In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

In [2]:
```
# Load the CSV file into a DataFrame
df = pd.read_csv('mcdonalds.csv')
```

In [3]:
```
# Display the first few rows of the DataFrame to verify it's loaded correctly
print(df.head())
```

```
  yummy convenient spicy fattening greasy fast cheap tasty expensive healthy  \
0    No        Yes    No       Yes     No  Yes   Yes    No       Yes      No
1   Yes        Yes    No       Yes    Yes  Yes   Yes   Yes       Yes      No
2    No        Yes   Yes       Yes    Yes  Yes    No   Yes       Yes     Yes
3   Yes        Yes    No       Yes    Yes  Yes   Yes   Yes        No      No
4    No        Yes    No       Yes    Yes  Yes   Yes    No        No     Yes

  disgusting Like  Age      VisitFrequency  Gender
0         No   -3   61  Every three months  Female
1         No   +2   51  Every three months  Female
2         No   +1   62  Every three months  Female
3        Yes   +4   69        Once a week  Female
4         No   +2   49        Once a month    Male
```

In [4]:
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1453 entries, 0 to 1452
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   yummy           1453 non-null   object
 1   convenient      1453 non-null   object
 2   spicy           1453 non-null   object
 3   fattening       1453 non-null   object
 4   greasy          1453 non-null   object
 5   fast            1453 non-null   object
 6   cheap           1453 non-null   object
 7   tasty           1453 non-null   object
 8   expensive       1453 non-null   object
 9   healthy         1453 non-null   object
 10  disgusting      1453 non-null   object
 11  Like            1453 non-null   object
 12  Age             1453 non-null   int64
 13  VisitFrequency  1453 non-null   object
 14  Gender          1453 non-null   object
dtypes: int64(1), object(14)
memory usage: 170.4+ KB
None
```

In [5]:
```
print(df.describe())
```

```
               Age
count  1453.000000
mean     44.604955
std      14.221178
min      18.000000
25%      33.000000
50%      45.000000
75%      57.000000
max      71.000000
```

In [6]:
```
print(df.isnull().sum())
```

```
yummy             0
convenient        0
spicy             0
fattening         0
greasy            0
fast              0
cheap             0
tasty             0
expensive         0
healthy           0
disgusting        0
Like              0
Age               0
VisitFrequency    0
Gender            0
dtype: int64
```

In [7]:
```
print(df.columns)
```

```
Index(['yummy', 'convenient', 'spicy', 'fattening', 'greasy', 'fast', 'cheap',
       'tasty', 'expensive', 'healthy', 'disgusting', 'Like', 'Age',
       'VisitFrequency', 'Gender'],
      dtype='object')
```

In [8]:
```
print(df.dtypes)
```

```
yummy             object
convenient        object
spicy             object
fattening         object
greasy            object
fast              object
cheap             object
tasty             object
expensive         object
healthy           object
disgusting        object
Like              object
Age                int64
VisitFrequency    object
Gender            object
dtype: object
```

In [9]:
```
# Check for duplicate rows
print(df.duplicated().sum())
```

```
22
```

In [10]:
```
# Display unique values for each column
for column in df.columns:
    print(f"{column}: {df[column].nunique()} unique values")
```

```
yummy: 2 unique values
convenient: 2 unique values
spicy: 2 unique values
fattening: 2 unique values
greasy: 2 unique values
fast: 2 unique values
cheap: 2 unique values
tasty: 2 unique values
expensive: 2 unique values
healthy: 2 unique values
disgusting: 2 unique values
Like: 11 unique values
Age: 54 unique values
VisitFrequency: 6 unique values
Gender: 2 unique values
```

In [11]:
```
# Preprocess the data (ensure non-numeric columns are handled appropriately, e.g., label encoding or dropping)
# For demonstration, assuming all columns are numeric or processed as such
data = df.select_dtypes(include=[np.number])  # Select only numeric columns for clustering
```

In [12]:
```
# Standardize or normalize the data if necessary
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)
```

In [13]:
```
# Run k-means for 2 to 8 segments with 10 random restarts (n_init)
segment_results = {}
for n_clusters in range(2, 9):  # 2 to 8 clusters
    kmeans = KMeans(n_clusters=n_clusters, n_init=10, random_state=42)
    kmeans.fit(data_scaled)
    segment_results[n_clusters] = kmeans.labels_

    # Print a summary for each k-means solution
    print(f"Number of clusters: {n_clusters}")
    print(f"Cluster labels: {np.unique(kmeans.labels_)}")
    print()
```

```
Number of clusters: 2
Cluster labels: [0 1]

Number of clusters: 3
Cluster labels: [0 1 2]

Number of clusters: 4
Cluster labels: [0 1 2 3]

Number of clusters: 5
Cluster labels: [0 1 2 3 4]

Number of clusters: 6
Cluster labels: [0 1 2 3 4 5]

Number of clusters: 7
Cluster labels: [0 1 2 3 4 5 6]

Number of clusters: 8
Cluster labels: [0 1 2 3 4 5 6 7]
```

In [14]:
```
# Optional: relabel segment numbers to be consistent across segmentations
# This step is complex and may require domain knowledge or matching centroids for consistency.

# Display the results for review
for n_clusters, labels in segment_results.items():
    df[f'Cluster_{n_clusters}'] = labels
```

In [15]:
```
# Save or display DataFrame with added cluster columns for analysis
print(df.head())
```

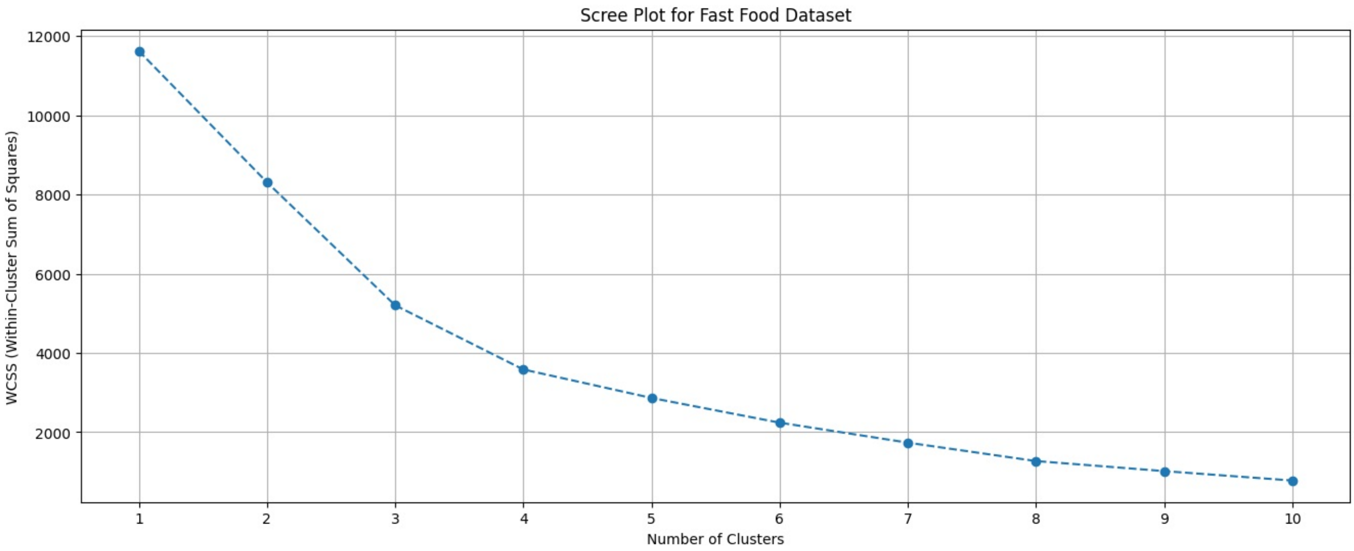
```
  yummy convenient spicy fattening greasy fast cheap tasty expensive healthy  \
0    No        Yes    No       Yes     No  Yes   Yes    No       Yes      No
1   Yes        Yes    No       Yes    Yes  Yes   Yes   Yes       Yes      No
2    No        Yes   Yes       Yes    Yes  Yes    No   Yes       Yes     Yes
3   Yes        Yes    No       Yes    Yes  Yes   Yes   Yes        No      No
4    No        Yes    No       Yes    Yes  Yes   Yes    No        No     Yes

