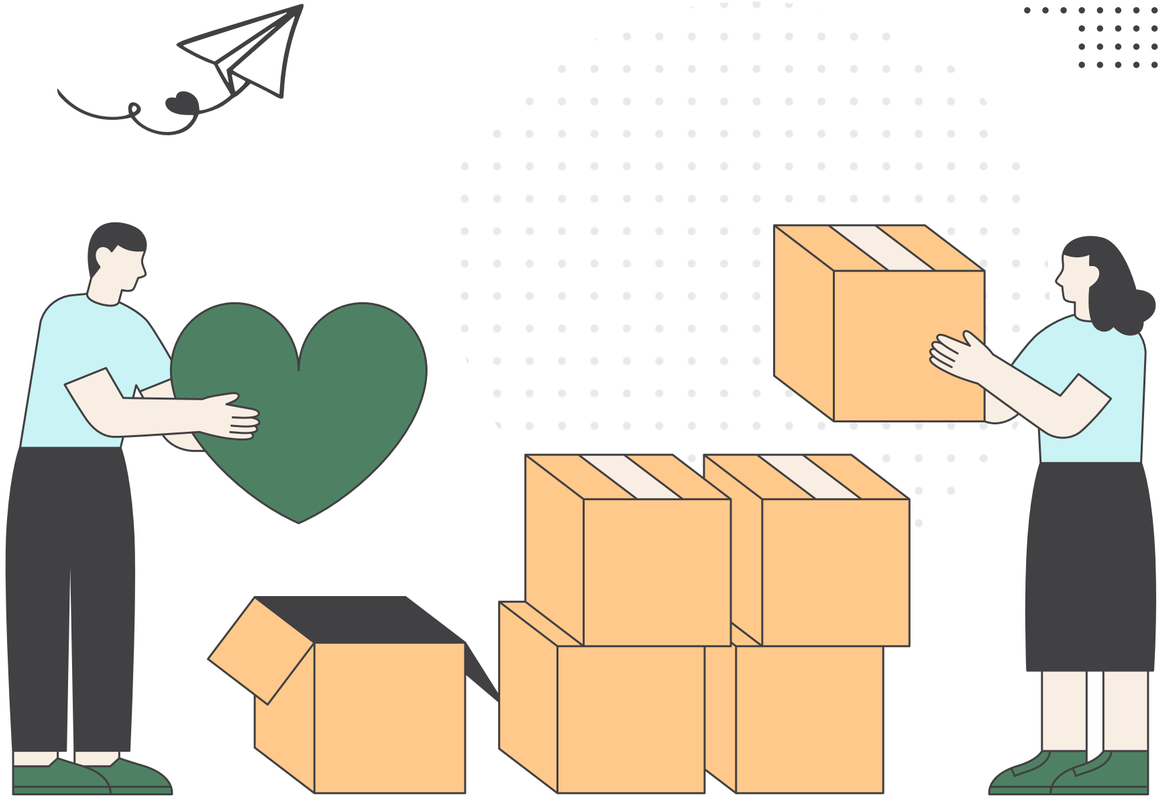
CLUSTERING

OBESITY LEVELS

TEAM:

- *Amira Nafisha Tsaqifa / 0706022310033*
- *Angeline Octavia Liemanto / 0706022310006*
- *Talitha Celin Widjaja / 0706022310019*



DATASET

This dataset is used to analyze factors that influence obesity levels in individuals, including demographic information such as age, gender, height, and weight, as well as variables related to eating habits, physical activity, water consumption, use of technology, and other lifestyle behaviors. The data provides a comprehensive view of how daily habits and physical conditions contribute to different obesity categories.

The dataset is structured and consists of 17 variables with numerical, categorical, and ordinal data types. It is complete and consistent, with no missing values, allowing it to be directly used for data analysis and interpretation of patterns related to obesity levels without requiring extensive data cleaning.

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

LIBRARY IMPORTS

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

INSTALL PACKAGE

```
pip install xlrd
```

The xlrd package is installed to enable reading and processing Excel files, particularly older .xls formats, ensuring the dataset can be successfully loaded into the analysis environment.

The imported libraries are used to support the entire data analysis and clustering process, including data loading and manipulation, numerical computation, and data visualization. They enable feature scaling and dimensionality reduction, the implementation of clustering methods such as K-Means and Hierarchical Clustering, visualization of cluster structures using dendrograms, and evaluation of clustering performance using silhouette analysis.

DATA UNDERSTANDING

1 LOAD DATA

```
# Displaying the First Rows of the Dataset
url = "https://raw.githubusercontent.com/amiranafisha/Obesity-Levels-Based-on-Eating-Habits-and-Physical-Condition/main/ObesityDataSet_raw_and_data_

df = pd.read_excel(url)
df.head()
```

✓ 0.8s

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC
0	Female	21.0	1.62	64.0	yes	no	2.0	3.0	Sometimes	no	2.0	no	0.0	1.0	no
1	Female	21.0	1.52	56.0	yes	no	3.0	3.0	Sometimes	yes	3.0	yes	3.0	0.0	Sometimes
2	Male	23.0	1.80	77.0	yes	no	2.0	3.0	Sometimes	no	2.0	no	2.0	1.0	Frequently
3	Male	27.0	1.80	87.0	no	no	3.0	3.0	Sometimes	no	2.0	no	2.0	0.0	Frequently
4	Male	22.0	1.78	89.8	no	no	2.0	1.0	Sometimes	no	2.0	no	0.0	0.0	Sometimes

MTRANS	NObeyesdad
Public_Transportation	Normal_Weight
Public_Transportation	Normal_Weight
Public_Transportation	Normal_Weight
Walking	Overweight_Level_I
Public_Transportation	Overweight_Level_II

This step loads the dataset from an external source and displays the first few rows of the data to provide an initial overview of the dataset structure, variables, and sample values.

2 DATA STRUCTURE OVERVIEW

```
# Checking Data Structure and Data Types
df.info()
```

```
<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
```

df.info() is used to display the dataset structure, including the number of rows, columns, data types, and missing values.

3 DESCRIPTIVE STATISTICS

```
# Descriptive Statistics of Numerical Variables
df.describe()
```

df.describe() is used to generate a statistical summary of numerical variables, including mean, standard deviation, minimum, maximum, and quartiles.

	Age	Height	Weight	FCVC	NCP	CH2O	FAF	TUE
count	2111.000000	2111.000000	2111.000000	2111.000000	2111.000000	2111.000000	2111.000000	2111.000000
mean	24.312600	1.701677	86.586058	2.419043	2.685628	2.008011	1.010298	0.657866
std	6.345968	0.093305	26.191172	0.533927	0.778039	0.612953	0.850592	0.608927
min	14.000000	1.450000	39.000000	1.000000	1.000000	1.000000	0.000000	0.000000
25%	19.947192	1.630000	65.473343	2.000000	2.658738	1.584812	0.124505	0.000000
50%	22.777890	1.700499	83.000000	2.385502	3.000000	2.000000	1.000000	0.625350
75%	26.000000	1.768464	107.430682	3.000000	3.000000	2.477420	1.666678	1.000000
max	61.000000	1.980000	173.000000	3.000000	4.000000	3.000000	3.000000	2.000000

4 UNIQUE VALUES EXPLORATION

```
# Display unique values for each column
for col in df.columns:
    print(f"Unique values for column '{col}':")
    print(df[col].unique())
    print("-" * 20)
```

✓ 0.0s

This step is used to display the unique values in each column to better understand the range of data, especially for categorical variables.

```
Unique values for column 'Gender':
['Female' 'Male']
--------------------
Unique values for column 'Age':
[21.       23.       27.       ... 22.524036 24.361936 23.664709]
--------------------
Unique values for column 'Height':
[1.62     1.52     1.8      ... 1.752206 1.73945  1.738836]
--------------------
Unique values for column 'Weight':
[ 64.        56.        77.       ... 133.689352 133.346641 133.472641]
--------------------
Unique values for column 'family_history_with_overweight':
['yes' 'no']
--------------------
Unique values for column 'FAVC':
['no' 'yes']
--------------------
Unique values for column 'FCVC':
[2.       3.       1.       2.450218 2.880161 2.00876  2.596579 2.591439
 2.392665 1.123939 2.027574 2.658112 2.88626  2.714447 2.750715 1.4925
 2.205439 2.059138 2.310423 2.823179 2.052932 2.596364 2.767731 2.815157
 2.737762 2.568063 2.524428 2.971574 1.0816   1.270448 1.344854 2.959658
 2.725282 2.844607 2.44004  2.432302 2.592247 2.449267 2.929889 2.015258
 1.031149 1.592183 1.21498  1.522001 2.703436 2.362918 2.14084  2.5596
...
['Normal_Weight' 'Overweight_Level_I' 'Overweight_Level_II'
 'Obesity_Type_I' 'Insufficient_Weight' 'Obesity_Type_II'
 'Obesity_Type_III']
```

DATA PREPARATION

1

```
# Checking for Missing Values - no missing values
df.isnull().sum()
```

```
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
```

2

```
# Check duplicates
df.duplicated().sum()
df.shape
```

✓ 0.0s

```
(2111, 17)
```

3

```
# Removing Duplicate Records and Verifying the Result
df = df.drop_duplicates()
df.duplicated().sum()
```

✓ 0.0s

```
0
```

4

```
# After duplicate
df.shape
```

✓ 0.0s

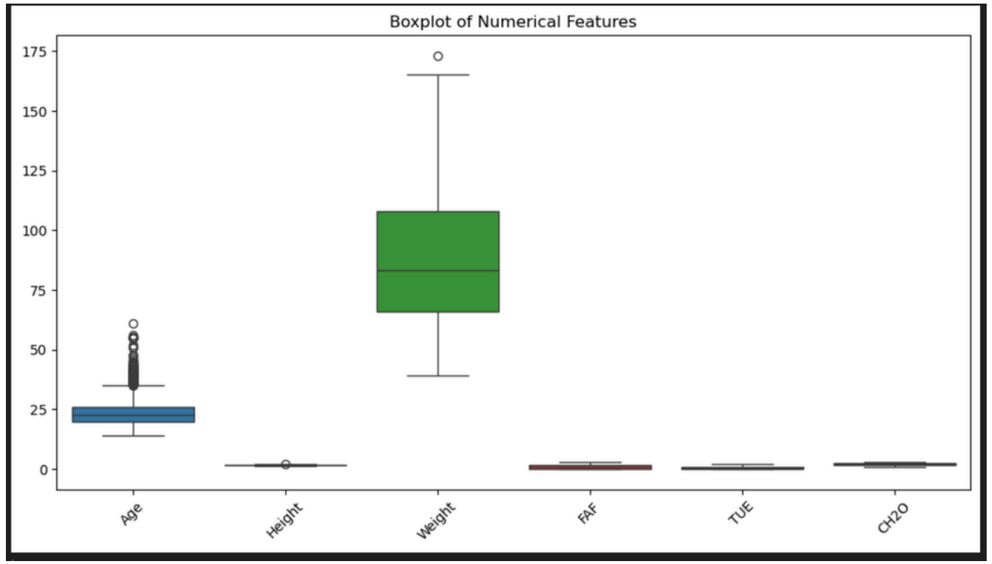
```
(2087, 17)
```

This step checks the dataset for missing values and duplicate records to ensure data quality. Since no missing values are found, the focus is on identifying and removing duplicate entries, followed by verification of the dataset size to confirm that duplicates have been successfully removed.

EXPLORATORY DATA ANALYSIS

1

```
# Boxplot of Numerical Features for Outlier Detection
numerical_cols = ['Age', 'Height', 'Weight', 'FAF', 'TUE', 'CH2O']

plt.figure(figsize=(12,6))
sns.boxplot(data=df[numerical_cols])
plt.xticks(rotation=45)
plt.title('Boxplot of Numerical Features')
plt.show()
```




```
# Ages over 50 removed
outlier = df[df['Age'] >= 50].shape[0]
print(f"Number of rows: {outlier}")
df = df[df['Age'] <= 50]
```

✓ 0.0s

```
Number of rows: 10
```

```
# Weight over 170 removed
outlier = df[df['Weight'] >= 170].shape[0]
print(f"Number of rows: {outlier}")
df = df[df['Weight'] <= 170]
```

✓ 0.0s

```
Number of rows: 1
```

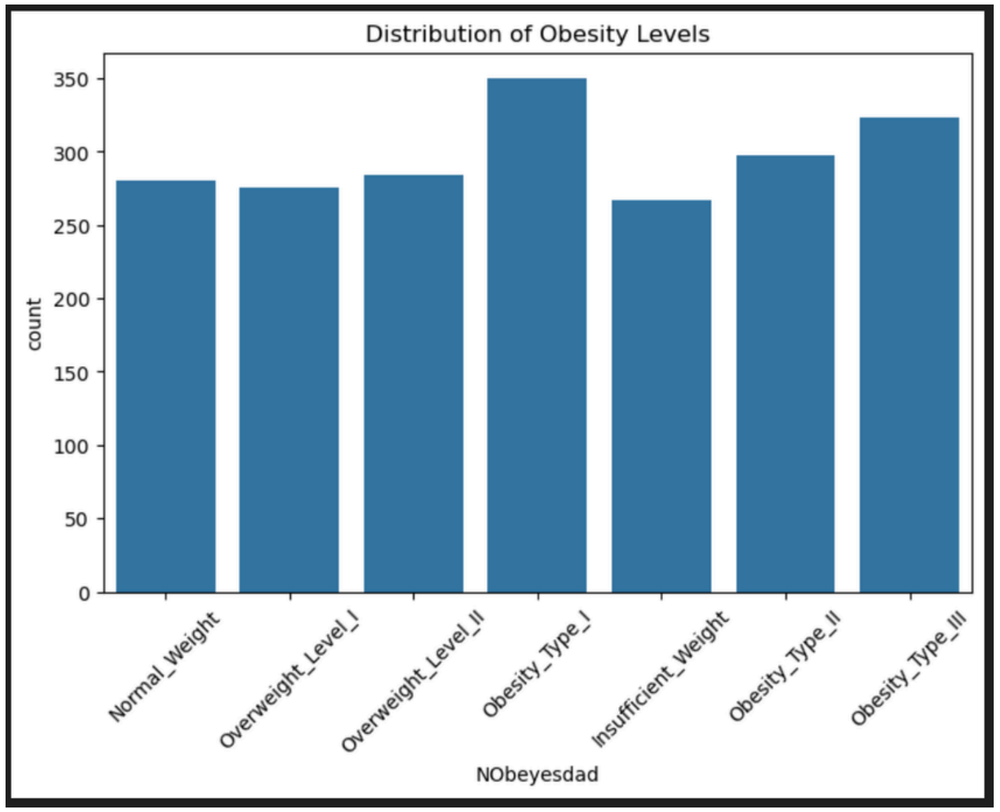
This step removes extreme values from the dataset by applying threshold-based filtering on the Age and Weight variables. Records with age above 50 and weight above 170 kg are excluded to reduce the influence of extreme outliers and improve the robustness of the subsequent analysis.

This boxplot is used to identify potential outliers in the numerical features of the dataset, including age, height, weight, physical activity frequency (FAF), time using technological devices (TUE), and daily water consumption (CH2O). Several potential outliers are observed in variables such as Age and Weight, which may influence the analysis and should be further examined before modeling.

2

```
# Distribution of Obesity Levels
plt.figure(figsize=(8,5))
sns.countplot(data=df, x='NObeyesdad')
plt.xticks(rotation=45)
plt.title('Distribution of Obesity Levels')
plt.show()
```

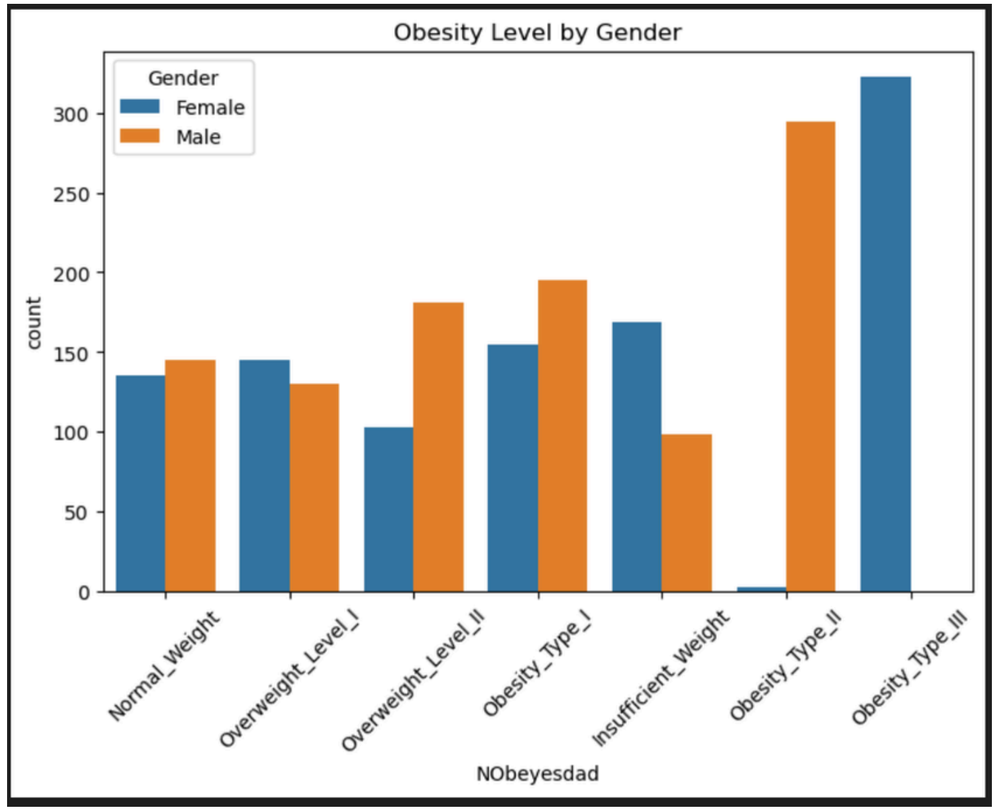
The distribution of obesity levels shows how individuals are spread across different obesity categories in the dataset. This visualization provides an overview of the prevalence of each obesity level and helps identify underlying patterns in the data. Understanding this distribution supports the interpretation of clustering results and helps evaluate whether the formed clusters correspond to meaningful obesity-related groups.



3

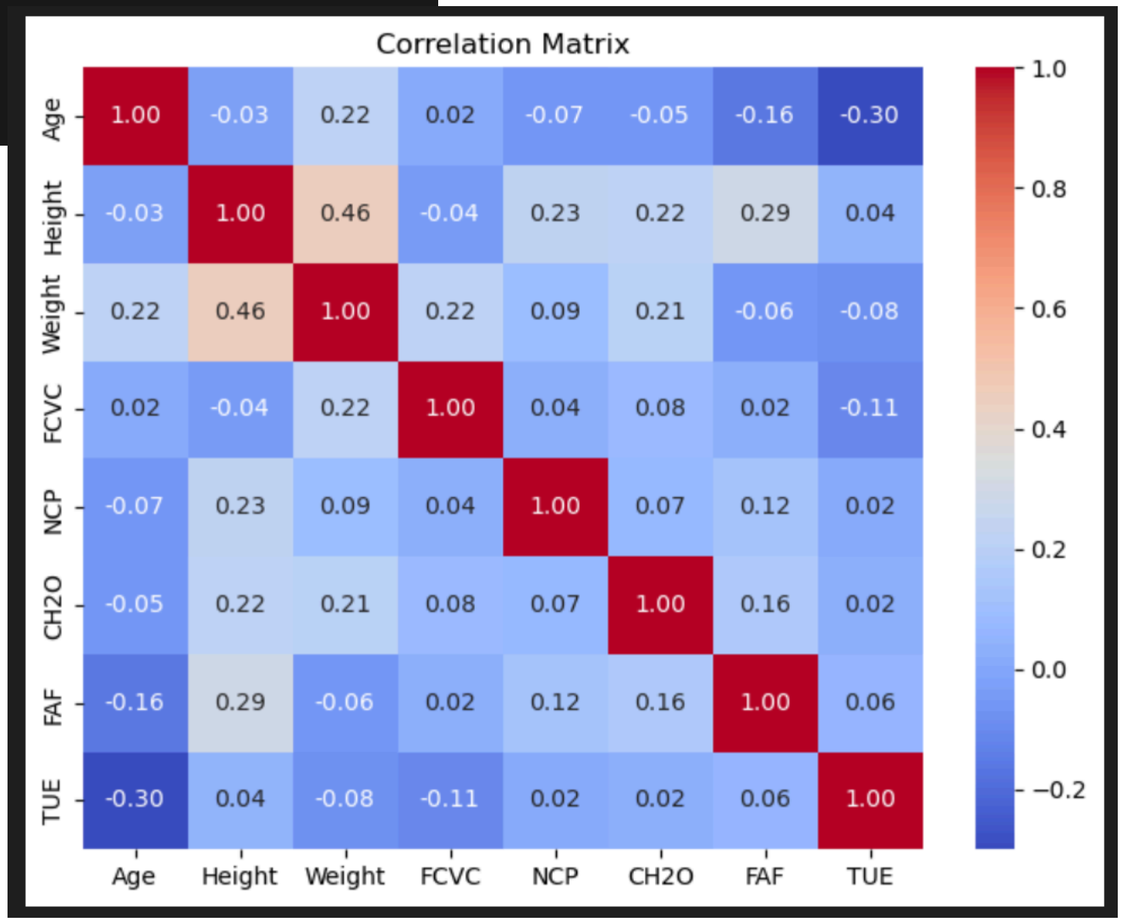
```
# Gender vs Obesity Level
plt.figure(figsize=(8,5))
sns.countplot(data=df, x='NObeyesdad', hue='Gender')
plt.xticks(rotation=45)
plt.title('Obesity Level by Gender')
plt.show()
```

This visualization compares obesity levels across genders to identify potential differences in obesity patterns between male and female individuals. It highlights gender-related trends in obesity prevalence, providing additional context for understanding how gender may influence patterns within the data. These insights help support the interpretation of clustering results by indicating whether gender-related differences align with the clusters formed in the analysis.



4

```
# Correlation Matrix of Numerical Variables
plt.figure(figsize=(8,6))
sns.heatmap(df[['Age','Height','Weight','FCVC','NCP','CH2O','FAF','TUE']].corr(),
            annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```



The correlation matrix visualizes the relationships among numerical variables by displaying their correlation coefficients. It helps identify variables with strong positive or negative associations, which is useful for understanding feature dependencies and detecting potential multicollinearity. This insight supports feature selection and aids in interpreting variables that may influence obesity patterns and the formation of meaningful clusters.

FEATURE ENGINEERING

1 ENCODING

```python
df_encoded_one = df.copy()

# Label Encoding (binary columns)
label_encode = {
    'Gender': {'Female': 0, 'Male': 1},
    'family_history_with_overweight' : {'no': 0, 'yes': 1},
    'FAVC' : {'no': 0, 'yes': 1},
    'SMOKE' : {'no': 0, 'yes': 1},
    'SCC' : {'no': 0, 'yes': 1},
}

for col, mapping in label_encode.items():
    df_encoded_one[col] = df_encoded_one[col].replace(mapping)

# One Hot Encoding (categorical columns)
df_encoded_one = pd.get_dummies(
    df_encoded_one,
    columns=['MTRANS', 'CAEC', 'CALC'],
    drop_first=True
)
```

```python
# Ordinal Encoding (for CAEC and CALC)
CAEC_mapping = {
    'no': 0,
    'Sometimes': 1,
    'Frequently': 2,
    'Always': 3
}

CALC_mapping = {
    'no': 0,
    'Sometimes': 1,
    'Frequently': 2,
    'Always': 3
}

df_encoded['CAEC'] = df_encoded['CAEC'].map(CAEC_mapping)
df_encoded['CALC'] = df_encoded['CALC'].map(CALC_mapping)
```

```python
# Nominal Encoding
nominal_cols = ['MTRANS']
encoder = OrdinalEncoder()
df_encoded[nominal_cols] = encoder.fit_transform(df_encoded[nominal_cols])
```

Previously, we tried using ordinal and nominal encoding (not one-hot) but the silhouette score was lower than one-hot, so we decided to use one-hot.

We applied **label encoding** to binary columns by converting their text values into numbers (0 and 1) so the model can understand them. Then, we applied **one-hot encoding** to categorical columns with more than two categories and has no order (ranking).

MTRANS_Motorbike	MTRANS_Public_Transportation	MTRANS_Walking
0	1	0
0		
0		

Gender	family_history_with_overweight
0	1
0	1

2 ADDING NEW FEATURE (VARIABLE)

```
df_encoded_one['BMI'] = df_encoded_one['Weight'] / (df_encoded_one['Height'] ** 2)
```

BMI
24.386526
24.238227
23.765432
26.851852

Next, we added a new feature, BMI, to improve the model's performance.

3 SCALING

```
from sklearn.preprocessing import StandardScaler

# Columns to scale
scale_cols = ['Age', 'Height', 'Weight', 'BMI', 'FCVC', 'NCP', 'CH2O', 'FAF', 'TUE']

# Columns not to scale
other_cols = [col for col in df_encoded_one.columns if col not in scale_cols]

sc = StandardScaler()
scaled_data = sc.fit_transform(df_encoded_one[scale_cols])
df_scaled = pd.DataFrame(scaled_data, columns=scale_cols)
df_encoded_one_sc = pd.concat([df_encoded_one[other_cols].reset_index(drop=True), df_scaled.reset_index(drop=True)], axis=1)
```

Age	Height	Weight	BMI	FCVC	NCP	CH2O	FAF	
-0.533092	-0.886059	-0.873587	-0.670617	-0.787768	0.392179	-0.006437	-1.184973	0.5
-0.533092	-1.958658	-1.179312	-0.689090	1.082123	0.392179	1.635753	2.329662	-1.0
-0.200982	1.044619	-0.376784	-0.747987	-0.787768	0.392179	-0.006437	1.158117	0.5
0.463237	1.044619	0.005372	-0.363507	1.082123	0.392179	-0.006437	1.158117	-1.0
-0.367037	0.830099	0.112376	-0.177829	-0.787768	-2.223023	-0.006437	-1.184973	-1.0

We created two datasets to compare the modeling results with and without scaling. We used **StandardScaler**. The first dataset (df_encoded_one) is unscaled and the second dataset (df_encoded_one_sc) is scaled, where numerical columns are standardized so they have a similar scale.

```python
# Drop the target variable
df_encoded_one = df_encoded_one.drop(columns=['NObeyesdad'])
df_encoded_one_sc = df_encoded_one_sc.drop(columns=['NObeyesdad'])


for col in df_encoded_one.select_dtypes(include='bool').columns:
    df_encoded_one[col] = df_encoded_one[col].astype(int)

for col in df_encoded_one_sc.select_dtypes(include='bool').columns:
    df_encoded_one_sc[col] = df_encoded_one_sc[col].astype(int)
```

After that, we removed the target variable (NObeyesdad) from both datasets and converted any boolean columns into 0 and 1 so all features are fully numeric and ready for modeling.

MODELING

1 K-MEANS

```
from sklearn.cluster import KMeans
from kneed import KneeLocator
```

```
wcss = []
for i in range(1, 11):   #clusters 1-10
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
    kmeans.fit(df_encoded_one)
    wcss.append(kmeans.inertia_)

# Plot the elbow method graph
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

```
# Create a DataFrame to display the WCSS values
wcss_table = pd.DataFrame({
    "Number of Clusters": range(1, 11),
    "WCSS": wcss
})
```

First we used the **Elbow Method** to find the best number of clusters for K-Means. We run K-Means with 1 to 10 clusters, calculate the **WCSS** (how compact the clusters are), and store the values. Then we plot WCSS to see where the decrease starts to slow down (the “elbow”). We also use **KneeLocator** to automatically detect that elbow point and choose it as the optimal number of clusters.

```
# Example WCSS values and cluster numbers
clusters = range(1, 11)

# Automatically find the elbow point using the KneeLocator
knee_locator = KneeLocator(clusters, wcss, curve="convex", direction="decreasing")
optimal_clusters = knee_locator.knee

# Plot the Elbow Method graph
plt.figure(figsize=(8, 5))
plt.plot(clusters, wcss, marker='o', linestyle='--', label="WCSS")
plt.axvline(optimal_clusters, linestyle='--', color='red', label=f'Optimal Clusters: {optimal_clusters}')
plt.scatter(optimal_clusters, wcss[optimal_clusters-1], c='red', s=100, zorder=5)  # Highlight elbow point
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.legend()
plt.grid(True)
plt.show()

# Print the optimal number of clusters
print(f"The optimal number of clusters is: {optimal_clusters}")
```

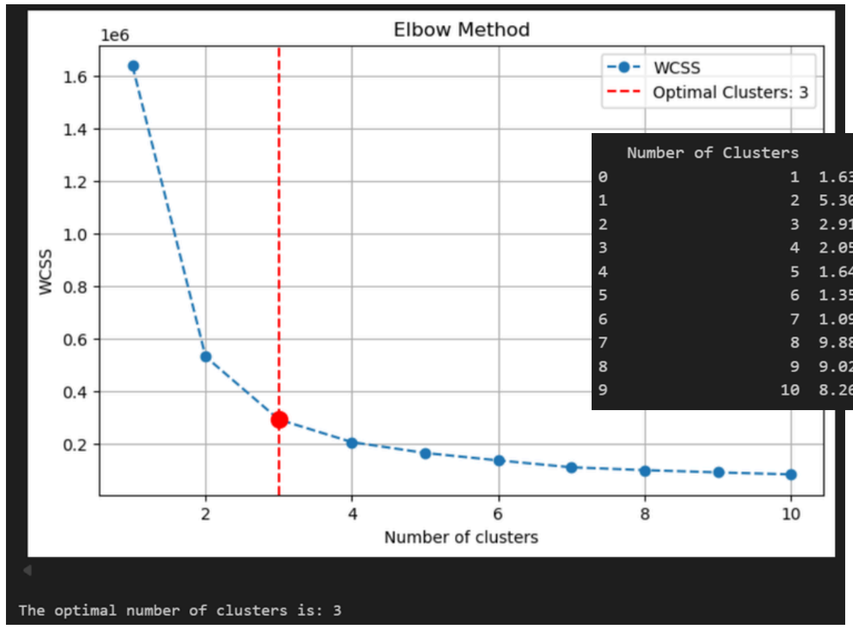
```
# Fit K-Means to the data with the optimal number of K cluster
kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=300, random_state=0)
clusters = kmeans.fit_predict(df_encoded_one)

# Add the cluster labels to the DataFrame
df['KMeans_Cluster'] = clusters
sil_score = silhouette_score(df_encoded_one, clusters)
print("Silhouette Score:", sil_score)
```

After deciding the optimal number of clusters, we fit the **K-Means model** then the cluster labels are then added to the dataset. Finally, we calculate the **Silhouette Score** to measure how well the data points are grouped.

K-MEANS RESULT

The number of clusters in **unscaled data is 3**, but in **scaled data is 4**. And the silhouette score of unscaled is higher than the scaled data. We assume that the unscaled dataset has better clustering quality.

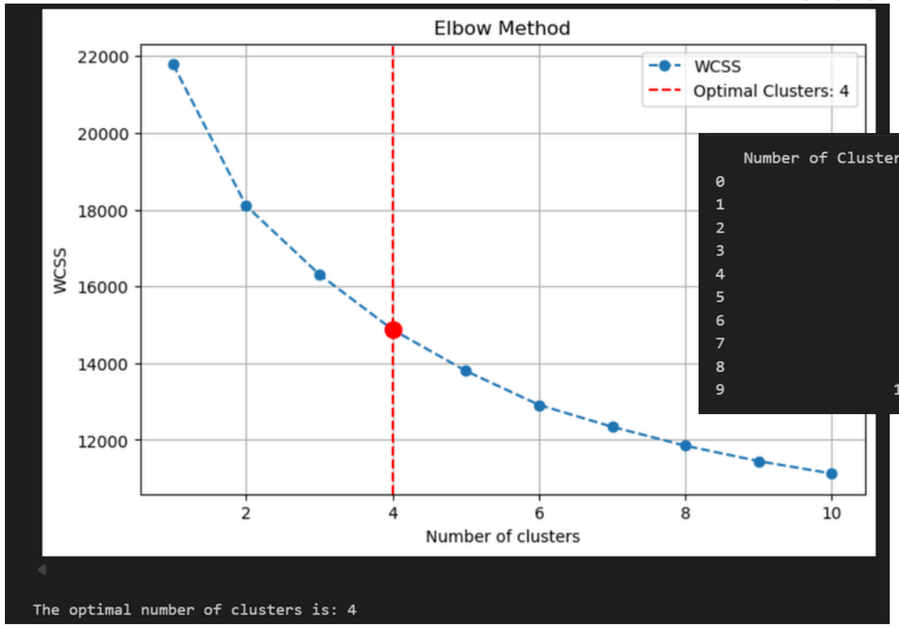
UNSCALED DATA



	Number of Clusters	WCSS
0	1	1.638542e+06
1	2	5.308059e+05
2	3	2.915403e+05
3	4	2.054610e+05
4	5	1.642046e+05
5	6	1.358987e+05
6	7	1.096328e+05
7	8	9.884481e+04
8	9	9.029207e+04
9	10	8.261226e+04

```
Silhouette Score: 0.5035448905724916
```

SCALED DATA



	Number of Clusters	WCSS
0	1	21787.766378
1	2	18096.503588
2	3	16309.998203
3	4	14870.453619
4	5	13797.007992
5	6	12909.812696
6	7	12340.035037
7	8	11846.257970
8	9	11445.164969
9	10	11121.898471

```
Silhouette Score: 0.10759946645770473
```

2 HIERARCHICAL AGGLOMERATIVE CLUSTERING

```
from sklearn.cluster import AgglomerativeClustering
```

```
silhouette_scores = []

#tentuin sndr rangenya (mau brp k)
for i in range(2, 11):
    # Create a new AgglomerativeClustering instance for each number of clusters
    agglo_model = AgglomerativeClustering(n_clusters=i, linkage='single')
    cluster_labels = agglo_model.fit_predict(df_encoded_one)
    silhouette_avg = silhouette_score(df_encoded_one, cluster_labels)
    silhouette_scores.append(silhouette_avg)

# Plot Silhouette Score = the higher the better
plt.plot(range(2, 11), silhouette_scores, marker='o')
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Score")
plt.title("Silhouette Score")
plt.show()
```

```
# Define the range of clusters to test and linkage methods
range_n_clusters = range(2, 11)

#linkage - jarak antar clusternya
linkage_methods = ['ward', 'complete', 'average', 'single']

# metric (Euclidean) - jarak antar data points, 1 titik ke titik lain
metrics = ['euclidean', 'manhattan']

best_silhouette_score = -1
best_n_clusters = 0
best_linkage = ''
best_metric = ''

results_for_plotting = []
```

```
for linkage_method in linkage_methods:
    for metric in metrics:
        # Skip 'ward' linkage with 'manhattan' metric as it's not supported
        # khusus WARD, ga bisa dipake bareng manhattan
        if linkage_method == 'ward' and metric == 'manhattan':
            print(f"Skipping Ward linkage with Manhattan metric (not supported).")
            continue

        current_scores = []
        for n_clusters in range_n_clusters:
            agglo = AgglomerativeClustering(n_clusters=n_clusters, linkage=linkage_method, metric=metric)
            y_agglo = agglo.fit_predict(df_encoded_one)
            silhouette_avg = silhouette_score(df_encoded_one, y_agglo)
            current_scores.append(silhouette_avg)

            if silhouette_avg > best_silhouette_score:
                best_silhouette_score = silhouette_avg
                best_n_clusters = n_clusters
                best_linkage = linkage_method
                best_metric = metric
        results_for_plotting.append({
            'linkage': linkage_method,
            'metric': metric,
            'scores': current_scores
        })
```

We used **Silhouette Score** to evaluate Agglomerative (Hierarchical) Clustering and find the best clustering setup.

First, it calculates and plots silhouette scores for different numbers of clusters (from 2 to 10) using a single linkage, where a higher score means better clustering. Then, it tests multiple combinations of linkage methods and distance metrics to see which combination gives the highest silhouette score.

All results are plotted for comparison, and the best number of clusters, linkage method, and metric are selected based on the highest silhouette score.

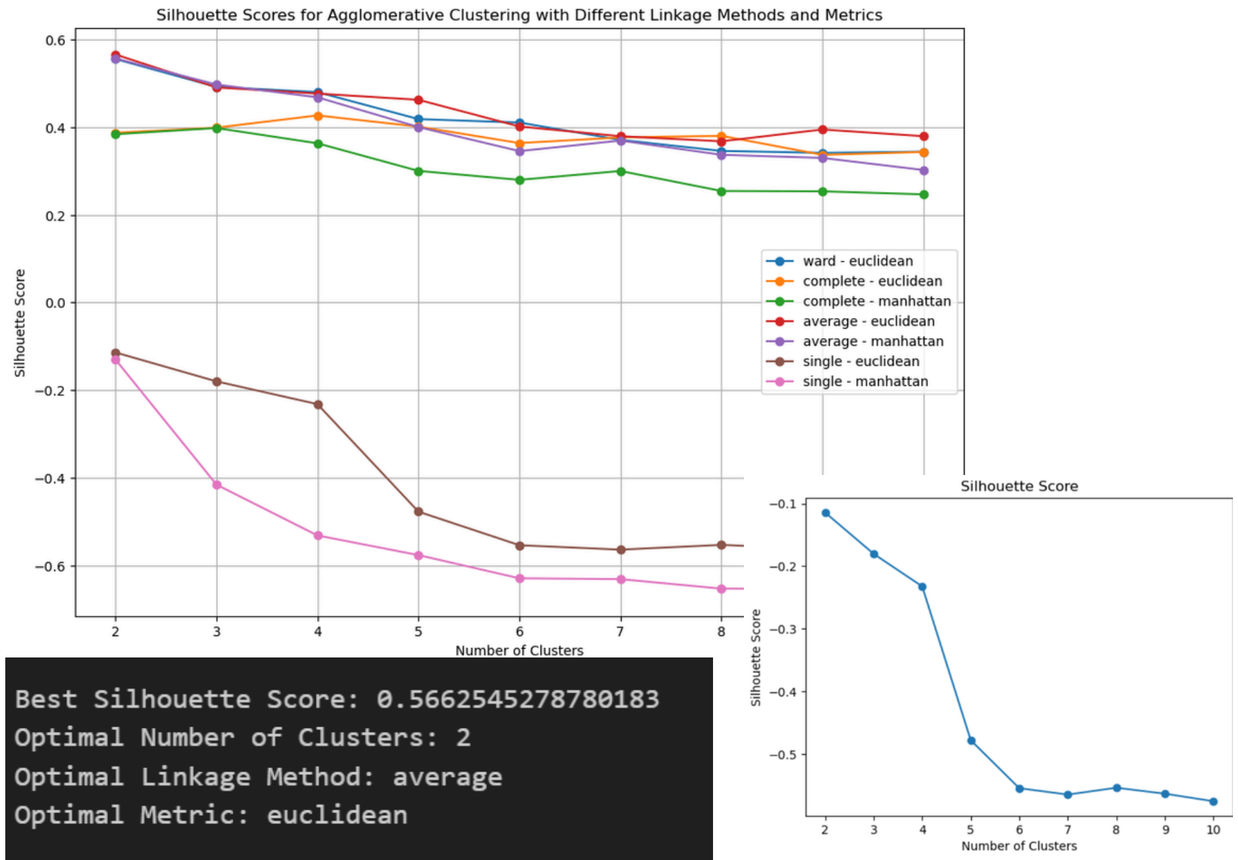
```
# Hierarchical Clustering
agglo_unscaled = AgglomerativeClustering(n_clusters=best_n_clusters, linkage=best_linkage, metric=best_metric)
y_unscaled = agglo_unscaled.fit_predict(df_encoded_one)
```

Finally, the Agglomerative Clustering model is fitted using these best parameters to produce the final cluster labels.

HAC RESULT

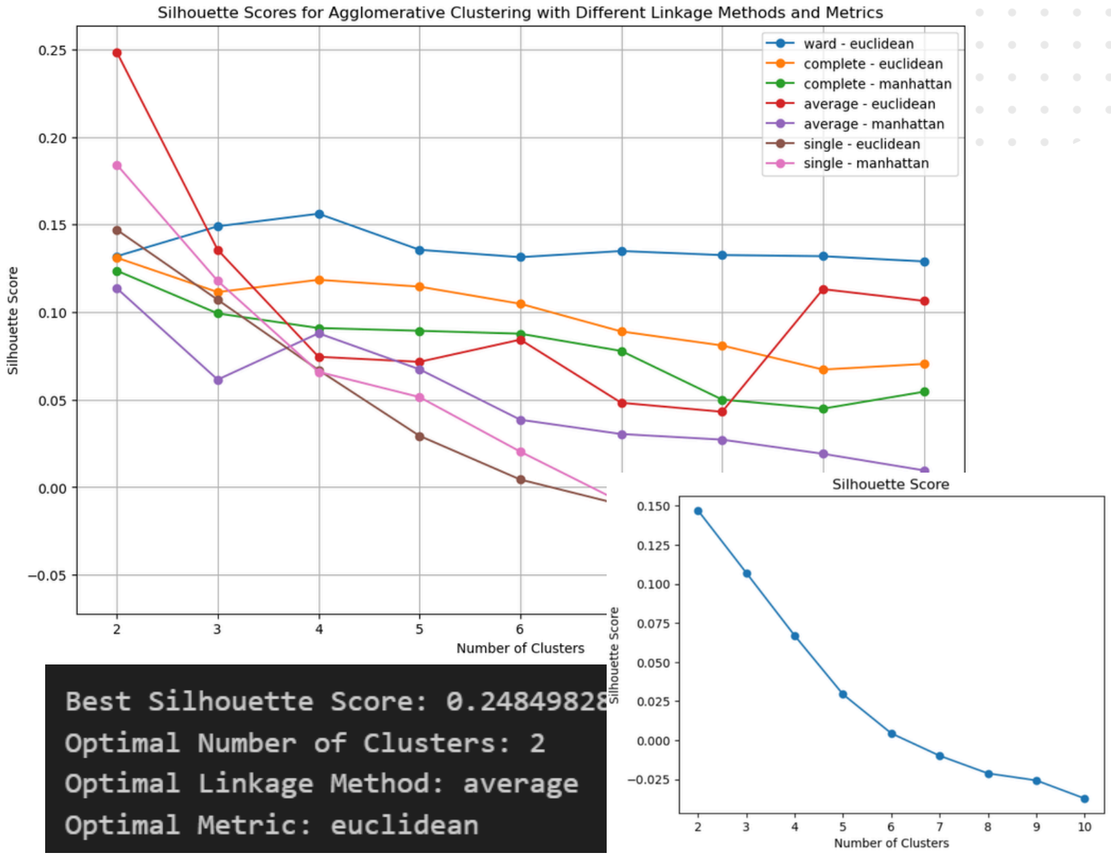
The number of clusters, linkage method and metric are the same for both unscaled and scaled data. But the silhouette score result of unscaled data is much higher than the scaled data. We aasume the unscaled data has better clustering quality.

UNSCALED DATA




```
Best Silhouette Score: 0.5662545278780183
Optimal Number of Clusters: 2
Optimal Linkage Method: average
Optimal Metric: euclidean
```

SCALED DATA




```
Best Silhouette Score: 0.24849828
Optimal Number of Clusters: 2
Optimal Linkage Method: average
Optimal Metric: euclidean
```

3 PCA + HIERARCHICAL AGGLOMERATIVE CLUSTERING

```
# PCA AFTER scaling
pca = PCA(n_components=0.9, random_state=42)
X_pca = pca.fit_transform(df_encoded_one_sc)

print("Number of PCA components:", X_pca.shape[1])

# Agglomerative with stable configuration
agglo_pca = AgglomerativeClustering(
    n_clusters=3,
    linkage='ward'   # FIXED ON PURPOSE
)

y_agglo_pca = agglo_pca.fit_predict(X_pca)

print("Agglo + PCA Silhouette:",
      silhouette_score(X_pca, y_agglo_pca))

print("Cluster distribution (Agglo + PCA):")
print(pd.Series(y_agglo_pca).value_counts())
```

```
Number of PCA components: 9
Agglo + PCA Silhouette: 0.1615775774671269
Cluster distribution (Agglo + PCA):
2    903
1    658
0    515
Name: count, dtype: int64
```

We applied feature scaling and then used Principal Component Analysis (PCA) to reduce dimensionality while preserving 90% of the total variance. This resulted in 9 principal components that summarize the original features and reduce noise and redundancy.

Then, we performed Hierarchical Agglomerative Clustering using Ward linkage with 3 clusters on the PCA-transformed data. Ward linkage was chosen to minimize within-cluster variance and produce more compact clusters.

Next, we evaluated the clustering result using the Silhouette Score. Although the silhouette score after PCA is lower compared to the non-PCA model, the resulting clusters are more balanced and stable, without extreme or singleton clusters.

Finally, we assigned the cluster labels back to the original dataset to enable cluster profiling and interpretation using the original variables. This allows further analysis and supports managerial insights.

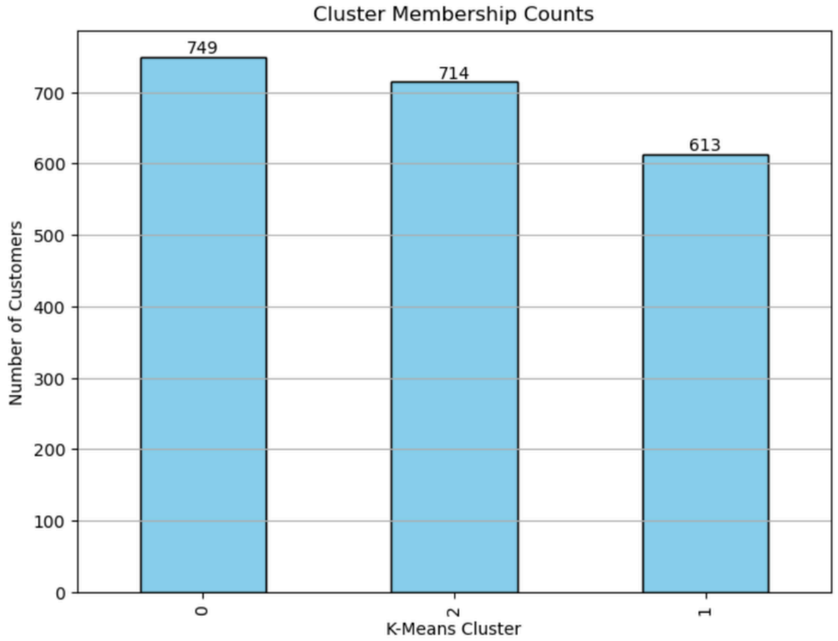
```
# Simpan hasil clustering PCA ke dataframe utama
df['Agglo_PCA'] = y_agglo_pca
```

```
# Add the cluster labels to the original data
df['Agglo_Scaled'] = y_scaled
df['Agglo_Unscaled'] = y_unscaled

# Analyze the clusters with the original values
df.head()
```

	FCVC	NCP	CAEC	SMOKE	...	FAF	TUE	CALC	MTRANS	NObeyesdad	KMeans_Cluster	KMeans_Cluster_SC	Agglo_PCA	Agglo_Scaled	Agglo_Unscaled
...	2.0	3.0	Sometimes	no	...	0.0	1.0	no	Public_Transportation	Normal_Weight	1	1	2	0	1
...	3.0	3.0	Sometimes	yes	...	3.0	0.0	Sometimes	Public_Transportation	Normal_Weight	1	1	2	0	1
...	2.0	3.0	Sometimes	no	...	2.0	1.0	Frequently	Public_Transportation	Normal_Weight	0	1	2	0	1
...	3.0	3.0	Sometimes	no	...	2.0	0.0	Frequently	Walking	Overweight_Level_I	0	0	0	0	1
...	2.0	1.0	Sometimes	no	...	0.0	0.0	Sometimes	Public_Transportation	Overweight_Level_II	0	2	0	0	1

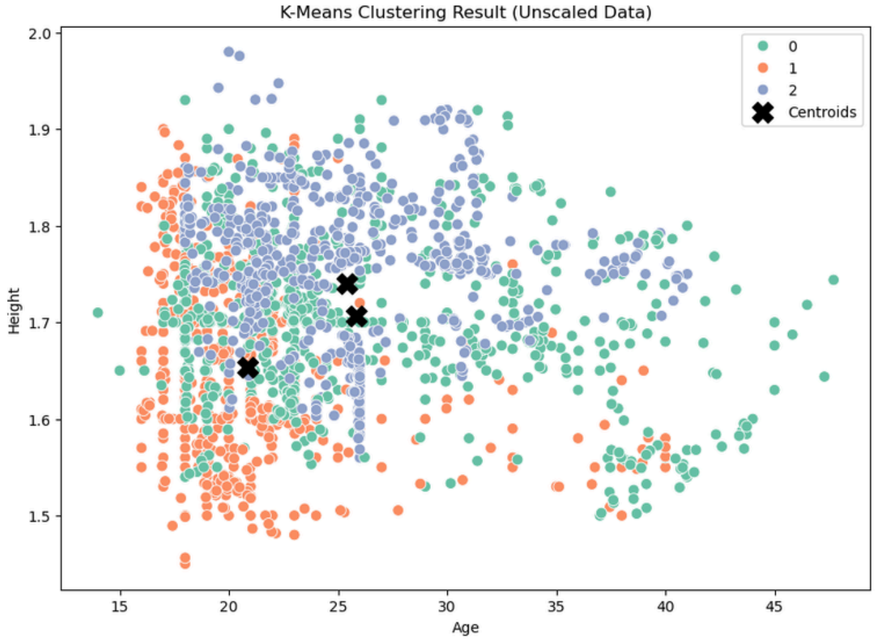
K-MEANS MODEL RESULT

UNSCALED DATA



Cluster 0 : 749 members
Cluster 1 : 613 members
Cluster 2 : 714 members

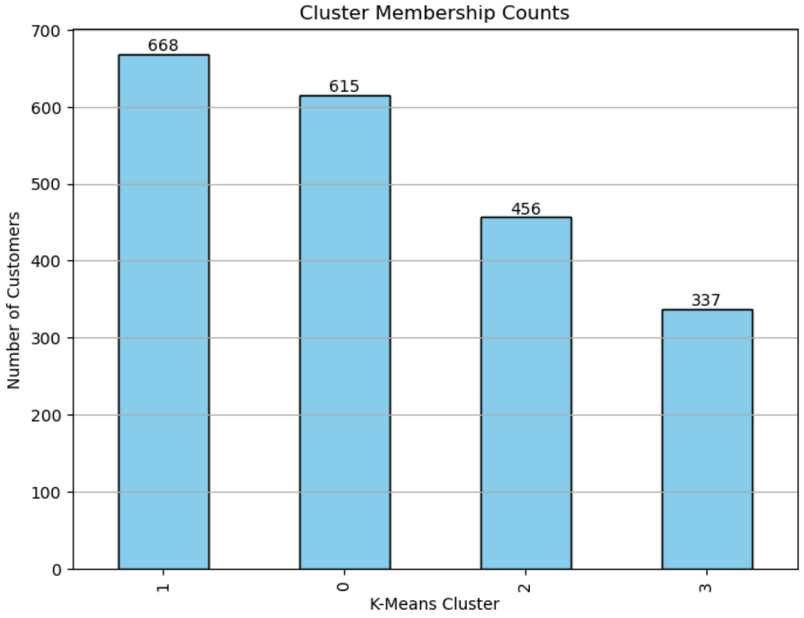
The clustering results are quite evenly distributed across clusters.



The data points are grouped into three clusters based on age and height. Each cluster represents a different pattern, but there is some overlap, meaning the differences are not very sharp. Overall, the clustering shows a reasonable separation without one cluster dominating the others.

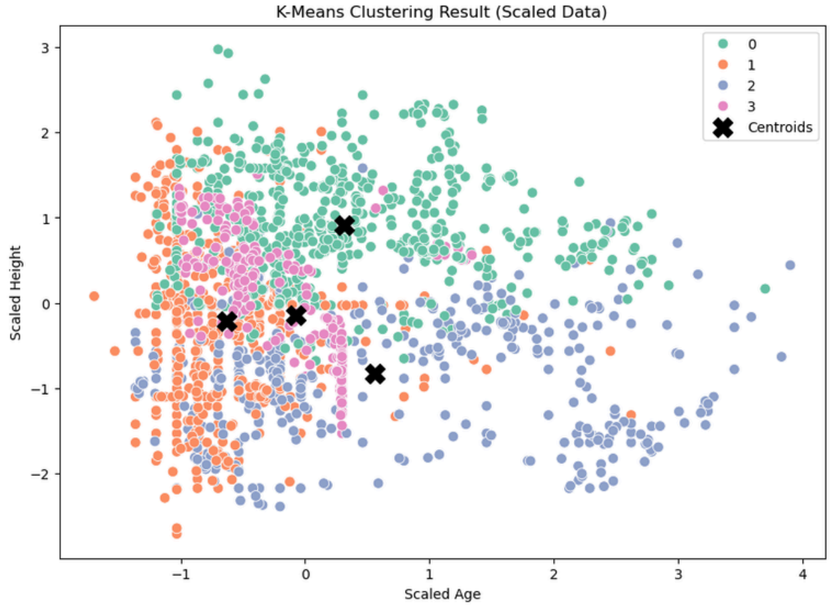
K-MEANS MODEL RESULT

SCALED DATA



Cluster 0 : 615 members
Cluster 1 : 668 members
Cluster 2 : 456 members
Cluster 3 : 337 members

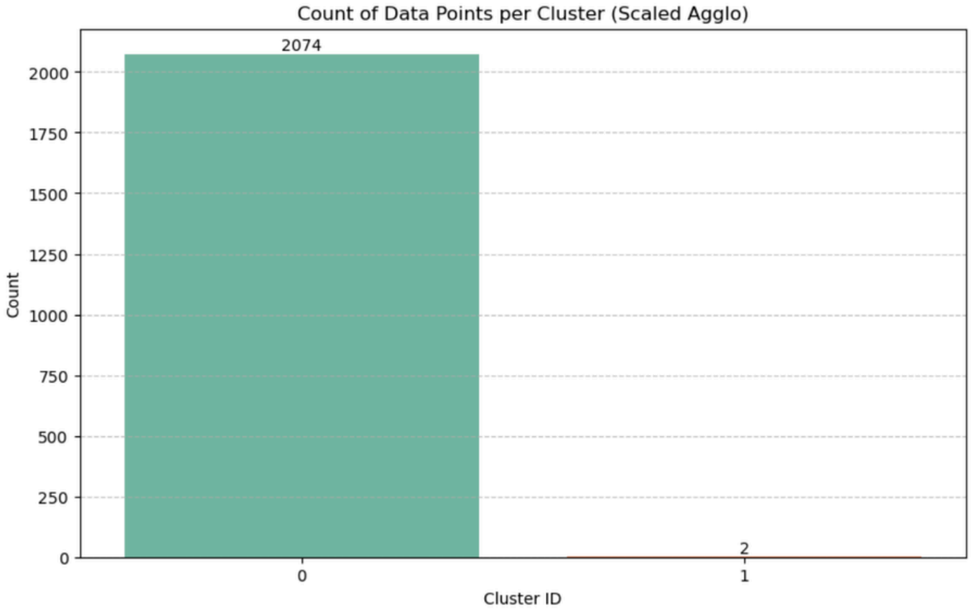
The clustering results are quite evenly distributed across clusters, with cluster 3 as minority.

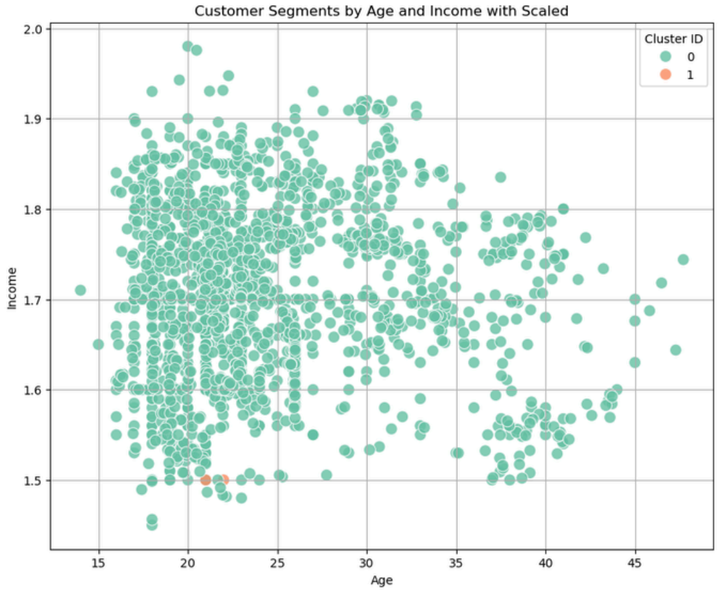


The scaled data shows four clearer clusters, with points more spread out compared to the unscaled version. Scaling helps separate the groups better, although some overlap between clusters still exists. Overall, the clusters are more distinct and easier to interpret after scaling.

AGGLO MODEL RESULT

SCALED DATA

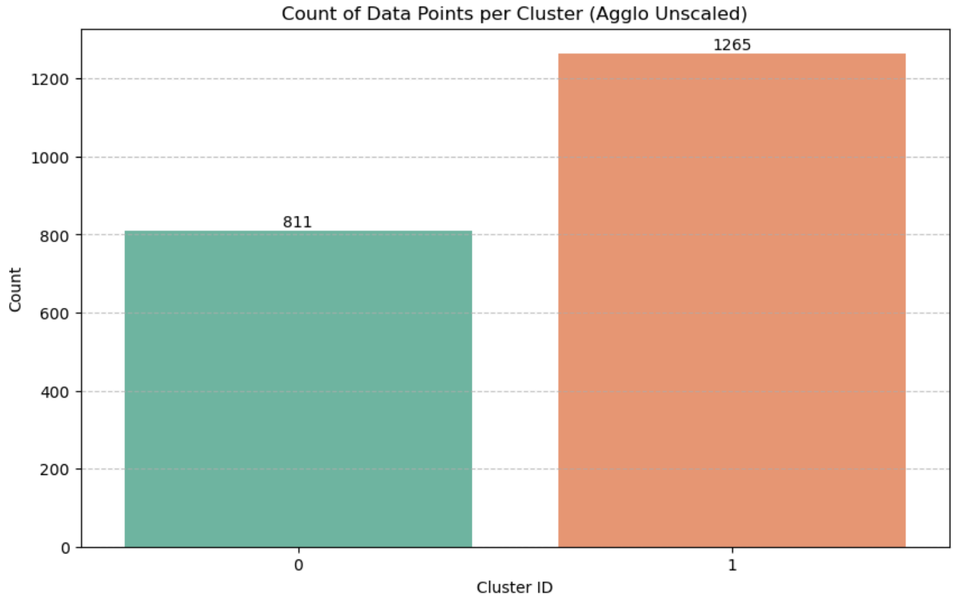


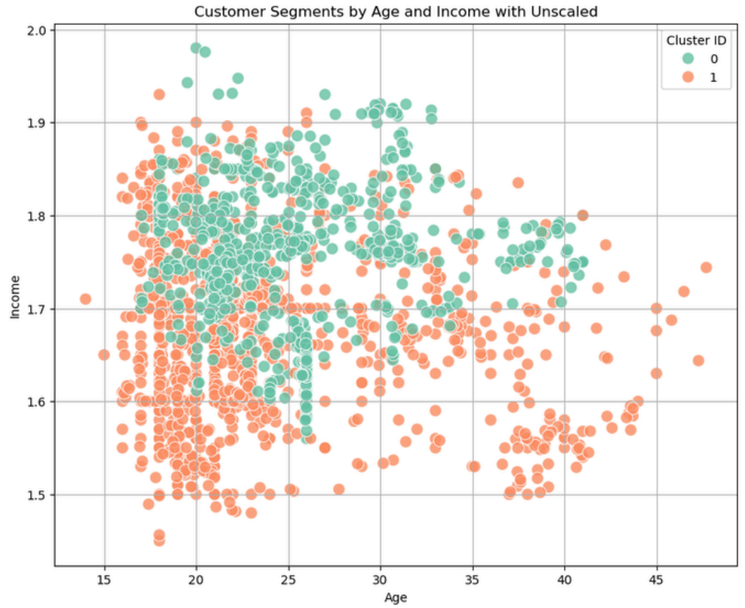


This plot shows the result of Agglomerative (hierarchical) clustering on scaled data using age and income.
Most customers fall into one main cluster, while only 2 members are grouped into a second cluster, indicating the data is quite similar overall. This suggests Agglomerative clustering is very sensitive to the data structure, and with this setting it did not strongly separate customers into distinct groups.

AGGLO MODEL RESULT

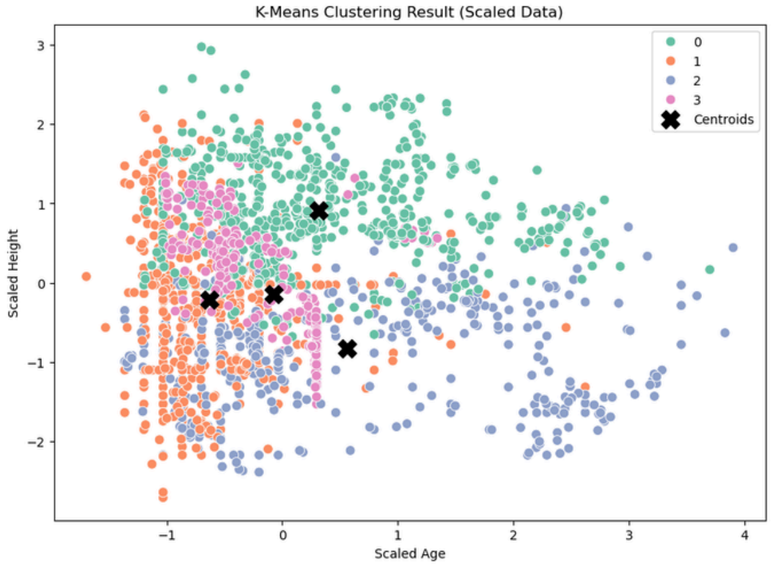
UNSCALED DATA





This plot shows the result of Agglomerative clustering on unscaled age and income data.
Compared to the scaled version, the two clusters are more mixed and overlap heavily, meaning the separation is not very clear. This suggests that without scaling, Agglomerative clustering struggles to clearly distinguish customer segments based on age and income.

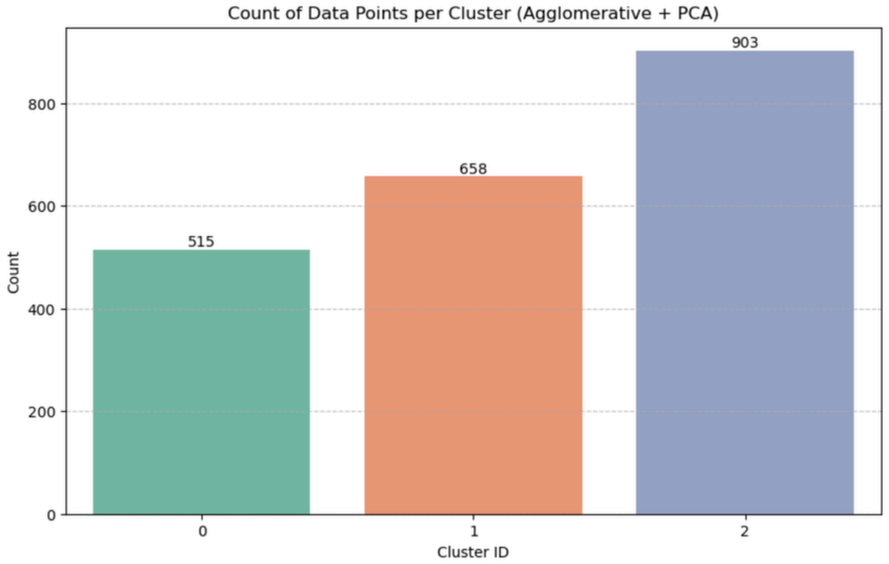
OVERALL MODEL RESULT



For K-Means, the Scaled Data has a better clustering result then the Unscaled Data. This does NOT align with the silhoutte score, where Scaled Data (0.1075) actually has a lower score then the Unscaled Data (0.5035)

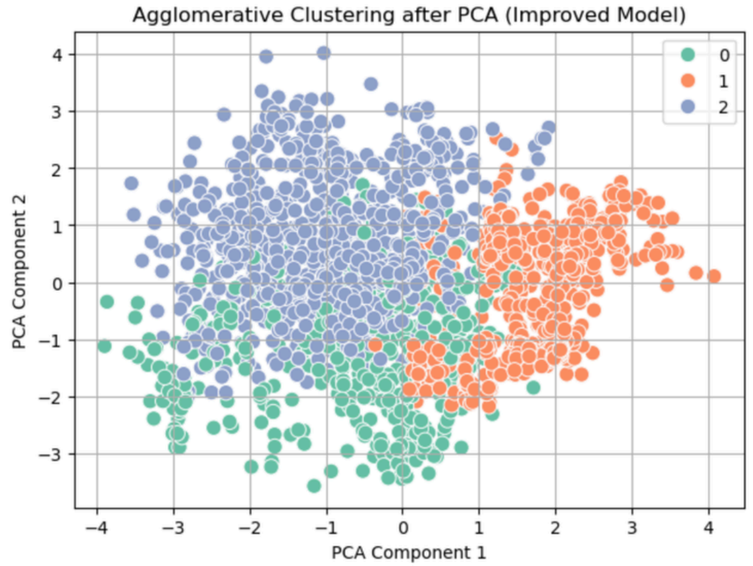
For Hierarchical Agglomerative modeling, the Unscaled Data has a better clustering result then the Scaled Data. This does align with the silhoutte score, where unscaled data (0.5662) has better score then scaled data (0.2484)

AGGLO + PCA MODEL RESULT



Cluster 0 : 515 members
Cluster 1 : 658 members
Cluster 2 : 903 members

The clustering results are quite evenly distributed across clusters, with cluster 0 as minority.



After applying PCA, the agglomerative clustering results show three clearer clusters in a reduced two-dimensional space. The data points are more compact and structured, indicating that PCA helps capture the main variance in the data. Although some overlap between clusters is still visible, the overall cluster separation is improved and easier to interpret.

MODEL EVALUATION

EVALUATION METHOD: SILHOUETTE SCORE, WCSS, VISUALIZATION

K-MEANS: BEST BALANCE (4 CLUSTERS, SCALED)

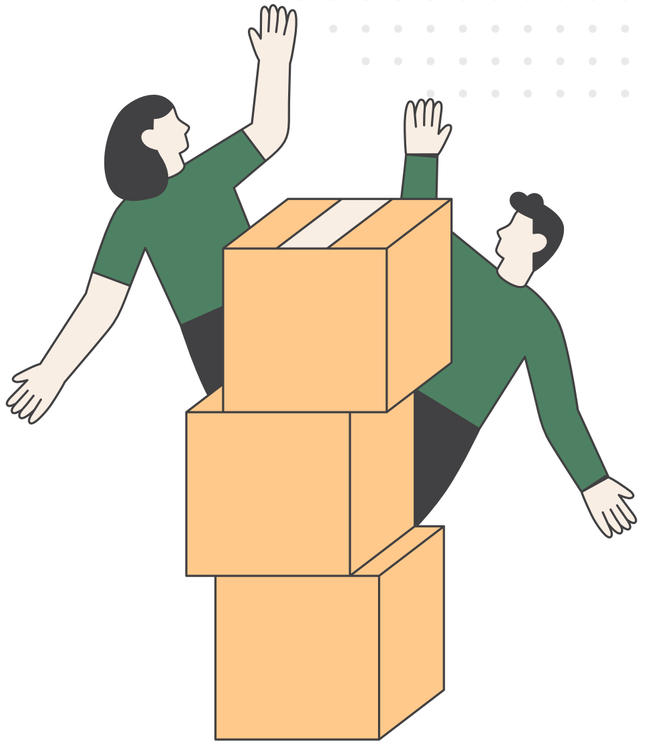
AGGLOMERATIVE: HIGHER SCORE, LESS DETAILED

PCA: BETTER VISUALIZATION, LOWER SCORE

FINAL MODEL: K-MEANS (SCALED)

INSIGHTS, INTERPRETATIONS AND CONCLUSIONS

- Dataset Quality: The dataset is complete, consistent, and suitable for clustering analysis without extensive data cleaning.
- Clustering Outcome: K-Means applied to the scaled dataset identified four distinct clusters, with a reasonably balanced distribution, indicating meaningful population segmentation.
- Behavioral Interpretation: The clusters capture differences in physical conditions, eating habits, and activity levels, demonstrating that obesity is influenced by multiple lifestyle-related factors rather than a single variable.
- Role of PCA: PCA improved visual interpretability by reducing dimensionality and clarifying cluster separation, although it slightly reduced clustering performance based on internal validation metrics.
- Final Conclusion: K-Means with four clusters on the scaled dataset provides the best balance between clustering performance, interpretability, and practical relevance for this study.



CLUSTERING

THANK YOU