https://archive.ics.uci.edu/dataset/544/estimation+of+obesity+levels+based+on+eating+habits+and+physical+condition

Review and Update - lengthen problem description and code ummaries

Problem Description: We aim to perform unsupervised learning to analyze and group individuals based on their eating habits and physical conditions. This dataset contains demographic, dietary, and activity-based attributes collected from individuals in Colombia, Mexico, and Peru.

The goal:

1. Use clustering techniques (KMeans, DBSCAN) to identify distinct groups of individuals.
2. Analyze the characteristics of each group using descriptive statistics and visualization.
3. Optionally compare discovered clusters with the target variable `NObeyesdad` for interpretation.

In [37]:
```
# Import Necessary Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# For data preprocessing
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# For dimensionality reduction
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

# For clustering
from sklearn.cluster import KMeans, DBSCAN
from sklearn.mixture import GaussianMixture
from scipy.cluster.hierarchy import dendrogram, linkage

# For evaluation metrics
from sklearn.metrics import silhouette_score, silhouette_samples
from sklearn.metrics import adjusted_rand_score, normalized_mutual_info_score, confusion_matrix, davies_bouldin_score, calinski

# For parallel coordinates plot
from pandas.plotting import parallel_coordinates

# Suppress warnings for cleaner output
import warnings
warnings.filterwarnings('ignore')
```

In [49]:
```
# Set plot styles for better aesthetics
sns.set(style="whitegrid", palette="muted", color_codes=True)
plt.rcParams['figure.figsize'] = (12, 8)
```

In [42]:
```
# Define the path to the ZIP file
zip_file_path = 'estimation+of+obesity+levels+based+on+eating+habits+and+physical+condition.zip'
extract_dir = 'obesity_dataset'
```

In [43]:
```
# Extract ZIP file contents
os.makedirs(extract_dir, exist_ok=True)
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extract_dir)

print(f"Files extracted to '{extract_dir}' directory.")
```

```
Files extracted to 'obesity_dataset' directory.
```

In [44]:
```
# List extracted files
extracted_files = os.listdir(extract_dir)
print("Extracted Files:")
print(extracted_files)
```

```
Extracted Files:
['ObesityDataSet_raw_and_data_sinthetic.csv']
```

LOAD DATA

In [45]:
```
# Load the dataset into a DataFrame
csv_file_path = os.path.join(extract_dir, 'ObesityDataSet_raw_and_data_sinthetic.csv')
df = pd.read_csv(csv_file_path)
```

In [46]:
```
# Display the first few rows of the dataset
print("\nFirst 5 rows of the dataset:")
print(df.head())
```

```
First 5 rows of the dataset:
   Gender   Age  Height  Weight family_history_with_overweight FAVC  FCVC  \
0  Female  21.0    1.62    64.0                            yes   no   2.0
1  Female  21.0    1.52    56.0                            yes   no   3.0
2    Male  23.0    1.80    77.0                            yes   no   2.0
3    Male  27.0    1.80    87.0                             no   no   3.0
4    Male  22.0    1.78    89.8                             no   no   2.0

   NCP       CAEC SMOKE  CH2O  SCC  FAF  TUE        CALC  \
0  3.0  Sometimes    no   2.0   no  0.0  1.0          no
1  3.0  Sometimes   yes   3.0  yes  3.0  0.0   Sometimes
2  3.0  Sometimes    no   2.0   no  2.0  1.0  Frequently
3  3.0  Sometimes    no   2.0   no  2.0  0.0  Frequently
4  1.0  Sometimes    no   2.0   no  0.0  0.0   Sometimes

                 MTRANS           NObeyesdad
0  Public_Transportation        Normal_Weight
1  Public_Transportation        Normal_Weight
2  Public_Transportation        Normal_Weight
3                Walking   Overweight_Level_I
4  Public_Transportation  Overweight_Level_II
```

The data is clean with no missing values displayed. It includes both numerical and categorical variables. The target column NObeyesdad classifies individuals into weight categories (e.g., "Normal_Weight", "Overweight_Level_I").

EXPLORATORY DATA ANALYSIS (EDA)

In [25]:
```
# Basic dataset information
print("\nDataset Information:")
print(df.info())
```

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2111 entries, 0 to 2110
Data columns (total 17 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Gender                          2111 non-null   object
 1   Age                             2111 non-null   float64
 2   Height                          2111 non-null   float64
 3   Weight                          2111 non-null   float64
 4   family_history_with_overweight  2111 non-null   object
 5   FAVC                            2111 non-null   object
 6   FCVC                            2111 non-null   float64
 7   NCP                             2111 non-null   float64
 8   CAEC                            2111 non-null   object
 9   SMOKE                           2111 non-null   object
 10  CH2O                            2111 non-null   float64
 11  SCC                             2111 non-null   object
 12  FAF                             2111 non-null   float64
 13  TUE                             2111 non-null   float64
 14  CALC                            2111 non-null   object
 15  MTRANS                          2111 non-null   object
 16  NObeyesdad                      2111 non-null   object
dtypes: float64(8), object(9)
memory usage: 280.5+ KB
None
```

In [26]:
```
# Summary statistics
print("\nSummary Statistics:")
print(df.describe(include='all'))
```

```
Summary Statistics:
       Gender          Age       Height       Weight  \
count    2111  2111.000000  2111.000000  2111.000000
unique      2          NaN          NaN          NaN
top      Male          NaN          NaN          NaN
freq     1068          NaN          NaN          NaN
mean      NaN    24.312600     1.701677    86.586058
std       NaN     6.345968     0.093305    26.191172
min       NaN    14.000000     1.450000    39.000000
25%       NaN    19.947192     1.630000    65.473343
50%       NaN    22.777890     1.700499    83.000000
75%       NaN    26.000000     1.768464   107.430682
max       NaN    61.000000     1.980000   173.000000

       family_history_with_overweight  FAVC         FCVC          NCP  \
count                            2111  2111  2111.000000  2111.000000
unique                              2     2          NaN          NaN
top                               yes   yes          NaN          NaN
freq                             1726  1866          NaN          NaN
mean                              NaN   NaN     2.419043     2.685628
std                               NaN   NaN     0.533927     0.778039
min                               NaN   NaN     1.000000     1.000000
25%                               NaN   NaN     2.000000     2.658738
50%                               NaN   NaN     2.385502     3.000000
75%                               NaN   NaN     3.000000     3.000000
max                               NaN   NaN     3.000000     4.000000

            CAEC SMOKE         CH2O   SCC          FAF          TUE  \
count       2111  2111  2111.000000  2111  2111.000000  2111.000000
unique         4     2          NaN     2          NaN          NaN
top     Sometimes    no          NaN    no          NaN          NaN
freq        1765  2067          NaN  2015          NaN          NaN
mean         NaN   NaN     2.008011   NaN     1.010298     0.657866
std          NaN   NaN     0.612953   NaN     0.850592     0.608927
min          NaN   NaN     1.000000   NaN     0.000000     0.000000
25%          NaN   NaN     1.584812   NaN     0.124505     0.000000
50%          NaN   NaN     2.000000   NaN     1.000000     0.625350
75%          NaN   NaN     2.477420   NaN     1.666678     1.000000
max          NaN   NaN     3.000000   NaN     3.000000     2.000000

            CALC                 MTRANS      NObeyesdad
count       2111                   2111            2111
unique         4                      5               7
top     Sometimes  Public_Transportation  Obesity_Type_I
freq        1401                   1580             351
mean         NaN                    NaN             NaN
std          NaN                    NaN             NaN
min          NaN                    NaN             NaN
25%          NaN                    NaN             NaN
50%          NaN                    NaN             NaN
75%          NaN                    NaN             NaN
max          NaN                    NaN             NaN
```

Number of Instances: 2111 rows. Features:

- 8 numerical features (e.g., Age, Height, Weight, etc.).
- 9 categorical features (e.g., Gender, family_history_with_overweight, FAVC, etc.).
- 1 target variable: NObeyesdad with 7 unique obesity levels

Obesity Trends:

- High-calorie food (FAVC) consumption and family history of overweight appear to dominate in this dataset, which may be significant contributors to obesity.
- Physical activity (FAF) is relatively low overall, which could also play a role in obesity levels.

Behavioral Factors:

- Most individuals do not monitor their caloric intake (SCC = no) and consume alcohol occasionally.
- Occasional snacking (CAEC = Sometimes) is the most frequent behavior.

Demographics:

- The majority of the individuals are young (mean age ~24) and use public transportation as their primary mode of travel.

Physical Metrics:

- The dataset includes individuals with a wide range of weights (39kg to 173kg), which suggests good diversity in body types.

Target Variable:

- NObeyesdad shows a range of obesity levels, with Obesity_Type_I being the most common. This indicates a need to examine which features are most strongly associated with obesity.

In [27]:
```python
# Check for missing values
print("\nMissing Values in Each Column:")
print(df.isnull().sum())
```

```
Missing Values in Each Column:
Gender                            0
Age                               0
Height                            0
Weight                            0
family_history_with_overweight    0
FAVC                              0
FCVC                              0
NCP                               0
CAEC                              0
SMOKE                             0
CH2O                              0
SCC                               0
FAF                               0
TUE                               0
CALC                              0
MTRANS                            0
NObeyesdad                        0
dtype: int64
```

In [33]:
```
df.columns
```

Out[33]:
```
Index(['Gender', 'Age', 'Height', 'Weight', 'family_history_with_overweight',
       'FAVC', 'FCVC', 'NCP', 'CAEC', 'SMOKE', 'CH2O', 'SCC', 'FAF', 'TUE',
       'CALC', 'MTRANS', 'NObeyesdad'],
      dtype='object')
```

In [39]:
```
# Identify categorical features
categorical_features = ['Gender', 'MTRANS', 'CALC', 'CAEC', 'SMOKE',
                        'SCC', 'family_history_with_overweight', 'FAVC', 'NObeyesdad']

# Identify ordinal features (assuming 'CALC', 'CAEC', 'SMOKE', 'SCC', 'family_history_with_overweight', 'FAVC' have ordinal rel
ordinal_features = ['CALC', 'CAEC', 'SMOKE', 'SCC', 'family_history_with_overweight', 'FAVC']

# One-Hot Encoding for nominal categorical variables
nominal_features = ['Gender', 'MTRANS', 'NObeyesdad']
```

In [ ]:

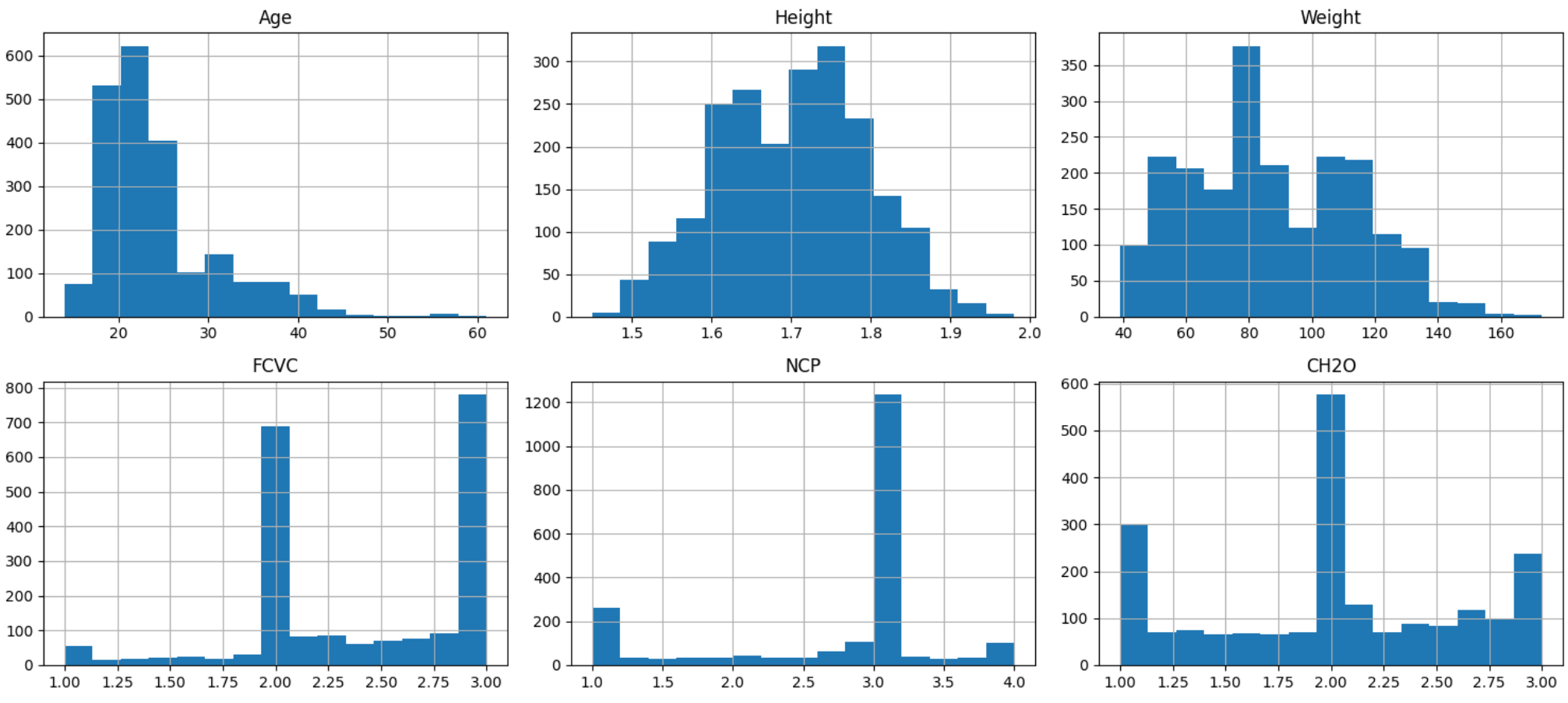
In [ ]:

In [ ]:

In [ ]:

VISUALIZE DATA DISTRIBUTIONS

In [28]:
```
# Plot distributions for numeric columns
numeric_features = ['Age', 'Height', 'Weight', 'FCVC', 'NCP', 'CH2O', 'FAF', 'TUE']
df[numeric_features].hist(bins=15, figsize=(15, 10))
plt.tight_layout()
plt.show()
```



The histograms display the distributions of key **numerical features** in the dataset. Below is an analysis of each feature's distribution:

---

1. Age

- **Distribution**: Right-skewed with most values between **18 and 30**.
- **Observation**: The majority of individuals are **young adults**, with very few individuals over 50. This aligns with the earlier insight from the summary statistics (mean age ~24).

---

2. Height

- **Distribution**: Approximately **normal distribution** centered around **1.7 meters**.
- **Observation**: Heights cluster around the mean (1.7m) with a narrow spread (1.5m to 2.0m), indicating minimal variability.

---

3. Weight

- **Distribution**: Bimodal distribution with peaks around **65-80 kg** and a smaller cluster around **110-130 kg**.
- **Observation**: There are distinct groups in the data:
  - One group corresponds to individuals of average weight.
  - The second group represents individuals with significantly higher weights, possibly indicating **obesity**.

---

4. FCVC (Frequency of Vegetable Consumption)

- **Distribution**: Concentrated around **2 and 3**.
- **Observation**: Most individuals have a moderate-to-high vegetable consumption, with very few reporting low consumption.

---

5. NCP (Number of Main Meals)

- **Distribution**: Strongly **right-skewed** with a sharp peak at **3 meals per day**.
- **Observation**: A large majority consume **3 main meals daily**, which is a common eating pattern.

---

6. CH2O (Daily Water Consumption)

- **Distribution**: Peaks at **2 liters per day**, with fewer individuals reporting very low or very high water consumption.
- **Observation**: Most individuals consume a moderate amount of water daily.

---

7. FAF (Physical Activity Frequency)

- **Distribution**: Peaks at **0 (no activity)** and **2 (moderate activity)**.
- **Observation**: A significant portion of individuals report **no physical activity**, which could be a contributing factor to obesity.

---

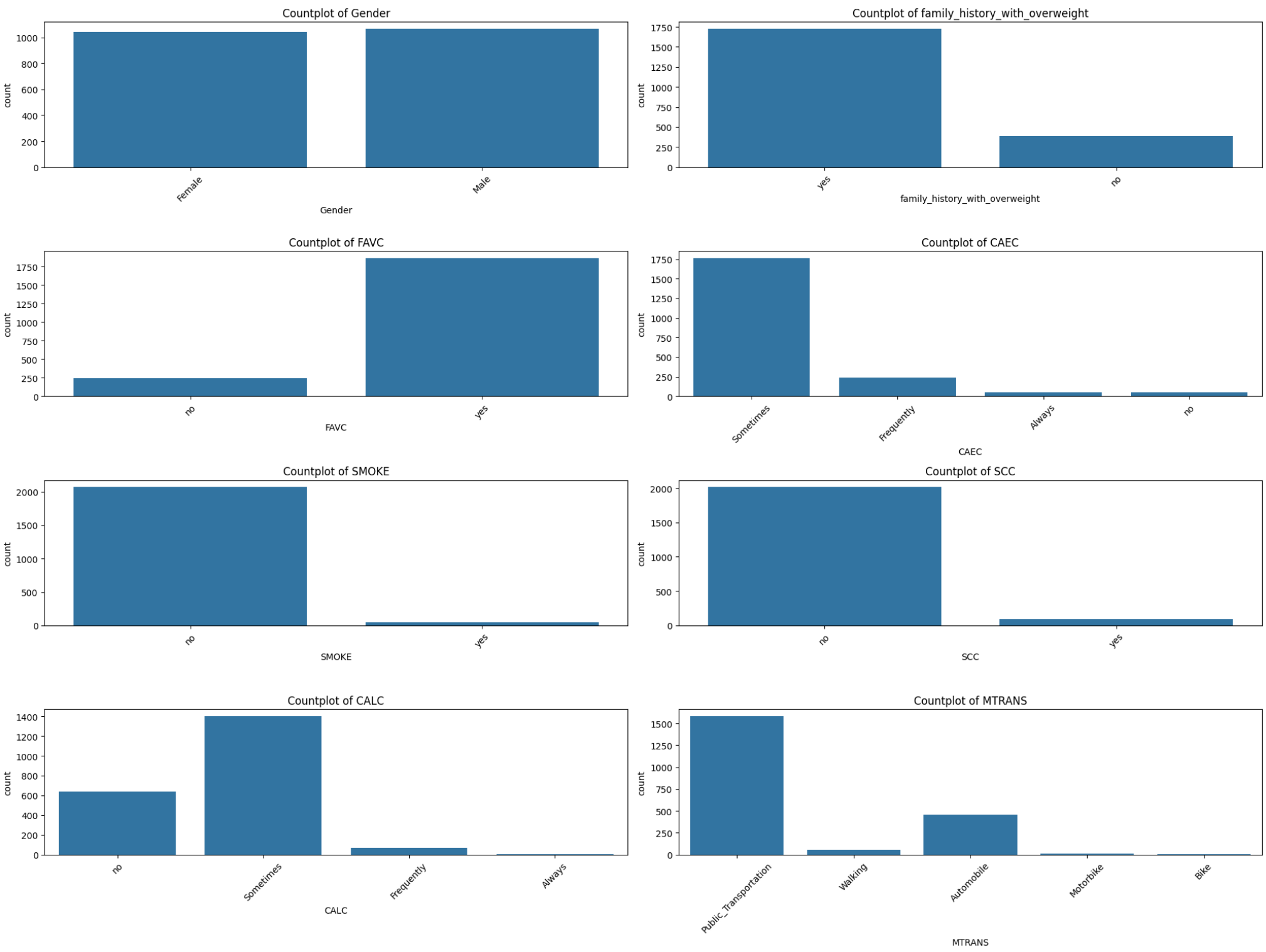
8. TUE (Time Using Technology)

- **Distribution**: Bimodal with peaks around **0.5** and **1.75 units**.
- **Observation**: There is a clear divide in technology usage:
  - One group spends minimal time on technology.
  - The other group spends **significantly more time**, possibly indicating sedentary behavior.

Key Observations and Insights

1. **Age** and **Height** distributions suggest a primarily **young and average-height population**.
2. **Weight** distribution highlights two distinct groups, likely corresponding to normal-weight and obese individuals.
3. Behavioral factors:
   - **FCVC** (vegetable consumption) is high for most individuals, which is a positive habit.
   - **NCP** indicates a consistent eating pattern with 3 meals/day.
   - **CH2O** shows good water consumption levels.
   - **FAF** (physical activity) reveals **low activity levels** for many individuals, which could correlate with higher weights.
4. **Technology Use**:
   - The bimodal distribution in **TUE** indicates two behavioral groups: those who spend minimal time and those heavily engaged in technology, which may reflect **sedentary lifestyles**.

In [29]:
```
# Countplot for categorical features
categorical_features = ['Gender', 'family_history_with_overweight', 'FAVC', 'CAEC',
                        'SMOKE', 'SCC', 'CALC', 'MTRANS']
plt.figure(figsize=(20, 15))
for i, feature in enumerate(categorical_features):
    plt.subplot(4, 2, i + 1)
    sns.countplot(data=df, x=feature)
    plt.title(f"Countplot of {feature}")
    plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