   ... Age      VisitFrequency  Gender Cluster_2 Cluster_3  Cluster_4  \
0  ...  61  Every three months  Female         0         0          1
1  ...  51  Every three months  Female         0         2          2
2  ...  62  Every three months  Female         0         0          1
3  ...  69        Once a week  Female         0         0          1
4  ...  49       Once a month    Male         0         2          2

   Cluster_5  Cluster_6  Cluster_7  Cluster_8
0          3          2          5          5
1          1          1          0          1
2          3          4          2          5
3          3          4          2          3
4          1          1          0          7

[5 rows x 22 columns]
```

In [16]:
```
# Preprocess the data (select numeric columns)
data = df.select_dtypes(include=[np.number])
```

In [17]:
```
# Standardize the data
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)
```

In [18]:
```
# Calculate WCSS for different number of clusters
wcss = []
for i in range(1, 11):  # Range from 1 to 10 clusters
    kmeans = KMeans(n_clusters=i, n_init=10, random_state=42)
    kmeans.fit(data_scaled)
    wcss.append(kmeans.inertia_)
```

In [19]:
```
# Plot the scree plot
plt.figure(figsize=(16, 6))
plt.plot(range(1, 11), wcss, marker='o', linestyle='--')
plt.title('Scree Plot for Fast Food Dataset')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS (Within-Cluster Sum of Squares)')
plt.xticks(range(1, 11))
plt.grid(True)
plt.show()
```



In [20]:
```
from sklearn.metrics import adjusted_rand_score
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Preprocess the data (select numeric columns)
data = df.select_dtypes(include=[np.number])
```

In [21]:
```
# Standardize the data
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)
```

In [22]:
```
# Run k-means clustering multiple times and evaluate stability
num_clusters = 4  # Choose the number of clusters to evaluate (can be adjusted)
num_runs = 10     # Number of runs for evaluating stability
cluster_labels_list = []
```

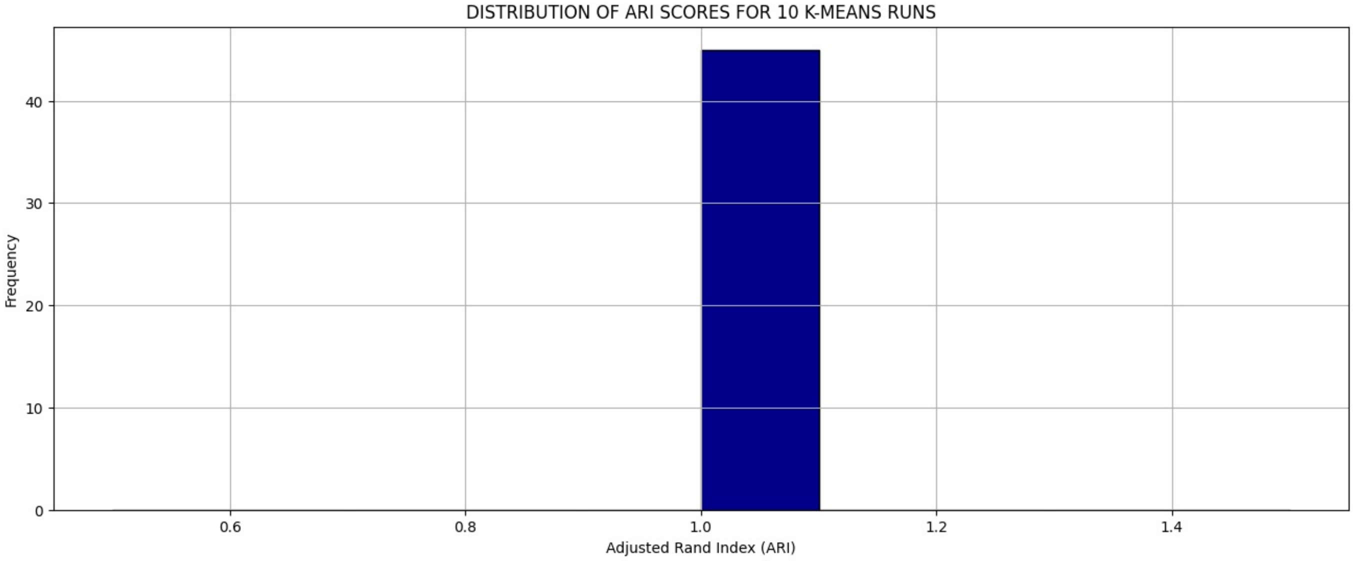
In [23]:
```
# Run k-means multiple times
for i in range(num_runs):
    kmeans = KMeans(n_clusters=num_clusters, n_init=10, random_state=i)
    kmeans.fit(data_scaled)
    cluster_labels_list.append(kmeans.labels_)
```

In [24]:
```
# Compute the Adjusted Rand Index (ARI) between all pairs of clustering results
ari_scores = []
for i in range(len(cluster_labels_list)):
    for j in range(i + 1, len(cluster_labels_list)):
        ari = adjusted_rand_score(cluster_labels_list[i], cluster_labels_list[j])
        ari_scores.append(ari)
```

In [25]:
```
# Display the average ARI score
average_ari = np.mean(ari_scores)
print(f"Average Adjusted Rand Index (ARI) for {num_runs} runs with {num_clusters} clusters: {average_ari:.3f}")
```

```
Average Adjusted Rand Index (ARI) for 10 runs with 4 clusters: 1.000
```

In [26]:
```
# Plot the distribution of ARI scores
plt.figure(figsize=(16, 6))
plt.hist(ari_scores, bins=10, color='darkblue', edgecolor='black')
plt.title(f'DISTRIBUTION OF ARI SCORES FOR {num_runs} K-MEANS RUNS')
plt.xlabel('Adjusted Rand Index (ARI)')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```



