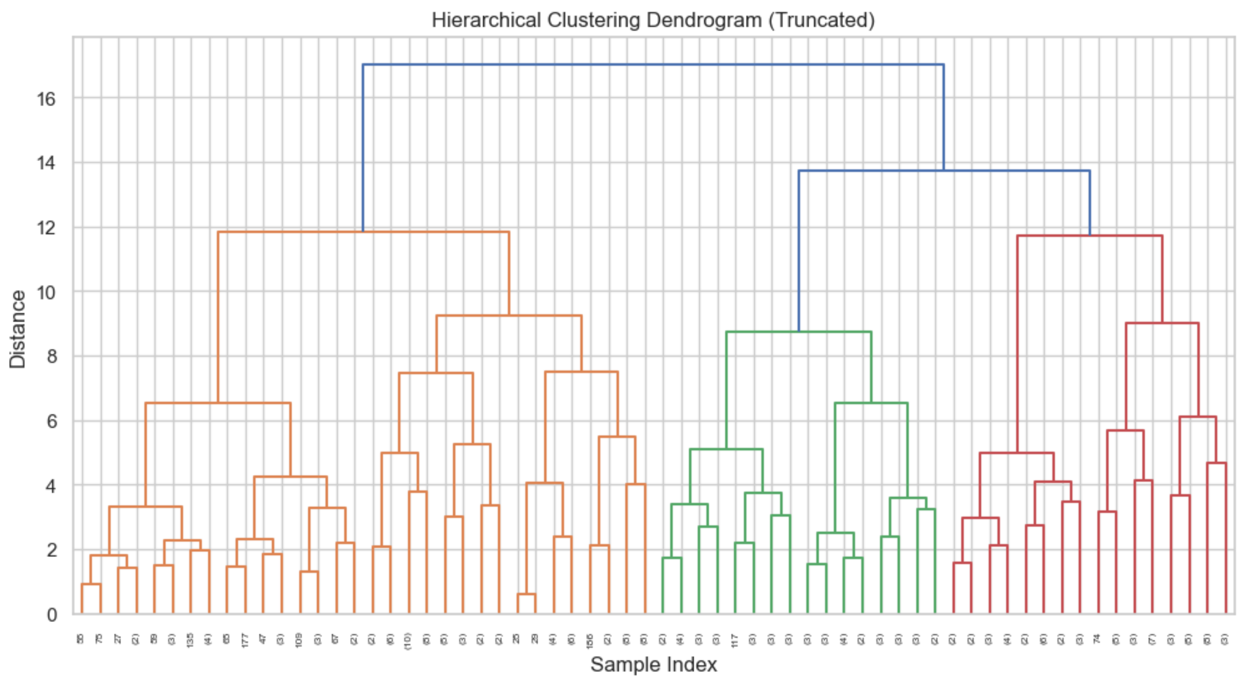
```python
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.scatterplot(
    x=df['Exercise_Time_Min'],
    y=df['BMI'],
    hue=df['Cluster_KMeans'],
    palette='tab10',
    s=80,
    edgecolor='black'
)
plt.title("K-Means Clustering (k=6) – Based on Exercise & BMI")
plt.xlabel("Exercise Time (Minutes)")
plt.ylabel("BMI")
plt.legend(title='Cluster', bbox_to_anchor=(1.05, 1), loc='upper
left')
plt.grid(True)
plt.tight_layout()
plt.show()
```



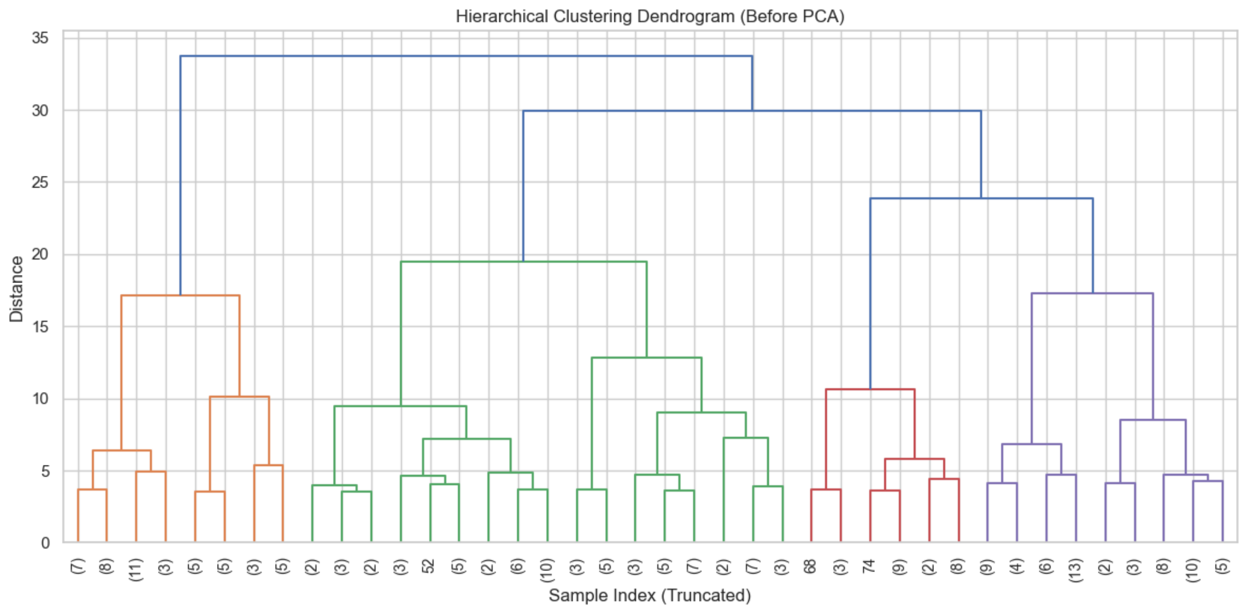
```
agglo = AgglomerativeClustering(n_clusters=6)
df_scaled['Cluster_Agglo'] =
agglo.fit_predict(df_scaled.drop(columns=['Cluster_KMeans']))

# Evaluation
print("Agglomerative Silhouette Score:",
silhouette_score(df_scaled.drop(['Cluster_KMeans', 'Cluster_Agglo'],
axis=1), df_scaled['Cluster_Agglo']))
print("Agglomerative Davies–Bouldin Index:",
davies_bouldin_score(df_scaled.drop(['Cluster_KMeans',
'Cluster_Agglo'], axis=1), df_scaled['Cluster_Agglo']))
print("Agglomerative Calinski–Harabasz Score:",
calinski_harabasz_score(df_scaled.drop(['Cluster_KMeans',
'Cluster_Agglo'], axis=1), df_scaled['Cluster_Agglo']))
```

```
Agglomerative Silhouette Score: 0.12756292890549378
Agglomerative Davies–Bouldin Index: 1.7066416828134539
Agglomerative Calinski–Harabasz Score: 28.16456858482108
```

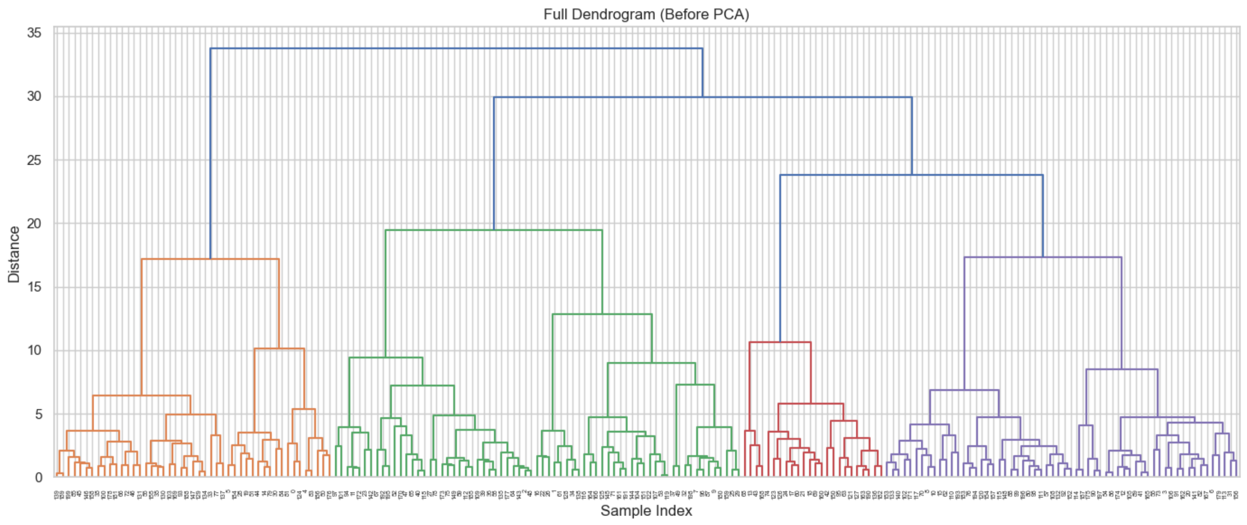
```
from scipy.cluster.hierarchy import dendrogram, linkage

linked = linkage(df_scaled.drop(['Cluster_KMeans', 'Cluster_Agglo'],
axis=1), method='ward')

plt.figure(figsize=(12, 6))
dendrogram(linked, truncate_mode='level', p=5)
plt.title("Hierarchical Clustering Dendrogram (Truncated)")
plt.xlabel("Sample Index")
plt.ylabel("Distance")
plt.show()
```



```
from scipy.cluster.hierarchy import linkage, dendrogram
import matplotlib.pyplot as plt

# Create linkage matrix using Ward method
linkage_matrix = linkage(df_scaled, method='ward')

# Plot dendrogram
plt.figure(figsize=(12, 6))
dendrogram(linkage_matrix, truncate_mode='lastp', p=40,
leaf_rotation=90, leaf_font_size=10)
plt.title("Hierarchical Clustering Dendrogram (Before PCA)")
plt.xlabel("Sample Index (Truncated)")
plt.ylabel("Distance")
plt.tight_layout()
plt.show()
```



```
plt.figure(figsize=(14, 6))
dendrogram(linkage_matrix)
plt.title("Full Dendrogram (Before PCA)")
plt.xlabel("Sample Index")
plt.ylabel("Distance")
plt.tight_layout()
plt.show()
```



```
from sklearn.cluster import DBSCAN

# Try with slightly relaxed parameters
dbscan = DBSCAN(eps=1.5, min_samples=4)
df_scaled['Cluster_DBSCAN'] =
dbscan.fit_predict(df_scaled.drop(columns=['Cluster_KMeans',
'Cluster_Agglo']))

# Count clusters
n_clusters = len(set(df_scaled['Cluster_DBSCAN'])) - (1 if -1 in
df_scaled['Cluster_DBSCAN'].values else 0)
print(f"Number of clusters found by DBSCAN (excluding noise):
{n_clusters}")

# Evaluate only if at least 2 clusters
if n_clusters >= 2:
    mask = df_scaled['Cluster_DBSCAN'] != -1  # ignore noise points
    X_db = df_scaled[mask].drop(columns=['Cluster_KMeans',
'Cluster_Agglo', 'Cluster_DBSCAN'])
    y_db = df_scaled[mask]['Cluster_DBSCAN']

    print("DBSCAN Silhouette Score:", silhouette_score(X_db, y_db))
    print("DBSCAN Davies–Bouldin Index:", davies_bouldin_score(X_db,
y_db))
    print("DBSCAN Calinski–Harabasz Score:",
calinski_harabasz_score(X_db, y_db))
else:
    print("DBSCAN could not form at least 2 clusters. Try tuning eps
and min_samples.")
```

```
Number of clusters found by DBSCAN (excluding noise): 1
DBSCAN could not form at least 2 clusters. Try tuning eps and
min_samples.
```

```
# Try more lenient parameters
dbscan = DBSCAN(eps=2.0, min_samples=3)
df_scaled['Cluster_DBSCAN'] =
dbscan.fit_predict(df_scaled.drop(columns=['Cluster_KMeans',
'Cluster_Agglo']))

n_clusters = len(set(df_scaled['Cluster_DBSCAN'])) - (1 if -1 in
df_scaled['Cluster_DBSCAN'].values else 0)
print(f"Number of clusters found by DBSCAN (excluding noise):
{n_clusters}")

if n_clusters >= 2:
    mask = df_scaled['Cluster_DBSCAN'] != -1
    X_db = df_scaled[mask].drop(columns=['Cluster_KMeans',
'Cluster_Agglo', 'Cluster_DBSCAN'])
    y_db = df_scaled[mask]['Cluster_DBSCAN']

    print("DBSCAN Silhouette Score:", silhouette_score(X_db, y_db))
    print("DBSCAN Davies–Bouldin Index:", davies_bouldin_score(X_db,
y_db))
    print("DBSCAN Calinski–Harabasz Score:",
calinski_harabasz_score(X_db, y_db))
```