The countplots provide a visual distribution of the categorical features in the dataset. Below are key insights based on each plot:

---

1. Gender

- **Observation**: There is a slight imbalance between the two gender categories:
  - **Males** are slightly more represented (around 1068) compared to **Females** (approximately 1043).
- **Insight**: The dataset has a fairly balanced gender distribution, which reduces the risk of gender-related bias in analysis.

---

2. Family History of Overweight (`family_history_with_overweight`)

- **Observation**: The majority of individuals (**~80%**) reported having a **family history of overweight**.
- **Insight**: Family history seems to play a significant role in obesity and weight-related issues, which may be explored as a key feature in clustering and analysis.

---

3. FAVC (Frequent Consumption of High-Calorie Food)

- **Observation**: Most individuals (**~88%**) reported consuming high-calorie food frequently (**yes**).
- **Insight**: Frequent high-calorie food consumption is common and could be a major contributor to higher obesity levels.

---

4. CAEC (Food Consumption Between Meals)

- **Observation**:
  - **"Sometimes"** is the most frequent response, with the majority of individuals falling in this category.
  - Few individuals reported **"Frequently"** or **"Always"** consuming food between meals.
- **Insight**: Occasional snacking is prevalent. However, frequent snacking may contribute more to higher weight levels.

---

5. SMOKE

- **Observation**: A vast majority (**~98%**) reported **not smoking**.
- **Insight**: Smoking is not a common behavior in this dataset. It may have little to no impact on clustering or obesity levels.

---

6. SCC (Caloric Monitoring)

- **Observation**: Most individuals (**~95%**) do **not monitor their daily caloric intake**.
- **Insight**: A lack of caloric monitoring might contribute to poor eating habits and increased obesity levels.

---

7. CALC (Alcohol Consumption)

- **Observation**:
  - Most individuals consume alcohol **"Sometimes"**.
  - A smaller group consumes alcohol **"Frequently"** or **"Always"**.
  - A notable proportion reported **no alcohol consumption**.
- **Insight**: Occasional alcohol consumption is common. Higher frequency could correlate with obesity levels.

---

8. MTRANS (Mode of Transportation)

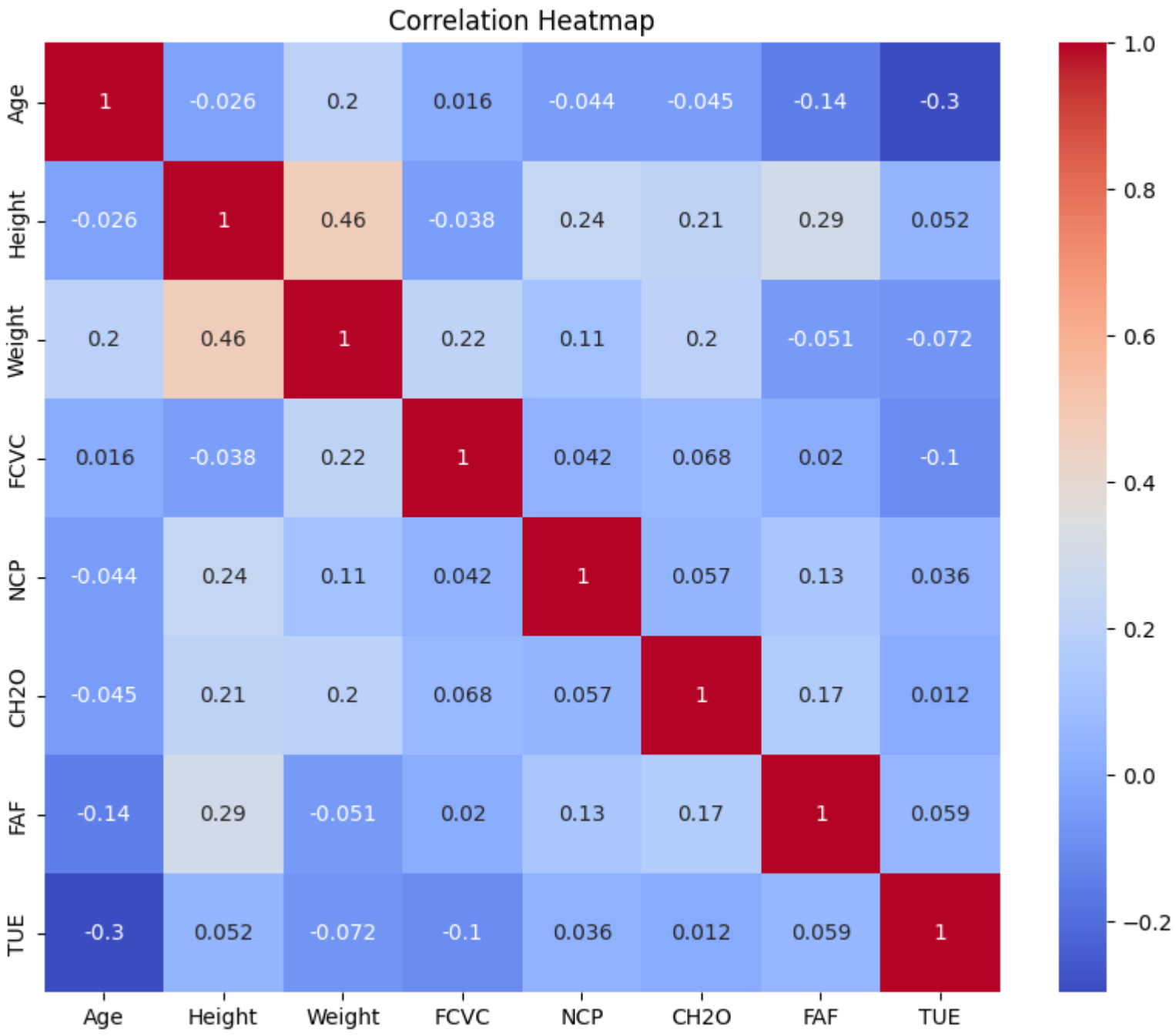
- **Observation**:
  - The majority rely on **Public Transportation** (~75%).
  - A smaller group uses **Walking** and other forms like **Automobile** or **Motorbike**.
- **Insight**: Heavy reliance on **public transportation** may suggest a sedentary lifestyle, while individuals who **walk** could exhibit healthier weight trends.

Overall Observations and Insights

1. **Behavioral Patterns**:
   - Most individuals report frequent consumption of **high-calorie food (FAVC)** and **occasional snacking (CAEC)**, but they do not monitor calories (`SCC = no`).
   - These habits might align with higher obesity levels.
2. **Family History**:
   - A significant majority reported a **family history of overweight**, suggesting a genetic predisposition to weight gain.
3. **Lifestyle**:
   - High reliance on **public transportation** and low levels of physical activity (as inferred earlier) may contribute to sedentary behavior.
4. **Smoking and Alcohol**:
   - Smoking is uncommon.
   - Alcohol is consumed **sometimes** by most individuals, with smaller groups reporting frequent consumption.

In [30]:
```
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

```
/var/folders/j_/555m2zps099832fjh_m8jjnc0000gn/T/ipykernel_14975/2943803821.py:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
  sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
```



DATA PREPROCESSING

In [12]:
```
# One-hot encode categorical features
column_transformer = ColumnTransformer([
    ('encoder', OneHotEncoder(drop='first'), categorical_features)
], remainder='passthrough')

X = column_transformer.fit_transform(df.drop('NObeyesdad', axis=1))  # Drop target variable for unsupervised learning
```

In [13]:
```
# Feature names after encoding
encoded_feature_names = column_transformer.named_transformers_['encoder'].get_feature_names_out(categorical_features)
feature_names = list(encoded_feature_names) + numeric_features
```

In [14]:
```
# Standardize numerical features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

The code prepares the dataset by:

1. Encoding categorical features into numerical form.
2. Combining and labeling all feature names.
3. Standardizing the features for clustering or other unsupervised learning techniques.

This ensures the data is clean, numeric, and scaled appropriately for further analysis.

KMEANS CLUSTERING

In [15]:
```
# Determine the optimal number of clusters using the Elbow Method
wcss = []
for k in range(2, 25):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)
```

```
/Users/ryantalbot/opt/anaconda3/envs/tf2/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/ryantalbot/opt/anaconda3/envs/tf2/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/ryantalbot/opt/anaconda3/envs/tf2/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/ryantalbot/opt/anaconda3/envs/tf2/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/ryantalbot/opt/anaconda3/envs/tf2/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/ryantalbot/opt/anaconda3/envs/tf2/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/ryantalbot/opt/anaconda3/envs/tf2/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/ryantalbot/opt/anaconda3/envs/tf2/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/ryantalbot/opt/anaconda3/envs/tf2/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/ryantalbot/opt/anaconda3/envs/tf2/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/ryantalbot/opt/anaconda3/envs/tf2/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/ryantalbot/opt/anaconda3/envs/tf2/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/ryantalbot/opt/anaconda3/envs/tf2/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/ryantalbot/opt/anaconda3/envs/tf2/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/ryantalbot/opt/anaconda3/envs/tf2/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/ryantalbot/opt/anaconda3/envs/tf2/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/ryantalbot/opt/anaconda3/envs/tf2/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/ryantalbot/opt/anaconda3/envs/tf2/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/ryantalbot/opt/anaconda3/envs/tf2/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/ryantalbot/opt/anaconda3/envs/tf2/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/ryantalbot/opt/anaconda3/envs/tf2/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/ryantalbot/opt/anaconda3/envs/tf2/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/Users/ryantalbot/opt/anaconda3/envs/tf2/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

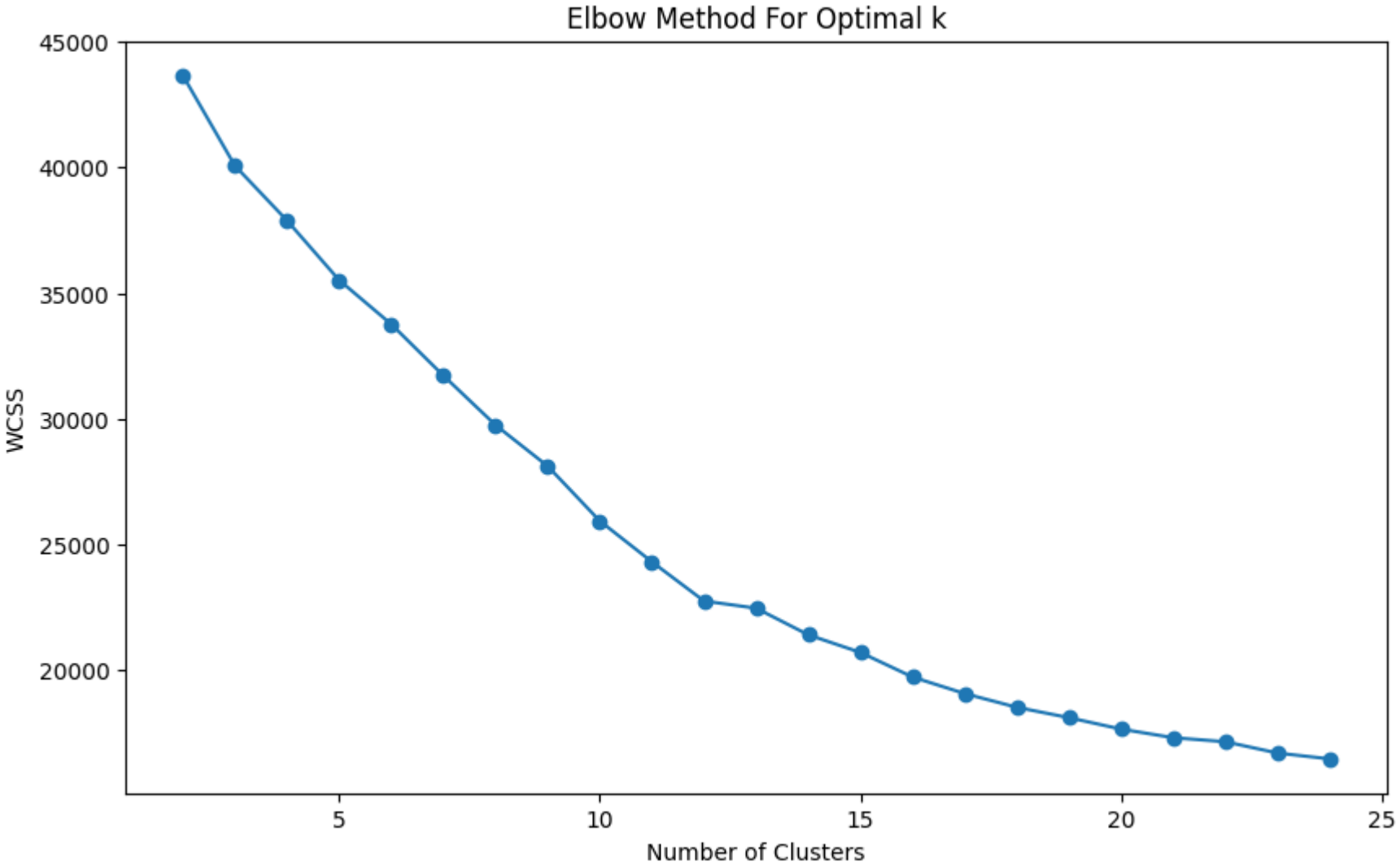
This code uses the Elbow Method to determine the optimal number of clusters (k) for K-Means Clustering.

The Elbow Method helps identify the optimal number of clusters by plotting k (number of clusters) against wcss. The "elbow" point in the plot indicates where the improvement in inertia starts to level off, suggesting the best k value.

This method ensures that clustering results are both accurate and interpretable.

In [16]:

```
# Plot the Elbow Curve
plt.figure(figsize=(10, 6))
plt.plot(range(2, 25), wcss, marker='o')
plt.title('Elbow Method For Optimal k')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```



The graph displays the Elbow Method to determine the optimal number of clusters (k) for K-Means Clustering.

Optimal Number of Clusters:

- The elbow point appears around k = 8 to 10, where further increasing k yields diminishing returns.

In [17]:
```
# Fit KMeans with optimal clusters
optimal_k = 2
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
kmeans_labels = kmeans.fit_predict(X_scaled)
```

```
/Users/ryantalbot/opt/anaconda3/envs/tf2/lib/python3.8/site-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

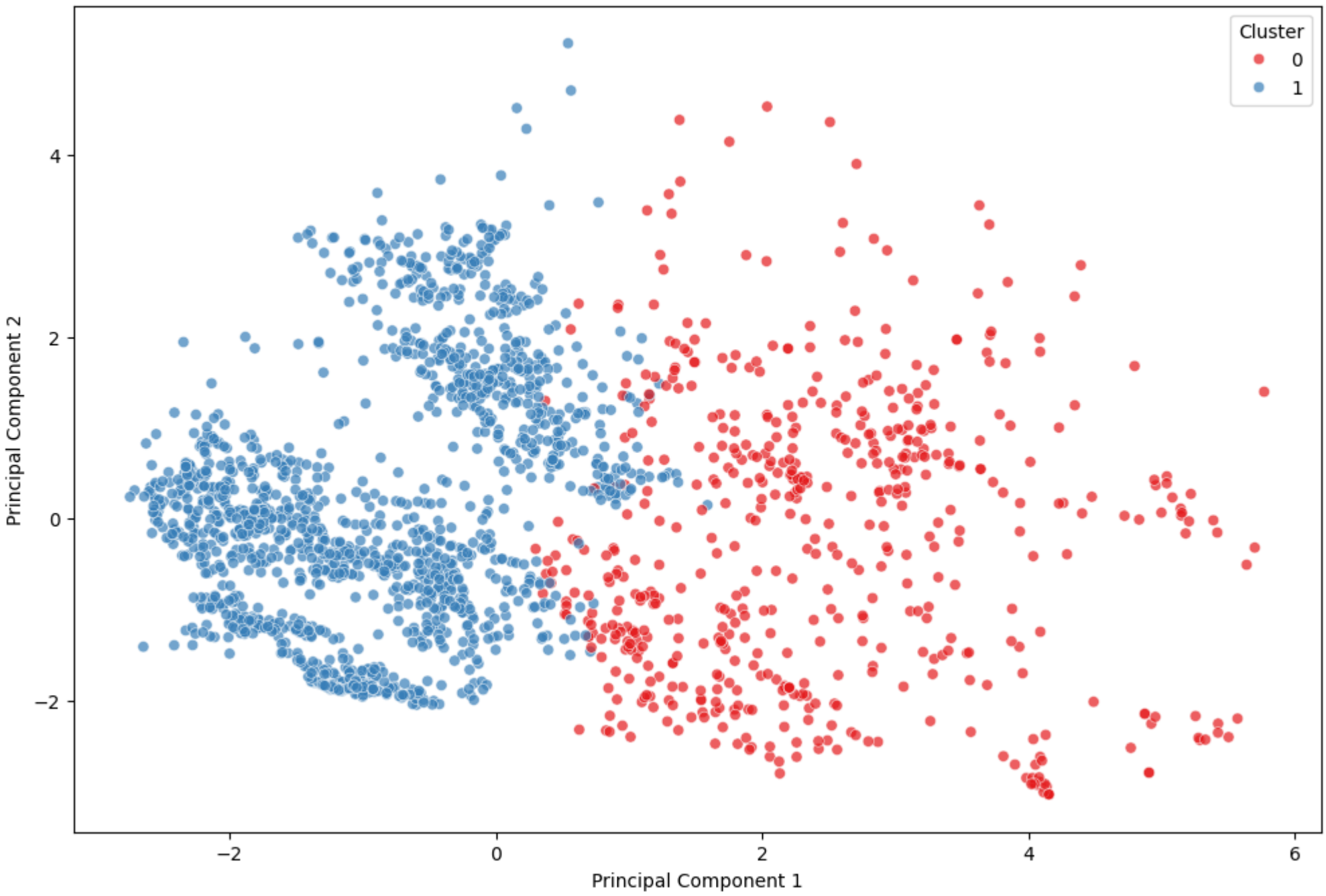
In [18]:
```
# Add cluster labels to the original DataFrame
df['Cluster'] = kmeans_labels
```

VISUALIZE CLUSTERS USING PCA

In [19]:
```
pca = PCA(n_components=2, random_state=42)
principal_components = pca.fit_transform(X_scaled)
```

In [20]:
```
# Create a DataFrame with PCA results and cluster labels
pca_df = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])
pca_df['Cluster'] = kmeans_labels
```

In [21]:
```
# Plot the clusters
plt.figure(figsize=(12, 8))
sns.scatterplot(data=pca_df, x='PC1', y='PC2', hue='Cluster', palette='Set1', alpha=0.7)
plt.title('PCA Visualization of K-Means Clusters')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Cluster')
plt.show()
```



EVALUATE CLUSTER QUALITY

In [22]:
```
silhouette = silhouette_score(X_scaled, kmeans_labels)
davies_bouldin = davies_bouldin_score(X_scaled, kmeans_labels)
calinski_harabasz = calinski_harabasz_score(X_scaled, kmeans_labels)

print("\nCluster Quality Metrics:")
print(f"Silhouette Score: {silhouette:.3f}")
print(f"Davies-Bouldin Score: {davies_bouldin:.3f}")
print(f"Calinski-Harabasz Index: {calinski_harabasz:.3f}")
```

```
Cluster Quality Metrics:
Silhouette Score: 0.226
Davies-Bouldin Score: 2.687
Calinski-Harabasz Index: 235.779
```

In [ ]:

Evaluation of Cluster Quality Across Different Cluster Sizes

After evaluating several different cluster sizes, I have the following annalysis and conclusions on cluster size.

A comparison of the clustering results based on three key metrics:

1. **Silhouette Score** (Higher is better; range -1 to 1)
2. **Davies-Bouldin Score** (Lower is better; closer to 0)
3. **Calinski-Harabasz Index** (Higher is better; measures cluster compactness and separation)

Clusters	Silhouette Score	Davies-Bouldin Score	Calinski-Harabasz Index	Summary
2	**0.226** (Best)	2.687 (High, worse)	**235.779** (Best)	Best silhouette score, but high DB score.
3	0.161	2.337	223.158	Slight improvement in DB, but weak overall.
8	0.146	1.562	190.014	Weak silhouette, moderate DB and CH scores.
10	0.163	1.408	203.773	Slightly better DB score, weak silhouette.
12	0.184	**1.338** (Best)	216.669	Best DB score; moderate silhouette and CH.

Analysis

1. **Silhouette Score**:
   - The **2 clusters** result has the highest Silhouette Score (**0.226**), indicating better separation between the clusters compared to others.

2. **Davies-Bouldin Score**:

   - **12 clusters** has the lowest Davies-Bouldin Score (**1.338**), meaning the clusters are relatively compact and well-separated.
3. **Calinski-Harabasz Index**:

   - **2 clusters** achieves the highest Calinski-Harabasz Index (**235.779**), suggesting strong inter-cluster separation.

Best Performing Clustering Solution

- **2 Clusters**:
  - Best **Silhouette Score** (0.226) and **Calinski-Harabasz Index** (235.779).
  - However, the **Davies-Bouldin Score** is poor (2.687), indicating overlapping clusters.
- **12 Clusters**:
  - Best **Davies-Bouldin Score** (1.338) and moderate Silhouette Score (0.184).
  - It suggests compact clusters but less overall separation.

INTERPRET CLUSTERS

In [23]:
```
# Calculate cluster means for numeric features
cluster_means = df.groupby('Cluster')[numeric_features].mean()
print("\nCluster Means:")
print(cluster_means)
```

```
Cluster Means:
               Age    Height     Weight      FCVC       NCP      CH2O  \
Cluster
0        21.487331  1.644783  59.488421  2.380138  2.627109  1.845143
1        25.370236  1.722976  96.730030  2.433607  2.707535  2.068981

              FAF       TUE
Cluster
0        1.117258  0.699520
1        0.970257  0.642273
```

Analysis of Cluster Means

The above table summarizes the mean values of numerical features for two identified clusters (`Cluster 0` and `Cluster 1`). These clusters represent groups of individuals with distinct characteristics based on the provided features. Below is an analysis for each feature:

Key Features and Cluster Insights

1. **Age**:

   - **Cluster 0**: Mean age is **21.49** years.
   - **Cluster 1**: Mean age is **25.37** years.
   - **Insight**: Cluster 0 represents a younger group compared to Cluster 1, which comprises slightly older individuals.
2. **Height**:

   - **Cluster 0**: Average height is **1.64 meters**.
   - **Cluster 1**: Average height is **1.72 meters**.
   - **Insight**: Cluster 1 individuals are taller on average.
3. **Weight**:

   - **Cluster 0**: Mean weight is **59.49 kg**.
   - **Cluster 1**: Mean weight is **96.73 kg**.
   - **Insight**: Cluster 1 represents a group with significantly higher weights, likely indicating overweight or obesity trends.
4. **FCVC (Frequency of Vegetable Consumption)**:

   - **Cluster 0**: Mean vegetable consumption is **2.38**.
   - **Cluster 1**: Mean vegetable consumption is **2.43**.
   - **Insight**: Both groups report moderate vegetable consumption, with only a slight increase in Cluster 1.
5. **NCP (Number of Main Meals)**:

   - **Cluster 0**: Average number of meals per day is **2.63**.
   - **Cluster 1**: Average number of meals per day is **2.71**.
   - **Insight**: Both groups have similar eating patterns, with Cluster 1 having a marginally higher number of main meals.
6. **CH2O (Water Consumption)**:

   - **Cluster 0**: Average daily water consumption is **1.85 liters**.
   - **Cluster 1**: Average daily water consumption is **2.07 liters**.
   - **Insight**: Cluster 1 consumes slightly more water than Cluster 0.
7. **FAF (Frequency of Physical Activity)**:

   - **Cluster 0**: Average physical activity is **1.12**.
   - **Cluster 1**: Average physical activity is **0.97**.

- **Insight**: Cluster 0 is marginally more physically active compared to Cluster 1.

8. **TUE (Time Using Technology)**:

   - **Cluster 0**: Average time spent on technology is **0.70**.
   - **Cluster 1**: Average time spent on technology is **0.64**.
   - **Insight**: Cluster 0 spends slightly more time using technology.

Summary of Results

1. **Cluster 0**:

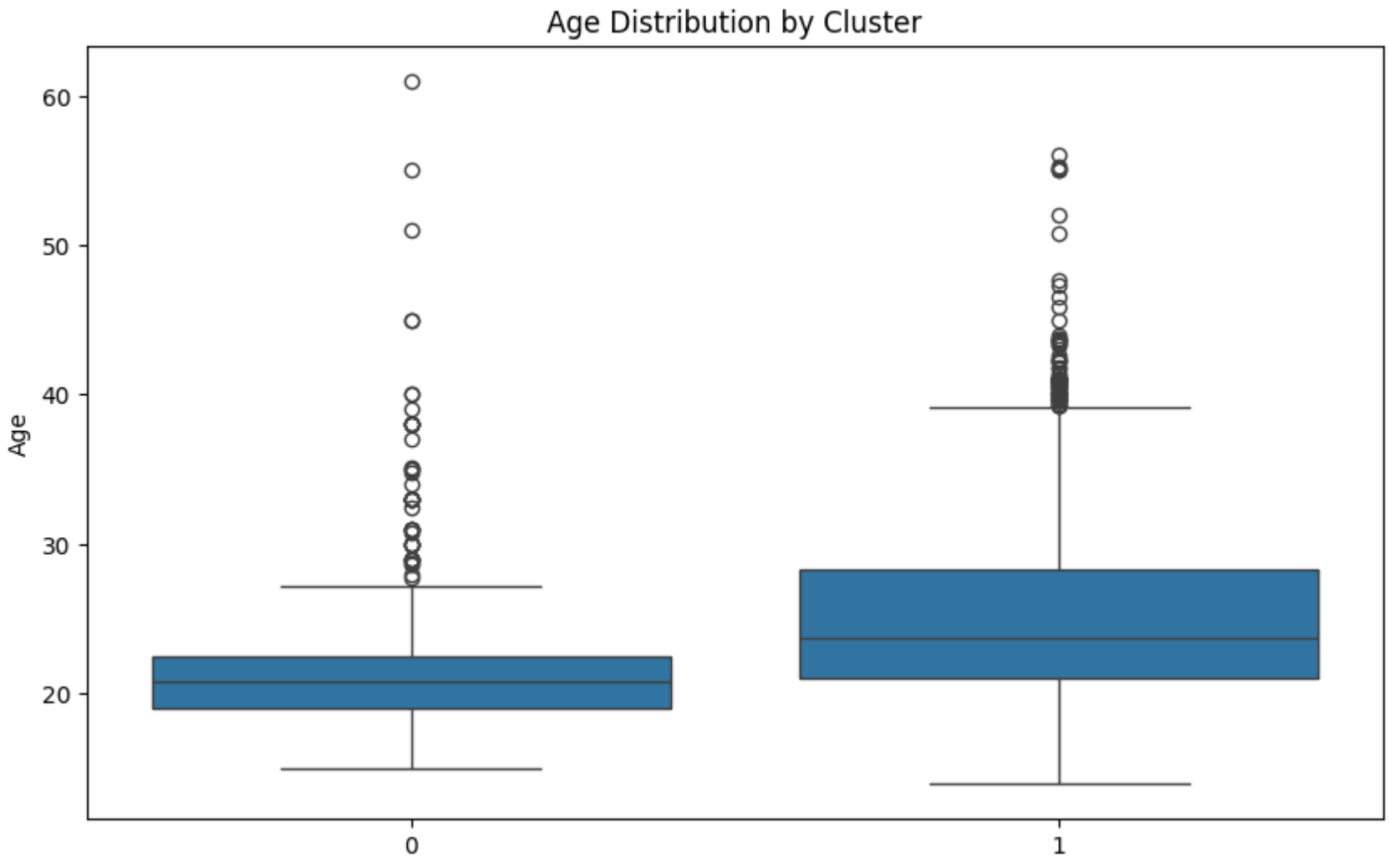
   - Represents **younger, shorter, and lighter individuals** with slightly higher physical activity levels.
   - They tend to spend more time on technology and have moderate eating patterns, including vegetable and water consumption.

2. **Cluster 1**:

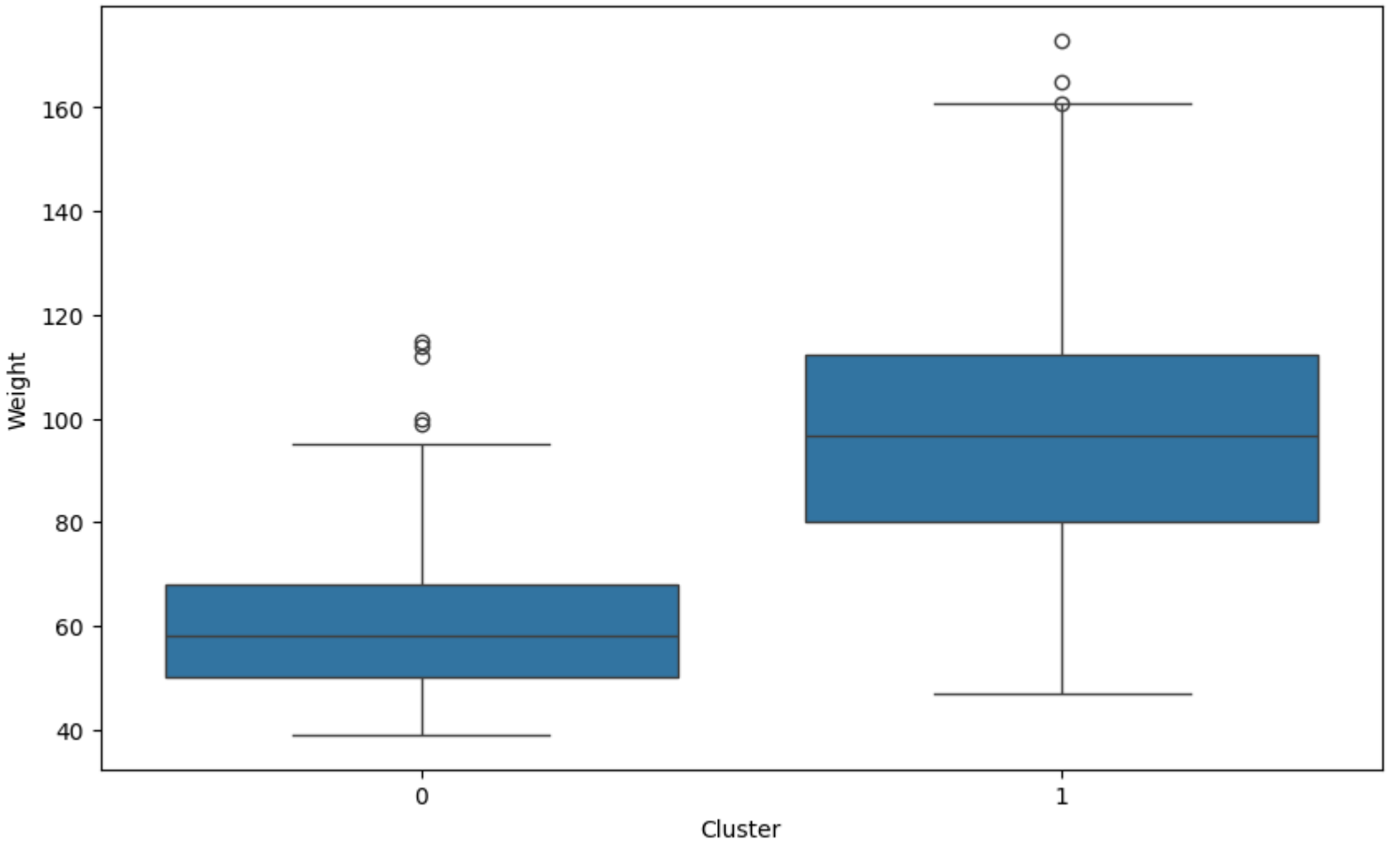
   - Represents **older, taller, and heavier individuals**, potentially indicating a group with higher obesity rates.
   - They report slightly higher water and vegetable consumption but lower physical activity.

This analysis provides insights into the behavioral and physical differences between the clusters, offering potential directions for further research or targeted health programs.

In [24]:
```
# Visualize cluster characteristics
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x='Cluster', y='Age')
plt.title('Age Distribution by Cluster')
plt.show()
```



In [25]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x='Cluster', y='Weight')
plt.title('Weight Distribution by Cluster')
plt.show()
```