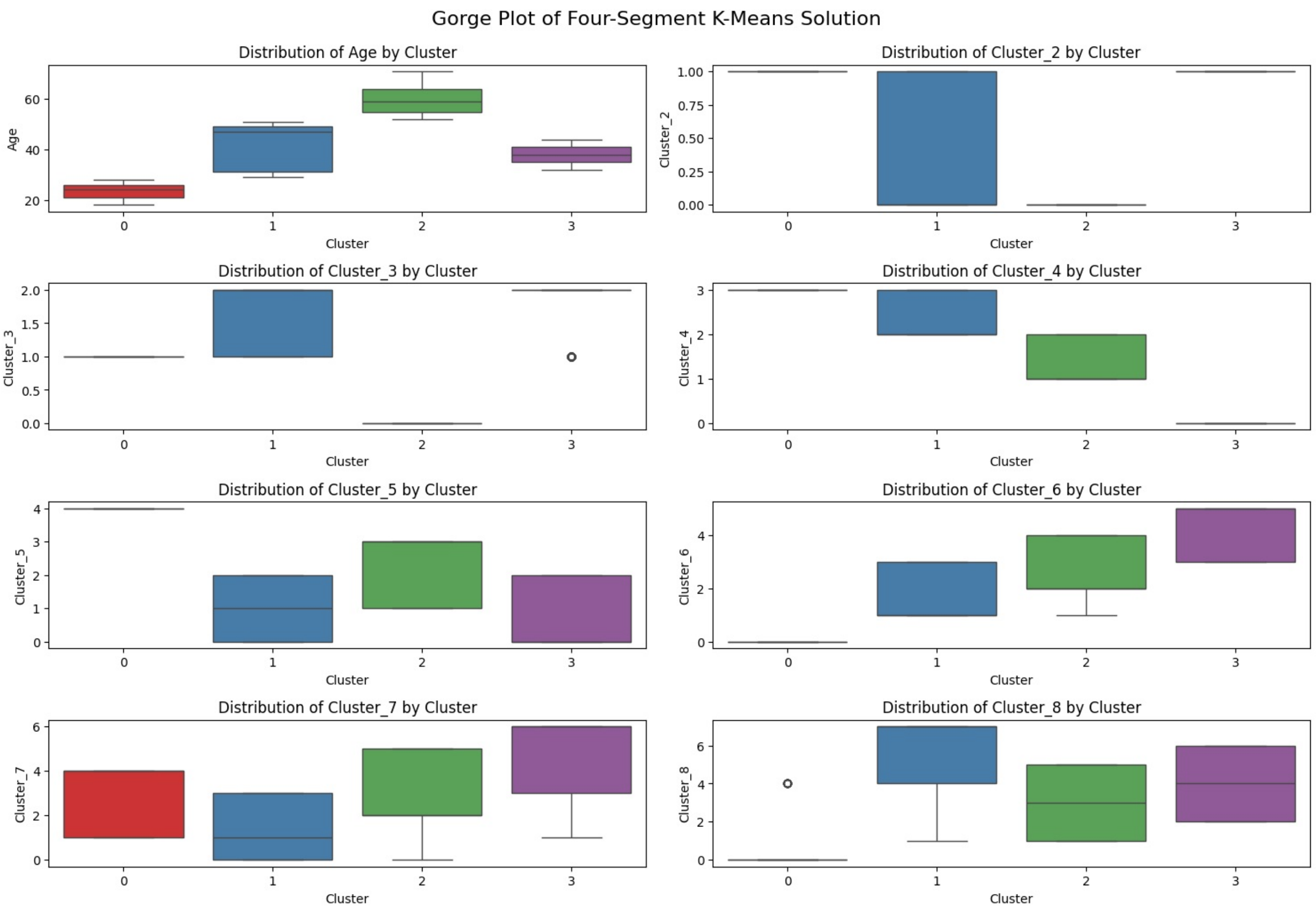
In [27]:
```
import seaborn as sns
from sklearn.preprocessing import StandardScaler

# Preprocess the data (select numeric columns)
data = df.select_dtypes(include=[np.number])
```

In [28]:
```
# Standardize the data
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)
```

In [29]:
```
# Run k-means clustering for 4 segments
num_clusters = 4
kmeans = KMeans(n_clusters=num_clusters, n_init=10, random_state=42)
df['Cluster'] = kmeans.fit_predict(data_scaled)
```

In [30]:
```
# Create a gorge plot for the four-segment k-means solution
plt.figure(figsize=(15, 12))
for feature in data.columns:
    plt.subplot(len(data.columns) // 2 + 1, 2, list(data.columns).index(feature) + 1)
    sns.boxplot(x='Cluster', y=feature, data=df, palette='Set1')
    plt.title(f'Distribution of {feature} by Cluster')
    plt.xlabel('Cluster')
    plt.ylabel(feature)

plt.tight_layout()
plt.suptitle('Gorge Plot of Four-Segment K-Means Solution', y=1.02, fontsize=16)
plt.show()
```

```
C:\Users\LOKESH\AppData\Local\Temp\ipykernel_47236\284269802.py:5: FutureWarning:

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

  sns.boxplot(x='Cluster', y=feature, data=df, palette='Set1')
C:\Users\LOKESH\AppData\Local\Temp\ipykernel_47236\284269802.py:5: FutureWarning:

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

  sns.boxplot(x='Cluster', y=feature, data=df, palette='Set1')
C:\Users\LOKESH\AppData\Local\Temp\ipykernel_47236\284269802.py:5: FutureWarning:

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

  sns.boxplot(x='Cluster', y=feature, data=df, palette='Set1')
C:\Users\LOKESH\AppData\Local\Temp\ipykernel_47236\284269802.py:5: FutureWarning:

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

  sns.boxplot(x='Cluster', y=feature, data=df, palette='Set1')
C:\Users\LOKESH\AppData\Local\Temp\ipykernel_47236\284269802.py:5: FutureWarning:

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

  sns.boxplot(x='Cluster', y=feature, data=df, palette='Set1')
C:\Users\LOKESH\AppData\Local\Temp\ipykernel_47236\284269802.py:5: FutureWarning:

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

  sns.boxplot(x='Cluster', y=feature, data=df, palette='Set1')
C:\Users\LOKESH\AppData\Local\Temp\ipykernel_47236\284269802.py:5: FutureWarning:

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

  sns.boxplot(x='Cluster', y=feature, data=df, palette='Set1')
C:\Users\LOKESH\AppData\Local\Temp\ipykernel_47236\284269802.py:5: FutureWarning:

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

  sns.boxplot(x='Cluster', y=feature, data=df, palette='Set1')
```



In [31]:

```python
from sklearn.metrics import adjusted_rand_score
from sklearn.preprocessing import StandardScaler

# Preprocess the data (select numeric columns)
data = df.select_dtypes(include=[np.number])

# Standardize the data
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

# Run k-means for solutions from 2 to 8 segments and store the labels
solutions = {}
for n_clusters in range(2, 9):
    kmeans = KMeans(n_clusters=n_clusters, n_init=10, random_state=42)
    kmeans.fit(data_scaled)
    solutions[n_clusters] = kmeans.labels_
```

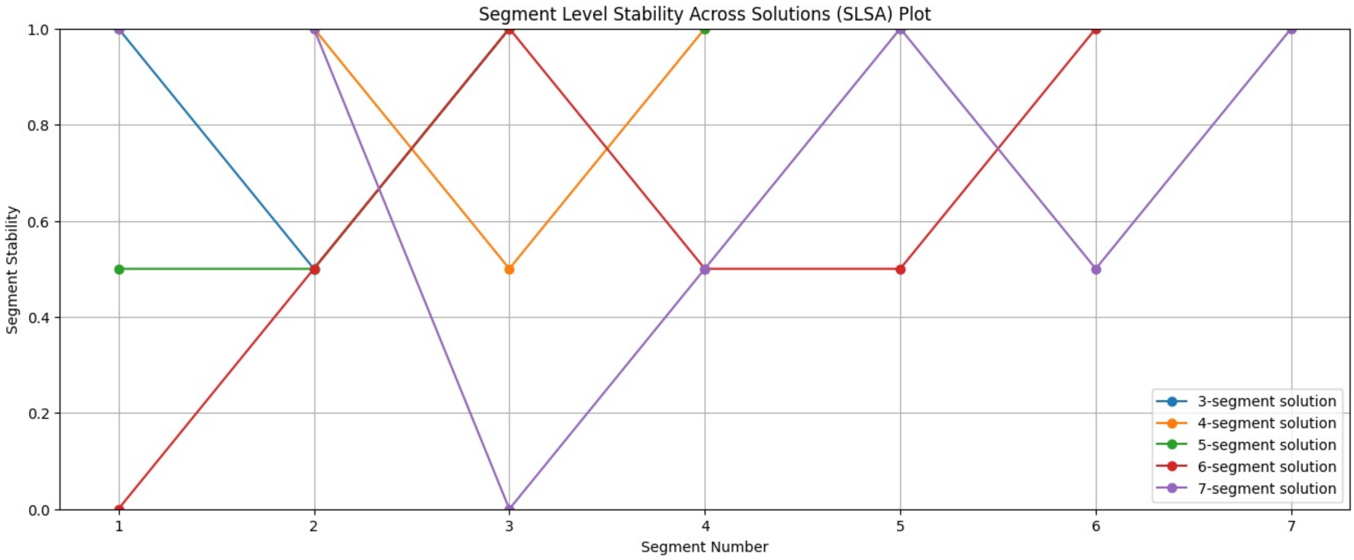
In [32]:

```python
# Calculate segment level stability
segment_stability = {}
for current_clusters in range(3, 8):
    previous_labels = solutions[current_clusters - 1]
    current_labels = solutions[current_clusters]
    next_labels = solutions[current_clusters + 1]

    stability_scores = []
    for cluster in np.unique(current_labels):
        # Find the indices of data points in the current cluster
        indices = np.where(current_labels == cluster)[0]

        # Calculate ARI for current cluster against previous and next solutions
        ari_prev = adjusted_rand_score(previous_labels[indices], current_labels[indices])
        ari_next = adjusted_rand_score(current_labels[indices], next_labels[indices])

        # Average stability for the current segment
        stability_score = (ari_prev + ari_next) / 2
        stability_scores.append(stability_score)

    segment_stability[current_clusters] = stability_scores
```

In [33]:

```python
# Plot the SLSA plot
plt.figure(figsize=(16, 6))
for clusters, stability in segment_stability.items():
    plt.plot(range(1, len(stability) + 1), stability, marker='o', label=f'{clusters}-segment solution')

plt.ylim(0, 1)
plt.xlabel("Segment Number")
```

```
plt.ylabel("Segment Stability")
plt.title("Segment Level Stability Across Solutions (SLSA) Plot")
plt.legend()
plt.grid(True)
plt.show()
```



In [34]:

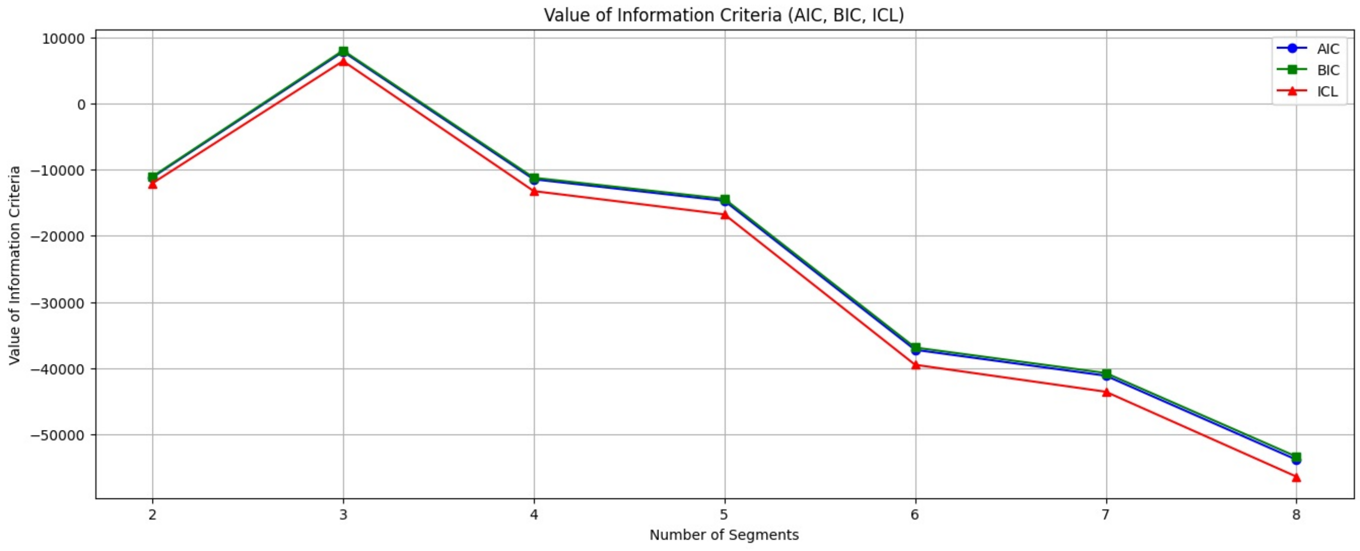
```
from sklearn.mixture import GaussianMixture
import matplotlib.pyplot as plt
from sklearn.preprocessing import Binarizer, StandardScaler

# Preprocess the data (select numeric columns and binarize them)
data = df.select_dtypes(include=[np.number])
binarizer = Binarizer(threshold=0.5)  # Adjust threshold as necessary
data_binarized = binarizer.fit_transform(StandardScaler().fit_transform(data))
# Fit mixture models for 2 to 8 segments
aic_values = []
bic_values = []
icl_values = []  # In Python, ICL may be approximated by custom calculations
```

In [35]:

```
for n_components in range(2, 9):
    gmm = GaussianMixture(n_components=n_components, covariance_type='spherical', random_state=1234, n_init=10)
    gmm.fit(data_binarized)
    aic_values.append(gmm.aic(data_binarized))
    bic_values.append(gmm.bic(data_binarized))
    # ICL can be approximated using penalized BIC; here is a simplified approach
    icl_values.append(gmm.bic(data_binarized) - np.log(n_components) * len(data_binarized))
```

In [36]:

```
# Plot the information criteria
plt.figure(figsize=(16, 6))
plt.plot(range(2, 9), aic_values, marker='o', label='AIC', color='blue')
plt.plot(range(2, 9), bic_values, marker='s', label='BIC', color='green')
plt.plot(range(2, 9), icl_values, marker='^', label='ICL', color='red')
plt.title('Value of Information Criteria (AIC, BIC, ICL)')
plt.xlabel('Number of Segments')
plt.ylabel('Value of Information Criteria')
plt.xticks(range(2, 9))
plt.legend()
plt.grid(True)
plt.show()
```



In [37]:
```
# Preprocess the data (select numeric columns and binarize them)
data = df.select_dtypes(include=[np.number])
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)
binarizer = Binarizer(threshold=0.5)
data_binarized = binarizer.fit_transform(data_scaled)

# Initialize lists to store AIC, BIC, and ICL values
aic_values = []
bic_values = []
icl_values = []  # ICL approximation using a penalized BIC
```

