This command opens a file chooser dialog, allowing me to manually upload a CSV file (simulated_health_wellness_data.csv) from my local machine into the Colab environment. Once uploaded, the file is saved and ready for data processing. This is a standard method for importing local files when working in Google Colab.

```
from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving simulated_health_wellness_data.csv to
simulated_health_wellness_data.csv
```

command opens a file upload dialog where I selected the simulated_health_wellness_data.csv file from my local machine.

Once uploaded, the file is saved in the Colab environment and is ready to be read and analyzed.

This step is necessary to bring external data into Colab for processing and analysis.

```
import pandas as pd

df = pd.read_csv('simulated_health_wellness_data.csv')
df.head()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 200,\n  \"fields\": [\
n    {\n      \"column\": \"Exercise_Time_Min\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
9.310039152070015,\n        \"min\": 3.8025489591025554,\n
\"max\": 57.20169166589619,\n        \"num_unique_values\": 200,\n
\"samples\": [\n          15.364850518678814,\n
24.377124707590276,\n          23.98293387770603\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Healthy_Meals_Per_Day\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1,\n        \"min\": 0,\n        \"max\": 9,\n
\"num_unique_values\": 10,\n        \"samples\": [\n          7,\n
8,\n          0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Sleep_Hours_Per_Night\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.422471146224773,\n
\"min\": 1.7787867568598743,\n        \"max\": 10.70841919340899,\n
\"num_unique_values\": 200,\n        \"samples\": [\n
7.005122126767616,\n          7.4707929452052015,\n
10.46458413310928\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Stress_Level\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2,\n        \"min\": 1,\n
\"max\": 9,\n        \"num_unique_values\": 9,\n        \"samples\":
[\n          4,\n          7,\n          6\n        ],\n
```

```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"BMI\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 5.070777532776108,\n
\"min\": 12.502971425632134,\n        \"max\": 37.8985466882716,\n
\"num_unique_values\": 200,\n        \"samples\": [\n
24.26498999185756,\n          17.79974558702853,\n
18.05213766618202\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df"}
```


Used df.shape, df.columns, and df.head() to check the dataset's dimensions, column names, and preview the first few rows.

```
print("Shape:", df.shape)
print("Columns:", df.columns)
df.head()
```

```
Shape: (200, 5)
Columns: Index(['Exercise_Time_Min', 'Healthy_Meals_Per_Day',
'Sleep_Hours_Per_Night',
       'Stress_Level', 'BMI'],
      dtype='object')
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 200,\n  \"fields\": [\
n    {\n      \"column\": \"Exercise_Time_Min\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
9.310039152070015,\n        \"min\": 3.8025489591025554,\n
\"max\": 57.20169166589619,\n        \"num_unique_values\": 200,\n
\"samples\": [\n          15.364850518678814,\n
24.377124707590276,\n          23.98293387770603\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Healthy_Meals_Per_Day\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1,\n        \"min\": 0,\n        \"max\": 9,\n
\"num_unique_values\": 10,\n        \"samples\": [\n          7,\n
8,\n          0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Sleep_Hours_Per_Night\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.422471146224773,\n
\"min\": 1.7787867568598743,\n        \"max\": 10.70841919340899,\n
\"num_unique_values\": 200,\n        \"samples\": [\n
7.005122126767616,\n          7.4707929452052015,\n
10.46458413310928\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Stress_Level\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2,\n        \"min\": 1,\n
\"max\": 9,\n        \"num_unique_values\": 9,\n        \"samples\":
[\n          4,\n          7,\n          6\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
```

```
n    },\n    {\n      \"column\": \"BMI\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 5.070777532776108,\n
\"min\": 12.502971425632134,\n        \"max\": 37.8985466882716,\n
\"num_unique_values\": 200,\n        \"samples\": [\n
24.26498999185756,\n          17.79974558702853,\n
18.05213766618202\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df"}
```