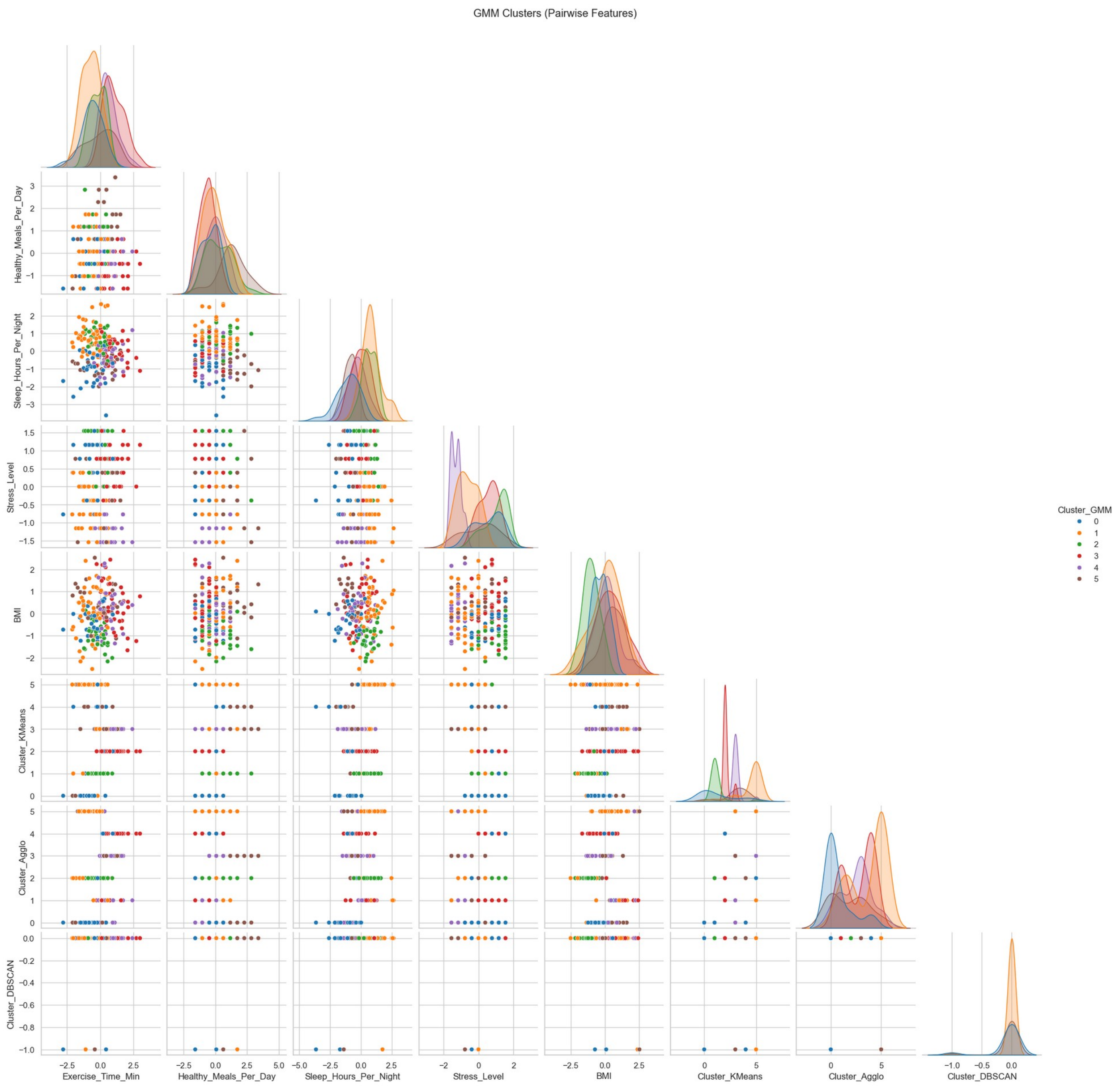
```python
else:
    print("Still fewer than 2 clusters — consider further tuning or
choosing another algorithm.")
```

```
Number of clusters found by DBSCAN (excluding noise): 1
Still fewer than 2 clusters — consider further tuning or choosing
another algorithm.
```

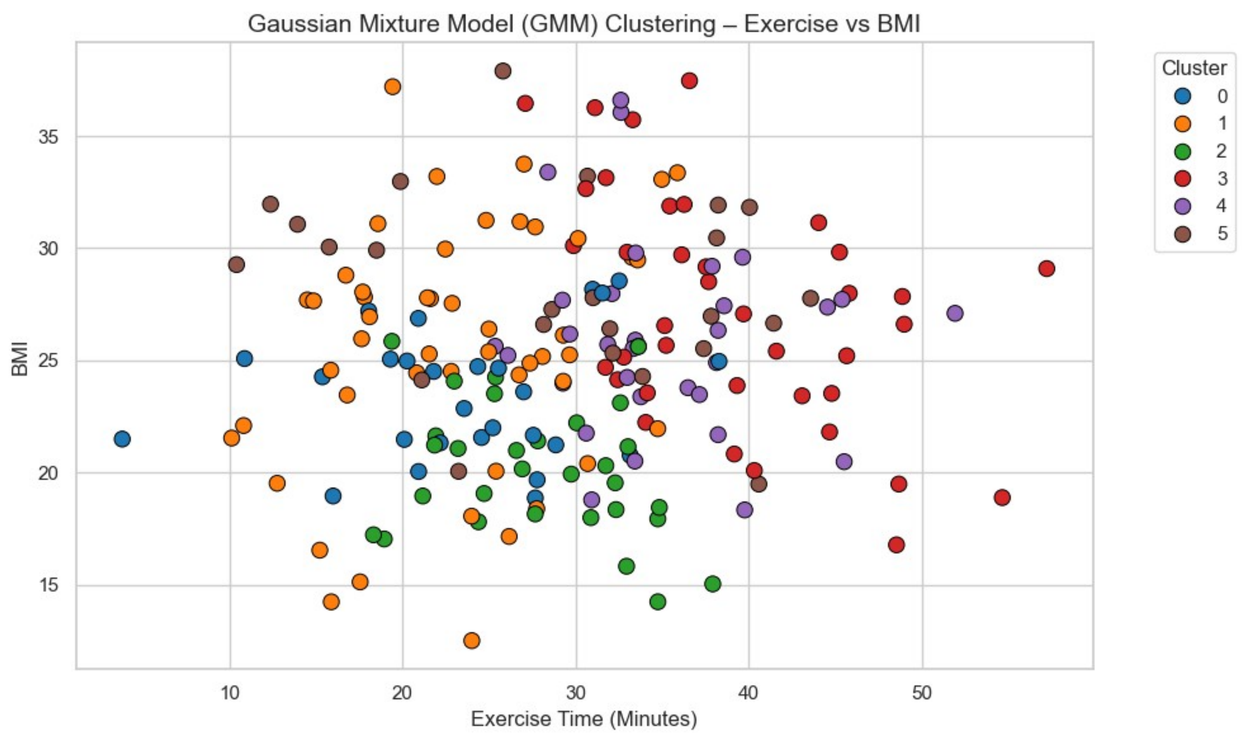
```python
from sklearn.mixture import GaussianMixture

# Fit GMM with 6 components (same as K-Means for comparison)
gmm = GaussianMixture(n_components=6, random_state=42)
df_scaled['Cluster_GMM'] =
gmm.fit_predict(df_scaled.drop(columns=['Cluster_KMeans',
'Cluster_Agglo', 'Cluster_DBSCAN']))

# Evaluation
print("GMM Silhouette Score:",
silhouette_score(df_scaled.drop(columns=['Cluster_KMeans',
'Cluster_Agglo', 'Cluster_DBSCAN', 'Cluster_GMM']),
df_scaled['Cluster_GMM']))
print("GMM Davies–Bouldin Index:",
davies_bouldin_score(df_scaled.drop(columns=['Cluster_KMeans',
'Cluster_Agglo', 'Cluster_DBSCAN', 'Cluster_GMM']),
df_scaled['Cluster_GMM']))
print("GMM Calinski–Harabasz Score:",
calinski_harabasz_score(df_scaled.drop(columns=['Cluster_KMeans',
'Cluster_Agglo', 'Cluster_DBSCAN', 'Cluster_GMM']),
df_scaled['Cluster_GMM']))
```

```
GMM Silhouette Score: 0.13191027665802493
GMM Davies–Bouldin Index: 1.636199385129544
GMM Calinski–Harabasz Score: 27.88383075490278
```

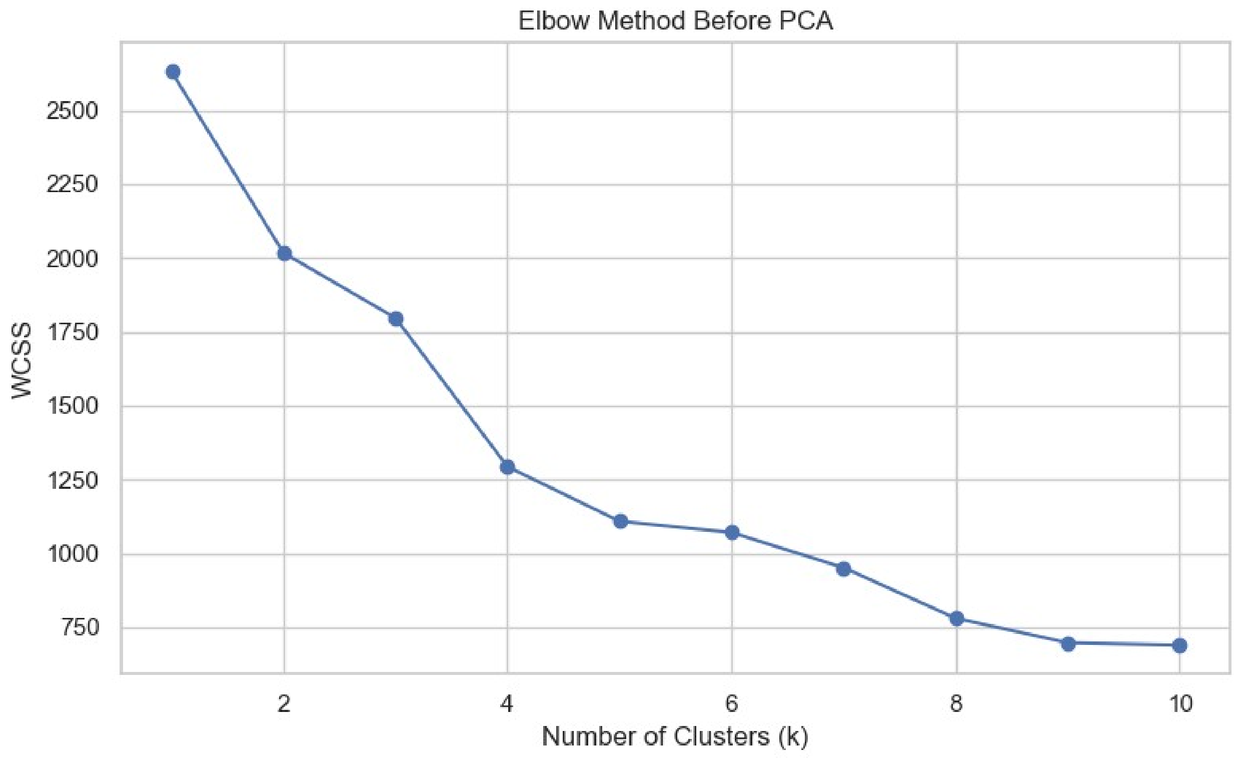
```python
sns.pairplot(df_scaled, hue='Cluster_GMM', palette='tab10',
corner=True)
plt.suptitle("GMM Clusters (Pairwise Features)", y=1.02)
plt.show()
```



```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.scatterplot(
    x=df['Exercise_Time_Min'],
    y=df['BMI'],
    hue=df_scaled['Cluster_GMM'],
    palette='tab10',
    s=80,
    edgecolor='black'
)
plt.title("Gaussian Mixture Model (GMM) Clustering – Exercise vs BMI",
fontsize=14)
plt.xlabel("Exercise Time (Minutes)")
plt.ylabel("BMI")
plt.legend(title='Cluster', bbox_to_anchor=(1.05, 1), loc='upper
left')
plt.grid(True)
plt.tight_layout()
plt.show()
```



```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# WCSS for raw standardized data
wcss_before = []
K = range(1, 11)

for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(df_scaled)  # df_scaled = standardized data before PCA
    wcss_before.append(kmeans.inertia_)

# Plot Elbow for Before PCA
plt.figure(figsize=(8, 5))
plt.plot(K, wcss_before, marker='o')
plt.title('Elbow Method Before PCA')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('WCSS')
plt.grid(True)
plt.tight_layout()
plt.show()
```



```
from sklearn.cluster import KMeans

# Ensure df_scaled is your standardized dataset (no cluster columns)
X_scaled = df_scaled.drop(columns=['Cluster_KMeans', 'Cluster_Agglo',
'Cluster_DBSCAN', 'Cluster_GMM'], errors='ignore')

# KMeans before PCA
kmeans_before = KMeans(n_clusters=3, random_state=42)
kmeans_before.fit(X_scaled)
wcss_kmeans_before = kmeans_before.inertia_  # WCSS
print("WCSS (K-Means - Before PCA):", round(wcss_kmeans_before, 4))

WCSS (K-Means - Before PCA): 740.4663

import numpy as np

def calculate_wcss(data, labels):
    wcss = 0
    for label in np.unique(labels):
        cluster_points = data[labels == label]
        if len(cluster_points) == 0:
            continue
        centroid = np.mean(cluster_points, axis=0)
        wcss += np.sum((cluster_points - centroid) ** 2)
    return wcss

from sklearn.cluster import AgglomerativeClustering
from sklearn.mixture import GaussianMixture
import numpy as np

# Function to calculate WCSS
def calculate_wcss(data, labels):
    wcss = 0
    for label in np.unique(labels):
        cluster_points = data[labels == label]
        if len(cluster_points) == 0:
            continue
        centroid = np.mean(cluster_points, axis=0)
        wcss += np.sum((cluster_points - centroid) ** 2)
    return wcss

# Convert df_scaled to NumPy array if not already
X_scaled = df_scaled.drop(columns=['Cluster_KMeans', 'Cluster_Agglo',
'Cluster_DBSCAN', 'Cluster_GMM'], errors='ignore').to_numpy()

# Agglomerative Clustering (Before PCA)
agg_model = AgglomerativeClustering(n_clusters=3)
labels_agg = agg_model.fit_predict(X_scaled)
wcss_agg = calculate_wcss(X_scaled, labels_agg)
print("WCSS (Agglomerative - Before PCA):", round(wcss_agg, 4))

# GMM (Before PCA)
gmm_model = GaussianMixture(n_components=3, random_state=42)
labels_gmm = gmm_model.fit_predict(X_scaled)
wcss_gmm = calculate_wcss(X_scaled, labels_gmm)
print("WCSS (GMM - Before PCA):", round(wcss_gmm, 4))
```

```
WCSS (Agglomerative - Before PCA): 760.6833
WCSS (GMM - Before PCA): 759.8581
```