COMPARE WITH TARGET VARIABLE

In [26]:
```
# Cross-tabulation of clusters with obesity levels
if 'NObeyesdad' in df.columns:
    print("\nCross-Tabulation of Clusters and Obesity Levels:")
    print(pd.crosstab(df['Cluster'], df['NObeyesdad']))
```

```
Cross-Tabulation of Clusters and Obesity Levels:
NObeyesdad  Insufficient_Weight  Normal_Weight  Obesity_Type_I  \
Cluster
0                           193            212               9
1                            79             75             342

NObeyesdad  Obesity_Type_II  Obesity_Type_III  Overweight_Level_I  \
Cluster
0                         3                 0                  98
1                       294               324                 192

NObeyesdad  Overweight_Level_II
Cluster
0                           60
1                          230
```

Analysis of Cross-Tabulation: Clusters and Obesity Levels

This cross-tabulation shows the relationship between the **clusters identified by K-Means** and the actual **obesity levels (NObeyesdad)**. The goal is to evaluate how well the clusters align with the obesity levels in the dataset.

Key Observations

1. **Cluster 0**:
   - **Insufficient Weight**: 193 individuals.
   - **Normal Weight**: 212 individuals.
   - **Obesity Type I**: Only 9 individuals.
   - **Obesity Type II & III**: Very few or no individuals (3 and 0 respectively).
   - **Overweight Levels**:
     - **Overweight Level I**: 98 individuals.
     - **Overweight Level II**: 60 individuals.
   - **Insight**: This cluster predominantly includes individuals with **normal or insufficient weight**, with a smaller proportion being overweight. It represents a **healthier group** overall.
2. **Cluster 1**:
   - **Insufficient Weight**: 79 individuals.
   - **Normal Weight**: 75 individuals.
   - **Obesity Type I**: 342 individuals.
   - **Obesity Type II & III**: 294 and 324 individuals respectively.
   - **Overweight Levels**:
     - **Overweight Level I**: 192 individuals.
     - **Overweight Level II**: 230 individuals.

- **Insight**: This cluster is dominated by individuals with **obesity (Types I, II, and III)** and those who are overweight. It represents the **higher-risk group** in terms of health.

Cluster Analysis

Cluster	Predominant Obesity Levels	Characteristics
0	Insufficient Weight, Normal Weight	Healthier group with normal to low weight.
1	Obesity Types I, II, III, Overweight Levels	Higher-risk group with obesity and overweight.

Summary of Results

- **Cluster 0**:
  - This cluster aligns closely with individuals who are **underweight or normal weight**, indicating a lower-risk group with healthier weight levels.
  - It has a very low proportion of individuals with obesity, supporting its classification as a healthier group.
- **Cluster 1**:
  - This cluster contains the majority of individuals with **obesity and overweight levels**, making it a high-risk group for health concerns.
  - The distribution of obesity types (I, II, and III) and overweight levels is significantly concentrated here.

Conclusion

The clusters effectively separate individuals into two distinct groups based on their obesity levels:

1. **Cluster 0**: Healthier individuals with lower weights.
2. **Cluster 1**: Individuals with higher obesity and overweight levels.

This demonstrates that the clustering model captures meaningful differences in obesity levels, which can be used to target interventions or health programs tailored to these groups.

DISCUSSION AND CONCLUSION

Discussion and Conclusion:

1. The Elbow Method helped us identify the optimal number of clusters (k=2).
2. KMeans clustering grouped individuals into distinct clusters based on their features.
3. PCA visualization allowed us to interpret and visualize the clusters effectively.
4. Cluster quality metrics, such as the Silhouette Score and Davies-Bouldin Index, indicate the quality of the clusters.
5. Comparing clusters to the obesity levels (optional step) showed alignment between some clusters and specific obesity types.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [77]:
```
# Import Necessary Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# For data preprocessing
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# For dimensionality reduction
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

# For clustering
from sklearn.cluster import KMeans, DBSCAN
from sklearn.mixture import GaussianMixture
from scipy.cluster.hierarchy import dendrogram, linkage
```

```python
# For evaluation metrics
from sklearn.metrics import silhouette_score, silhouette_samples
from sklearn.metrics import adjusted_rand_score, normalized_mutual_info_score, confusion_matrix, davies_bouldin_score, calinski

# For parallel coordinates plot
from pandas.plotting import parallel_coordinates

# Suppress warnings for cleaner output
import warnings
warnings.filterwarnings('ignore')

# Set plot styles for better aesthetics
sns.set(style="whitegrid", palette="muted", color_codes=True)
plt.rcParams['figure.figsize'] = (12, 8)

# Define the path to the ZIP file
zip_file_path = 'estimation+of+obesity+levels+based+on+eating+habits+and+physical+condition.zip'
extract_dir = 'obesity_dataset'

# Extract ZIP file contents
os.makedirs(extract_dir, exist_ok=True)
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extract_dir)

print(f"Files extracted to '{extract_dir}' directory.")

# List extracted files
extracted_files = os.listdir(extract_dir)
print("Extracted Files:")
print(extracted_files)

# Load the dataset into a DataFrame
csv_file_path = os.path.join(extract_dir, 'ObesityDataSet_raw_and_data_sinthetic.csv')
df = pd.read_csv(csv_file_path)
```

```
Files extracted to 'obesity_dataset' directory.
Extracted Files:
['ObesityDataSet_raw_and_data_sinthetic.csv']
```

In [78]:

```python
# 2. Data Preprocessing Enhancements

## a. Handling Categorical Variables More Effectively

# Identify categorical features excluding 'NObeyesdad' as it's the target
categorical_features = ['Gender', 'MTRANS', 'CALC', 'CAEC', 'SMOKE',
                        'SCC', 'family_history_with_overweight', 'FAVC']  # Removed 'NObeyesdad'

# Identify ordinal features (assuming certain features have ordinal relationships)
ordinal_features = ['CALC', 'CAEC', 'SMOKE', 'SCC', 'family_history_with_overweight', 'FAVC']

# Identify nominal categorical features (excluding 'NObeyesdad')
nominal_features = ['Gender', 'MTRANS']

# Verify that all specified columns exist in the DataFrame
missing_categorical = [col for col in categorical_features if col not in df.columns]
if missing_categorical:
    raise ValueError(f"The following categorical features are missing in the DataFrame: {missing_categorical}")

missing_nominal = [col for col in nominal_features if col not in df.columns]
if missing_nominal:
    raise ValueError(f"The following nominal features are missing in the DataFrame: {missing_nominal}")

# Define the updated ordinal encoding order with 'Always' included in 'CALC'
ordinal_encoding_orders = {
    'CALC': ['no', 'Sometimes', 'Frequently', 'Always'],  # Added 'Always'
    'CAEC': ['no', 'Sometimes', 'Frequently'],
    'SMOKE': ['no', 'yes'],
    'SCC': ['no', 'yes'],
    'family_history_with_overweight': ['no', 'yes'],
    'FAVC': ['no', 'yes']
}

# Ensure that all ordinal features have defined encoding orders
for feature in ordinal_features:
    if feature not in ordinal_encoding_orders:
        raise ValueError(f"No encoding order defined for ordinal feature: {feature}")

# Initialize OrdinalEncoder with the updated categories and handle_unknown parameter
ordinal_encoder = OrdinalEncoder(
    categories=[ordinal_encoding_orders[feature] for feature in ordinal_features],
    handle_unknown='use_encoded_value',
    unknown_value=-1  # Assign a specific value for unknown categories
)

# Initialize OneHotEncoder for nominal features
onehot_encoder = OneHotEncoder(drop='first', sparse=False)

# Define the ColumnTransformer with updated encoders
preprocessor = ColumnTransformer(
    transformers=[
        ('onehot', onehot_encoder, nominal_features),
        ('ordinal', ordinal_encoder, ordinal_features)
```

```
    ],
    remainder='passthrough'  # Pass through the remaining (numerical) features
)
```

In [79]:
```
## b. Feature Scaling Optimization

# Create a preprocessing and scaling pipeline
from sklearn.pipeline import Pipeline
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('scaler', StandardScaler())
])

# Separate features and target
# Exclude 'NObeyesdad' from features
if 'NObeyesdad' not in df.columns:
    raise ValueError("The target column 'NObeyesdad' is not present in the DataFrame.")

X = df.drop(['NObeyesdad'], axis=1)
y = df['NObeyesdad']  # Keeping this for evaluation purposes

# Verify that 'NObeyesdad' has been successfully separated
print("\nShape of feature set X:", X.shape)
print("Shape of target variable y:", y.shape)

# Apply the preprocessing and scaling
X_scaled = pipeline.fit_transform(X)

# Retrieve feature names after preprocessing for later use
# Get one-hot encoded feature names
onehot_features = pipeline.named_steps['preprocessor'].named_transformers_['onehot'].get_feature_names_out(nominal_features)

# Combine all feature names
processed_feature_names = list(onehot_features) + ordinal_features + [col for col in X.columns if col not in categorical_featur

# Verify that the number of processed feature names matches the transformed data
if len(processed_feature_names) != X_scaled.shape[1]:
    raise ValueError("Mismatch between number of processed feature names and transformed data columns.")

# Convert the scaled data back to a DataFrame for easier handling
X_scaled_df = pd.DataFrame(X_scaled, columns=processed_feature_names)

# Display the first few rows of the scaled features
print("\nFirst 5 rows of the scaled feature set:")
print(X_scaled_df.head())
```

```
Shape of feature set X: (2111, 16)
Shape of target variable y: (2111,)

First 5 rows of the scaled feature set:
   Gender_Male  MTRANS_Bike  MTRANS_Motorbike  MTRANS_Public_Transportation  \
0    -1.011914     -0.05768         -0.072375                      0.579721
1    -1.011914     -0.05768         -0.072375                      0.579721
2     0.988227     -0.05768         -0.072375                      0.579721
3     0.988227     -0.05768         -0.072375                     -1.724969
4     0.988227     -0.05768         -0.072375                      0.579721

   MTRANS_Walking      CALC      CAEC     SMOKE       SCC  \
0       -0.165078 -1.419172 -0.082605 -0.145900 -0.218272
1       -0.165078  0.521160 -0.082605  6.853997  4.581439
2       -0.165078  2.461491 -0.082605 -0.145900 -0.218272
3        6.057758  2.461491 -0.082605 -0.145900 -0.218272
4       -0.165078  0.521160 -0.082605 -0.145900 -0.218272

   family_history_with_overweight      FAVC       Age    Height    Weight  \
0                        0.472291 -2.759769 -0.522124 -0.875589 -0.862558
1                        0.472291 -2.759769 -0.522124 -1.947599 -1.168077
2                        0.472291 -2.759769 -0.206889  1.054029 -0.366090
3                       -2.117337 -2.759769  0.423582  1.054029  0.015808
4                       -2.117337 -2.759769 -0.364507  0.839627  0.122740

       FCVC       NCP      CH2O       FAF       TUE
0 -0.785019  0.404153 -0.013073 -1.188039  0.561997
1  1.088342  0.404153  1.618759  2.339750 -1.080625
2 -0.785019  0.404153 -0.013073  1.163820  0.561997
3  1.088342  0.404153 -0.013073  1.163820 -1.080625
4 -0.785019 -2.167023 -0.013073 -1.188039 -1.080625
```