In [38]:
```
# Check the first few rows of the dataset
print(df.head())

# Ensure the 'Like' column exists
if 'Like' not in df.columns:
    raise ValueError("The 'Like' column is missing in the dataset.")

# Convert the dependent variable 'LIKE' to a numeric variable
# Mapping the categorical variable with 11 levels from 'ILOVEIT!' (+5) to 'IHATE IT!' (-5)
like_mapping = {
    'I LOVE IT!': 5, '4': 4, '3': 3, '2': 2, '1': 1, '0': 0,
    '-1': -1, '-2': -2, '-3': -3, '-4': -4, 'I HATE IT!': -5
}
```

```
  yummy convenient spicy fattening greasy fast cheap tasty expensive healthy  \
0    No         Yes    No        Yes     No  Yes   Yes    No       Yes      No
1   Yes         Yes    No        Yes    Yes  Yes   Yes   Yes       Yes      No
2    No         Yes   Yes        Yes    Yes  Yes    No   Yes       Yes     Yes
3   Yes         Yes    No        Yes    Yes  Yes   Yes   Yes        No      No
4    No         Yes    No        Yes    Yes  Yes   Yes    No        No     Yes

   ...      VisitFrequency  Gender  Cluster_2 Cluster_3 Cluster_4  Cluster_5  \
0  ...  Every three months  Female          0         0         1          3
1  ...  Every three months  Female          0         2         2          1
2  ...  Every three months  Female          0         0         1          3
3  ...        Once a week  Female          0         0         1          3
4  ...        Once a month    Male          0         2         2          1

   Cluster_6  Cluster_7  Cluster_8  Cluster
0          2          5          5        2
1          1          0          1        1
2          4          2          5        2
3          4          2          3        2
4          1          0          7        1

[5 rows x 23 columns]
```

In [39]:
```
# Replace the categorical 'Like' with numeric 'Like.n'
df['Like.n'] = df['Like'].map(like_mapping)

# Check the mapping results
print(df['Like.n'].value_counts().sort_index())

# Define the independent variables (perceptions of McDonald's)
independent_vars = ['yummy', 'convenient', 'spicy', 'fattening', 'greasy',
                    'fast', 'cheap', 'tasty', 'expensive', 'healthy', 'disgusting']

# Check if all independent variables exist in the DataFrame
missing_vars = [var for var in independent_vars if var not in df.columns]
```

```
    if missing_vars:
        raise ValueError(f"The following independent variables are missing: {missing_vars}")
```

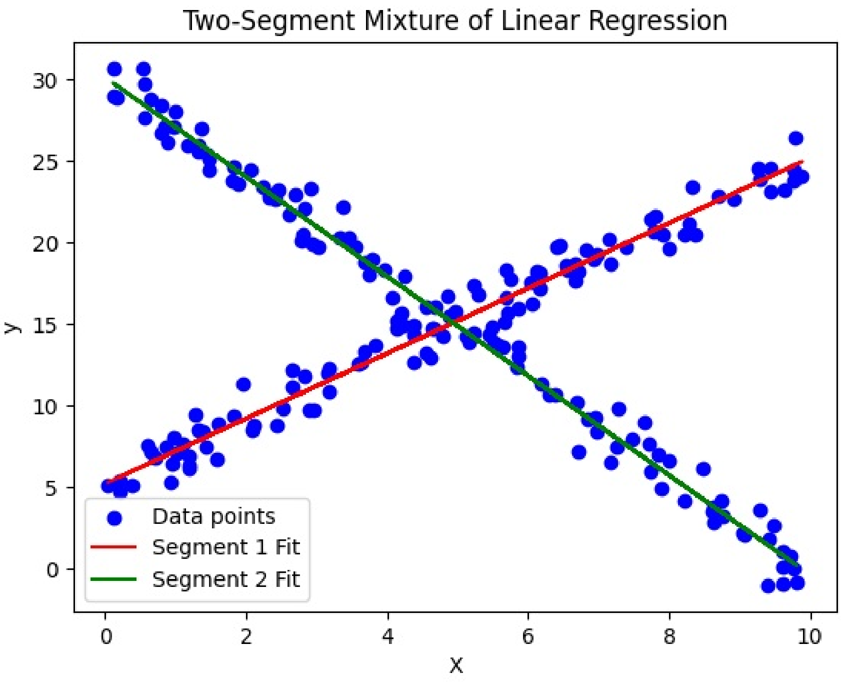
```
Like.n
-4.0     71
-3.0     73
-2.0     59
-1.0     58
 0.0    169
Name: count, dtype: int64
```

In [40]:
```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Example data generation (replace this with your actual data)
np.random.seed(0)
X1 = np.random.rand(100, 1) * 10
y1 = 2 * X1 + 5 + np.random.randn(100, 1)  # Segment 1
X2 = np.random.rand(100, 1) * 10
y2 = -3 * X2 + 30 + np.random.randn(100, 1)  # Segment 2

X = np.vstack((X1, X2))
y = np.vstack((y1, y2))
```

In [41]:
```
# Create a function to fit the two segments
def fit_mixture_regression(X, y):
    # Fit the first linear regression model
    model1 = LinearRegression()
    model1.fit(X[:100], y[:100])

    # Fit the second linear regression model
    model2 = LinearRegression()
    model2.fit(X[100:], y[100:])

    return model1, model2
```

In [42]:
```
# Fit the mixture regression
model1, model2 = fit_mixture_regression(X, y)

# Print the regression coefficients
print("Segment 1 Coefficients (Intercept, Slope):", model1.intercept_[0], model1.coef_[0][0])
print("Segment 2 Coefficients (Intercept, Slope):", model2.intercept_[0], model2.coef_[0][0])

# Plotting the results
plt.scatter(X, y, color='blue', label='Data points')
plt.plot(X[:100], model1.predict(X[:100]), color='red', label='Segment 1 Fit')
plt.plot(X[100:], model2.predict(X[100:]), color='green', label='Segment 2 Fit')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Two-Segment Mixture of Linear Regression')
plt.legend()
plt.show()
```

```
Segment 1 Coefficients (Intercept, Slope): 5.222151077447226 1.9936935021402045
Segment 2 Coefficients (Intercept, Slope): 30.101897754959293 -3.044003252201276
```



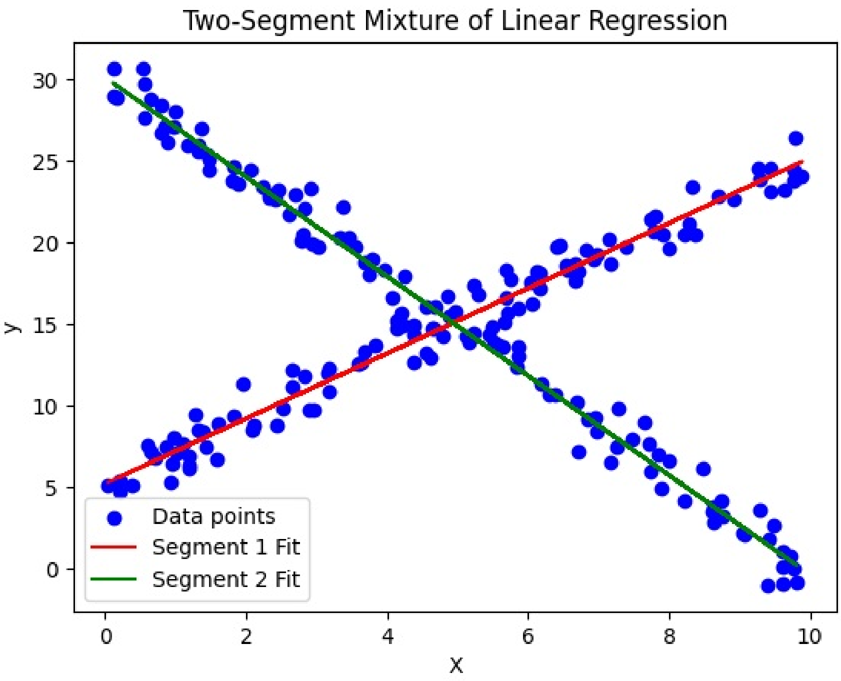
In [43]:
```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

In [44]:
```
# Example data generation (replace this with your actual data)
np.random.seed(0)
X1 = np.random.rand(100, 1) * 10
y1 = 2 * X1 + 5 + np.random.randn(100, 1)  # Segment 1
X2 = np.random.rand(100, 1) * 10
y2 = -3 * X2 + 30 + np.random.randn(100, 1)  # Segment 2

