fvpwnpsre

June 23, 2025

[1]:
```
# Install plotly
!pip install plotly --quiet

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import silhouette_score
from scipy.cluster.hierarchy import dendrogram, linkage
```

Block 1 sets up everything needed for the analysis. It imports Python libraries like pandas, seaborn, scikit-learn, and plotly for data handling, clustering, and visualizations. After that, we load the dataset into a DataFrame so we can start exploring the health data of the patients.

[5]:
```
#Block 1
# Load the dataset from file or upload
from google.colab import files
uploaded = files.upload()

# Assuming file is named like this:
df = pd.read_csv('simulated_health_wellness_data.csv')
df.head()
```

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

Saving simulated_health_wellness_data.csv to simulated_health_wellness_data
(2).csv
```

[5]:

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

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

Block 2 we're trying to understand the shape and structure of the data. We check for missing values, get summary statistics, and use pair plots to see how variables like sleep, diet, and stress interact. This helps us spot patterns or outliers before we jump into clustering.

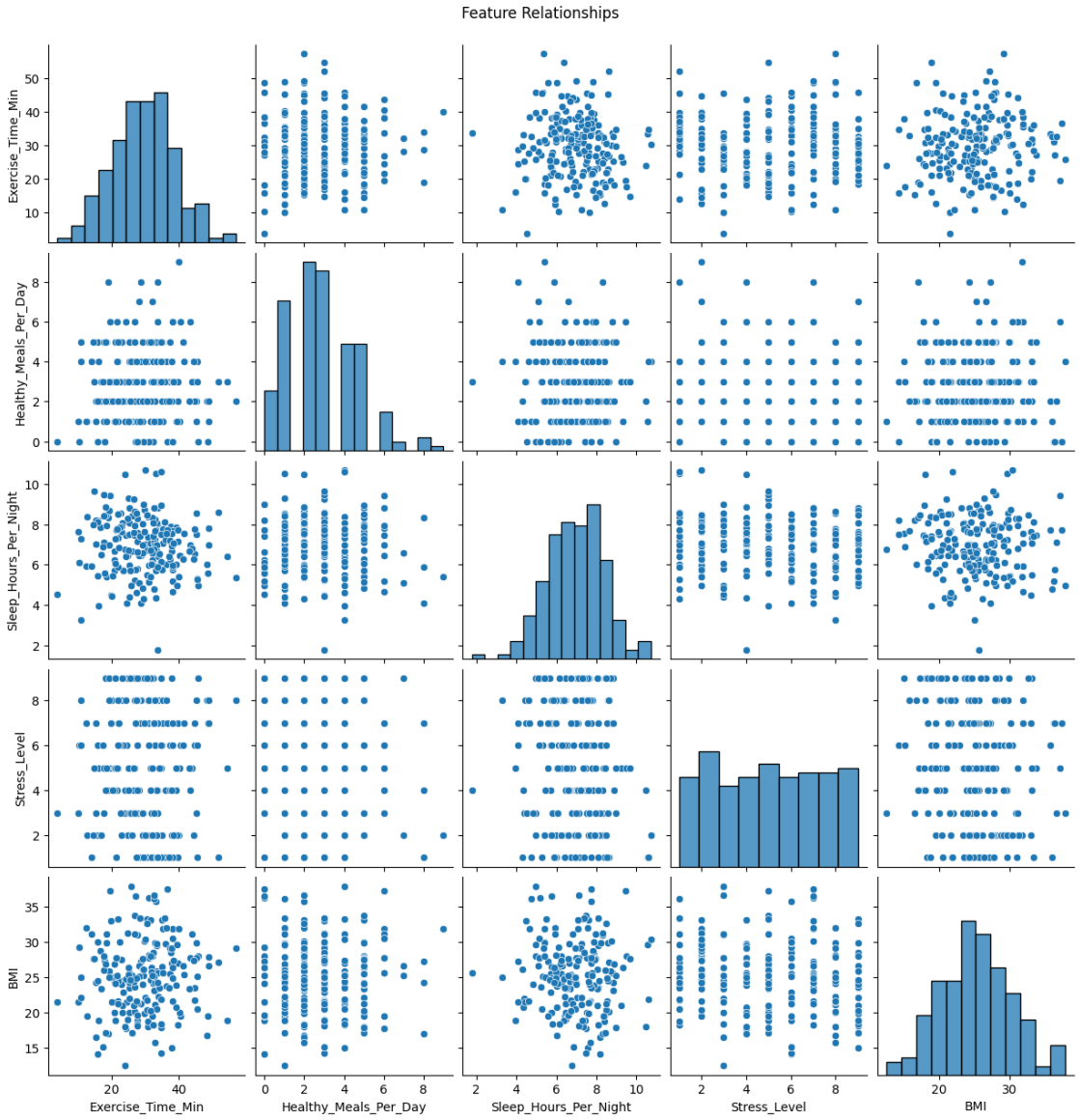
[7]:
```
#Block 2
# Basic info and stats
print(df.info())
print(df.describe())