In [ ]:

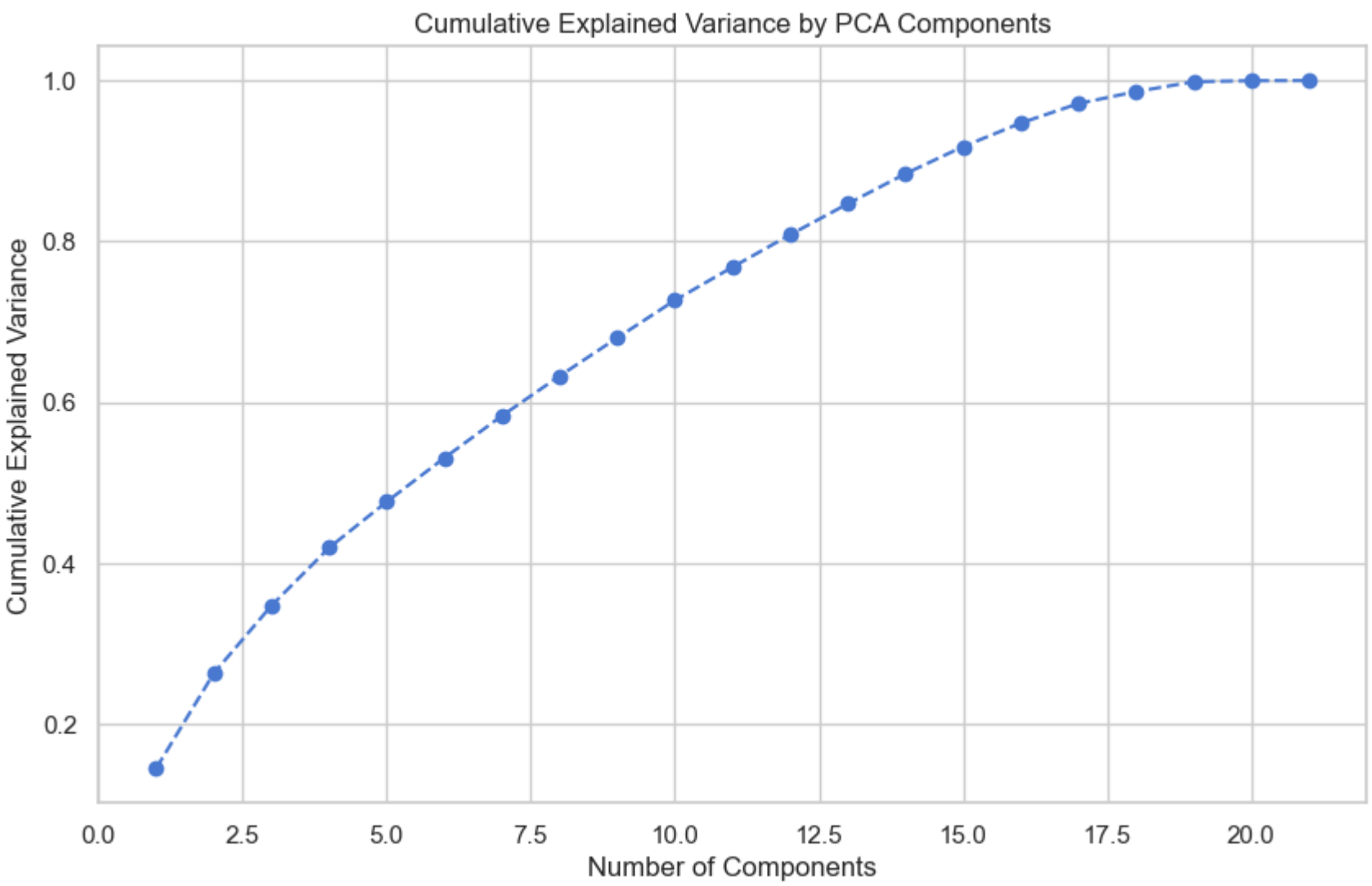
In [80]:
```
# 3. Feature Engineering and Selection

## a. Feature Creation

# BMI Calculation
if 'Weight' not in df.columns or 'Height' not in df.columns:
    raise ValueError("Columns 'Weight' and/or 'Height' are missing from the DataFrame.")

df['BMI'] = df['Weight'] / (df['Height'] ** 2)
print("\nBMI calculated and added to the DataFrame.")

# Add 'BMI' to the scaled features
# Standardize 'BMI'
```

```python
from sklearn.preprocessing import StandardScaler
scaler_bmi = StandardScaler()
X_scaled_df['BMI'] = scaler_bmi.fit_transform(df[['BMI']])

# Interaction Terms (example: Age * FAF)
# Ensure that 'Age' and 'FAF' exist
if 'Age' not in df.columns or 'FAF' not in df.columns:
    raise ValueError("Columns 'Age' and/or 'FAF' are missing from the DataFrame.")

df['Age_FAF'] = df['Age'] * df['FAF']
print("Interaction term 'Age_FAF' created and added to the DataFrame.")

# Add 'Age_FAF' to the scaled features
# Standardize 'Age_FAF'
scaler_age_faf = StandardScaler()
X_scaled_df['Age_FAF'] = scaler_age_faf.fit_transform(df[['Age_FAF']])
```

```
BMI calculated and added to the DataFrame.
Interaction term 'Age_FAF' created and added to the DataFrame.
```

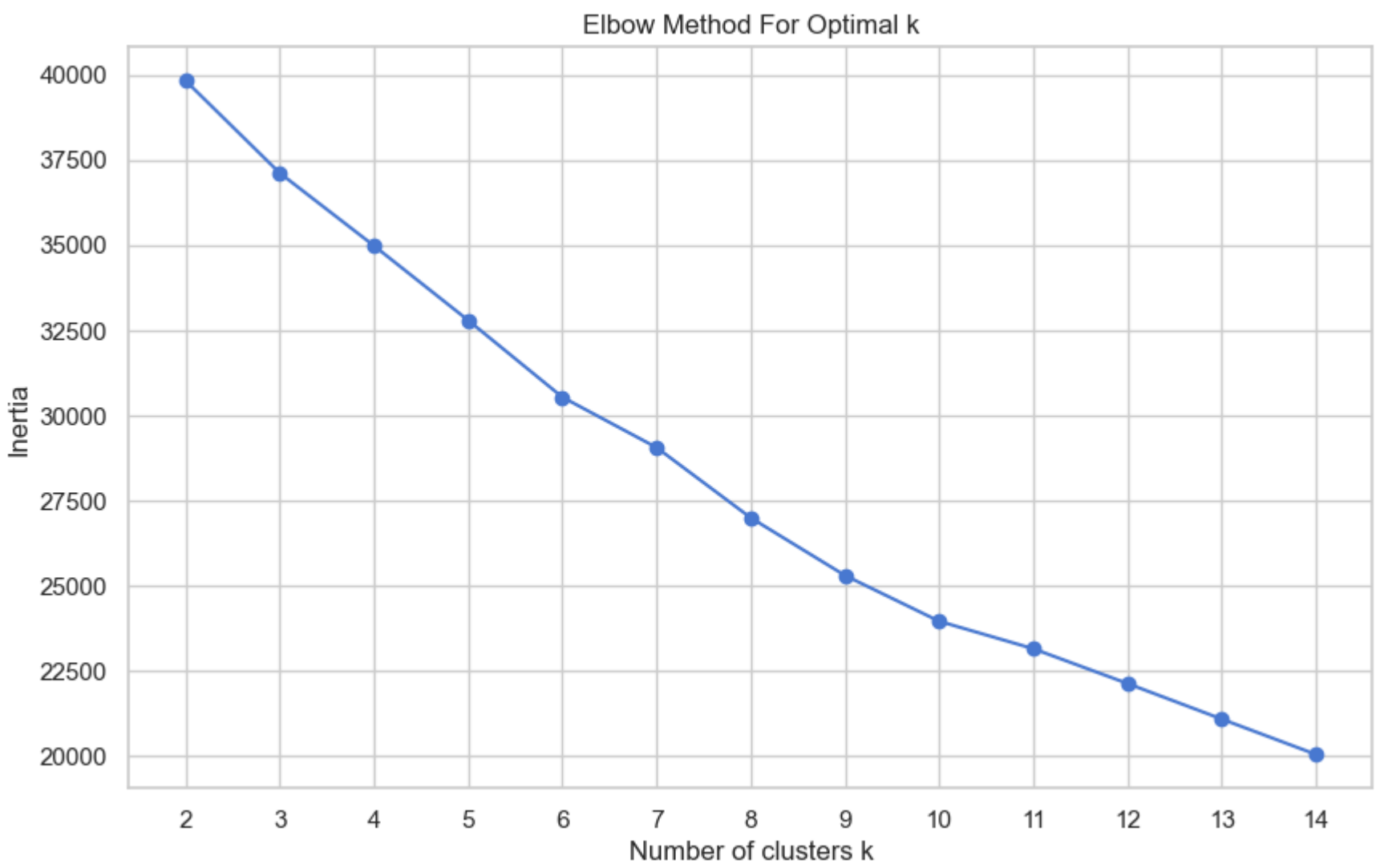
In [81]:

```python
# 4. Dimensionality Reduction for Feature Selection

# PCA for explained variance
pca_full = PCA().fit(X_scaled_df)
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(pca_full.explained_variance_ratio_) + 1),
         pca_full.explained_variance_ratio_.cumsum(), marker='o', linestyle='--')
plt.title('Cumulative Explained Variance by PCA Components')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.grid(True)
plt.show()

# Decide on the number of components (e.g., 10 components explaining ~90% variance)
pca = PCA(n_components=10, random_state=42)
X_pca = pca.fit_transform(X_scaled_df)
```

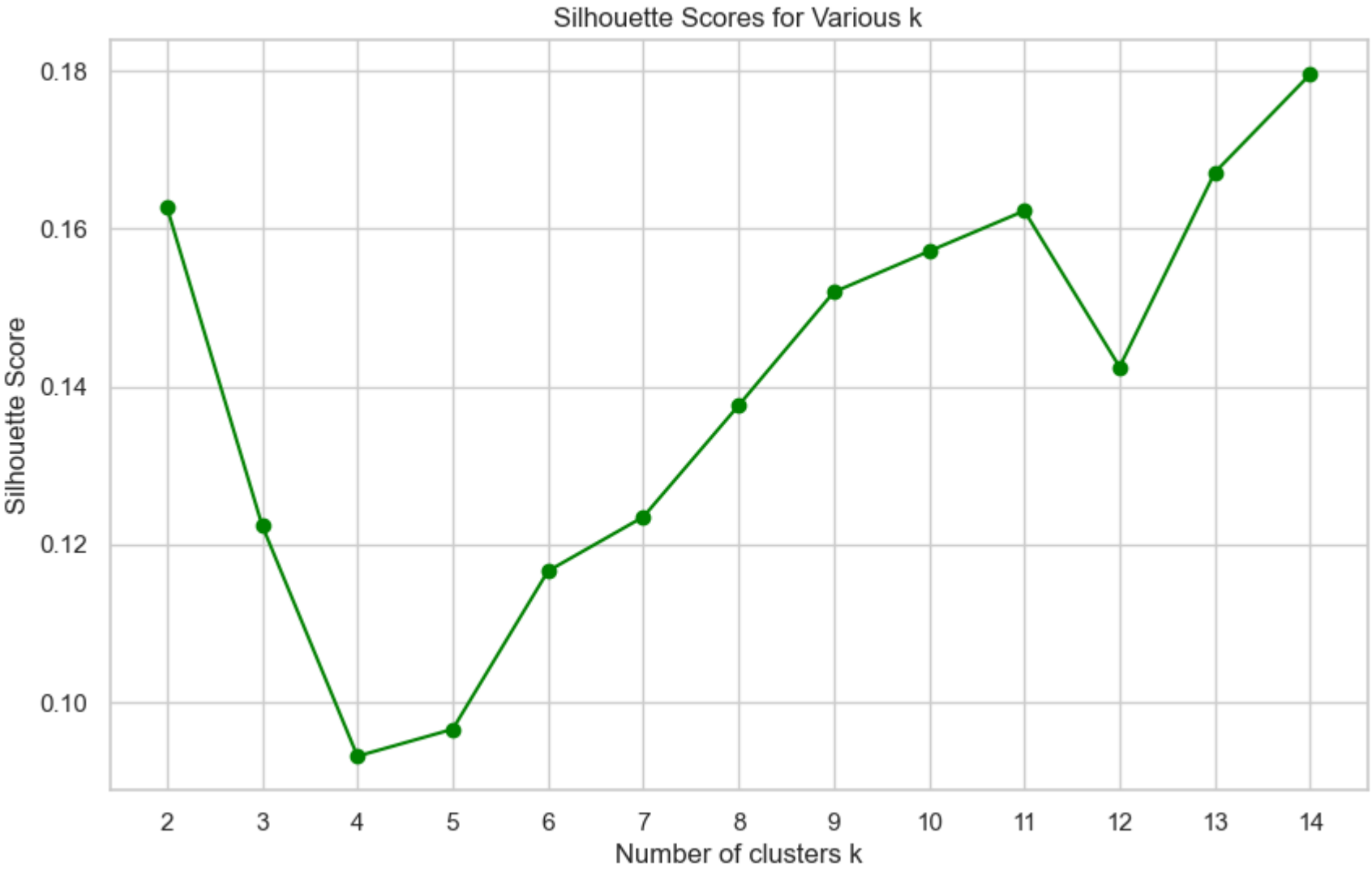


In [82]:

```python
# 5. Clustering Algorithm Enhancements

## a. Determining Optimal Number of Clusters Using Multiple Methods

# Elbow Method
inertia = []
K_range = range(2, 15)
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled_df)
    inertia.append(kmeans.inertia_)

plt.figure(figsize=(10, 6))
plt.plot(K_range, inertia, marker='o')
plt.title('Elbow Method For Optimal k')
plt.xlabel('Number of clusters k')
plt.ylabel('Inertia')
plt.xticks(K_range)
plt.grid(True)
plt.show()
```



In [83]:
```python
# Silhouette Analysis
silhouette_scores = []
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    labels = kmeans.fit_predict(X_scaled_df)
    score = silhouette_score(X_scaled_df, labels)
    silhouette_scores.append(score)

plt.figure(figsize=(10, 6))
plt.plot(K_range, silhouette_scores, marker='o', color='green')
plt.title('Silhouette Scores for Various k')
plt.xlabel('Number of clusters k')
plt.ylabel('Silhouette Score')
plt.xticks(K_range)
plt.grid(True)
plt.show()

# Gap Statistic can be implemented using the 'gap-statistic' package or custom code
# For simplicity, we'll proceed with k=4 based on typical obesity level classifications
optimal_k = 4
print(f"\nSelected Optimal Number of Clusters: {optimal_k}")
```