X = np.vstack((X1, X2))
y = np.vstack((y1, y2))
```

In [45]:
```
# Create a function to fit the two segments
def fit_mixture_regression(X, y):
    # Fit the first linear regression model
    model1 = LinearRegression()
    model1.fit(X[:100], y[:100])

    # Fit the second linear regression model
    model2 = LinearRegression()
    model2.fit(X[100:], y[100:])

    return model1, model2

# Fit the mixture regression
model1, model2 = fit_mixture_regression(X, y)
```

In [46]:
```
# Print the regression coefficients
print("Segment 1 Coefficients (Intercept, Slope):", model1.intercept_[0], model1.coef_[0][0])
print("Segment 2 Coefficients (Intercept, Slope):", model2.intercept_[0], model2.coef_[0][0])

# Plotting the results
plt.scatter(X, y, color='blue', label='Data points')
plt.plot(X[:100], model1.predict(X[:100]), color='red', label='Segment 1 Fit')
plt.plot(X[100:], model2.predict(X[100:]), color='green', label='Segment 2 Fit')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Two-Segment Mixture of Linear Regression')
plt.legend()
plt.show()
```

```
Segment 1 Coefficients (Intercept, Slope): 5.222151077447226 1.9936935021402045
Segment 2 Coefficients (Intercept, Slope): 30.101897754959293 -3.044003252201276
```



In [47]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

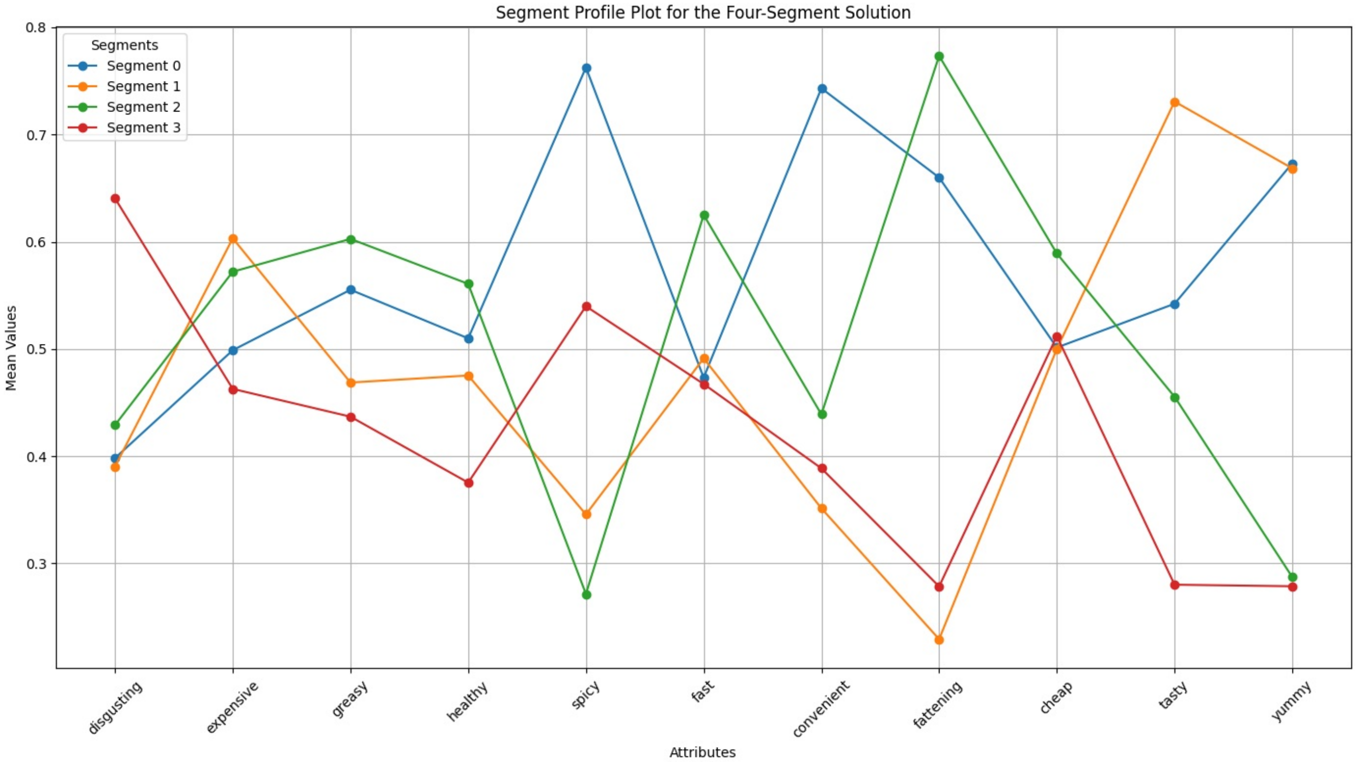
In [48]:
```
# Example data generation (replace this with your actual data)
np.random.seed(0)
data = {
    'disgusting': np.random.rand(100),
    'expensive': np.random.rand(100),
```

```
    'greasy': np.random.rand(100),
    'healthy': np.random.rand(100),
    'spicy': np.random.rand(100),
    'fast': np.random.rand(100),
    'convenient': np.random.rand(100),
    'fattening': np.random.rand(100),
    'cheap': np.random.rand(100),
    'tasty': np.random.rand(100),
    'yummy': np.random.rand(100)
}
df = pd.DataFrame(data)
```

In [49]:

```
# Perform K-Means clustering
kmeans = KMeans(n_clusters=4, random_state=0)
df['segment'] = kmeans.fit_predict(df)

# Calculate mean values for each segment
segment_profile = df.groupby('segment').mean().reset_index()

# Plot the segment profile
plt.figure(figsize=(14, 8))
for i, segment in segment_profile.iterrows():
    plt.plot(segment_profile.columns[1:], segment.values[1:], marker='o', label=f'Segment {int(segment["segment

plt.title('Segment Profile Plot for the Four-Segment Solution')
plt.xlabel('Attributes')
plt.ylabel('Mean Values')
plt.xticks(rotation=45)
plt.legend(title='Segments')
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
C:\Users\LOKESH\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: Futur
eWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicit
ly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```



In [50]:

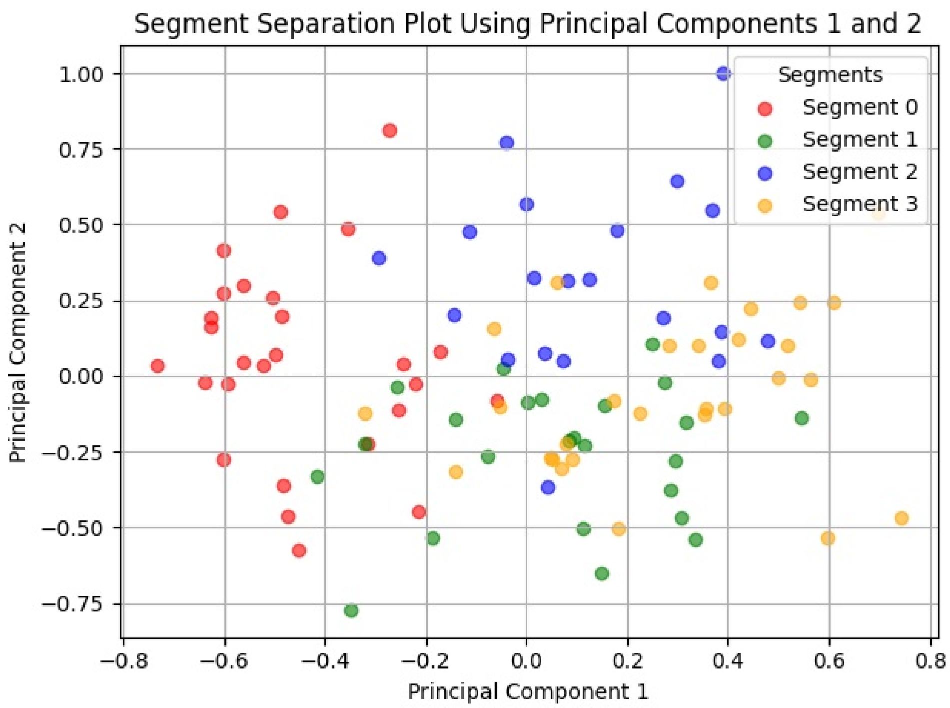
```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans

# Example data generation (replace this with your actual data)
np.random.seed(0)
data = {
    'disgusting': np.random.rand(100),
    'expensive': np.random.rand(100),
    'greasy': np.random.rand(100),
    'healthy': np.random.rand(100),
    'spicy': np.random.rand(100),
    'fast': np.random.rand(100),
    'convenient': np.random.rand(100),
    'fattening': np.random.rand(100),
```

```
    'cheap': np.random.rand(100),
    'tasty': np.random.rand(100),
    'yummy': np.random.rand(100)
}
df = pd.DataFrame(data)
```

In [51]:
```
# Perform K-Means clustering
kmeans = KMeans(n_clusters=4, random_state=0)
df['segment'] = kmeans.fit_predict(df)

# Perform PCA for dimensionality reduction
pca = PCA(n_components=2)
pca_result = pca.fit_transform(df.drop('segment', axis=1))

# Add PCA results to the DataFrame
df['PCA1'] = pca_result[:, 0]
df['PCA2'] = pca_result[:, 1]

# Plot the segments in the PCA space
plt.figure(figsize=(10, 7))
colors = ['red', 'green', 'blue', 'orange']
```

```
C:\Users\LOKESH\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: Futur
eWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicit
ly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
<Figure size 1000x700 with 0 Axes>
```

In [52]:
```
for i in range(4):  # Number of clusters/segments
    plt.scatter(
        df[df['segment'] == i]['PCA1'],
        df[df['segment'] == i]['PCA2'],
        label=f'Segment {i}',
        color=colors[i],
        alpha=0.6
    )

plt.title('Segment Separation Plot Using Principal Components 1 and 2')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Segments')
plt.grid(True)
plt.tight_layout()
plt.show()
```



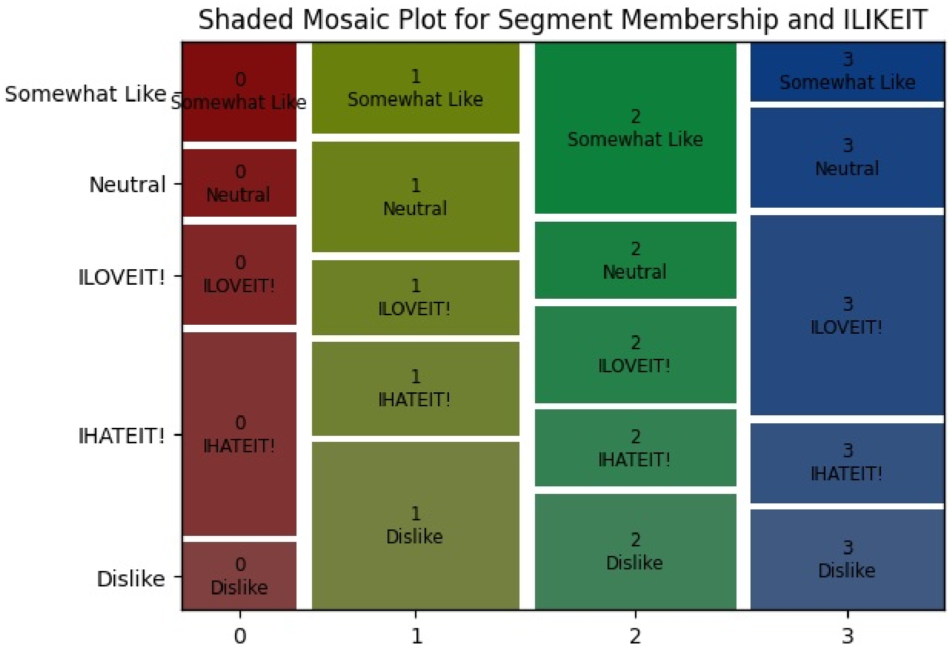
In [53]:
```
import matplotlib.pyplot as plt
from statsmodels.graphics.mosaicplot import mosaic
from sklearn.cluster import KMeans

# Example data generation (replace this with your actual data)
np.random.seed(0)
data = {
    'ILIKEIT': np.random.choice(['ILOVEIT!', 'Somewhat Like', 'Neutral', 'Dislike', 'IHATEIT!'], size=100),
    'disgusting': np.random.rand(100),
    'expensive': np.random.rand(100),
```

```
    'greasy': np.random.rand(100),
    'healthy': np.random.rand(100),
    'spicy': np.random.rand(100),
    'fast': np.random.rand(100),
    'convenient': np.random.rand(100),
    'fattening': np.random.rand(100),
    'cheap': np.random.rand(100),
    'tasty': np.random.rand(100),
    'yummy': np.random.rand(100)
}
df = pd.DataFrame(data)
```

In [54]:

```
# Perform K-Means clustering
kmeans = KMeans(n_clusters=4, random_state=0)
df['segment'] = kmeans.fit_predict(df.drop('ILIKEIT', axis=1))

# Create a cross-tabulation and convert to a dictionary
crosstab = pd.crosstab(df['segment'], df['ILIKEIT'])
crosstab_dict = crosstab.stack().to_dict()

# Plot the mosaic plot
plt.figure(figsize=(12, 8))
mosaic(crosstab_dict, title='Shaded Mosaic Plot for Segment Membership and ILIKEIT', gap=0.02)
plt.show()
```

```
C:\Users\LOKESH\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: 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)
<Figure size 1200x800 with 0 Axes>
```



In [55]:

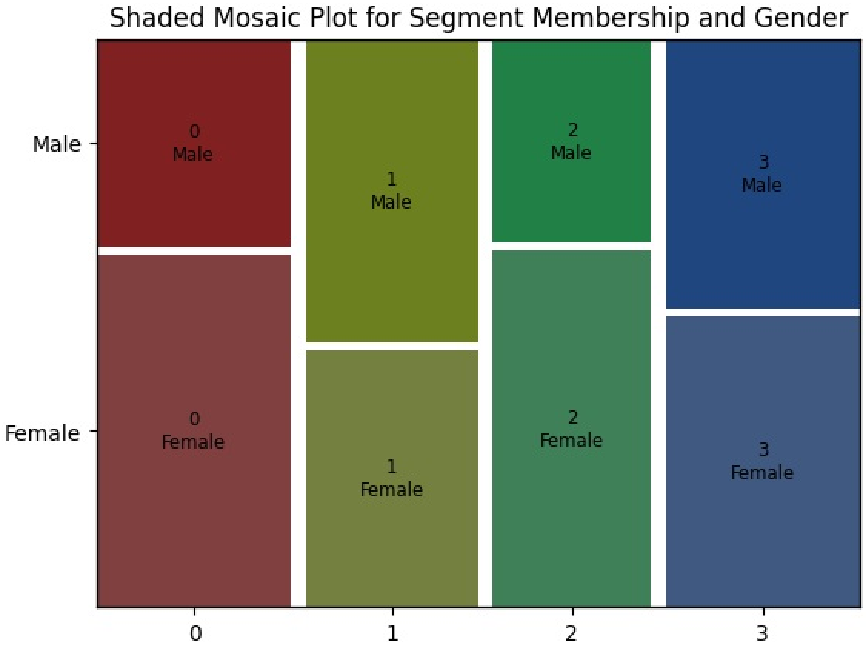
```
import matplotlib.pyplot as plt
from statsmodels.graphics.mosaicplot import mosaic
from sklearn.cluster import KMeans

# Example data generation (replace this with your actual data)
np.random.seed(0)
data = {
    'Gender': np.random.choice(['Male', 'Female'], size=100),
    'disgusting': np.random.rand(100),
    'expensive': np.random.rand(100),
    'greasy': np.random.rand(100),
    'healthy': np.random.rand(100),
    'spicy': np.random.rand(100),
    'fast': np.random.rand(100),
    'convenient': np.random.rand(100),
    'fattening': np.random.rand(100),
    'cheap': np.random.rand(100),
    'tasty': np.random.rand(100),
    'yummy': np.random.rand(100)
}
df = pd.DataFrame(data)
```

In [56]:

```
# Perform K-Means clustering
kmeans = KMeans(n_clusters=4, random_state=0)
df['segment'] = kmeans.fit_predict(df.drop('Gender', axis=1))
```

```
# Create a cross-tabulation and convert to a dictionary
crosstab = pd.crosstab(df['segment'], df['Gender'])
crosstab_dict = crosstab.stack().to_dict()

# Plot the mosaic plot
plt.figure(figsize=(12, 8))
mosaic(crosstab_dict, title='Shaded Mosaic Plot for Segment Membership and Gender', gap=0.02)
plt.show()
```

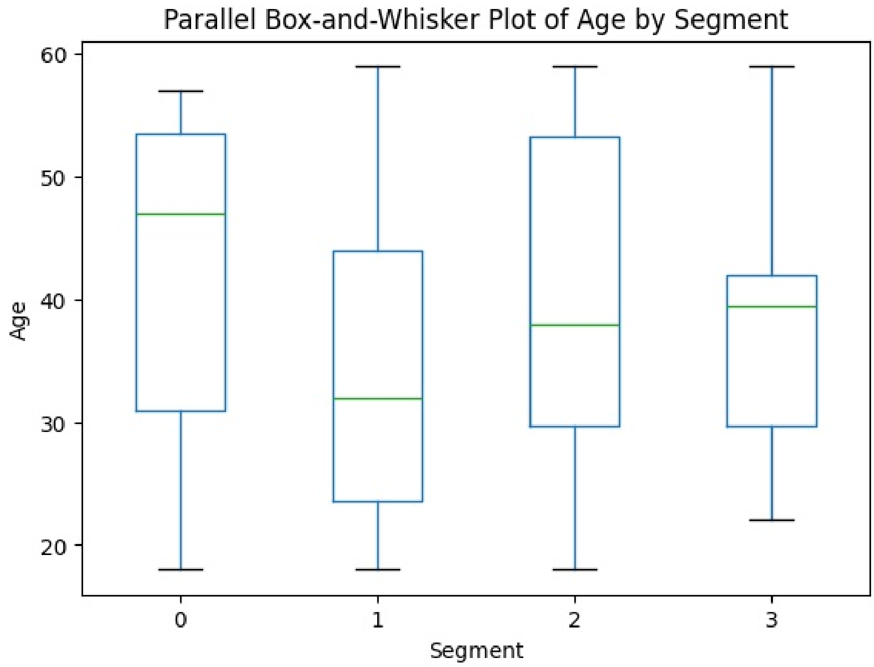
```
C:\Users\LOKESH\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
<Figure size 1200x800 with 0 Axes>
```



In [57]:
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Example data generation (replace this with your actual data)
np.random.seed(0)
data = {
    'Age': np.random.randint(18, 60, size=100),
    'disgusting': np.random.rand(100),
    'expensive': np.random.rand(100),
    'greasy': np.random.rand(100),
    'healthy': np.random.rand(100),
    'spicy': np.random.rand(100),
    'fast': np.random.rand(100),
    'convenient': np.random.rand(100),
    'fattening': np.random.rand(100),
    'cheap': np.random.rand(100),
    'tasty': np.random.rand(100),
    'yummy': np.random.rand(100)
}
df = pd.DataFrame(data)
```

In [58]:
```
# Perform K-Means clustering
kmeans = KMeans(n_clusters=4, random_state=0)
df['segment'] = kmeans.fit_predict(df.drop('Age', axis=1))

# Create a parallel box-and-whisker plot for age by segment
plt.figure(figsize=(12, 8))
df.boxplot(column='Age', by='segment', grid=False)
plt.title('Parallel Box-and-Whisker Plot of Age by Segment')
plt.suptitle('')  # Remove the default subtitle
plt.xlabel('Segment')
plt.ylabel('Age')
plt.show()
```

```
C:\Users\LOKESH\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: 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)
<Figure size 1200x800 with 0 Axes>
```



In [59]:
```
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

# Example data generation
np.random.seed(0)
data = {
    'Age': np.random.randint(18, 60, size=100),
    'disgusting': np.random.rand(100),
    'expensive': np.random.rand(100),
    'greasy': np.random.rand(100),
    'healthy': np.random.rand(100),
    'spicy': np.random.rand(100),
    'fast': np.random.rand(100),
    'convenient': np.random.rand(100),
    'fattening': np.random.rand(100),
    'cheap': np.random.rand(100),
    'tasty': np.random.rand(100),
    'yummy': np.random.rand(100),
}
df = pd.DataFrame(data)
```

In [60]:
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
np.random.seed(0)
data = {
    'disgusting': np.random.rand(100),
    'expensive': np.random.rand(100),
    'greasy': np.random.rand(100),
    'healthy': np.random.rand(100),
    'spicy': np.random.rand(100),
    'fast': np.random.rand(100),
    'convenient': np.random.rand(100),
    'fattening': np.random.rand(100),
    'cheap': np.random.rand(100),
    'tasty': np.random.rand(100),
    'yummy': np.random.rand(100),
}
df = pd.DataFrame(data)
```

In [61]:
```
# Perform K-Means clustering to create segments
kmeans = KMeans(n_clusters=4, random_state=0)
df['segment'] = kmeans.fit_predict(df)
# Calculate the mean of each feature for each segment
segment_profile = df.groupby('segment').mean()
plt.figure(figsize=(14, 8))
for segment in segment_profile.index:
    plt.plot(segment_profile.columns, segment_profile.loc[segment], marker='o', label=f'Segment {segment}')
plt.title('Simple Segment Evaluation Plot for the Fast Food Dataset')
plt.xlabel('Attributes')
```

```
plt.ylabel('Mean Value')
plt.xticks(rotation=45)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
C:\Users\LOKESH\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: 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)
```