Used df.describe() to generate summary statistics and understand the distribution of numerical features in the dataset.

```
df.describe()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 8,\n  \"fields\": [\n
{\n      \"column\": \"Exercise_Time_Min\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 63.355992523769196,\n
\"min\": 3.8025489591025554,\n        \"max\": 200.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
29.59229034827915,\n          29.958081159522635,\n          200.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Healthy_Meals_Per_Day\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
69.61391689891552,\n        \"min\": 0.0,\n        \"max\": 200.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          2.875,\n
3.0,\n          200.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Sleep_Hours_Per_Night\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 68.67505457157586,\n
\"min\": 1.422471146224773,\n        \"max\": 200.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
6.9335824132468495,\n          6.972330909373908,\n          200.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Stress_Level\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
69.11053428151432,\n        \"min\": 1.0,\n        \"max\": 200.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n          4.995,\n
5.0,\n          200.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"BMI\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 63.64372273395261,\n        \"min\": 5.070777532776108,\n
\"max\": 200.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          25.150008369947745,\n
25.155661564070886,\n          200.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```

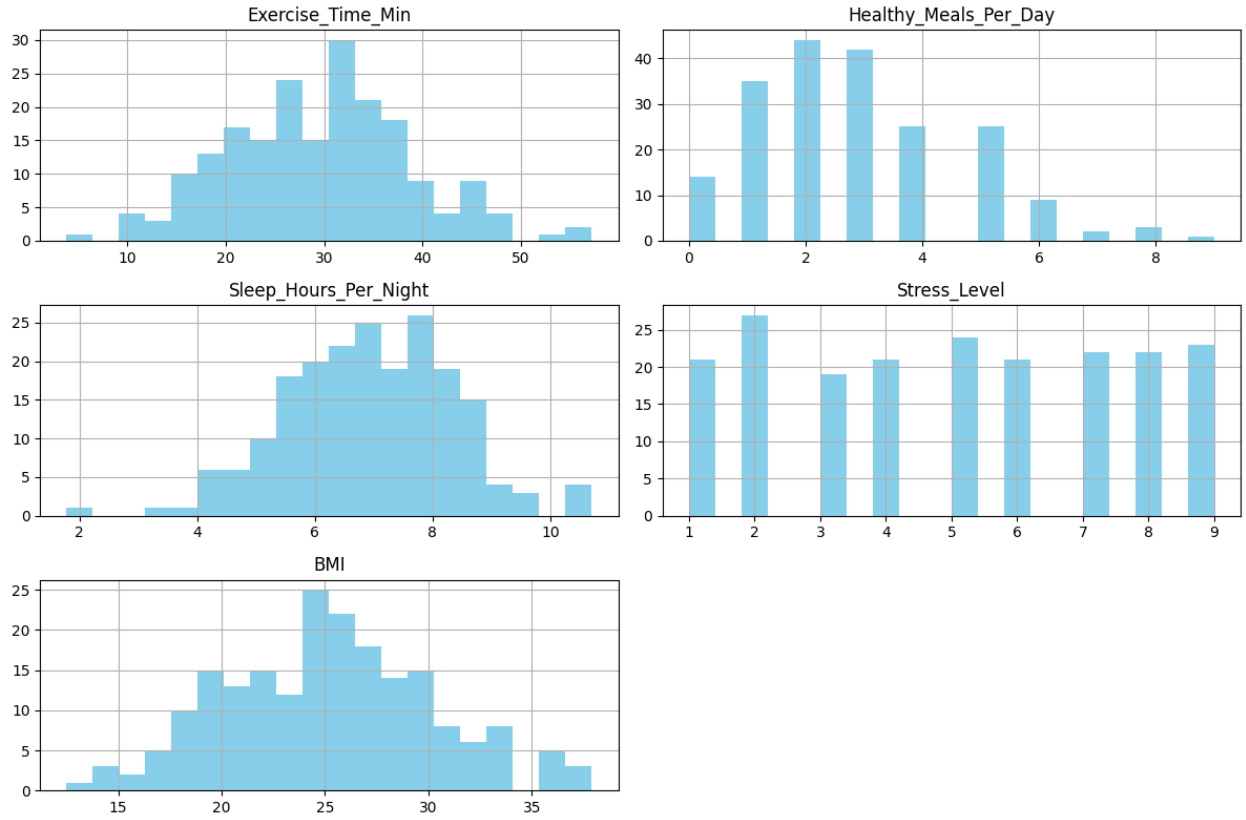

Checked for missing values in each column using df.isnull().sum() to identify any data cleaning needs.

```
df.isnull().sum()
```

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

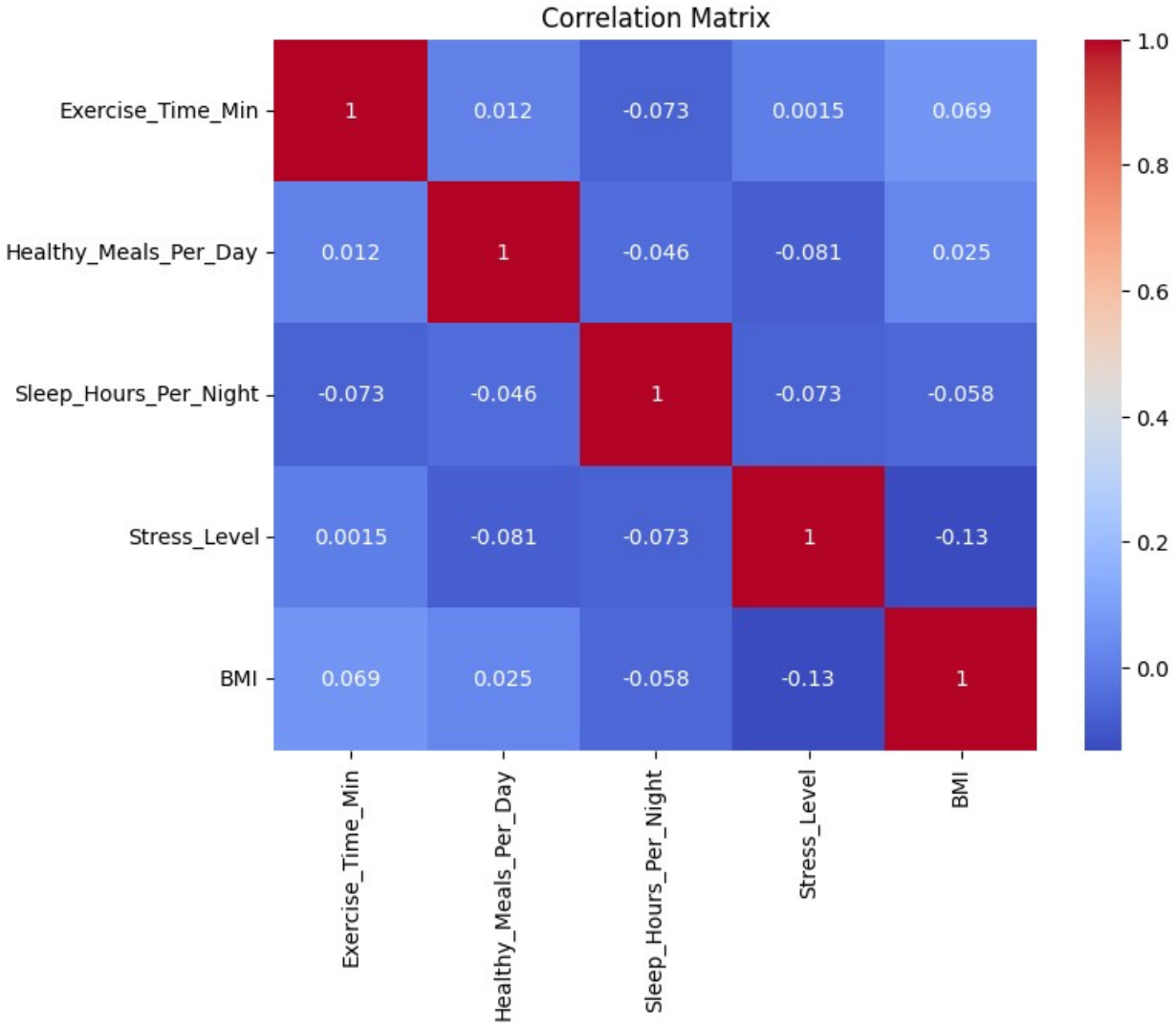
Visualized the distribution of all numerical variables using histograms with 20 bins and a skyblue color scheme to identify underlying patterns and outliers.

```
import matplotlib.pyplot as plt

df.hist(bins=20, figsize=(12, 8), color='skyblue')
plt.tight_layout()
plt.show()
```



To examine relationships between numerical variables, I created a correlation heatmap using Seaborn. The heatmap displays correlation coefficients between features, with values annotated in each cell for clarity. A coolwarm color palette was applied to easily distinguish positive (red) and negative (blue) correlations. This helped in identifying potential multicollinearity and understanding how variables interact within the dataset.

```
import seaborn as sns

plt.figure(figsize=(8, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```



To check the symmetry of the data distribution, I used the .skew() function. This function calculates the skewness of each numerical column in the DataFrame. A value close to 0 indicates a symmetric distribution, while positive or negative values suggest right or left skew, respectively. Understanding skewness is important for deciding whether transformation or normalization is required before modeling.

```
df.skew()
```

```
Exercise_Time_Min        0.133476
Healthy_Meals_Per_Day    0.620414
```

```
Sleep_Hours_Per_Night   -0.162912
Stress_Level             0.010693
BMI                      0.159342
dtype: float64
```

To standardize the dataset, I applied the StandardScaler from sklearn.preprocessing. This process scales each feature to have a mean of 0 and a standard deviation of 1, which is essential for many machine learning algorithms to perform optimally. The result was stored in scaled_df, a DataFrame that preserves the original column names for better interpretability

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_data = scaler.fit_transform(df)

# Optional: convert to DataFrame
scaled_df = pd.DataFrame(scaled_data, columns=df.columns)
scaled_df.head()
```


```
{"summary":"{\n  \"name\": \"scaled_df\",\n  \"rows\": 200,\n
\"fields\": [\n    {\n      \"column\": \"Exercise_Time_Min\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.0025094142341713,\n        \"min\": -2.777051536627066,\n
\"max\": 2.9729933773829798,\n        \"num_unique_values\": 200,\n
\"samples\": [\n          -1.5320174423169044,\n          -
0.5615714999886672,\n          -0.6040181547780695\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Healthy_Meals_Per_Day\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.0025094142341697,\n        \"min\": -1.5876044525782826,\n
\"max\": 3.3822877467972106,\n        \"num_unique_values\": 10,\n
\"samples\": [\n          2.277867258047101,\n
2.830077502422156,\n          -1.5876044525782826\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Sleep_Hours_Per_Night\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.0025094142341713,\n        \"min\": -3.632925130113516,\n
\"max\": 2.6603769217769218,\n        \"num_unique_values\": 200,\n
\"samples\": [\n          0.050418763492339584,\n
0.3786077609681751,\n          2.4885302420607283\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Stress_Level\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.0025094142341722,\n        \"min\": -1.537109867826259,\n
\"max\": 1.5409574519760116,\n        \"num_unique_values\": 9,\n
\"samples\": [\n          -0.38283462290040743,\n
0.7714406220254441,\n          0.3866822070501602\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"BMI\",\n      \"properties\": {\n
```

```
\"dtype\": \"number\",\n        \"std\": 1.002509414234171,\n
\"min\": -2.5003608454308366,\n        \"max\": 2.5204280012749347,\n
\"num_unique_values\": 200,\n        \"samples\": [\n          -
0.17497104735334887,\n          -1.453171153603619,\n          -
1.403272408530694\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"scaled_df"}
```

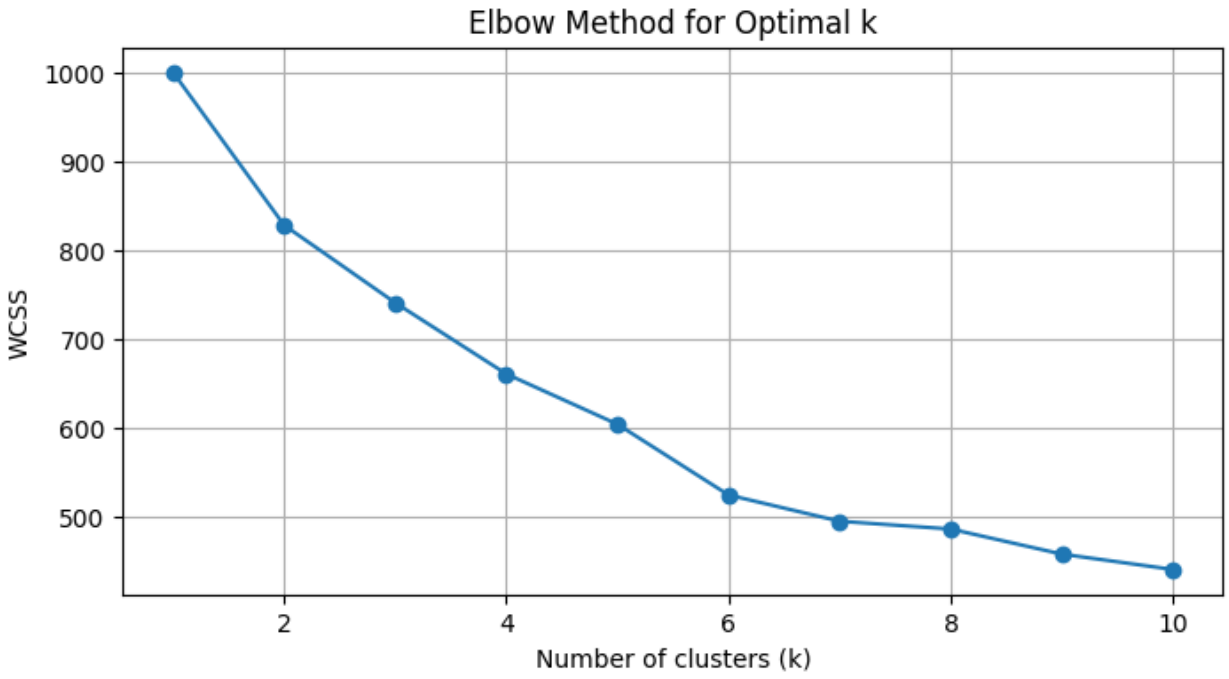

To examine the asymmetry in the distribution of each feature, I used the .skew() function. This helps identify whether a variable is normally distributed or skewed, which can influence the choice of preprocessing techniques and models.

```
df.skew()
```

```
Exercise_Time_Min        0.133476
Healthy_Meals_Per_Day    0.620414
Sleep_Hours_Per_Night   -0.162912
Stress_Level             0.010693
BMI                      0.159342
dtype: float64
```

To determine the optimal number of clusters for K-Means, I implemented the Elbow Method. Using a range of k values from 1 to 10, I calculated the Within-Cluster Sum of Squares (WCSS) for each and plotted the results. The "elbow point" in the plot indicates the value of k where adding more clusters does not significantly reduce WCSS, suggesting the most appropriate number of clusters for the dataset.

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

wcss = []  # WCSS for each k
K_range = range(1, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, init='k-means++', random_state=42)
    kmeans.fit(scaled_data)
    wcss.append(kmeans.inertia_)  # inertia_ is WCSS

# Plot Elbow
plt.figure(figsize=(8, 4))
plt.plot(K_range, wcss, marker='o')
plt.xlabel('Number of clusters (k)')
plt.ylabel('WCSS')
plt.title('Elbow Method for Optimal k')
plt.grid(True)
plt.show()
```



After identifying the optimal number of clusters using the Elbow Method, I applied K-Means clustering with k=7. The model was trained on the scaled dataset, and the resulting cluster labels were extracted and added as a new column ('Cluster') in the original (unscaled) DataFrame. This allowed for further analysis and visualization of the clustered data.

```
from sklearn.cluster import KMeans

# Fit KMeans with 7 clusters
kmeans = KMeans(n_clusters=7, init='k-means++', random_state=42)
kmeans.fit(scaled_data)

# Get cluster labels
labels = kmeans.labels_

# Add to original (unscaled) DataFrame
df['Cluster'] = labels

# View first few rows
df.head()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 200,\n  \"fields\": [\
n    {\n      \"column\": \"Exercise_Time_Min\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
9.310039152070015,\n        \"min\": 3.8025489591025554,\n
\"max\": 57.20169166589619,\n        \"num_unique_values\": 200,\n
\"samples\": [\n          15.364850518678814,\n
24.377124707590276,\n          23.98293387770603\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Healthy_Meals_Per_Day\",\n
```

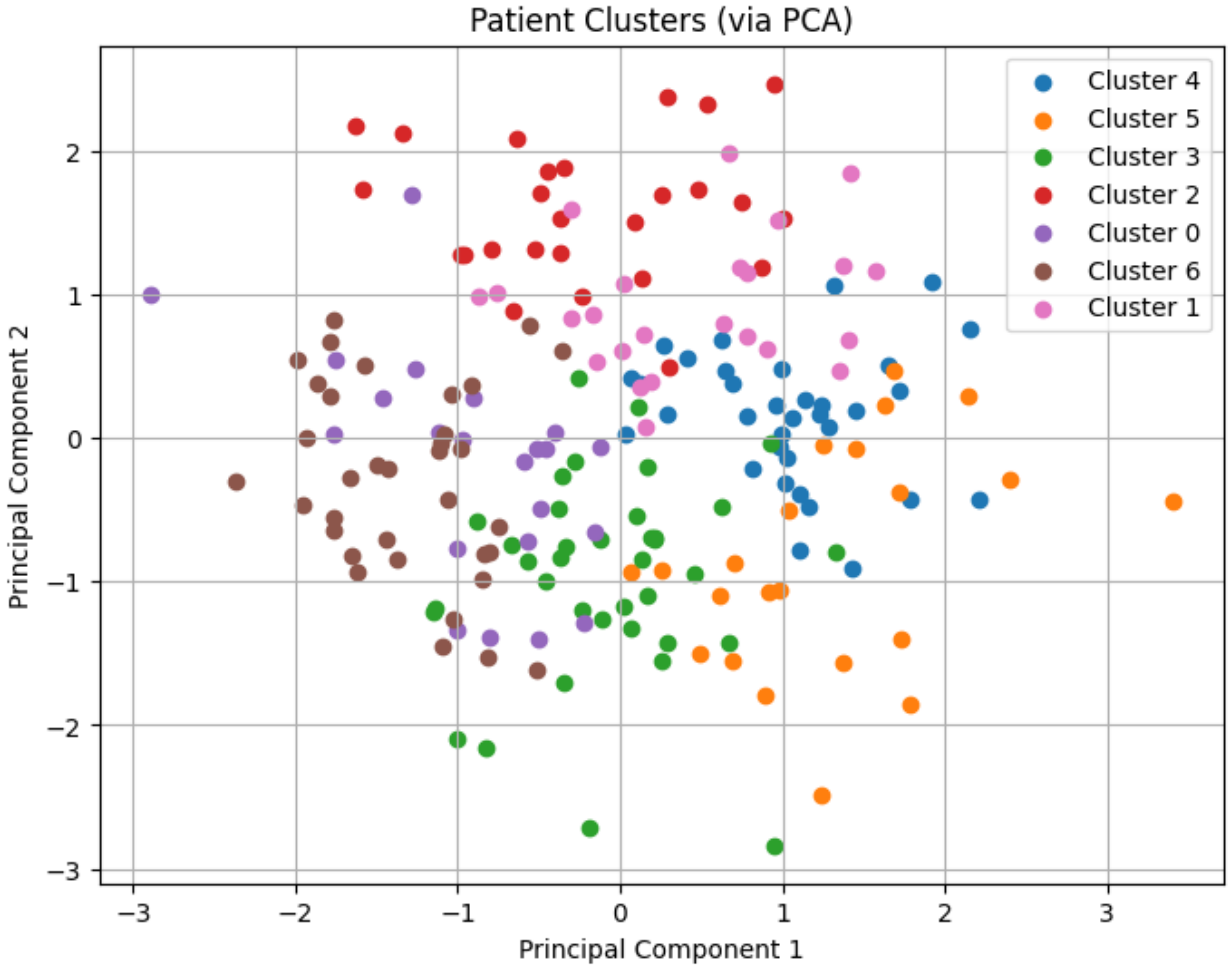
```
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1,\n        \"min\": 0,\n        \"max\": 9,\n
\"num_unique_values\": 10,\n        \"samples\": [\n          7,\n
8,\n          0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Sleep_Hours_Per_Night\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.422471146224773,\n
\"min\": 1.7787867568598743,\n        \"max\": 10.70841919340899,\n
\"num_unique_values\": 200,\n        \"samples\": [\n
7.005122126767616,\n          7.4707929452052015,\n
10.46458413310928\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Stress_Level\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2,\n        \"min\": 1,\n
\"max\": 9,\n        \"num_unique_values\": 9,\n        \"samples\":
[\n          4,\n          7,\n          6\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"BMI\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 5.070777532776108,\n
\"min\": 12.502971425632134,\n        \"max\": 37.8985466882716,\n
\"num_unique_values\": 200,\n        \"samples\": [\n
24.26498999185756,\n          17.79974558702853,\n
18.05213766618202\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Cluster\",\n      \"properties\": {\n        \"dtype\": \"int32\",\n
\"num_unique_values\": 7,\n        \"samples\": [\n          4,\n
5,\n          6\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df"}
```


To visualize the clusters in a reduced dimensional space, I applied Principal Component Analysis (PCA) to transform the high-dimensional data into two principal components. I then created a new DataFrame containing the PCA-transformed data and the cluster labels obtained from K-Means.

Using a scatter plot, I visualized the clusters by plotting Principal Component 1 (PC1) against Principal Component 2 (PC2), with each cluster represented in a different color. This helped me observe how well-separated the clusters were in the reduced 2D space and gain insights into group distinctions among patient profiles.

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Apply PCA
pca = PCA(n_components=2)
pca_data = pca.fit_transform(scaled_data)

# Create a new DataFrame for plotting
pca_df = pd.DataFrame(pca_data, columns=['PC1', 'PC2'])
```

```
pca_df['Cluster'] = labels

# Plot
plt.figure(figsize=(8, 6))
for cluster in pca_df['Cluster'].unique():
    cluster_points = pca_df[pca_df['Cluster'] == cluster]
    plt.scatter(cluster_points['PC1'], cluster_points['PC2'],
label=f'Cluster {cluster}')

plt.title('Patient Clusters (via PCA)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.grid(True)
plt.show()
```



To evaluate the quality of clustering, I calculated the Silhouette Score using the silhouette_score function from sklearn.metrics. This score measures how similar each data point is to its own cluster compared to other clusters. A higher silhouette score (closer to 1) indicates well-defined and separated clusters.

```
from sklearn.metrics import silhouette_score

score = silhouette_score(scaled_data, labels)
print(f'Silhouette Score: {score:.3f}')
```

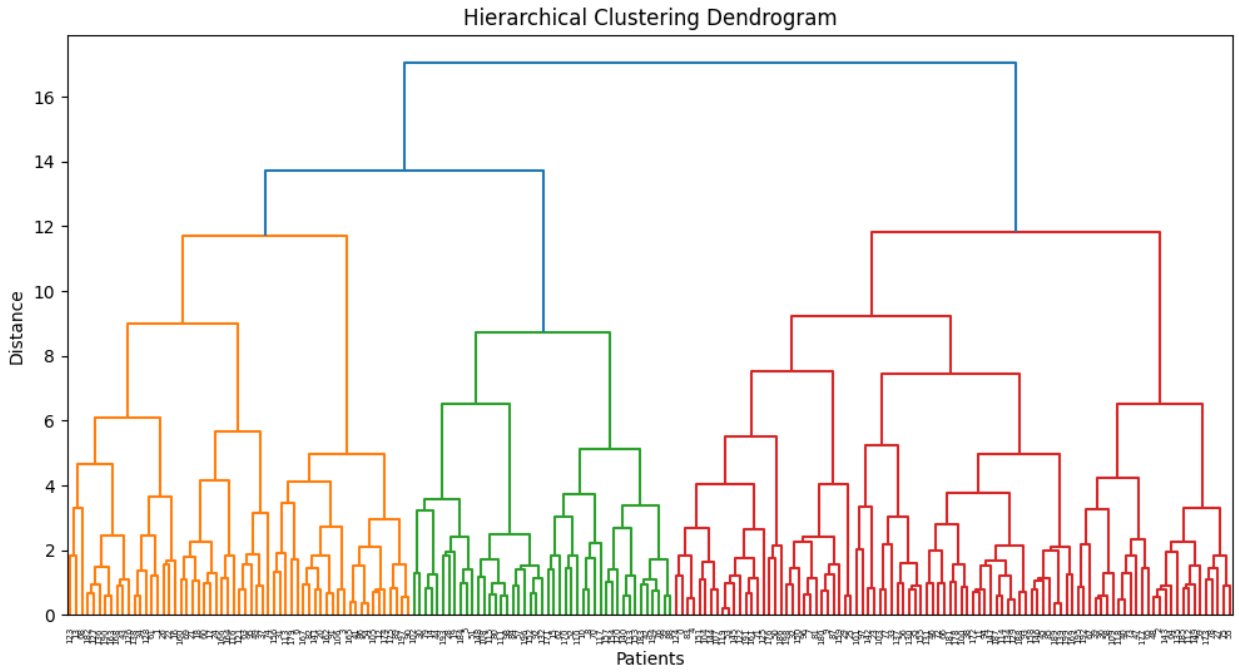
```
Silhouette Score: 0.178
```

To perform hierarchical clustering, a linkage matrix was created using Ward's method via the linkage function from scipy.cluster.hierarchy. Ward's method minimizes the total within-cluster variance, making it a popular choice for forming compact, spherical clusters.

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

# Create linkage matrix using Ward’s method (minimizes variance)
linked = linkage(scaled_data, method='ward')
```

The dendrogram was plotted using matplotlib and the dendrogram function from scipy.cluster.hierarchy. It visually represents the hierarchical clustering process, showing the distances between merged clusters. The orientation was set to 'top' and the distances were sorted in descending order for better clarity of cluster formation.

```
plt.figure(figsize=(12, 6))
dendrogram(linked,
           orientation='top',
           distance_sort='descending',
           show_leaf_counts=False)
plt.title("Hierarchical Clustering Dendrogram")
plt.xlabel("Patients")
plt.ylabel("Distance")
plt.show()
```



Using the fcluster function from scipy.cluster.hierarchy, I extracted 4 clusters from the hierarchical linkage matrix by setting the criterion to 'maxclust'. These cluster labels were then added to the original DataFrame under the column 'HC_Cluster' for further analysis.

```
from scipy.cluster.hierarchy import fcluster

# Create 4 clusters
hc_labels = fcluster(linked, t=4, criterion='maxclust')

# Add to DataFrame
df['HC_Cluster'] = hc_labels
```

To interpret the hierarchical clusters, I calculated the mean values of all features for each cluster using the groupby('HC_Cluster').mean() function. This helped in understanding the average profile of patients in each cluster.

```
df.groupby('HC_Cluster').mean()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 4,\n  \"fields\": [\n
{\n      \"column\": \"HC_Cluster\",\n      \"properties\": {\n
\"dtype\": \"int32\",\n        \"num_unique_values\": 4,\n
\"samples\": [\n          2,\n          4,\n          1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Exercise_Time_Min\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
4.683754238939904,\n        \"min\": 25.733636052924673,\n
\"max\": 36.564915022861086,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          28.057460317613593,\n
31.139440334603943,\n          36.564915022861086\n        ],\n
```

```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Healthy_Meals_Per_Day\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.9192922725934547,\n        \"min\": 2.288135593220339,\n
\"max\": 4.407407407407407,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          2.710144927536232,\n
2.288135593220339,\n          4.407407407407407\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Sleep_Hours_Per_Night\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.9889990084971954,\n        \"min\": 5.7074850391905,\n
\"max\": 7.754137533152827,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          7.646350699925356,\n
5.7074850391905,\n          6.423721409274216\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Stress_Level\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2.0864788992838665,\n        \"min\": 2.4814814814814814,\n
\"max\": 6.8,\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          3.652173913043478,\n          6.338983050847458,\n
2.4814814814814814\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"BMI\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 3.8544157823535437,\n        \"min\": 19.741328766475338,\n
\"max\": 29.145182828297962,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          29.145182828297962,\n
25.02585576531992,\n          24.22588052561946\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Cluster\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.9361747354055059,\n        \"min\": 2.4927536231884058,\n
\"max\": 4.2592592592592595,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          2.4927536231884058,\n
2.7966101694915255,\n          4.2592592592592595\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


DBSCAN clustering was applied with eps=1.2 and min_samples=5, allowing detection of clusters based on density. The resulting cluster labels were added to the DataFrame under the column 'DBSCAN_Cluster'.

```
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score

# Run DBSCAN
dbscan = DBSCAN(eps=1.2, min_samples=5)  # eps value can be adjusted
dbscan_labels = dbscan.fit_predict(scaled_data)
```

```
# Add labels to your DataFrame
df['DBSCAN_Cluster'] = dbscan_labels
```

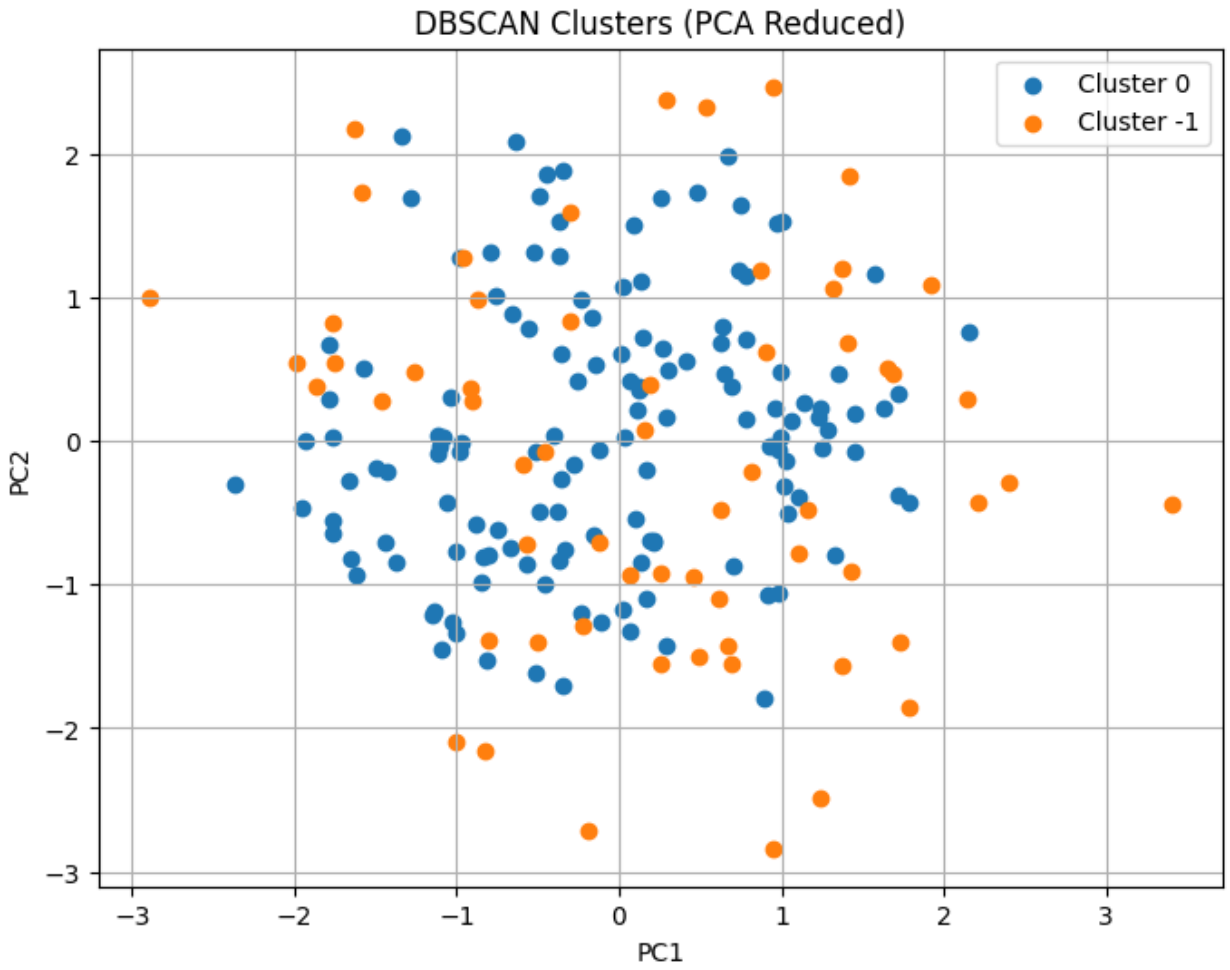
To evaluate DBSCAN results, we first identify the number of clusters (including noise points labeled -1) using np.unique(). Then, we count how many data points belong to each cluster with value_counts() on the DBSCAN_Cluster column.

```
# How many clusters (and noise points)?
import numpy as np
unique_labels = np.unique(dbscan_labels)
print("DBSCAN found clusters:", unique_labels)

# Count points per cluster
df['DBSCAN_Cluster'].value_counts()
```

```
DBSCAN found clusters: [-1  0]

DBSCAN_Cluster
 0    139
-1     61
Name: count, dtype: int64
```

We reused the PCA-reduced dataset to visualize the DBSCAN clustering results. Each point is plotted on the first two principal components (PC1 and PC2), with colors distinguishing different clusters identified by DBSCAN. Noise points (if any) are also included in the scatter plot.

```
# Reuse PCA from earlier
pca_df['DBSCAN_Cluster'] = dbscan_labels

plt.figure(figsize=(8, 6))
for cluster in pca_df['DBSCAN_Cluster'].unique():
    subset = pca_df[pca_df['DBSCAN_Cluster'] == cluster]
    plt.scatter(subset['PC1'], subset['PC2'], label=f'Cluster
{cluster}')

plt.title('DBSCAN Clusters (PCA Reduced)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.grid(True)
plt.show()
```



We reused the PCA-reduced dataset to visualize the DBSCAN clustering results. Each point is plotted on the first two principal components (PC1 and PC2), with colors distinguishing different clusters identified by DBSCAN. Noise points (if any) are also included in the scatter plot.

```
# Exclude noise (-1) for Silhouette Score
mask = dbscan_labels != -1

if len(np.unique(dbscan_labels[mask])) > 1:
    score = silhouette_score(scaled_data[mask], dbscan_labels[mask])
    print(f"Silhouette Score (DBSCAN, excluding noise): {score:.3f}")
else:
    print("Not enough clusters to calculate silhouette score.")
```

```
Not enough clusters to calculate silhouette score.
```

K-Means clustering was applied with k=7 to partition the scaled dataset into seven distinct clusters. The algorithm initialized centroids and iteratively updated them to minimize intra-cluster variance, using a fixed random state for reproducibility.

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=7, random_state=42)
kmeans.fit(scaled_data)
```

```
KMeans(n_clusters=7, random_state=42)
```

The silhouette score for Hierarchical Clustering (k=4) was computed after forming clusters using Ward's linkage method to evaluate cluster cohesion and separation.

```
from scipy.cluster.hierarchy import linkage, fcluster
from sklearn.metrics import silhouette_score

# Step 1: Compute linkage matrix using Ward's method
linked = linkage(scaled_data, method='ward')

# Step 2: Choose number of clusters (e.g., 4)
hc_labels = fcluster(linked, t=4, criterion='maxclust')

# Step 3: Compute silhouette score
hc_silhouette = silhouette_score(scaled_data, hc_labels)

print(f"Silhouette Score for Hierarchical Clustering (k=4):
{hc_silhouette:.3f}")
```

```
Silhouette Score for Hierarchical Clustering (k=4): 0.114
```