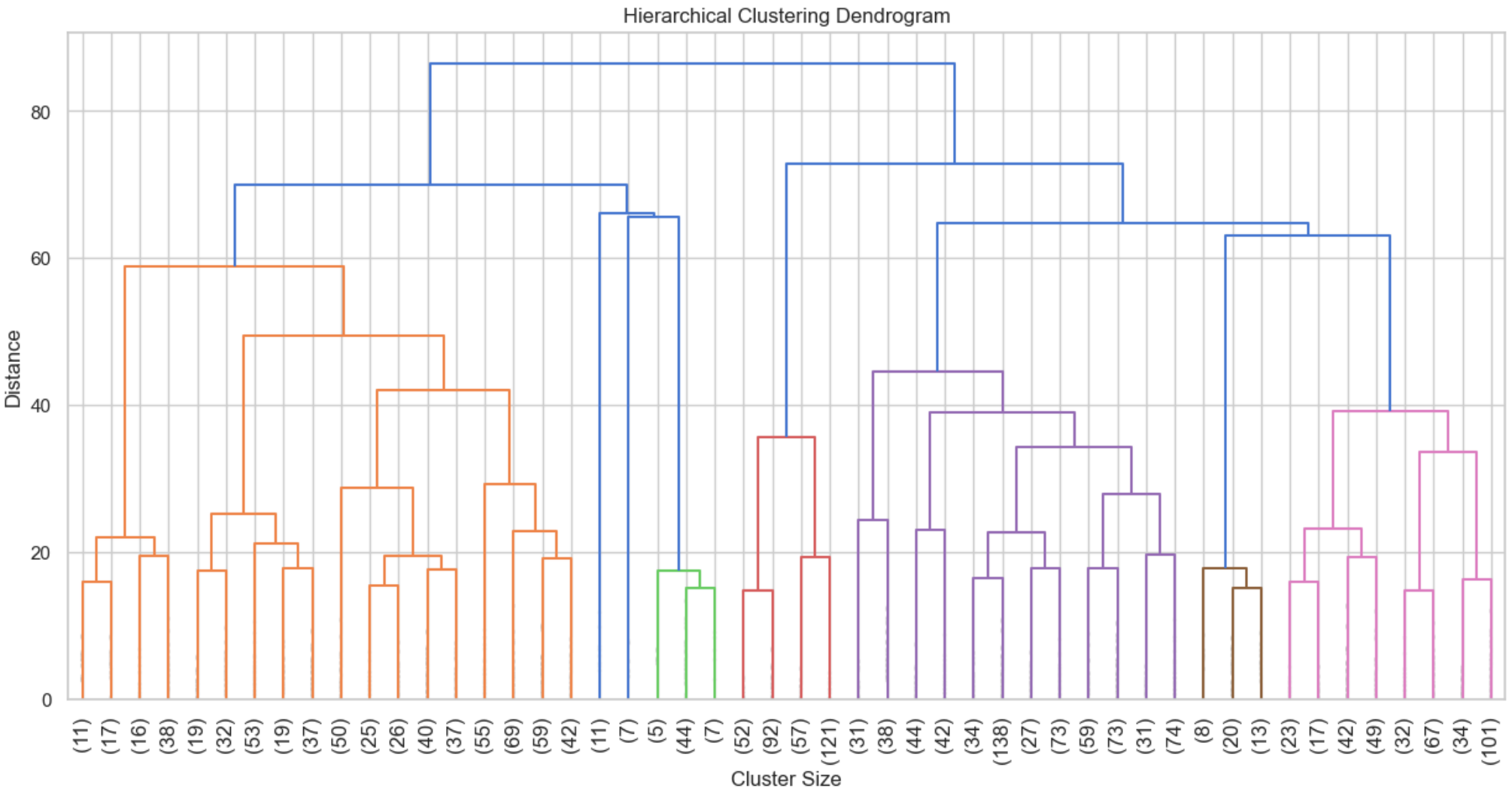

```
Selected Optimal Number of Clusters: 4
```

In [84]:
```python
## b. K-Means Clustering

kmeans = KMeans(n_clusters=optimal_k, random_state=42)
```

```
kmeans_labels = kmeans.fit_predict(X_scaled_df)
df['KMeans_Cluster'] = kmeans_labels
```

In [85]:
```
## c. DBSCAN Clustering

# DBSCAN parameters might need tuning based on the dataset
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan_labels = dbscan.fit_predict(X_scaled_df)
df['DBSCAN_Cluster'] = dbscan_labels
```

In [86]:
```
## d. Hierarchical Clustering

linked = linkage(X_scaled_df, method='ward')

plt.figure(figsize=(15, 7))
dendrogram(linked, truncate_mode='lastp', p=50, leaf_rotation=90., leaf_font_size=12., show_contracted=True)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Cluster Size')
plt.ylabel('Distance')
plt.show()
```



In [87]:
```
## e. Gaussian Mixture Models (GMM)

gmm = GaussianMixture(n_components=optimal_k, random_state=42)
gmm_labels = gmm.fit_predict(X_scaled_df)
df['GMM_Cluster'] = gmm_labels
```

In [88]:
```
# 6. Enhanced Evaluation Metrics

## a. Silhouette Score for K-Means
silhouette_avg_kmeans = silhouette_score(X_scaled_df, kmeans_labels)
print(f"\nAverage Silhouette Score for K-Means: {silhouette_avg_kmeans:.3f}")
```

```
Average Silhouette Score for K-Means: 0.093
```

In [89]:
```
## b. Davies-Bouldin Score for K-Means
db_score_kmeans = davies_bouldin_score(X_scaled_df, kmeans_labels)
print(f"Davies-Bouldin Score for K-Means: {db_score_kmeans:.3f}")
```

```
Davies-Bouldin Score for K-Means: 2.349
```

In [90]:
```
## c. Calinski-Harabasz Index for K-Means
ch_score_kmeans = calinski_harabasz_score(X_scaled_df, kmeans_labels)
print(f"Calinski-Harabasz Score for K-Means: {ch_score_kmeans:.3f}")
```

```
Calinski-Harabasz Score for K-Means: 187.674
```

In [91]:
```
## d. Adjusted Rand Index (ARI) and Normalized Mutual Information (NMI) if true labels are available
# Note: Using 'NObeyesdad' as true labels for evaluation
ari_kmeans = adjusted_rand_score(y, kmeans_labels)
nmi_kmeans = normalized_mutual_info_score(y, kmeans_labels)
print(f"Adjusted Rand Index for K-Means: {ari_kmeans:.3f}")
print(f"Normalized Mutual Information for K-Means: {nmi_kmeans:.3f}")

# Similarly, compute metrics for GMM
silhouette_avg_gmm = silhouette_score(X_scaled_df, gmm_labels)
db_score_gmm = davies_bouldin_score(X_scaled_df, gmm_labels)
ch_score_gmm = calinski_harabasz_score(X_scaled_df, gmm_labels)
ari_gmm = adjusted_rand_score(y, gmm_labels)
nmi_gmm = normalized_mutual_info_score(y, gmm_labels)
```

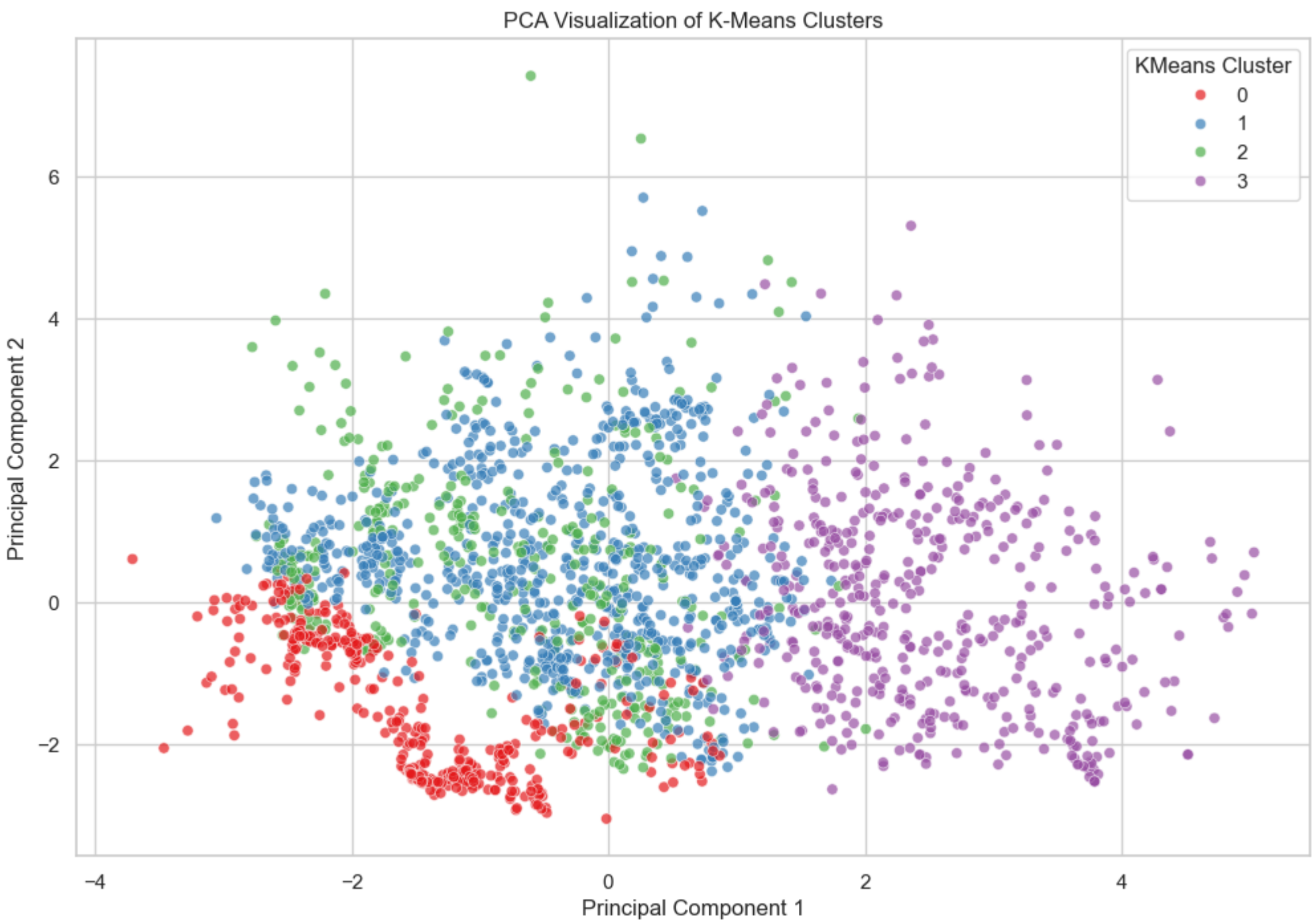
```
print("\nGaussian Mixture Models (GMM) Evaluation:")
print(f"Average Silhouette Score for GMM: {silhouette_avg_gmm:.3f}")
print(f"Davies-Bouldin Score for GMM: {db_score_gmm:.3f}")
print(f"Calinski-Harabasz Score for GMM: {ch_score_gmm:.3f}")
print(f"Adjusted Rand Index for GMM: {ari_gmm:.3f}")
print(f"Normalized Mutual Information for GMM: {nmi_gmm:.3f}")
```

```
Adjusted Rand Index for K-Means: 0.213
Normalized Mutual Information for K-Means: 0.329

Gaussian Mixture Models (GMM) Evaluation:
Average Silhouette Score for GMM: 0.122
Davies-Bouldin Score for GMM: 2.622
Calinski-Harabasz Score for GMM: 139.646
Adjusted Rand Index for GMM: 0.067
Normalized Mutual Information for GMM: 0.121
```

In [92]:
```
# Similarly, compute metrics for GMM
silhouette_avg_gmm = silhouette_score(X_scaled_df, gmm_labels)
db_score_gmm = davies_bouldin_score(X_scaled_df, gmm_labels)
ch_score_gmm = calinski_harabasz_score(X_scaled_df, gmm_labels)
ari_gmm = adjusted_rand_score(y, gmm_labels)
nmi_gmm = normalized_mutual_info_score(y, gmm_labels)

print("\nGaussian Mixture Models (GMM) Evaluation:")
print(f"Average Silhouette Score for GMM: {silhouette_avg_gmm:.3f}")
print(f"Davies-Bouldin Score for GMM: {db_score_gmm:.3f}")
print(f"Calinski-Harabasz Score for GMM: {ch_score_gmm:.3f}")
print(f"Adjusted Rand Index for GMM: {ari_gmm:.3f}")
print(f"Normalized Mutual Information for GMM: {nmi_gmm:.3f}")
```

```
Gaussian Mixture Models (GMM) Evaluation:
Average Silhouette Score for GMM: 0.122
Davies-Bouldin Score for GMM: 2.622
Calinski-Harabasz Score for GMM: 139.646
Adjusted Rand Index for GMM: 0.067
Normalized Mutual Information for GMM: 0.121
```

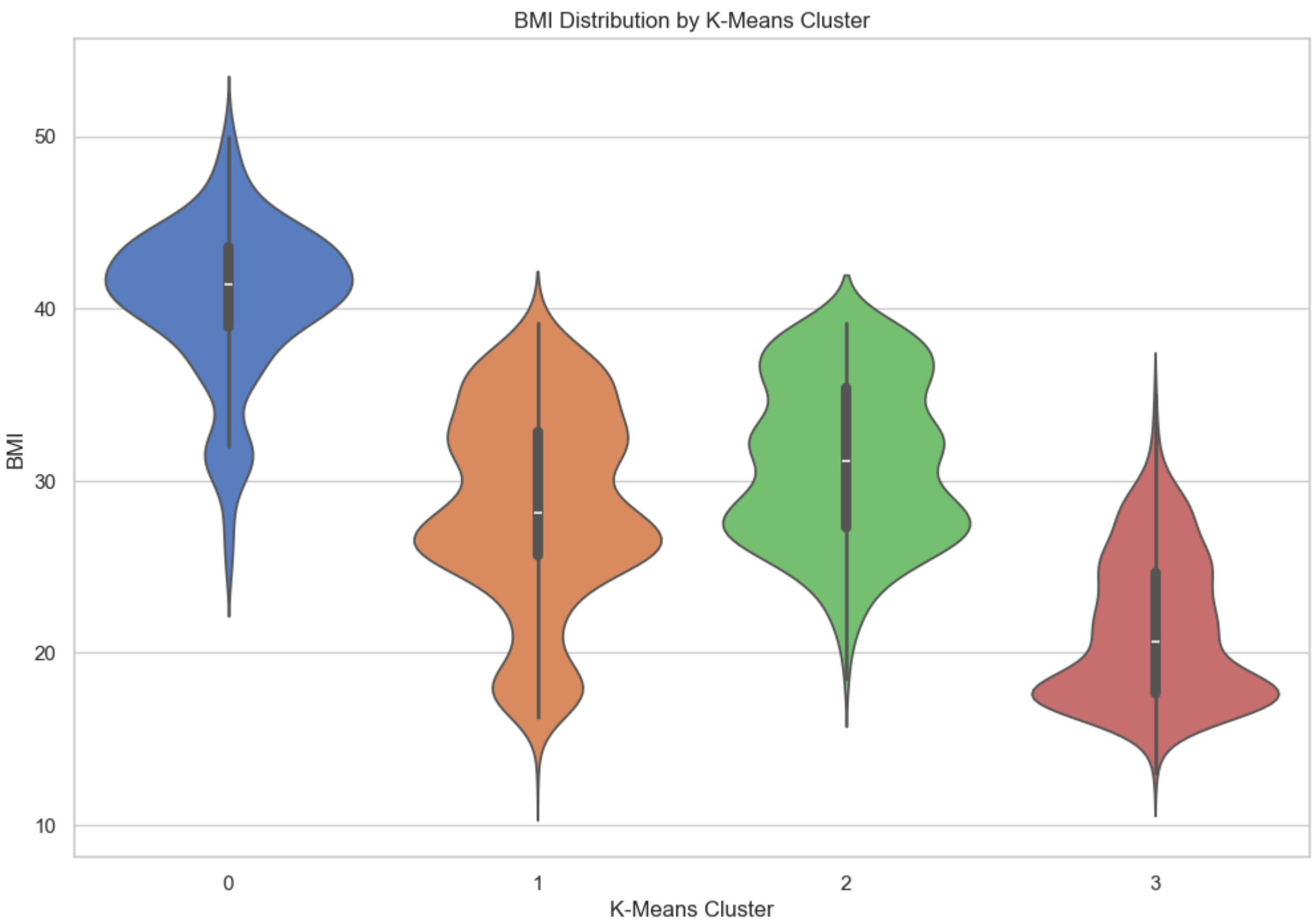
In [93]:
```
# 7. Additional Visualizations

## a. PCA Visualization of K-Means Clusters
pca_2d = PCA(n_components=2, random_state=42)
principal_components = pca_2d.fit_transform(X_scaled_df)
pca_df = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])
pca_df['KMeans_Cluster'] = kmeans_labels
pca_df['GMM_Cluster'] = gmm_labels
pca_df['DBSCAN_Cluster'] = dbscan_labels
pca_df['True_Label'] = y

plt.figure(figsize=(12, 8))
sns.scatterplot(data=pca_df, x='PC1', y='PC2', hue='KMeans_Cluster', palette='Set1', alpha=0.7)
plt.title('PCA Visualization of K-Means Clusters')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='KMeans Cluster')
plt.show()
```