# Check for nulls
print("Missing Values:\n", df.isnull().sum())

# Pairplot to visualize distributions
sns.pairplot(df)
plt.suptitle("Feature Relationships", y=1.02)
plt.show()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Exercise_Time_Min      200 non-null    float64
 1   Healthy_Meals_Per_Day  200 non-null    int64
 2   Sleep_Hours_Per_Night  200 non-null    float64
 3   Stress_Level           200 non-null    int64
 4   BMI                    200 non-null    float64
dtypes: float64(3), int64(2)
memory usage: 7.9 KB
None
       Exercise_Time_Min  Healthy_Meals_Per_Day  Sleep_Hours_Per_Night  \
count         200.000000             200.000000             200.000000
mean           29.592290               2.875000               6.933582
std             9.310039               1.815449               1.422471
min             3.802549               0.000000               1.778787
25%            22.948723               2.000000               5.967243
50%            29.958081               3.000000               6.972331
75%            35.008525               4.000000               7.886509
max            57.201692               9.000000              10.708419
```

```
       Stress_Level         BMI
count    200.000000  200.000000
mean       4.995000   25.150008
std        2.605556    5.070778
min        1.000000   12.502971
25%        3.000000   21.458196
50%        5.000000   25.155662
75%        7.000000   28.011155
max        9.000000   37.898547
Missing Values:
 Exercise_Time_Min        0
Healthy_Meals_Per_Day    0
Sleep_Hours_Per_Night    0
Stress_Level             0
BMI                      0
dtype: int64
```

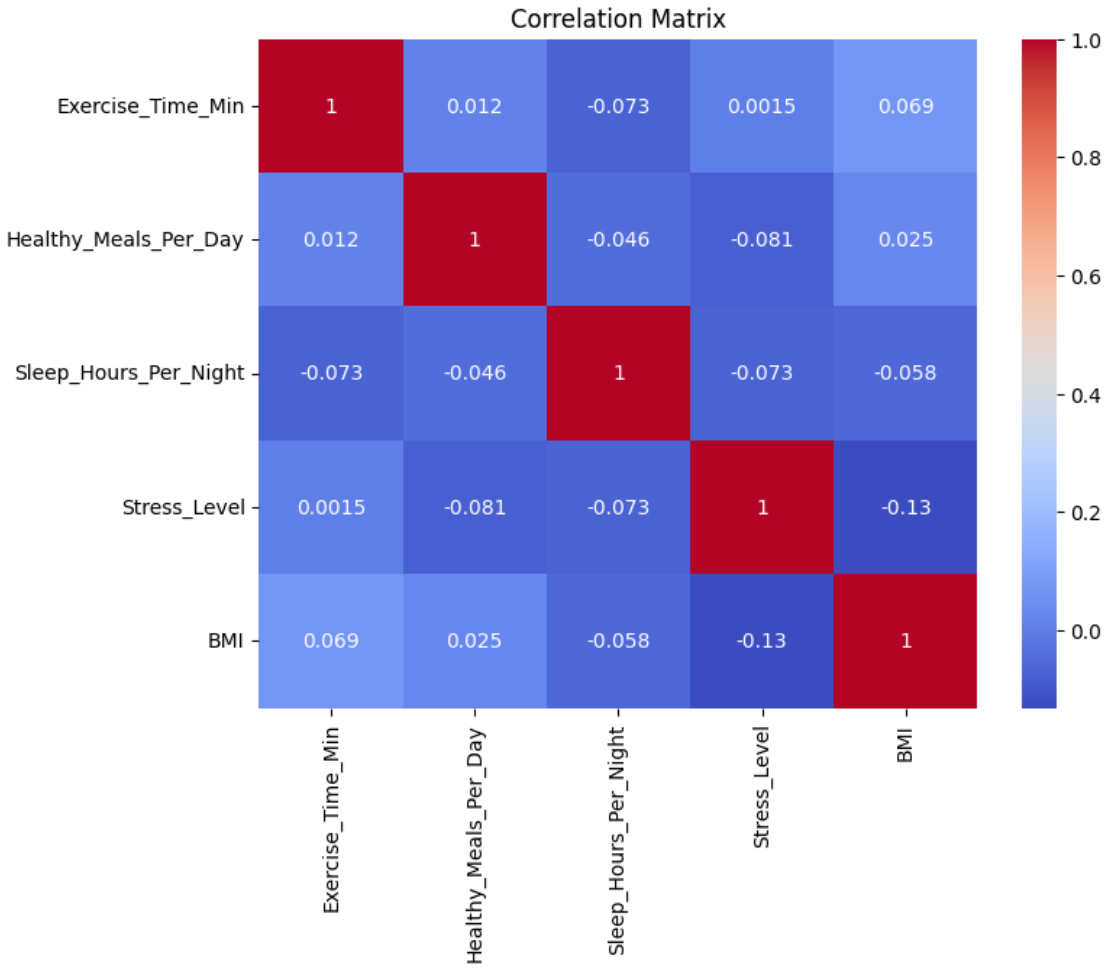


Block 3 Before clustering, it's important to know how the variables relate to each other and to normalize the data. We use a heatmap to visualize correlations, then standardize all features using a scaler so that variables like BMI or exercise time don't dominate others due to different units.

[8]:
```
#Block 3
# Correlation heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()

# Standardize the data
```

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



Block 4 PCA reduces the five wellness features into just two new dimensions. This makes it easier to visualize our data while keeping most of its meaning. The scatter plot at the end shows patients mapped into this new 2D space, helping us see natural separations between groups more clearly.

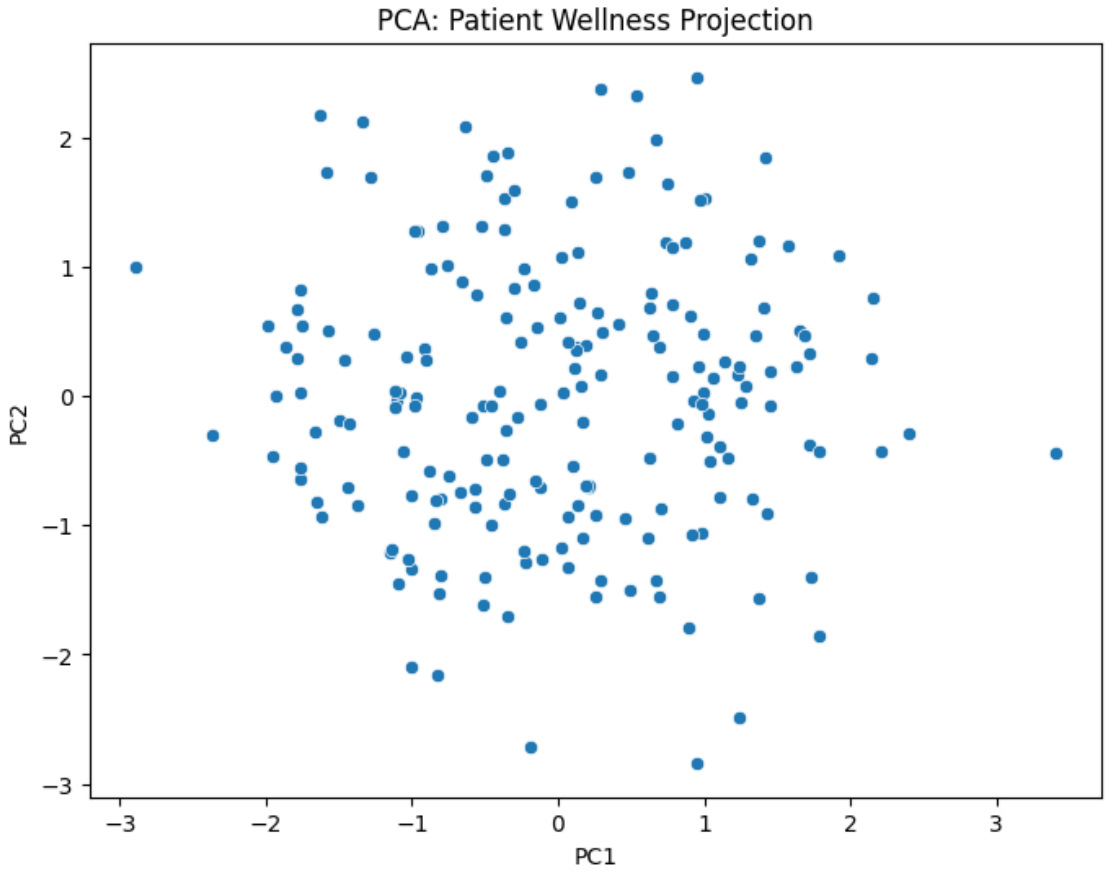
[11]:
```
#Block 4
# Apply PCA
pca = PCA(n_components=2)
pca_data = pca.fit_transform(scaled_data)