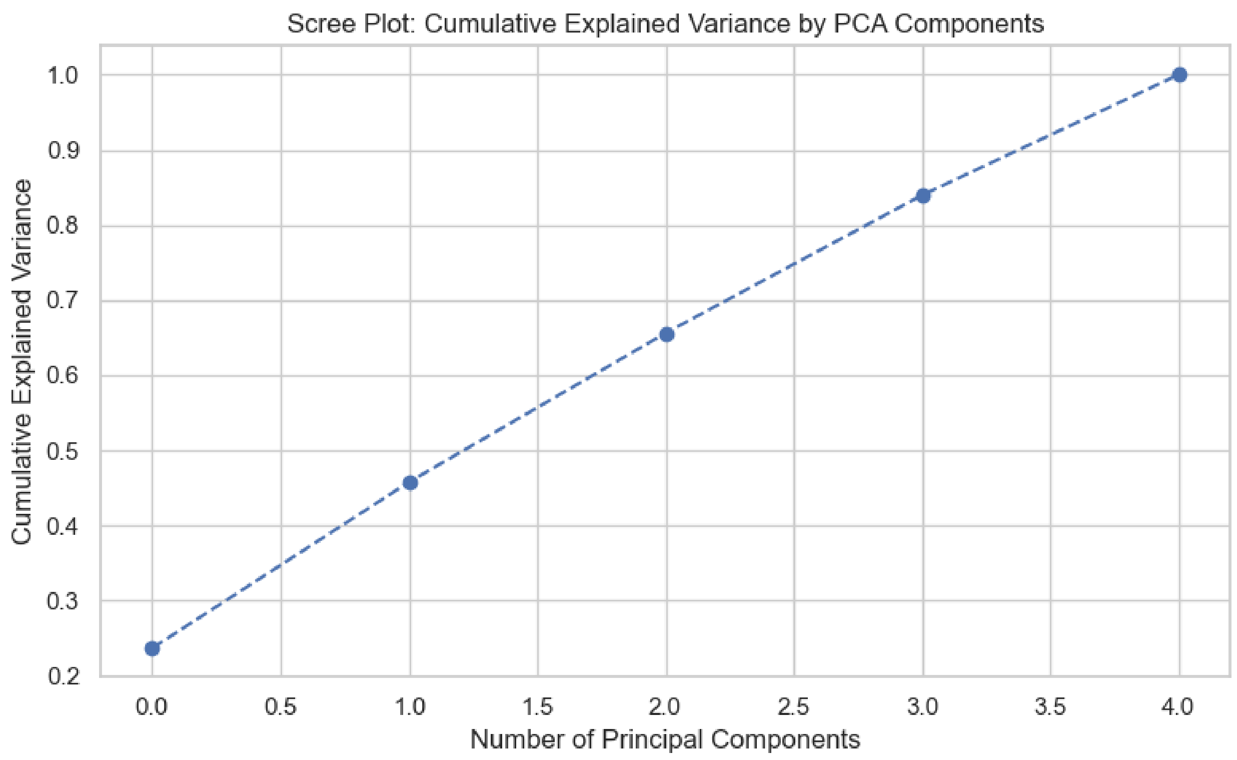
Dimensionality Reduction: Principal Component Analysis (PCA)

The correlation heatmap of the dataset indicating that multicollinearity amongst features was limited, PCA (Principal Component Analysis) was used as a way to improve clustering performance and visualization. But in an unsupervised learning context, PCA does not just limited to multicollinearity, it also provides a way of representing the data in an proper space where treatment of features amongst clusters is less complicated, and separation in a clustering form which becomes much sharper, especially in many dimensions (Lu & Uddin, 2024). Even in a multicollinearity free feature context, defining natural grouping of samples can still be obscured by the interplay of variations due to noise, scaling differences, or redundancy in contributions to variance.

In the case of patient wellness profiling, where even minor or subtle relationships between even features can make a difference, PCA can be very advantageous because it allows to get meaningful information into fewer dimensions while minimizing noise and keeping the right information or structure (Trezza et al., 2024). Therefore, it was impactful where clustering-based approach such as K-Means or GMM was sensitive. In this sense, by bringing the data into its top principal components, it was aimed to increase the interpretability of clusters and facilitate visual inspection of patient groups, a critical step towards actionable health segmentations.

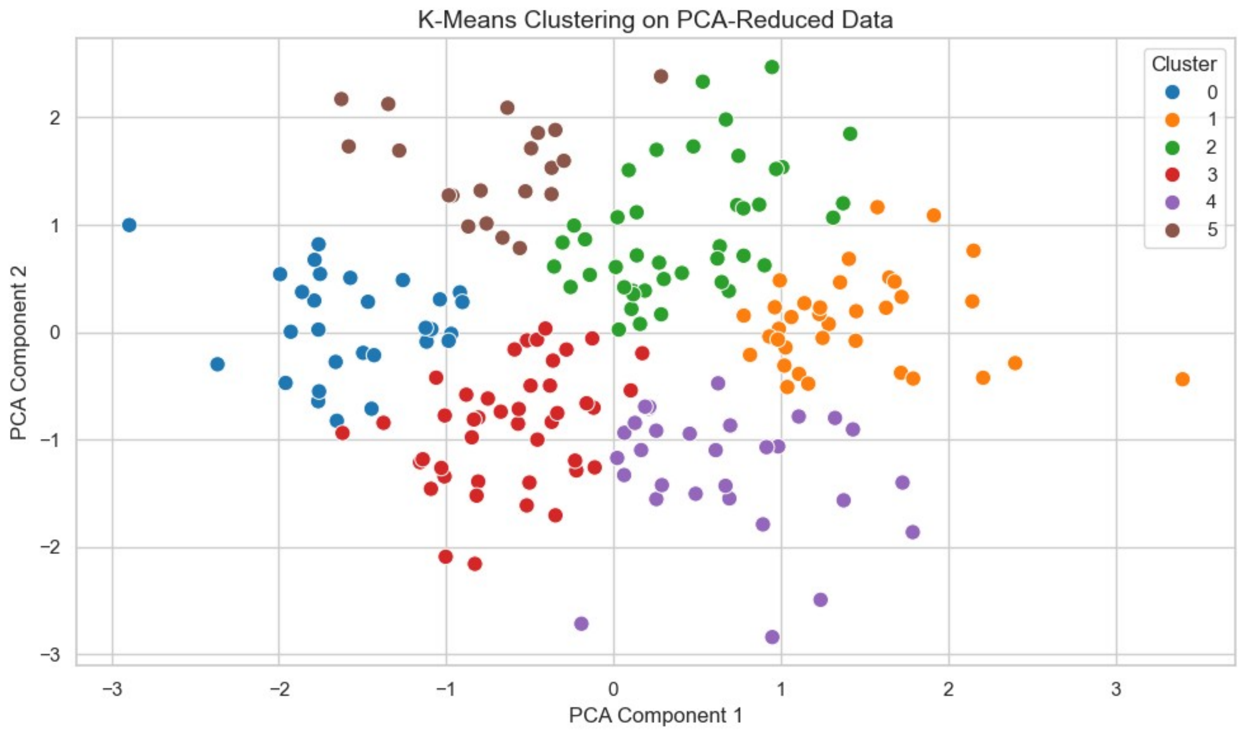
Once the dataset had been scaled, PCA was performed to reduce the five original features to five uncorrelated principal components. Each principal component explains a specified portion of the total variance in the data. While dimensionality reduction was the goal, it was essential to retain as much of the information as possible. The first two components represented approximately 46% of the total variance, and the five components explained 100 of the total variance. The table below explains the variance associated with each component.

```python
from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt

# PCA with all components
pca = PCA()
pca_data = pca.fit_transform(df_scaled.drop(columns=['Cluster_KMeans', 
'Cluster_Agglo', 'Cluster_DBSCAN', 'Cluster_GMM']))

# Scree plot (Explained Variance)
plt.figure(figsize=(8, 5))
plt.plot(np.cumsum(pca.explained_variance_ratio_), marker='o', 
linestyle='--')
plt.title("Scree Plot: Cumulative Explained Variance by PCA 
Components")
plt.xlabel("Number of Principal Components")
plt.ylabel("Cumulative Explained Variance")
plt.grid(True)
plt.tight_layout()
plt.show()
```

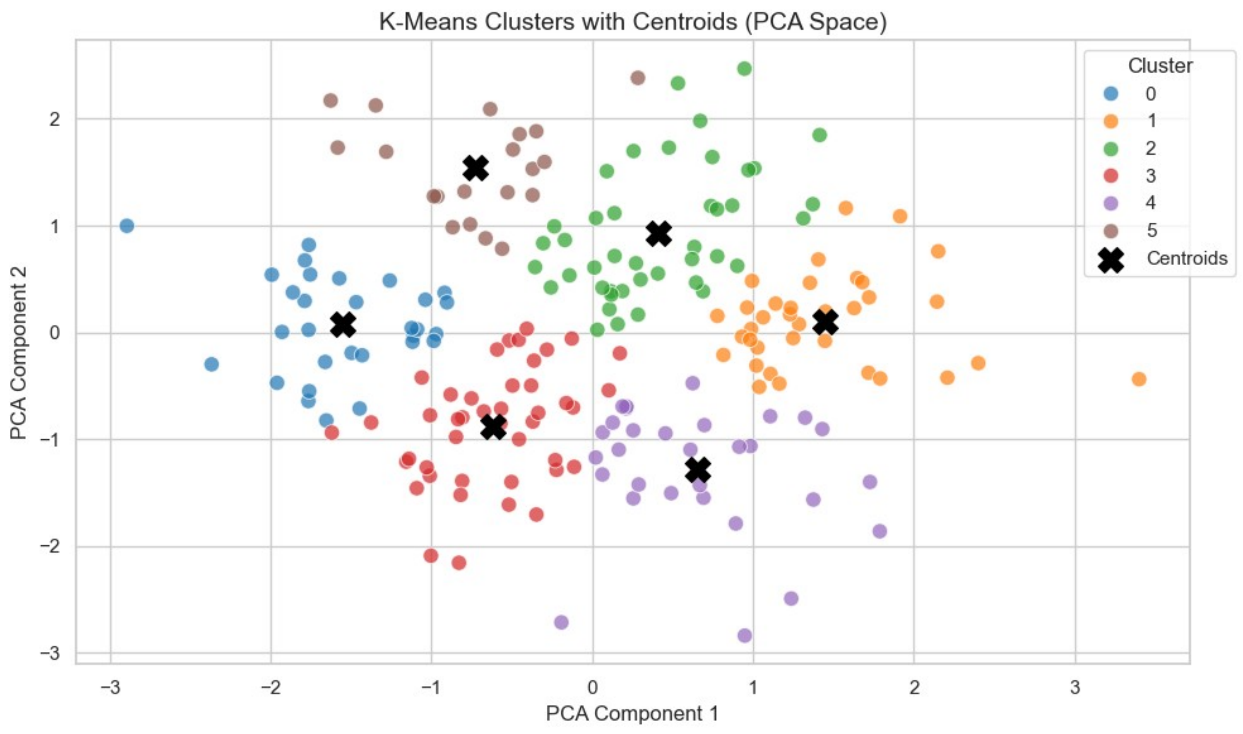


```
# Keep just first 2 PCA components
pca_2 = PCA(n_components=2)
pca_result =
pca_2.fit_transform(df_scaled.drop(columns=['Cluster_KMeans',
'Cluster_Agglo', 'Cluster_DBSCAN', 'Cluster_GMM']))

# Add PCA components to a new DataFrame
df_pca = pd.DataFrame(pca_result, columns=['PCA1', 'PCA2'])

# Perform PCA again with all components
pca_full = PCA()
pca_full.fit(df_scaled.drop(columns=['Cluster_KMeans',
'Cluster_Agglo', 'Cluster_DBSCAN', 'Cluster_GMM']))

# Create DataFrame for explained variance
pca_summary = pd.DataFrame({
    'Principal Component': [f'PC{i+1}' for i in
range(len(pca_full.explained_variance_ratio_))],
    'Explained Variance Ratio': pca_full.explained_variance_ratio_,
    'Cumulative Variance':
np.cumsum(pca_full.explained_variance_ratio_)
})

# Round for readability
pca_summary = pca_summary.round(4)
```

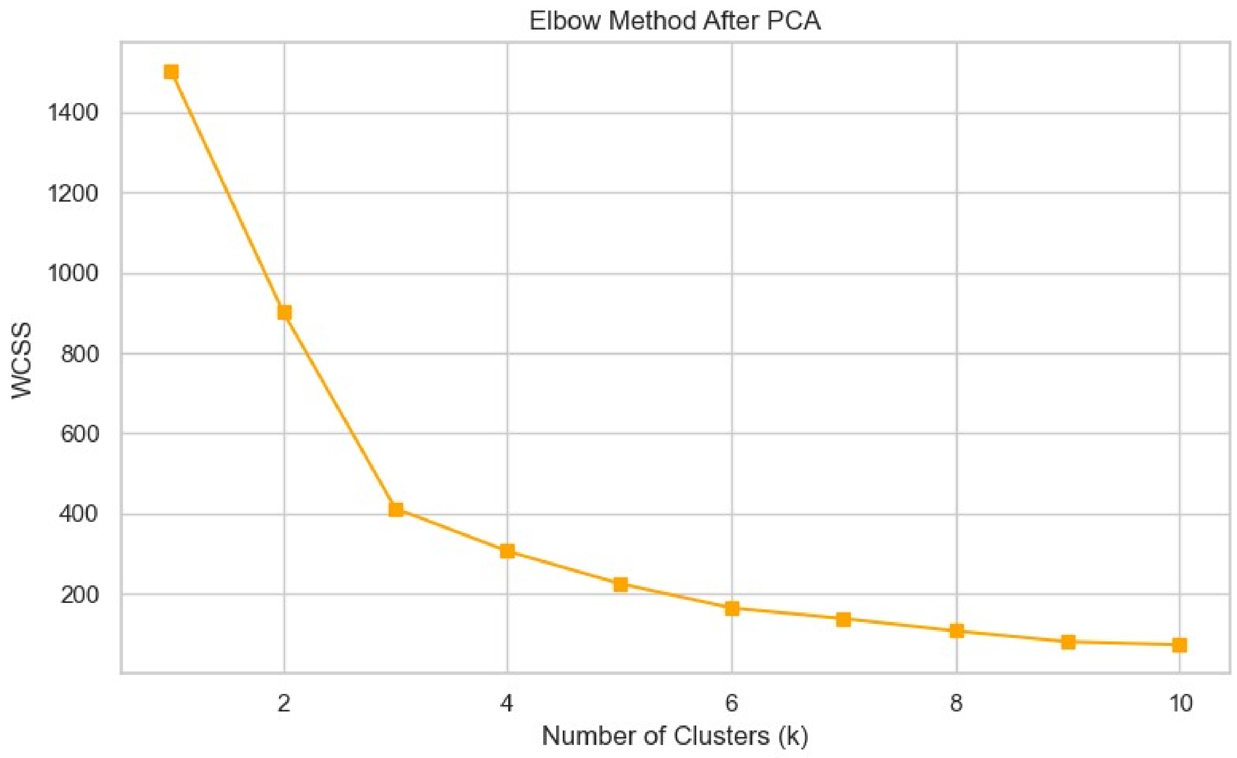
```
# Display
pca_summary

  Principal Component  Explained Variance Ratio  Cumulative Variance
0                 PC1                    0.2369               0.2369
1                 PC2                    0.2208               0.4577
2                 PC3                    0.1983               0.6560
3                 PC4                    0.1836               0.8397
4                 PC5                    0.1603               1.0000

# K-Means on PCA-reduced data
kmeans_pca = KMeans(n_clusters=6, random_state=42)
df_pca['Cluster_KMeans_PCA'] = kmeans_pca.fit_predict(df_pca)

# Visualize
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_pca, x='PCA1', y='PCA2',
hue='Cluster_KMeans_PCA', palette='tab10', s=80)
plt.title("K-Means Clustering on PCA-Reduced Data", fontsize=14)
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.legend(title='Cluster')
plt.grid(True)
plt.tight_layout()
plt.show()
```



```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_pca, x='PCA1', y='PCA2',
hue='Cluster_KMeans_PCA', palette='tab10', s=80, alpha=0.7)

# Plot centroids
centroids = kmeans_pca.cluster_centers_
plt.scatter(centroids[:, 0], centroids[:, 1], c='black', s=200,
marker='X', label='Centroids')

plt.title("K-Means Clusters with Centroids (PCA Space)", fontsize=14)
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.legend(title='Cluster', bbox_to_anchor=(1.05, 1))
plt.grid(True)
plt.tight_layout()
plt.show()
```



```
# AfterPCA
from sklearn.decomposition import PCA

pca = PCA(n_components=2)  # or n_components=5 if using full
df_pca = pca.fit_transform(df_scaled)

# WCSS for PCA-transformed data
wcss_after = []
for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(df_pca)
    wcss_after.append(kmeans.inertia_)

# Plot Elbow for After PCA
plt.figure(figsize=(8, 5))
plt.plot(K, wcss_after, marker='s', color='orange')
plt.title('Elbow Method After PCA')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('WCSS')
plt.grid(True)
plt.tight_layout()
plt.show()
```

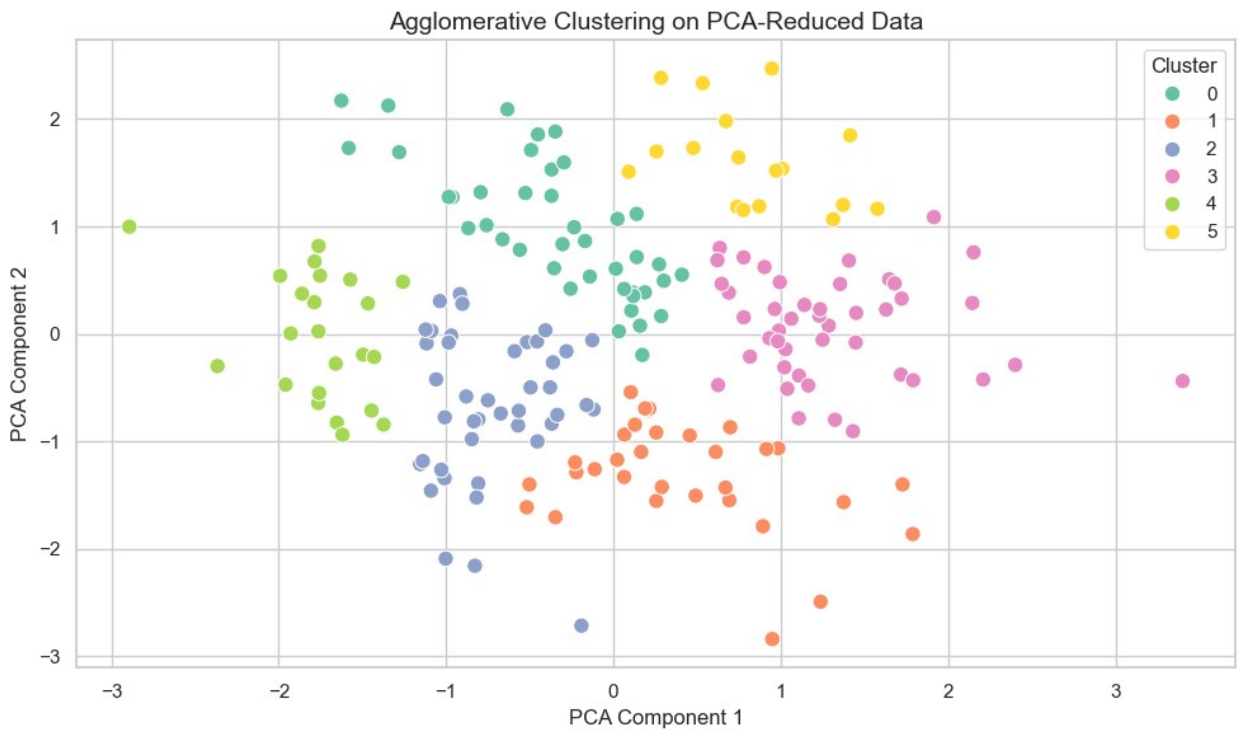


```
# Assuming df_pca is your PCA-transformed data
kmeans_after = KMeans(n_clusters=3, random_state=42)
kmeans_after.fit(df_pca)
wcss_kmeans_after = kmeans_after.inertia_
print("WCSS (K-Means - After PCA):", round(wcss_kmeans_after, 4))
```