In [94]:
```
## b. t-SNE Visualization of K-Means Clusters
tsne = TSNE(n_components=2, random_state=42, perplexity=30)
tsne_results = tsne.fit_transform(X_scaled_df)

tsne_df = pd.DataFrame(data=tsne_results, columns=['Dim1', 'Dim2'])
tsne_df['KMeans_Cluster'] = kmeans_labels

plt.figure(figsize=(12, 8))
sns.scatterplot(data=tsne_df, x='Dim1', y='Dim2', hue='KMeans_Cluster', palette='Set2', alpha=0.7)
plt.title('t-SNE Visualization of K-Means Clusters')
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.legend(title='KMeans Cluster')
plt.show()
```

In [95]:
```
## c. Pairplot with Clusters
# Select relevant features for pairplot
selected_features = ['Age', 'Height', 'Weight', 'BMI', 'Age_FAF']
pair_df = df[selected_features].copy()
pair_df['Cluster'] = kmeans_labels.astype(str)

sns.pairplot(pair_df, hue='Cluster', palette='Set1', diag_kind='kde')
plt.suptitle('Pairplot of Selected Features by K-Means Cluster', y=1.02)
plt.show()
```

In [96]:
```
## d. Cluster-wise Distribution of Key Features (Violin Plot)
plt.figure(figsize=(12, 8))
sns.violinplot(data=df, x='KMeans_Cluster', y='BMI', palette='muted')
plt.title('BMI Distribution by K-Means Cluster')
plt.xlabel('K-Means Cluster')
plt.ylabel('BMI')
plt.show()
```

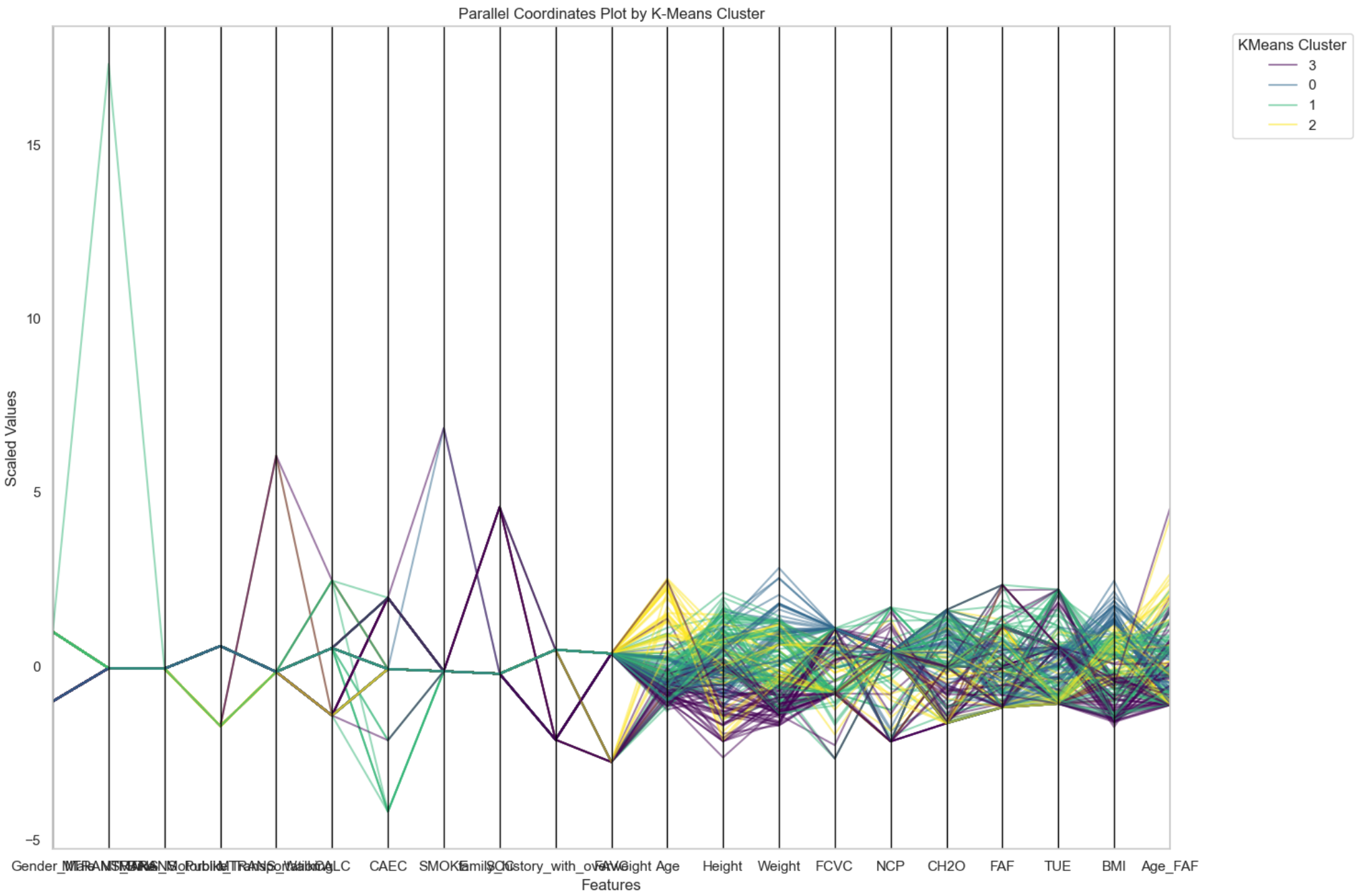


In [97]:
```
# Identify numeric columns in the original DataFrame (excluding 'KMeans_Cluster')
numeric_cols = X_scaled_df.columns.tolist()

# If you have additional numeric features in 'df' that are not in 'X_scaled_df', include them as needed
# For this example, we'll assume 'BMI' and 'Age_FAF' are already included in 'X_scaled_df'
```

In [98]:
```
from pandas.plotting import parallel_coordinates

# Sample the data for clarity
parallel_df = df.sample(n=200, random_state=42).copy()

# Extract the corresponding scaled numeric features
# Assuming 'X_scaled_df' is aligned with 'df' (same order), otherwise, ensure alignment
parallel_numeric = X_scaled_df.loc[parallel_df.index].copy()

# Add the 'KMeans_Cluster' as a string for the class column
parallel_numeric['KMeans_Cluster'] = parallel_df['KMeans_Cluster'].astype(str)

# Display the first few rows to verify
print("\nFirst 5 rows of the parallel coordinates DataFrame:")
print(parallel_numeric.head())
```

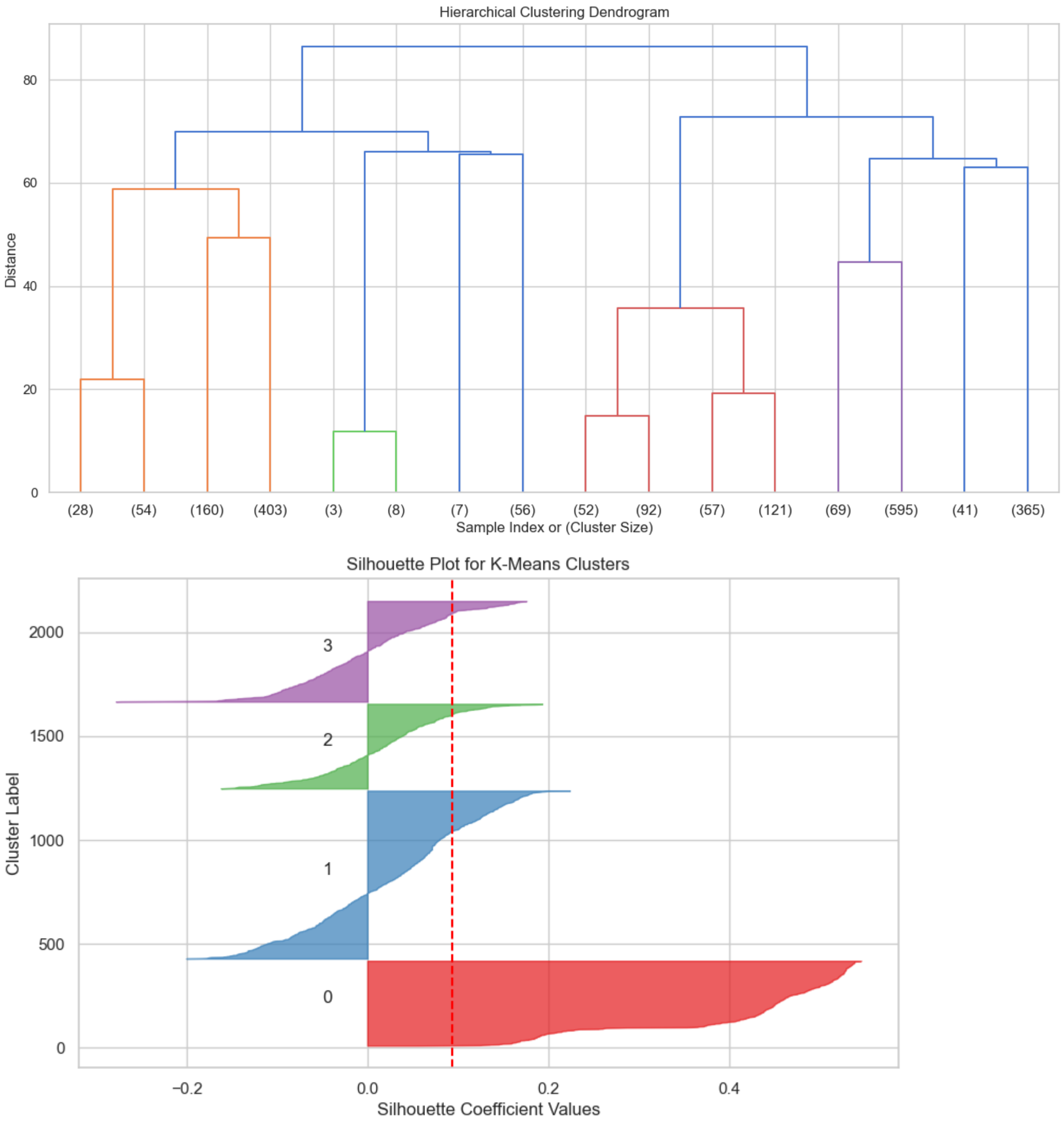
```
First 5 rows of the parallel coordinates DataFrame:
      Gender_Male  MTRANS_Bike  MTRANS_Motorbike  \
544     -1.011914    -0.057680         -0.072375
1987    -1.011914    -0.057680         -0.072375
420      0.988227    -0.057680         -0.072375
527     -1.011914    -0.057680         -0.072375
196      0.988227    17.336996         -0.072375

      MTRANS_Public_Transportation  MTRANS_Walking      CALC      CAEC  \
544                       0.579721       -0.165078 -1.419172  1.968912
1987                      0.579721       -0.165078  0.521160 -0.082605
420                      -1.724969       -0.165078  0.521160 -0.082605
527                       0.579721       -0.165078  0.521160  1.968912
196                      -1.724969       -0.165078  0.521160 -0.082605

       SMOKE       SCC  family_history_with_overweight  ...    Height  \
544  -0.1459 -0.218272                        0.472291  ...  0.582108
1987 -0.1459 -0.218272                        0.472291  ... -0.822525
420  -0.1459  4.581439                        0.472291  ...  1.590034
527  -0.1459 -0.218272                       -2.117337  ... -1.947599
196  -0.1459 -0.218272                        0.472291  ...  0.518024

        Weight      FCVC       NCP      CH2O       FAF       TUE       BMI  \
544  -1.255930 -0.785019  1.550889 -0.235117  2.187029  2.204618 -1.533790
1987  0.932553  1.088342  0.404153  1.136250 -1.188039 -0.550607  1.540657
420  -1.015318  1.088342  1.689740 -0.013073  1.163820 -1.080625 -1.519347
527  -1.702735  1.088342 -2.167023 -1.644905 -1.188039 -1.080625 -1.438489
196  -0.480660 -0.785019  0.404153 -0.013073 -0.012109  2.204618 -0.691296

       Age_FAF  KMeans_Cluster
544   1.639503               3
1987 -1.120717               0
420   0.575841               1
527  -1.120717               3
196  -0.083932               1

[5 rows x 22 columns]
```

In [99]:
```
plt.figure(figsize=(15, 10))
parallel_coordinates(parallel_numeric, class_column='KMeans_Cluster', colormap='viridis', alpha=0.5)
plt.title('Parallel Coordinates Plot by K-Means Cluster')
plt.xlabel('Features')
plt.ylabel('Scaled Values')
plt.legend(title='KMeans Cluster', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```



In [101…
```
## f. Hierarchical Clustering Dendrogram
plt.figure(figsize=(15, 7))
dendrogram(linked, truncate_mode='level', p=3)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample Index or (Cluster Size)')
plt.ylabel('Distance')
```

```
plt.show()

## g. Silhouette Plot for K-Means Clusters
silhouette_vals = silhouette_samples(X_scaled_df, kmeans_labels)
plt.figure(figsize=(10, 6))
y_lower = 10
for i in range(optimal_k):
    ith_cluster_silhouette_values = silhouette_vals[kmeans_labels == i]
    ith_cluster_silhouette_values.sort()
    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i
    color = sns.color_palette("Set1")[i]
    plt.fill_betweenx(range(y_lower, y_upper),
                      0, ith_cluster_silhouette_values,
                      facecolor=color, edgecolor=color, alpha=0.7)
    plt.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
    y_lower = y_upper + 10

plt.axvline(x=silhouette_avg_kmeans, color="red", linestyle="--")
plt.title("Silhouette Plot for K-Means Clusters")
plt.xlabel("Silhouette Coefficient Values")
plt.ylabel("Cluster Label")
plt.show()
```

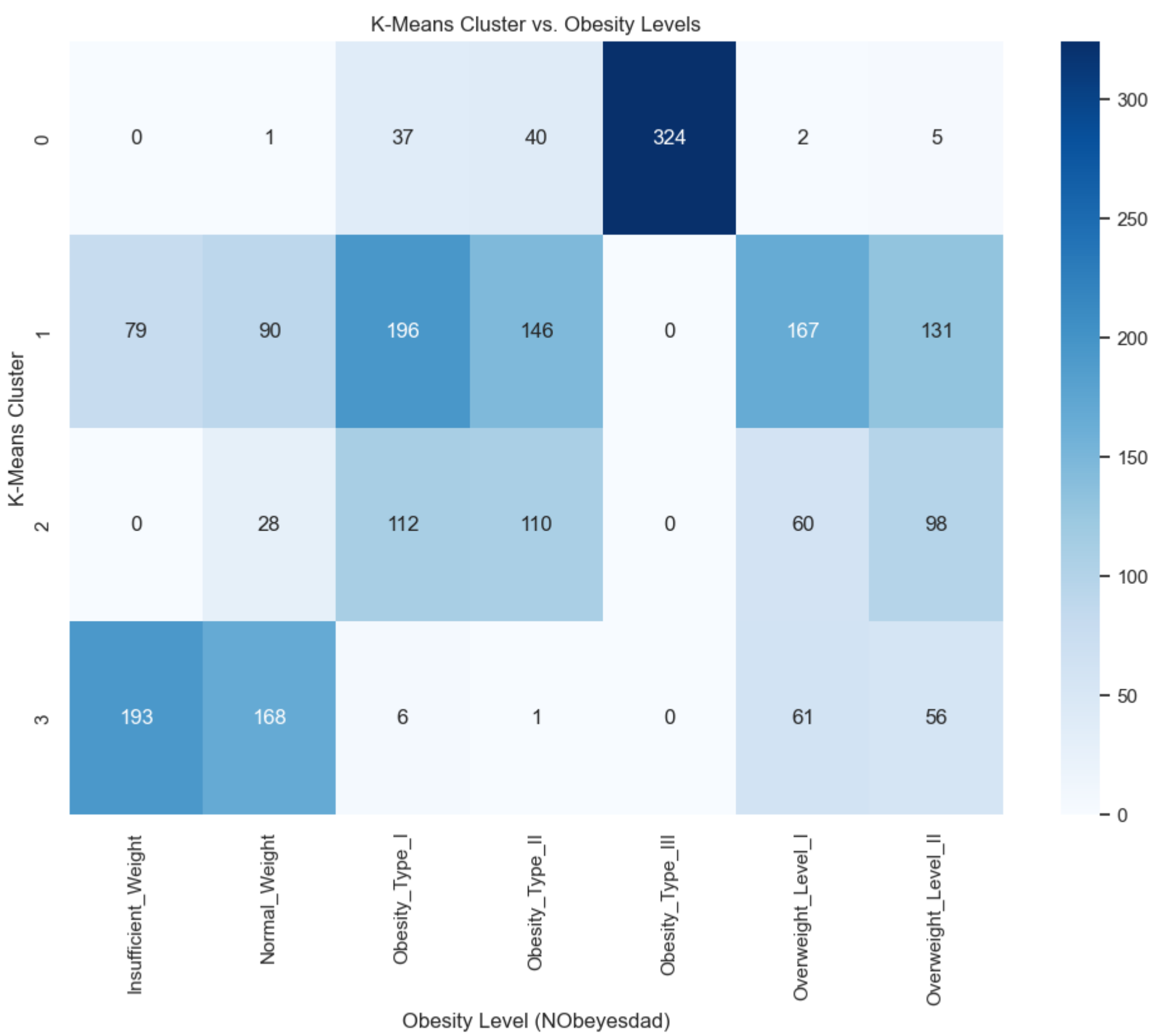


In [103…