# Create PCA DataFrame
pca_df = pd.DataFrame(pca_data, columns=['PC1', 'PC2'])
print("Explained Variance Ratio:", pca.explained_variance_ratio_)

# Plot PCA results
```

```
plt.figure(figsize=(8, 6))
sns.scatterplot(x='PC1', y='PC2', data=pca_df)
plt.title("PCA: Patient Wellness Projection")
plt.show()
```

```
Explained Variance Ratio: [0.23691549 0.22082517]
```



In this block 5, we apply K-Means clustering twice—once using the original data and once using the PCA-reduced data. Then we use the silhouette score to measure how well the clusters are formed. Lastly, we plot the clusters to visualize how well the patients separate into wellness groups.

[12]:
```
#Block 5
# K-Means on original data
kmeans = KMeans(n_clusters=3, random_state=42)
labels_original = kmeans.fit_predict(scaled_data)
sil_original = silhouette_score(scaled_data, labels_original)
print("Silhouette Score (Original Data):", sil_original)

# K-Means on PCA data
kmeans_pca = KMeans(n_clusters=3, random_state=42)
```

```
labels_pca = kmeans_pca.fit_predict(pca_df)
sil_pca = silhouette_score(pca_df, labels_pca)
print("Silhouette Score (PCA Data):", sil_pca)

# Visualize clusters in PCA space
pca_df['Cluster'] = labels_pca
fig = px.scatter(pca_df, x='PC1', y='PC2', color=pca_df['Cluster'].astype(str),
                 title="KMeans Clusters After PCA")
fig.show()
```

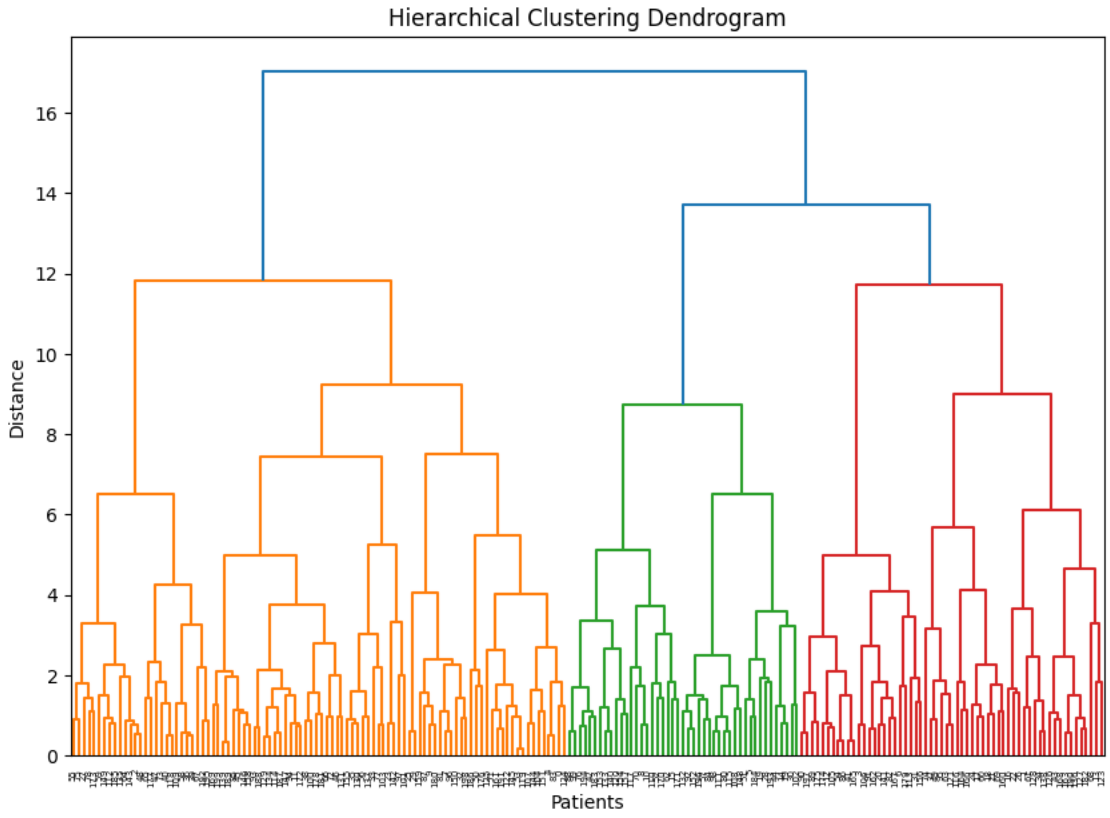
```
Silhouette Score (Original Data): 0.1516159911787657
Silhouette Score (PCA Data): 0.3625606718282867
```

Block 6, we use hierarchical clustering to group patients based on similarities, and visualize the relationships using a dendrogram. This lets us see how patients merge into clusters step by step.

[15]:
```
#Block 6
# Dendrogram
plt.figure(figsize=(10, 7))
linkage_matrix = linkage(scaled_data, method='ward')
dendrogram(linkage_matrix)
plt.title("Hierarchical Clustering Dendrogram")
plt.xlabel("Patients")
plt.ylabel("Distance")
plt.show()

# Agglomerative clustering
agg = AgglomerativeClustering(n_clusters=3)
agg_labels = agg.fit_predict(scaled_data)

# Add to PCA DataFrame for plotting
pca_df['Agg_Cluster'] = agg_labels
fig = px.scatter(pca_df, x='PC1', y='PC2', color=pca_df['Agg_Cluster'].
 ↪astype(str),
                 title="Agglomerative Clusters in PCA Space")
fig.show()
```



This final block compares how good our clustering methods were by showing the silhouette scores side-by-side. It helps us decide if using PCA improved our clusters and whether K-Means or Hierarchical worked better for segmenting patient wellness profiles. This step wraps up our entire analysis.

[16]:
```
#Block 7
print("  Model Evaluation Summary:")
print("-" * 40)
print(f"Silhouette Score (K-Means on Original Data): {sil_original:.3f}")
print(f"Silhouette Score (K-Means on PCA Data): {sil_pca:.3f}")
print("\nInterpretation:")
print("Higher silhouette scores mean better cluster separation.")
print("Compare visuals and scores to decide which model segments patient types␣
 ↪most effectively.")
```

```
  Model Evaluation Summary:
----------------------------------------
Silhouette Score (K-Means on Original Data): 0.152
Silhouette Score (K-Means on PCA Data): 0.363

Interpretation:
Higher silhouette scores mean better cluster separation.
```

Compare visuals and scores to decide which model segments patient types most effectively.