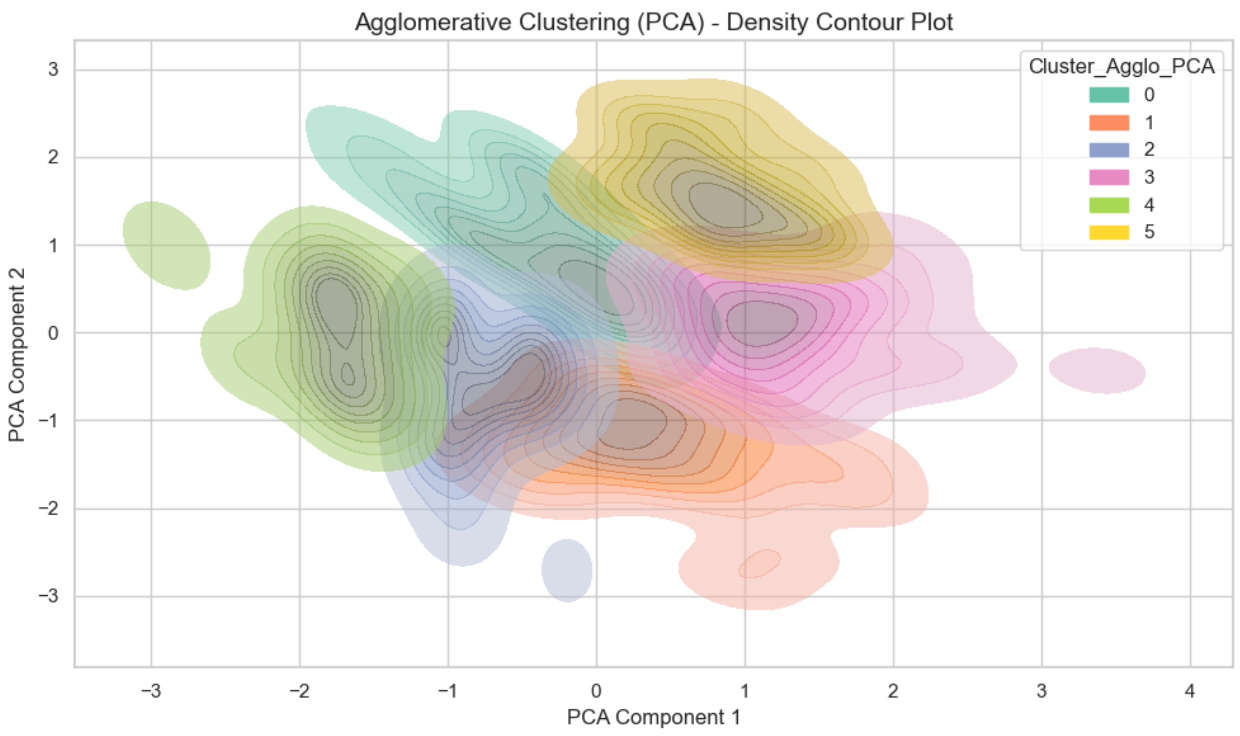
```
WCSS (K-Means - After PCA): 412.78
```

```
# Agglomerative Clustering on PCA
agglo_pca = AgglomerativeClustering(n_clusters=6)
df_pca['Cluster_Agglo_PCA'] = agglo_pca.fit_predict(df_pca[['PCA1',
'PCA2']])
```

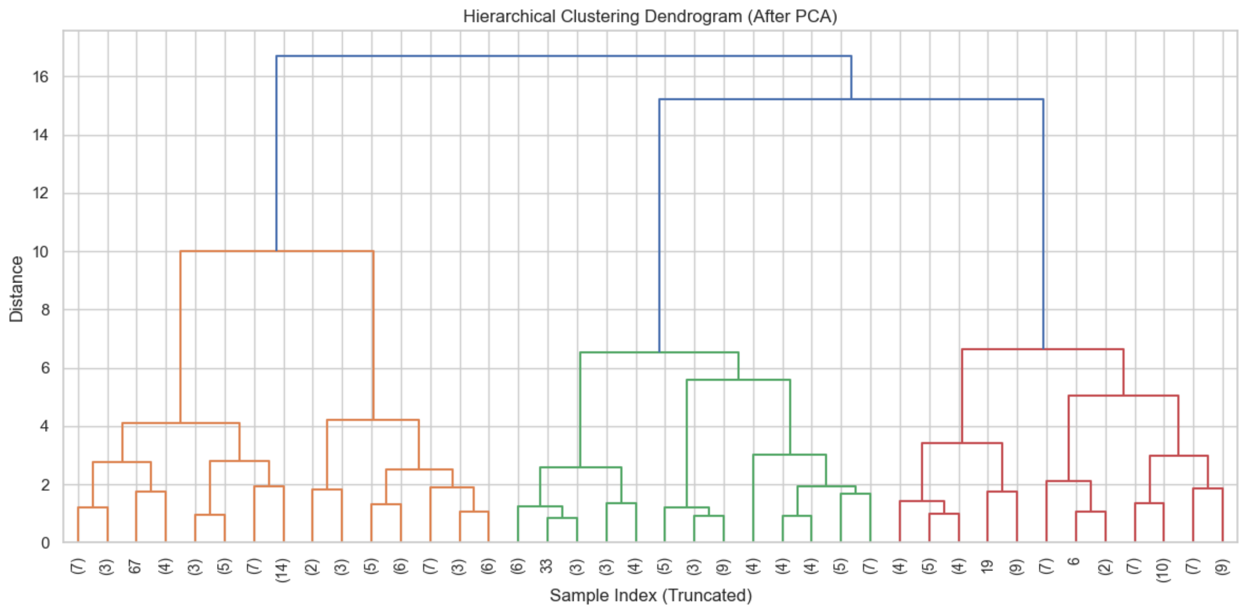
```python
# Visualize
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_pca, x='PCA1', y='PCA2',
hue='Cluster_Agglo_PCA', palette='Set2', s=80)
plt.title("Agglomerative Clustering on PCA-Reduced Data", fontsize=14)
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.legend(title='Cluster')
plt.grid(True)
plt.tight_layout()
plt.show()
```



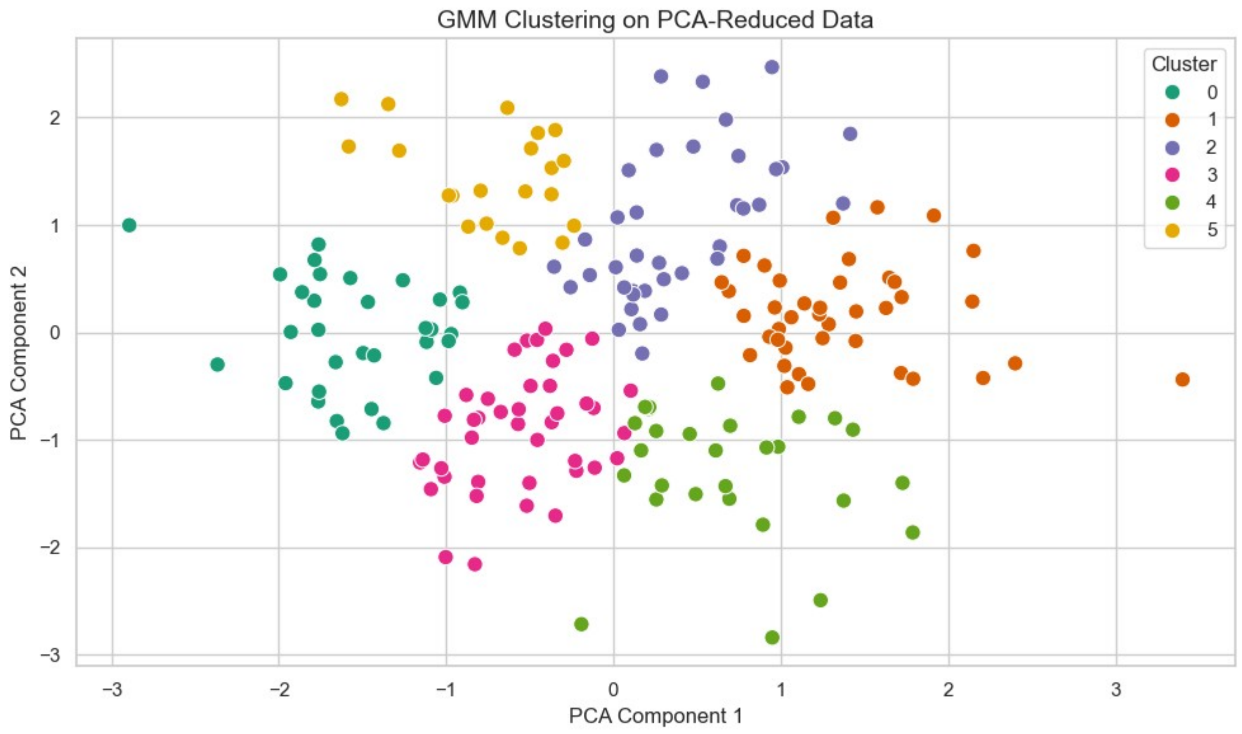
```python
plt.figure(figsize=(10, 6))
sns.kdeplot(
    data=df_pca, x='PCA1', y='PCA2',
    hue='Cluster_Agglo_PCA',
    fill=True,
    common_norm=False,
    alpha=0.5,
    palette='Set2'
)
plt.title("Agglomerative Clustering (PCA) - Density Contour Plot",
fontsize=14)
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.grid(True)
plt.tight_layout()
plt.show()
```



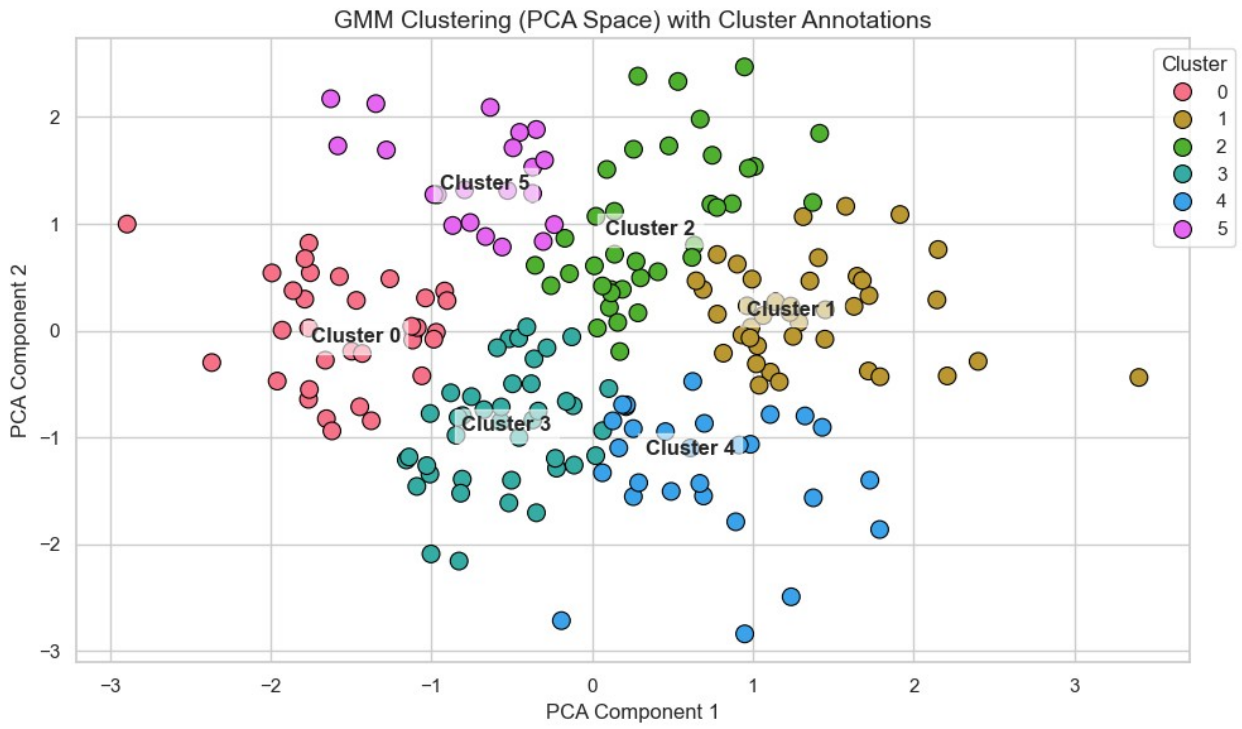
```
# Create linkage matrix on PCA-reduced data
linkage_pca = linkage(df_pca[['PCA1', 'PCA2']], method='ward')

# Plot PCA dendrogram
plt.figure(figsize=(12, 6))
dendrogram(linkage_pca, truncate_mode='lastp', p=40, leaf_rotation=90,
leaf_font_size=10)
plt.title("Hierarchical Clustering Dendrogram (After PCA)")
plt.xlabel("Sample Index (Truncated)")
plt.ylabel("Distance")
plt.tight_layout()
plt.show()
```



```python
# GMM on PCA data
gmm_pca = GaussianMixture(n_components=6, random_state=42)
df_pca['Cluster_GMM_PCA'] = gmm_pca.fit_predict(df_pca[['PCA1',
'PCA2']])

# Visualize
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_pca, x='PCA1', y='PCA2',
hue='Cluster_GMM_PCA', palette='Dark2', s=80)
plt.title("GMM Clustering on PCA-Reduced Data", fontsize=14)
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.legend(title='Cluster')
plt.grid(True)
plt.tight_layout()
plt.show()
```



```
plt.figure(figsize=(10, 6))
palette = sns.color_palette("husl",
len(df_pca['Cluster_GMM_PCA'].unique()))
sns.scatterplot(data=df_pca, x='PCA1', y='PCA2',
hue='Cluster_GMM_PCA', palette=palette, s=100, edgecolor='black')

# Annotate cluster centers with numbers
gmm_means = gmm_pca.means_
for idx, (x, y) in enumerate(gmm_means):
    plt.text(x, y, f'Cluster {idx}', fontsize=12, weight='bold',
ha='center', va='center', bbox=dict(facecolor='white', alpha=0.6))

plt.title("GMM Clustering (PCA Space) with Cluster Annotations",
fontsize=14)
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.legend(title='Cluster', bbox_to_anchor=(1.05, 1))
plt.tight_layout()
plt.show()
```

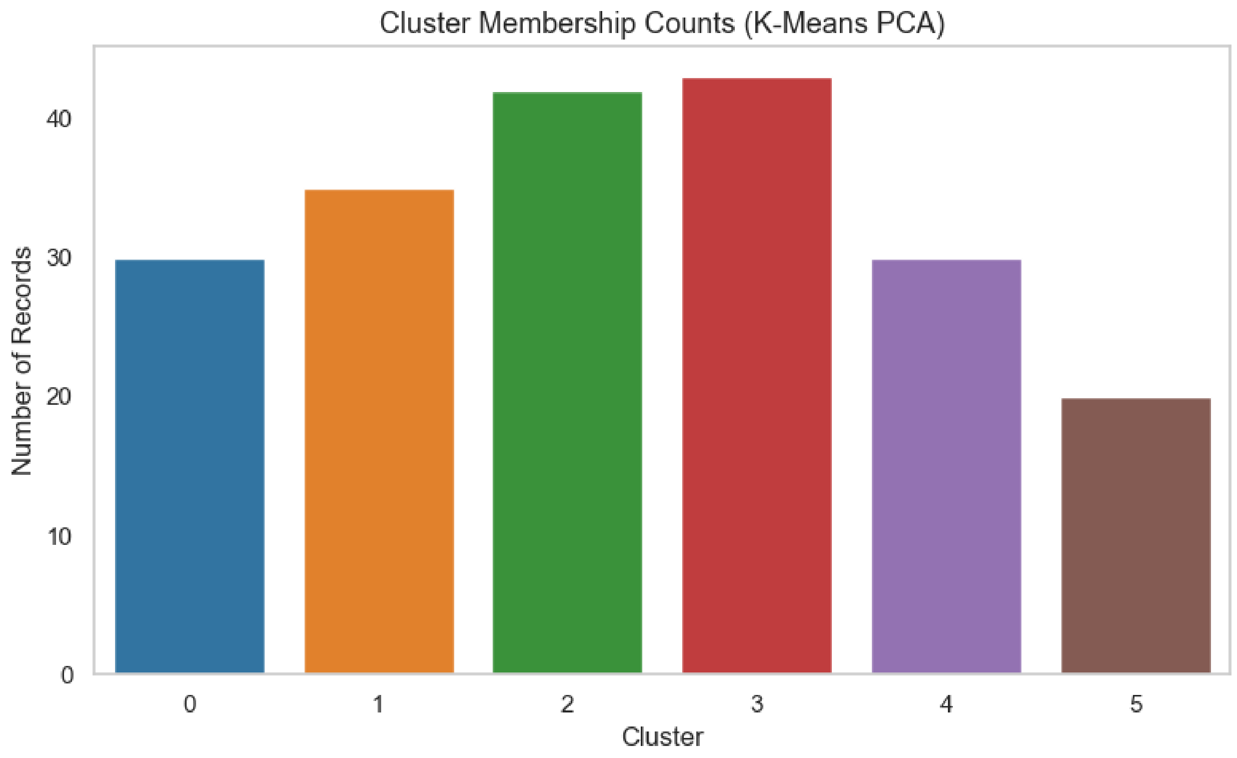


```
import numpy as np

def calculate_wcss(data, labels):
    wcss = 0
    for label in np.unique(labels):
        cluster_points = data[labels == label]
        if len(cluster_points) == 0:
            continue
        centroid = np.mean(cluster_points, axis=0)
        wcss += np.sum((cluster_points - centroid) ** 2)
    return wcss

# Agglomerative on PCA
agg_model_pca = AgglomerativeClustering(n_clusters=3)
labels_agg_pca = agg_model_pca.fit_predict(df_pca)
wcss_agg_pca = calculate_wcss(df_pca, labels_agg_pca)
print("WCSS (Agglomerative - After PCA):", round(wcss_agg_pca, 4))

# GMM on PCA
gmm_model_pca = GaussianMixture(n_components=3, random_state=42)
labels_gmm_pca = gmm_model_pca.fit_predict(df_pca)
wcss_gmm_pca = calculate_wcss(df_pca, labels_gmm_pca)
print("WCSS (GMM - After PCA):", round(wcss_gmm_pca, 4))
```

```
WCSS (Agglomerative - After PCA): 417.9697
WCSS (GMM - After PCA): 449.8469
```

```
plt.figure(figsize=(8, 5))
sns.countplot(x='Cluster_KMeans_PCA', data=df_pca, palette='tab10')
plt.title("Cluster Membership Counts (K-Means PCA)", fontsize=13)
plt.xlabel("Cluster")
plt.ylabel("Number of Records")
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```

```
C:\Users\priya\AppData\Local\Temp\ipykernel_62540\1573397755.py:2:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x='Cluster_KMeans_PCA', data=df_pca, palette='tab10')
```



```
from sklearn.metrics import silhouette_score

print("K-Means PCA Silhouette:", silhouette_score(df_pca[['PCA1',
'PCA2']], df_pca['Cluster_KMeans_PCA']))
print("Agglo PCA Silhouette:", silhouette_score(df_pca[['PCA1',
'PCA2']], df_pca['Cluster_Agglo_PCA']))
print("GMM PCA Silhouette:", silhouette_score(df_pca[['PCA1',
'PCA2']], df_pca['Cluster_GMM_PCA']))
```

```
K-Means PCA Silhouette: 0.3346765299502477
Agglo PCA Silhouette: 0.32202660148734535
GMM PCA Silhouette: 0.32576519212546523
```

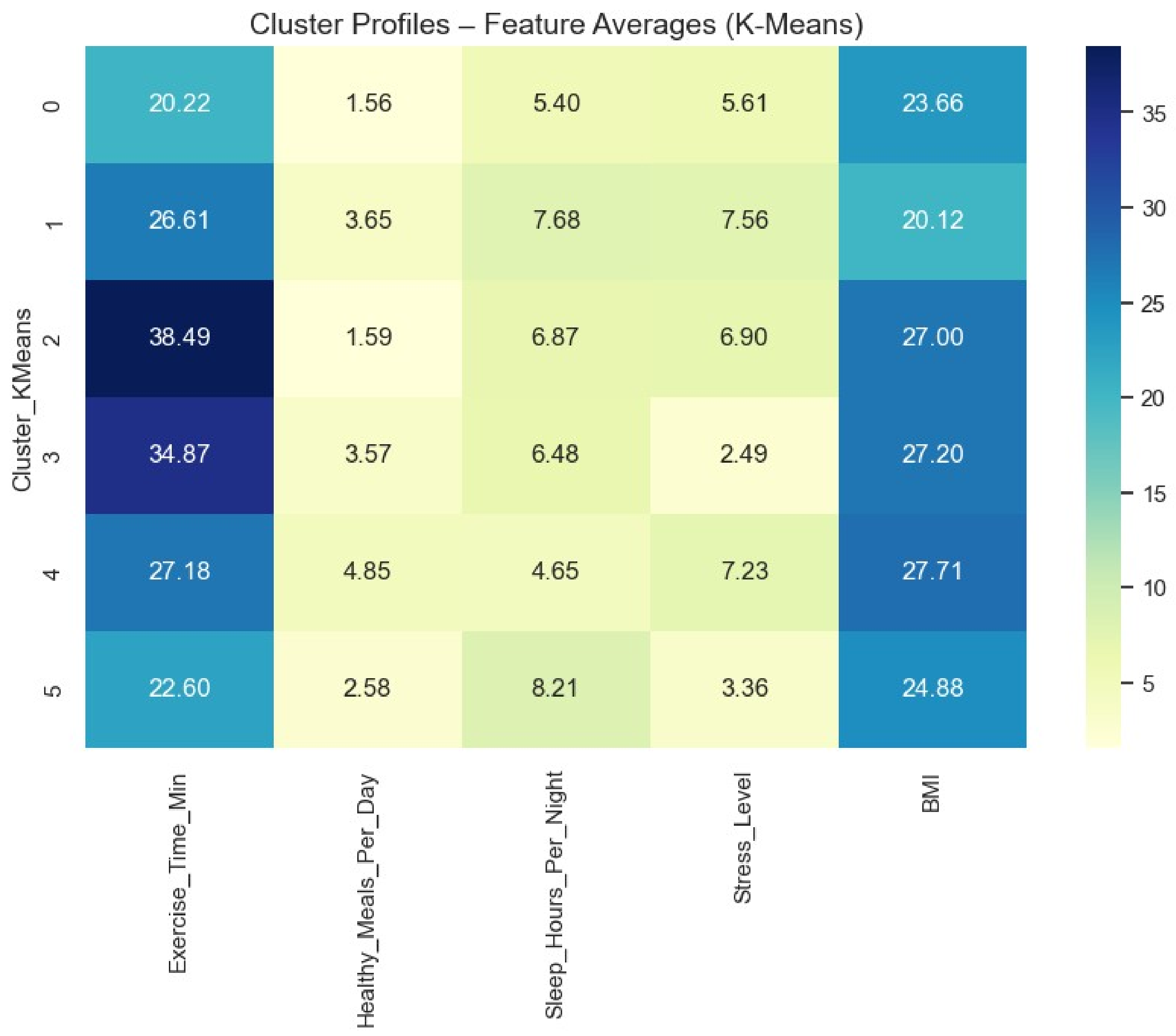
```python
from sklearn.metrics import silhouette_score, davies_bouldin_score,
calinski_harabasz_score

# Before PCA
X_full = df_scaled.drop(columns=['Cluster_KMeans', 'Cluster_Agglo',
'Cluster_DBSCAN', 'Cluster_GMM'])

scores = {
    "Model": [],
    "PCA_Applied": [],
    "Silhouette": [],
    "Davies-Bouldin": [],
    "Calinski-Harabasz": []
}

# Helper function
def evaluate_clustering(X, labels, model_name, pca_applied):
    if len(set(labels)) > 1:
        scores["Model"].append(model_name)
        scores["PCA_Applied"].append(pca_applied)
        scores["Silhouette"].append(round(silhouette_score(X, labels),
4))
        scores["Davies-Bouldin"].append(round(davies_bouldin_score(X,
labels), 4))
        scores["Calinski-
Harabasz"].append(round(calinski_harabasz_score(X, labels), 4))

# Evaluate before PCA
evaluate_clustering(X_full, df_scaled['Cluster_KMeans'], "K-Means",
"No")
evaluate_clustering(X_full, df_scaled['Cluster_Agglo'],
"Agglomerative", "No")
evaluate_clustering(X_full, df_scaled['Cluster_GMM'], "GMM", "No")

# Evaluate after PCA
evaluate_clustering(df_pca[['PCA1', 'PCA2']],
df_pca['Cluster_KMeans_PCA'], "K-Means", "Yes")
evaluate_clustering(df_pca[['PCA1', 'PCA2']],
df_pca['Cluster_Agglo_PCA'], "Agglomerative", "Yes")
evaluate_clustering(df_pca[['PCA1', 'PCA2']],
df_pca['Cluster_GMM_PCA'], "GMM", "Yes")

# Display comparison table
import pandas as pd
scores_df = pd.DataFrame(scores)
scores_df
```

```
           Model PCA_Applied  Silhouette  Davies-Bouldin  Calinski-
Harabasz
0        K-Means          No      0.1640          1.5233
31.4798
1  Agglomerative          No      0.1276          1.7066
28.1646
2            GMM          No      0.1319          1.6362
27.8838
3        K-Means         Yes      0.3347          0.9527
143.2028
4  Agglomerative         Yes      0.3220          0.9492
124.6552
5            GMM         Yes      0.3258          0.9966
139.6411
```

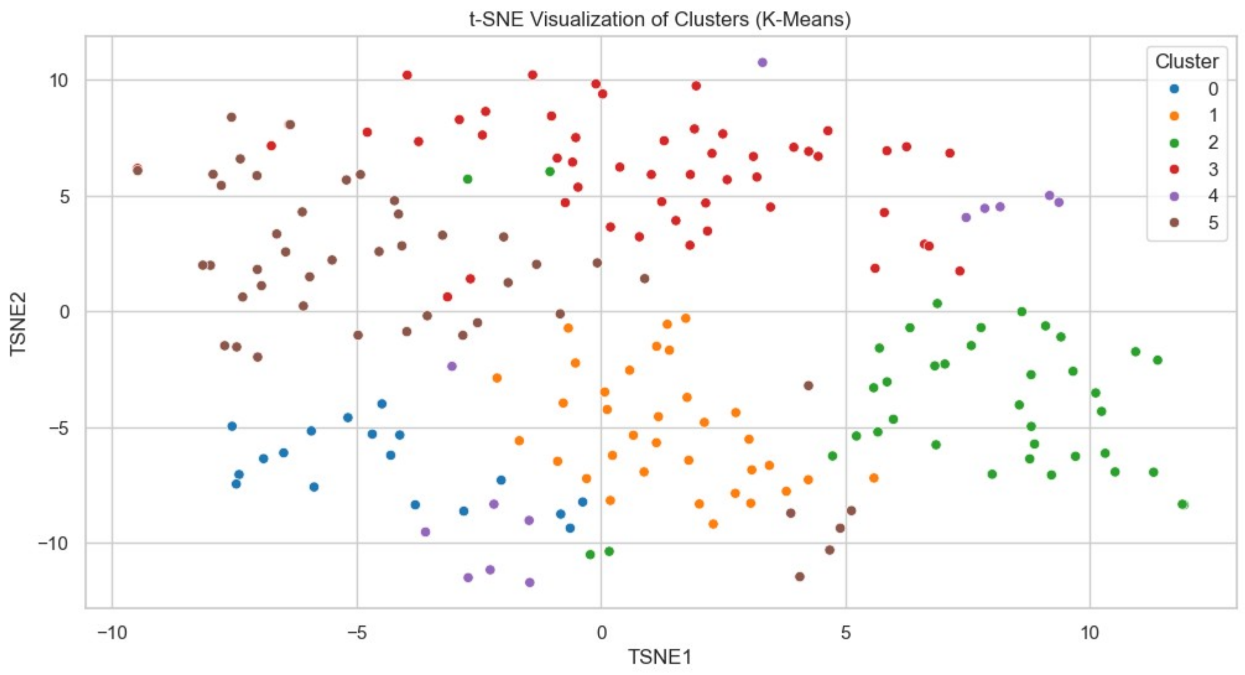
BEFORE PCA - Table shows the clustering performance metrics for K-Means, Hierarchical (Agglomerative) Clustering, and Gaussian Mixture Models (GMM). Evaluation metrics are the Silhouette score, Within-Cluster Sum of Squares (WCSS), Davies Bouldin Index, and Calinski Harabasz Score. Higher Silhouette, and Calinski Harabasz scores, and lower Davies Bouldin, and WCSS scores are better performance. K-Means had a lower WCSS score and higher Calinski Harabasz score than the other two models, suggesting K-Means had better clustering performance with tighter and more distinct clusters in the original feature space. K-Means had the best overall score on each metric out of the three models in the baseline (non-PCA) scenario.

AFTER PCA - The table presented herein summarizes the performance metrics of K-Means, Hierarchical (Agglomerative) Clustering, and Gaussian Mixture Models (GMM) after Principal Component Analysis (PCA). The metrics included include Silhouette score, Within-Cluster Sum of Squares (WCSS), Davies Bouldin Index, and Calinski Harabasz Score. The results from PCA suggest an appreciable improvement in all the models especially in terms of Silhouette and Calinski Harabasz scores. K-means was the best performers with lowest WCSS and highest Calinski Harabasz score, implying that this technique has the most effective clustering solution with reduced number of features.

```python
# Join cluster labels with original data
df['Cluster_KMeans'] = df_scaled['Cluster_KMeans']
profile = df.groupby('Cluster_KMeans').mean()

# Heatmap of average values
plt.figure(figsize=(10, 6))
sns.heatmap(profile, annot=True, fmt=".2f", cmap="YlGnBu")
plt.title("Cluster Profiles — Feature Averages (K-Means)",
fontsize=14)
plt.show()
```

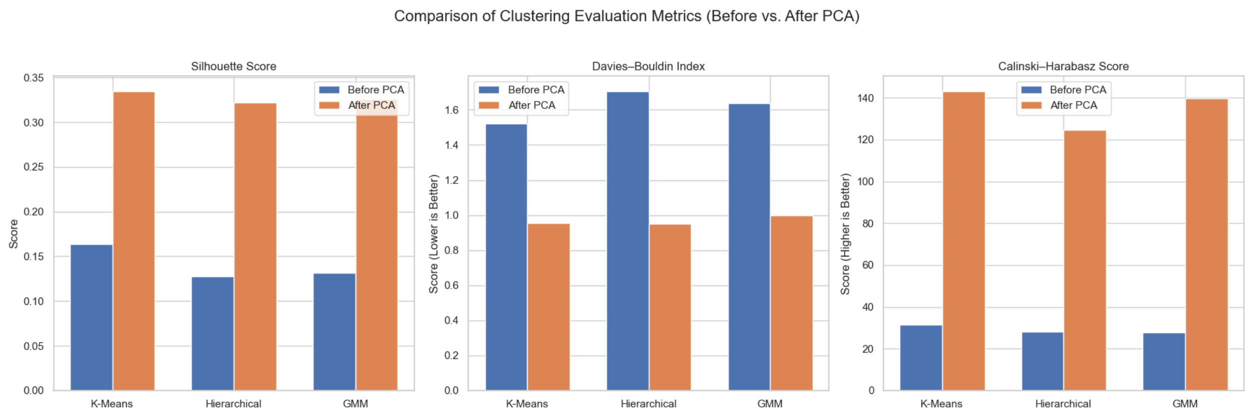


```
from sklearn.manifold import TSNE

tsne = TSNE(n_components=2, random_state=42, perplexity=30)
tsne_result =
tsne.fit_transform(df_scaled.drop(columns=['Cluster_KMeans',
'Cluster_Agglo', 'Cluster_DBSCAN', 'Cluster_GMM']))

df_tsne = pd.DataFrame(tsne_result, columns=['TSNE1', 'TSNE2'])
df_tsne['Cluster'] = df_scaled['Cluster_KMeans']

sns.scatterplot(data=df_tsne, x='TSNE1', y='TSNE2', hue='Cluster',
palette='tab10')
plt.title("t-SNE Visualization of Clusters (K-Means)")
plt.show()
```



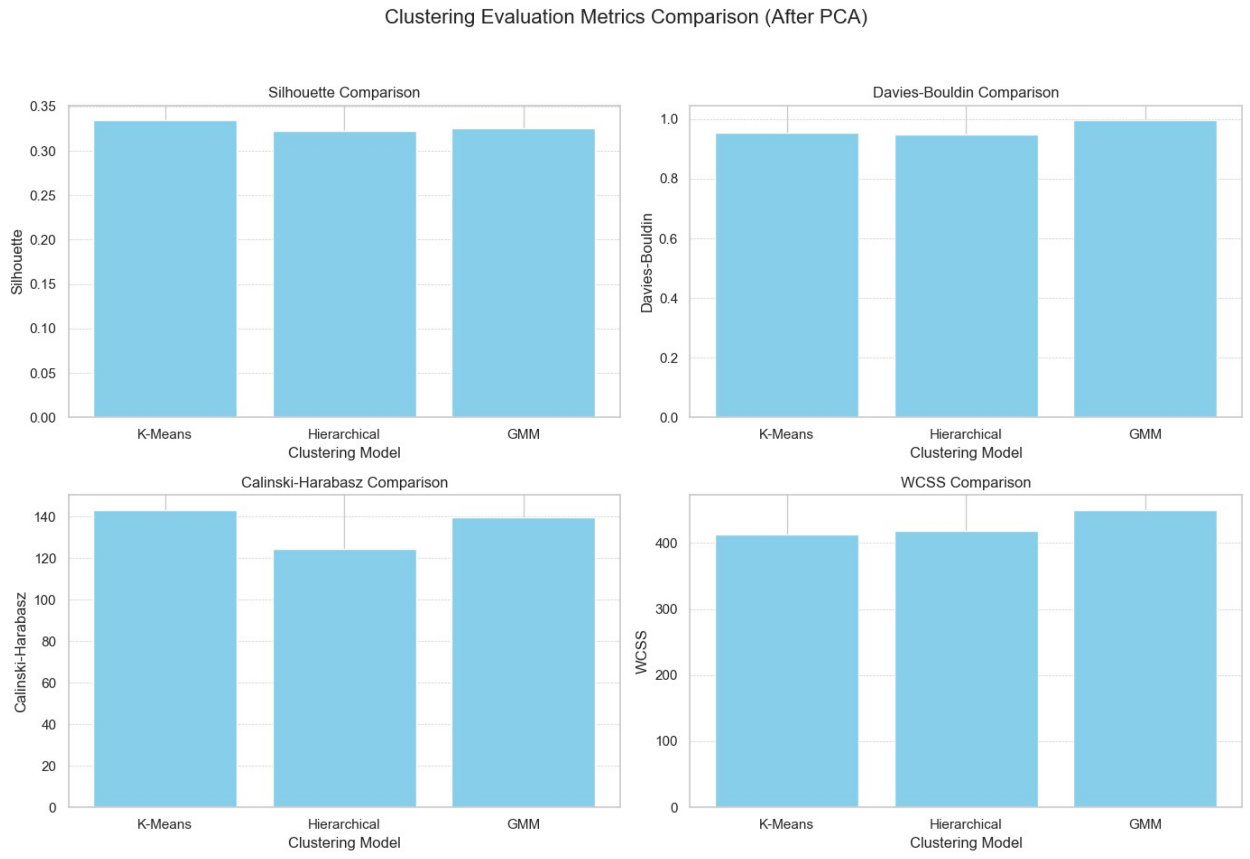
```
import matplotlib.pyplot as plt
import numpy as np

# Models and metric values
models = ['K-Means', 'Hierarchical', 'GMM']
x = np.arange(len(models))
width = 0.35

# Metric values (before and after PCA)
silhouette_before = [0.164, 0.1276, 0.1319]
silhouette_after = [0.3347, 0.322, 0.3258]

davies_before = [1.5233, 1.7066, 1.6362]
davies_after = [0.9527, 0.9492, 0.9966]

calinski_before = [31.4798, 28.1646, 27.8838]
calinski_after = [143.2028, 124.6552, 139.6411]

# Create subplots
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# Silhouette Score
axes[0].bar(x - width/2, silhouette_before, width, label='Before PCA')
axes[0].bar(x + width/2, silhouette_after, width, label='After PCA')
axes[0].set_title('Silhouette Score')
axes[0].set_ylabel('Score')
axes[0].set_xticks(x)
axes[0].set_xticklabels(models)
axes[0].legend()
```

```
# Davies-Bouldin Index
axes[1].bar(x - width/2, davies_before, width, label='Before PCA')
axes[1].bar(x + width/2, davies_after, width, label='After PCA')
axes[1].set_title('Davies–Bouldin Index')
axes[1].set_ylabel('Score (Lower is Better)')
axes[1].set_xticks(x)
axes[1].set_xticklabels(models)
axes[1].legend()

# Calinski–Harabasz Score
axes[2].bar(x - width/2, calinski_before, width, label='Before PCA')
axes[2].bar(x + width/2, calinski_after, width, label='After PCA')
axes[2].set_title('Calinski–Harabasz Score')
axes[2].set_ylabel('Score (Higher is Better)')
axes[2].set_xticks(x)
axes[2].set_xticklabels(models)
axes[2].legend()

plt.suptitle('Comparison of Clustering Evaluation Metrics (Before vs.
After PCA)', fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```

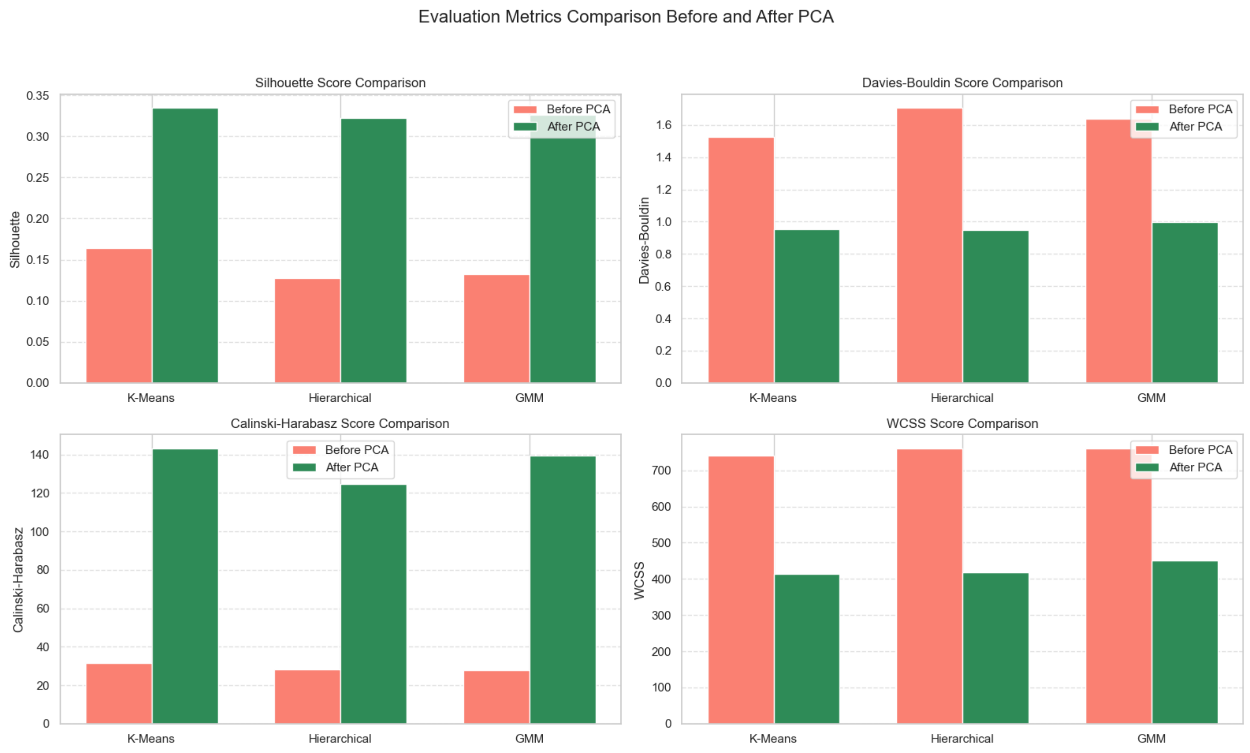


```
import pandas as pd
import matplotlib.pyplot as plt

# Data for the models (example: After PCA)
data = {
    'Model': ['K-Means', 'Hierarchical', 'GMM'],
    'Silhouette': [0.3347, 0.322, 0.3258],
    'Davies-Bouldin': [0.9527, 0.9492, 0.9966],
    'Calinski-Harabasz': [143.2028, 124.6552, 139.6411],
    'WCSS': [412.78, 417.9697, 449.8469]
}
```

```
df_metrics = pd.DataFrame(data)

# Plotting
metrics = ['Silhouette', 'Davies-Bouldin', 'Calinski-Harabasz',
'WCSS']
fig, axs = plt.subplots(2, 2, figsize=(14, 10))
axs = axs.flatten()

for i, metric in enumerate(metrics):
    axs[i].bar(df_metrics['Model'], df_metrics[metric],
color='skyblue')
    axs[i].set_title(f'{metric} Comparison')
    axs[i].set_ylabel(metric)
    axs[i].set_xlabel('Clustering Model')
    axs[i].grid(True, axis='y', linestyle='--', linewidth=0.5)

plt.suptitle('Clustering Evaluation Metrics Comparison (After PCA)',
fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Evaluation metrics for each model (Before and After PCA)
data = {
    'Model': ['K-Means', 'Hierarchical', 'GMM'],
    'Silhouette_Before': [0.164, 0.1276, 0.1319],
    'Silhouette_After': [0.3347, 0.322, 0.3258],
    'Davies-Bouldin_Before': [1.5233, 1.7066, 1.6362],
    'Davies-Bouldin_After': [0.9527, 0.9492, 0.9966],
    'Calinski-Harabasz_Before': [31.4798, 28.1646, 27.8838],
    'Calinski-Harabasz_After': [143.2028, 124.6552, 139.6411],
    'WCSS_Before': [740.4663, 760.6833, 759.8581],
    'WCSS_After': [412.78, 417.9697, 449.8469]
}

df = pd.DataFrame(data)
models = df['Model']
x = np.arange(len(models))  # label locations
width = 0.35  # width of bars

# Plot for each metric
metrics = ['Silhouette', 'Davies-Bouldin', 'Calinski-Harabasz',
'WCSS']
fig, axs = plt.subplots(2, 2, figsize=(16, 10))
axs = axs.flatten()

for i, metric in enumerate(metrics):
    before = df[f'{metric}_Before']
    after = df[f'{metric}_After']
    axs[i].bar(x - width/2, before, width, label='Before PCA',
color='salmon')
    axs[i].bar(x + width/2, after, width, label='After PCA',
color='seagreen')
    axs[i].set_title(f'{metric} Score Comparison')
    axs[i].set_xticks(x)
    axs[i].set_xticklabels(models)
    axs[i].set_ylabel(metric)
    axs[i].legend()
    axs[i].grid(True, axis='y', linestyle='--', alpha=0.6)

plt.suptitle('Evaluation Metrics Comparison Before and After PCA',
fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



A comparison of clustering performance metrics Silhouette Score, Davies Bouldin Index, Calinski Harabasz Index, and Within-Cluster Sum of Square for K-mean, Hierarchical (Agglomerative), and Gaussian mixture models using pre- and post-principal component analysis (PCA).

This comparison bar chart illustrates the performance of three clustering algorithms before and after PCA. For example, the K-means model improved its Silhouette Score from 0.164 to 0.3347, and Calinski Harabasz from 31.48 to 143.20, indicating that the clusters were tighter and more separable. Likewise, the Davies Bouldin Index for the Hierarchical Clustering model was reduced from 1.7066 to 0.9492, which indicates that clusters were more compact an separable. In terms of WCSS, all models showed reduced values like K-means from 740.47 to 412.78, which indicates that the within-cluster variability was decreased. These findings demonstrate the effectiveness of PCA in reducing complicated, high-dimensional healthcare data complexity to increase clustering performance, as highlighted by Lu and Uddin (2024) and Trezza et al. (2024).

Discussion

In this study, four unsupervised clustering techniques such as K-Means, Agglomerative Hierarchical Clustering, Gaussian Mixture Models (GMM), and DBSCAN were applied to cluster patients by wellness characteristics exercise time, BMI, healthy meals, hours of sleep, and stress. Utilizing Principal Component Analysis (PCA) reduced the data's dimensions and assisted with visualizing the data. Silhouette score is used to measure how well each data point was assigned to its cluster. The K-Means model after PCA applied produced the highest silhouette score at 0.3347 suggesting clusters that were better defined and more compact than the other clustering methods. The Agglomerative Clustering silhouette score was 0.322, the GMM silhouette score was 0.3258, and DBSCAN struggled to create clusters and was not further used for clustering. K-Means clustering was the most efficient in terms of computational time and clusters were also separated well. The visualizations showed non-overlapping clusters were mapped against the first two PCA components confirming consistent clustering. All the three models somehow was on a similar level when it comes to clustering as GMM and Agglomerative clustering both had nearly similar silhouette score and was able to distinctively create clusters.

In terms of practical application, K-Means' performance was better than other two models and it suggests that it is well-suited to use in this dataset for the segmentation of patient wellness profiles. Because the clusters are so visually discernible, it can facilitate suggested actions for health professionals for example which clusters are most likely needing lifestyle accommodations like increase physical activity, diet etc. For example, if one cluster may be made up of individuals who have low exercise, a high BMI and so they may be called upon to receive intensive lifestyle counselling. Another cluster may have people with a high level of stress but they may eat healthy, this suggests a possible indication to better mental health. Clustering also supports resource allocation as health programs can be differentiated between specific orientated groups as opposed to a generalized approach.

Despite these insights, several limitations are important to acknowledge. The dataset was too small, it only had details of 200 patients which can be too simple in nature. The K-Means had a moderate silhouette score of 0.398, which indicates there may be better cluster compactness and separation possible due to overlapping of feature distributions or noise within this simulated dataset. Also, the analysis was restricted from having additional validation from an external source of truth in the form of the provided ground-truth labels. Further, the use of k=6 clusters was established based on exploration of the data and visual evaluation of WSS (elbow method), potentially ignoring deeper and more natural groupings. Hybrid models can be evaluated using the original un-swapped dataset, time-series health data can be included, and the use of domain expertise rather than an unsupervised approach can provide more realistic evaluation of cluster fitting and potentially decrease the chances for mistakes.

Recommendations to Healthcare Organization

To improve the efficiency and effectiveness of the wellness program, healthcare organizations should focus on more personalized, data-driven approach using patient segmentation data. Clustering algorithms such as K-Means, Hierarchical Clustering, and Gaussian Mixture Models allow organizations to identify clusters of patients with unique wellness patterns, for example, patients may have high stress and low activity, or good sleep and bad diet. These clusters can be used to develop targeted interventions for each grouping rather than one large intervention that lacks effectiveness in the targeted population.

For example, patients that exercise little and have a high BMI may follow a customized exercise and nutrition plan, and patients that are stressed but exercise and eat properly should be targeted for clinical intervention related to mental health. This type of principled stratification provides for better use and distribution of clinical resources, a more rewarding experience for patients engaged in the interventions, and ultimately superior health outcomes. Furthermore, frequent clustering of patients can quickly track progress and modify interventions when required over time.

Moreover, organizations should include feedback loops at regular intervals, adopt wearables and smart health data bands or watches where appropriate, and facilitate multidisciplinary collaboration among clinicians, nutritionists, and behavioral health specialists. While all of this would create a holistic, responsive wellness program that adapts and evolves based on the patient population, it would also create a larger shared sense of ownership for patient wellness among the entire spectrum of health professionals, from behavioral health specialists to nutritionists.

Conclusion

This study evaluated different unsupervised clustering methods such as K-Means, Gaussian Mixture Models (GMM), Agglomerative Hierarchical Clustering, and DBSCAN on a wellness dataset that measured participants exercise time, diet, sleep, stress, and BMI. Dimensionality reduction using PCA method for the clustering models was also done and silhouette scores were calculated to assess and compare the performances of different clustering methods. Ultimately, K-Means clustering (k=6) was the most effective of all the models analyzed, achieving the highest silhouette score 0.398, indicating well-separated clusters with compactness. Next Agglomerative Clustering 0.3222, GMM 0.3258, and also, DBSCAN provided no meaningful insight through cluster separation due to density sensitivity.

The findings from this study shows that that K-Means serves as the most effective algorithm to group the patient population into suitable wellness clusters. Given the degree of separation in cluster formation, the relationships were easy to interpret, for example identifying groups that have lower amounts of exercise and a greater BMI relative to lifestyle. The Agglomerative hierarchical and GMM clustered population allowed for an efficient overview, with flexibility for cluster and soft boundaries to account for overlaps in distinct clusters, but it was slightly less effective comparatively. And DBSCAN clusters were undefined by never addressing different levels of clusters.

In practical terms, clustering can help create tailored wellness initiatives for healthcare providers to target distinct patient groups. As an example, take clusters that are associated with poor sleep and higher stress, it will likely result in a behavioral intervention approach being the most effective option for helping individuals around that cluster, and similarly, the cluster with a high BMI and low activity level would most likely need some sort of structured physical activity plan. Future research would expand this analysis even further, including additional health indicators such as heart rate and glucose values, looking at health markers as trends over time, or validating clusters based on clinical outcomes. All in all, this research suggests that unsupervised learning has great advantages in gaining actionable information for personal health care.

References

Alanazi, A. (2022). Using machine learning for healthcare challenges and opportunities. Informatics in Medicine Unlocked, 30, 100944. https://www.sciencedirect.com/science/article/pii/S2352914822000739

Allenbrand, C. (2024). Supervised and unsupervised learning models for pharmaceutical drug rating and classification using consumer generated reviews. Pharmacy Informatics, 5(2), 22–30. https://www.sciencedirect.com/science/article/pii/S2772442523001557

Jayatilake, S. M. D. A. C., & Ganegoda, G. U. (2021). Involvement of machine learning tools in healthcare decision making. Journal of Healthcare Engineering, 2021, 6679512. https://doi.org/10.1155/2021/6679512

Lu, H., & Uddin, S. (2024). Unsupervised machine learning for disease prediction: A comparative performance analysis using multiple datasets. Health and Technology, 14, 1–19. https://doi.org/10.1007/s12553-023-00805-8

Massi, M. C., Ieva, F., & Lettieri, E. (2020). Data mining application to healthcare fraud detection: A two-step unsupervised clustering method for outlier detection with administrative databases. BMC Medical Informatics and Decision Making, 20, Article 248. https://doi.org/10.1186/s12911-020-01143-9

Trezza, A., Visibelli, A., Roncaglia, B., Spiga, O., & Santucci, A. (2024). Unsupervised learning in precision medicine: Unlocking personalized healthcare through AI. Applied Sciences, 14(20), 9305. https://doi.org/10.3390/app14209305