```
# 8. Interpreting Clusters with Respect to Obesity Levels

## a. Analyzing Cluster Purity

conf_matrix = pd.crosstab(df['KMeans_Cluster'], df['NObeyesdad'])
```

```
print("\nConfusion Matrix between K-Means Clusters and True Obesity Levels:")
print(conf_matrix)

# Visualize the confusion matrix
plt.figure(figsize=(12, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('K-Means Cluster vs. Obesity Levels')
plt.xlabel('Obesity Level (NObeyesdad)')
plt.ylabel('K-Means Cluster')
plt.show()
```

```
Confusion Matrix between K-Means Clusters and True Obesity Levels:
NObeyesdad      Insufficient_Weight  Normal_Weight  Obesity_Type_I  \
KMeans_Cluster
0                                 0              1              37
1                                79             90             196
2                                 0             28             112
3                               193            168               6

NObeyesdad      Obesity_Type_II  Obesity_Type_III  Overweight_Level_I  \
KMeans_Cluster
0                            40               324                   2
1                           146                 0                 167
2                           110                 0                  60
3                             1                 0                  61

NObeyesdad      Overweight_Level_II
KMeans_Cluster
0                                 5
1                               131
2                                98
3                                56
```



In [104…

```
## b. Understanding Cluster Profiles

# Numerical Features Summary
numeric_features = ['Age', 'Height', 'Weight', 'BMI', 'Age_FAF']
num_summary = df.groupby('KMeans_Cluster')[numeric_features].mean()
print("\nNumerical Summary by K-Means Cluster:")
print(num_summary)

# Categorical Features Summary
categorical_features_summary = ['CALC', 'CAEC', 'SMOKE', 'SCC',
                                'family_history_with_overweight', 'FAVC', 'MTRANS']
```

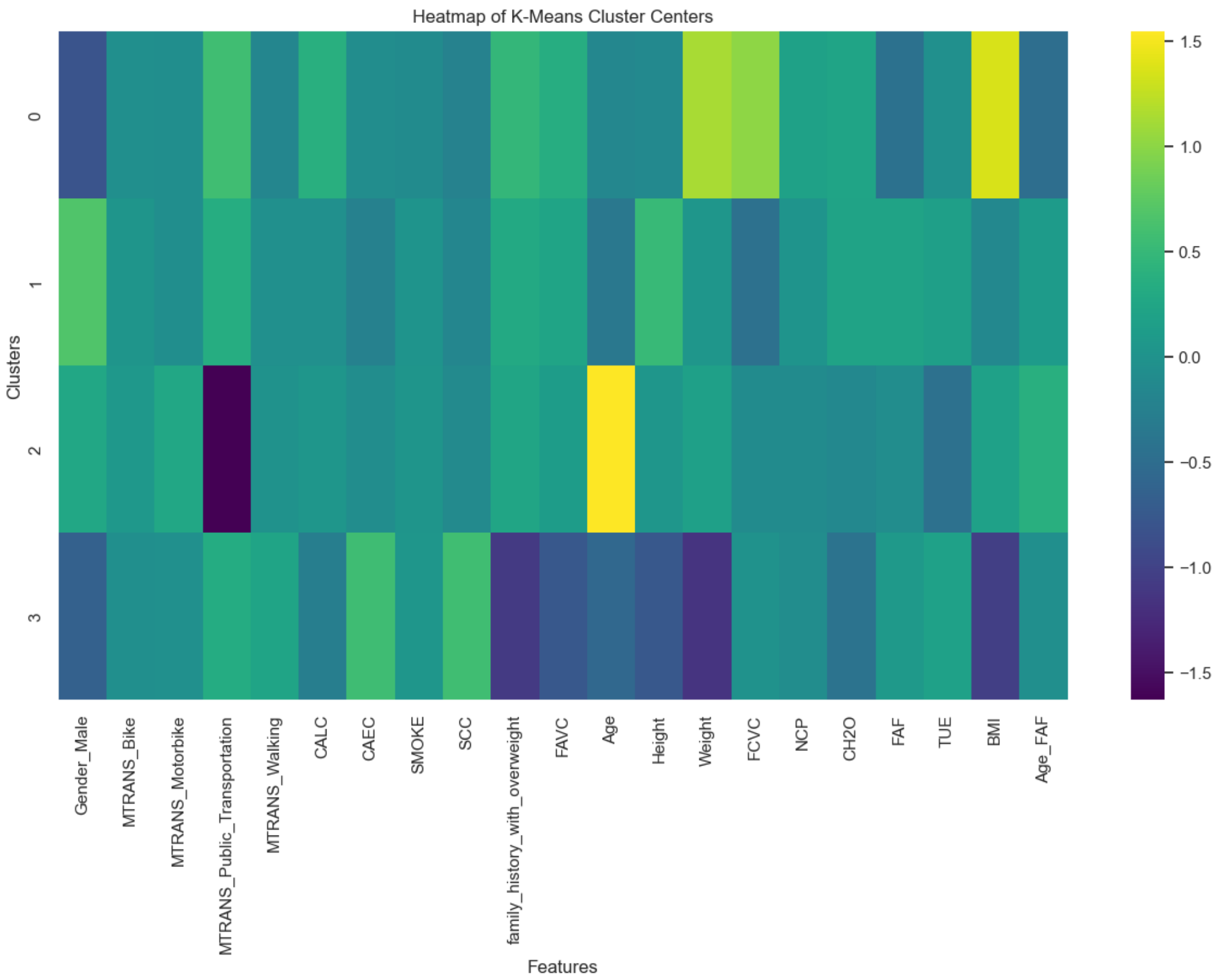
```
cat_summary = df.groupby('KMeans_Cluster')[categorical_features_summary].agg(lambda x: x.mode()[0])
print("\nCategorical Summary by K-Means Cluster:")
print(cat_summary)
```

```
Numerical Summary by K-Means Cluster:
                      Age    Height      Weight        BMI    Age_FAF
KMeans_Cluster
0               23.335497  1.689496  116.135398  40.507656  13.368886
1               22.037487  1.748891   87.501945  28.542206  25.916622
2               34.129994  1.705366   90.872117  31.101016  31.633493
3               20.672815  1.630091   56.533801  21.339269  22.393330

Categorical Summary by K-Means Cluster:
                      CALC       CAEC SMOKE SCC family_history_with_overweight  \
KMeans_Cluster
0               Sometimes  Sometimes    no  no                            yes
1               Sometimes  Sometimes    no  no                            yes
2               Sometimes  Sometimes    no  no                            yes
3               Sometimes  Sometimes    no  no                             no

                FAVC                MTRANS
KMeans_Cluster
0               yes  Public_Transportation
1               yes  Public_Transportation
2               yes             Automobile
3               yes  Public_Transportation
```

In [106…

```
# 9. Advanced Techniques and Considerations

## a. Feature Importance in Clustering

# Analyze K-Means cluster centers


# Use columns of X_scaled_df as feature names
processed_feature_names = X_scaled_df.columns.tolist()

# Now, create the DataFrame for the cluster centers
kmeans_centers = pd.DataFrame(kmeans.cluster_centers_, columns=processed_feature_names)

# Verify the cluster centers DataFrame
print("\nK-Means Cluster Centers (Feature-wise):")
print(kmeans_centers)


# Optionally, visualize cluster centers
plt.figure(figsize=(15, 8))
sns.heatmap(kmeans_centers, annot=False, cmap='viridis')
plt.title('Heatmap of K-Means Cluster Centers')
plt.xlabel('Features')
plt.ylabel('Clusters')
plt.show()
```

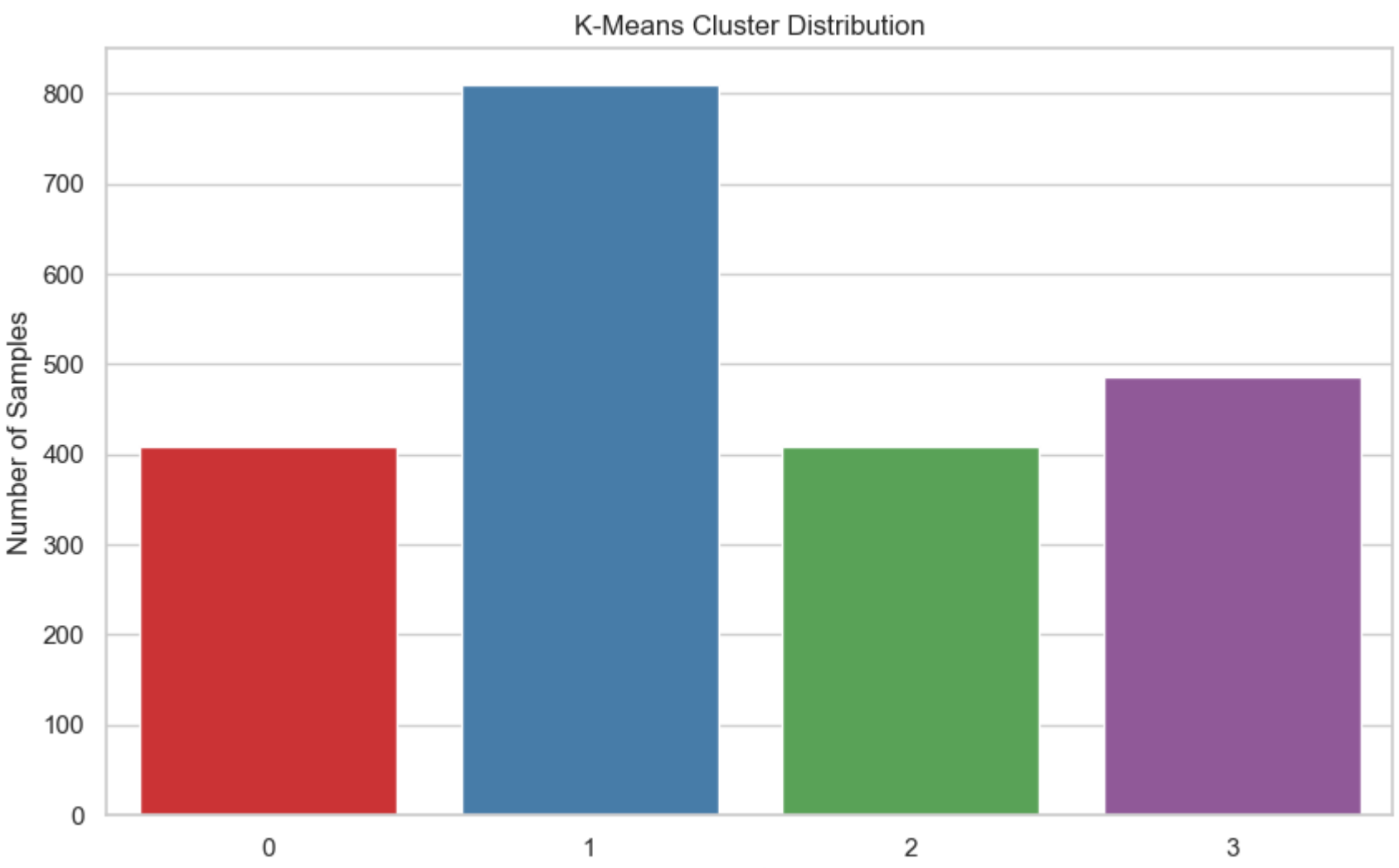
```
K-Means Cluster Centers (Feature-wise):
   Gender_Male  MTRANS_Bike  MTRANS_Motorbike  MTRANS_Public_Transportation  \
0    -0.806520    -0.057680         -0.072375                      0.574086
1     0.666820     0.028326         -0.072375                      0.337571
2     0.257783     0.070222          0.268051                     -1.628940
3    -0.649002    -0.057680         -0.043737                      0.323116

   MTRANS_Walking      CALC      CAEC     SMOKE       SCC  \
0       -0.165078  0.374093 -0.087621 -0.111671 -0.218272
1       -0.042005 -0.042472 -0.244900  0.018498 -0.170809
2       -0.012557  0.040832 -0.087633  0.025666 -0.124160
3        0.219840 -0.278977  0.556115  0.041726  0.573433

   family_history_with_overweight  ...       Age    Height    Weight  \
0                        0.472291  ... -0.154009 -0.130581  1.128485
1                        0.289833  ... -0.358598  0.506139  0.034978
2                        0.243795  ...  1.547395  0.039544  0.163684
3                       -1.086825  ... -0.573695 -0.767408 -1.147691

       FCVC       NCP      CH2O       FAF       TUE       BMI   Age_FAF
0  1.006559  0.184338  0.221469 -0.439961 -0.043328  1.349345 -0.490687
1 -0.452937  0.011431  0.215420  0.216481  0.146165 -0.144574  0.100646
2 -0.104879 -0.105876 -0.146007 -0.084521 -0.459126  0.174901  0.370062
3 -0.005086 -0.085452 -0.423267  0.081022  0.178963 -1.043880 -0.065395

[4 rows x 21 columns]
```



In [107…

```
## b. Cluster Balancing (if necessary)
# Check the distribution of clusters
cluster_counts = df['KMeans_Cluster'].value_counts().sort_index()
print("\nK-Means Cluster Distribution:")
print(cluster_counts)

# Plot cluster distribution
plt.figure(figsize=(10, 6))
sns.barplot(x=cluster_counts.index, y=cluster_counts.values, palette='Set1')
plt.title('K-Means Cluster Distribution')
plt.xlabel('Cluster')
plt.ylabel('Number of Samples')
plt.show()
```

```
K-Means Cluster Distribution:
0    409
1    809
2    408
3    485
Name: KMeans_Cluster, dtype: int64
```



In [108…
```
# If clusters are imbalanced, consider techniques like SMOTE for clustering or adjusting DBSCAN parameters

# 10. Automated Clustering Pipeline

# Define the complete pipeline with preprocessing, scaling, and clustering
clustering_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('scaler', StandardScaler()),
    ('kmeans', KMeans(n_clusters=optimal_k, random_state=42))
])
```

In [109…
```
# Fit the pipeline
clustering_pipeline.fit(X)

# Predict cluster labels
df['Pipeline_KMeans_Cluster'] = clustering_pipeline.predict(X)

# Verify the clustering
print("\nCluster Labels from Automated Pipeline:")
print(df['Pipeline_KMeans_Cluster'].value_counts())
```

```
Cluster Labels from Automated Pipeline:
1    893
0    731
2    476
3     11
Name: Pipeline_KMeans_Cluster, dtype: int64
```

In [ ]:

In [ ]:

In [ ]:
```
# 11. Documentation and Reporting

# Throughout the script, comments and print statements have been added to explain each step.
# For a comprehensive report, consider exporting the summaries and visualizations to a PDF or Jupyter Notebook.

# 12. Saving the Processed Data (Optional)

# Save the DataFrame with cluster labels for future use
df.to_csv('obesity_clustering_results.csv', index=False)
print("\nClustering results saved to 'obesity_clustering_results.csv'")
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: