EXPLORING DATA

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
mcdonalds = pd.read_csv("/content/mcdonalds.csv")
mcdonalds
```

	yummy	convenient	spicy	fattening	greasy	fast	cheap	tasty	expensive	healthy	disgusting	Like	Age	VisitFrequency	Gender
0	No	Yes	No	Yes	No	Yes	Yes	No	Yes	No	No	-3	61	Every three months	Female
1	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	No	No	+2	51	Every three months	Female
2	No	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	+1	62	Every three months	Female
3	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No	No	Yes	+4	69	Once a week	Female
4	No	Yes	No	Yes	Yes	Yes	Yes	No	No	Yes	No	+2	49	Once a month	Male
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1448	No	Yes	No	Yes	Yes	No	No	No	Yes	No	Yes	I hate it!-5	47	Once a year	Male
1449	Yes	Yes	No	Yes	No	No	Yes	Yes	No	Yes	No	+2	36	Once a week	Female
1450	Yes	Yes	No	Yes	No	Yes	No	Yes	Yes	No	No	+3	52	Once a month	Female

```
print(mcdonalds.columns.tolist())
```

```
['yummy', 'convenient', 'spicy', 'fattening', 'greasy', 'fast', 'cheap', 'tasty', 'expensive', 'healthy', 'disgusting', 'Like', 'Age', '
```

```
mcdonalds.shape
```

```
(1453, 15)
```

```
mcdonalds.head(3)
```

	yummy	convenient	spicy	fattening	greasy	fast	cheap	tasty	expensive	healthy	disgusting	Like	Age	VisitFrequency	Gender
0	No	Yes	No	Yes	No	Yes	Yes	No	Yes	No	No	-3	61	Every three months	Female
1	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	No	No	+2	51	Every three months	Female
2	No	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	+1	62	Every three months	Female

```
MD_x = mcdonalds.iloc[:, 0:11]
```

```
MD_x_binary = (MD_x == "Yes").astype(int)
```

```
col_means = MD_x_binary.mean().round(2)
```

```
print(col_means)
```

```
yummy         0.55
convenient    0.91
spicy         0.09
fattening     0.87
greasy        0.53
fast          0.90
cheap         0.60
tasty         0.64
expensive     0.36
healthy       0.20
disgusting    0.24
dtype: float64
```

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler


MD_x = (mcdonalds.iloc[:, 0:11] == "Yes").astype(int)


scaler = StandardScaler()
MD_x_scaled = scaler.fit_transform(MD_x)


pca = PCA()
pca.fit(MD_x_scaled)
```

```
PCA()
```

```
explained_variance = pca.explained_variance_
proportion_variance = pca.explained_variance_ratio_
cumulative_variance = proportion_variance.cumsum()



summary_df = pd.DataFrame({
    "Standard Deviation": explained_variance**0.5,
    "Proportion of Variance": proportion_variance,
    "Cumulative Proportion": cumulative_variance
})


print(summary_df.round(4))
```

```
    Standard Deviation  Proportion of Variance  Cumulative Proportion
0               1.6772                  0.2556                 0.2556
1               1.2779                  0.1483                 0.4039
2               1.1752                  0.1255                 0.5294
3               1.0401                  0.0983                 0.6277
4               0.9586                  0.0835                 0.7111
5               0.8846                  0.0711                 0.7822
6               0.8458                  0.0650                 0.8472
7               0.7699                  0.0538                 0.9011
8               0.7241                  0.0476                 0.9487
9               0.5548                  0.0280                 0.9766
10              0.5070                  0.0234                 1.0000
```

```
pca = PCA()
pca.fit(MD_x_scaled)
```

```
PCA()
```

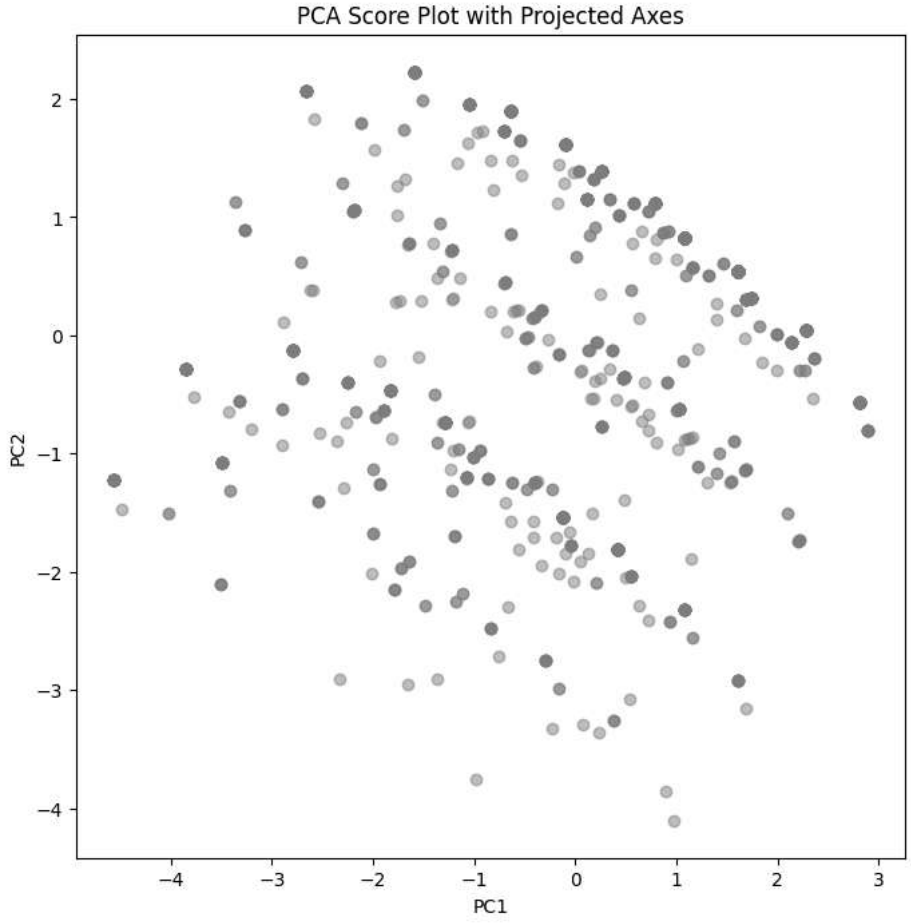
```
loadings = pd.DataFrame(pca.components_.T,
                        columns=[f'PC{i+1}' for i in range(MD_x.shape[1])],
                        index=MD_x.columns)



loadings_rounded = loadings.round(2)


print(loadings_rounded)
```

```
                PC1   PC2   PC3   PC4   PC5   PC6   PC7   PC8   PC9  PC10  PC11
yummy          0.41 -0.28  0.26  0.03 -0.35  0.12 -0.16 -0.02  0.20 -0.69 -0.09
convenient     0.31  0.05  0.32 -0.01  0.44 -0.19  0.66 -0.15  0.34 -0.03  0.02
spicy          0.02 -0.07 -0.02  0.85  0.19 -0.44 -0.21  0.03 -0.03 -0.06 -0.04
fattening     -0.18  0.21  0.61 -0.02 -0.09 -0.11  0.02  0.72 -0.11  0.02 -0.07
greasy        -0.27  0.14  0.39  0.33 -0.34  0.28  0.31 -0.49 -0.35  0.02 -0.03
fast           0.21  0.28  0.21  0.09  0.57  0.57 -0.39 -0.06 -0.10 -0.02 -0.05
cheap          0.29  0.58 -0.13  0.11 -0.26 -0.02  0.01  0.06  0.06 -0.06  0.69
tasty          0.43 -0.24  0.27  0.07 -0.29  0.05 -0.20 -0.06  0.21  0.71  0.01
expensive     -0.29 -0.57  0.18  0.04  0.21  0.17 -0.02  0.06  0.01 -0.03  0.69
healthy        0.27 -0.20 -0.37  0.30 -0.05  0.45  0.45  0.45 -0.22  0.05 -0.09
disgusting    -0.41  0.14 -0.09  0.24 -0.11  0.33  0.02  0.08  0.78  0.02 -0.11
```

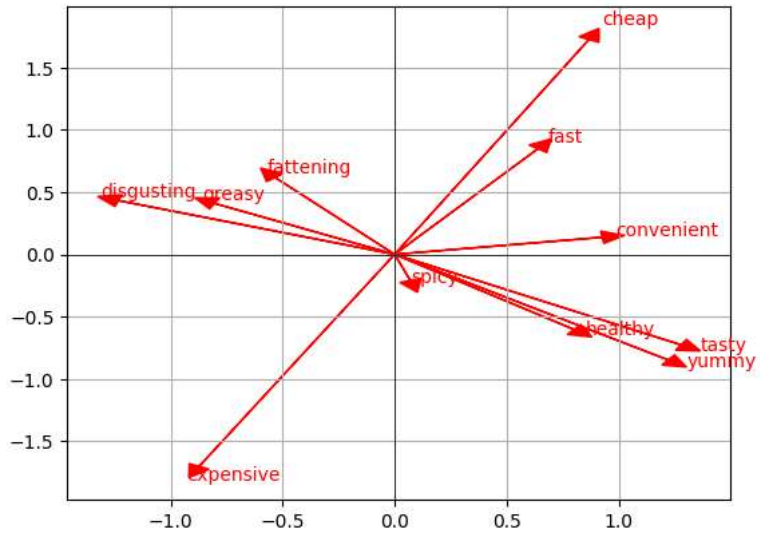
```
pca = PCA()
```

```
pca = PCA()
scores = pca.fit_transform(MD_x_scaled)
loadings = pca.components_.T


plt.figure(figsize=(8, 8))
plt.scatter(scores[:, 0], scores[:, 1], color='grey', alpha=0.5)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('PCA Score Plot with Projected Axes')
```

```
Text(0.5, 1.0, 'PCA Score Plot with Projected Axes')
```




```
for i, var in enumerate(MD_x.columns):
    plt.arrow(0, 0,
              loadings[i, 0]*3,  # Scale for visibility
              loadings[i, 1]*3,
              color='red',
              head_width=0.1, head_length=0.1)
    plt.text(loadings[i, 0]*3.2, loadings[i, 1]*3.2, var, color='red')

plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.grid(True)
plt.show()
```



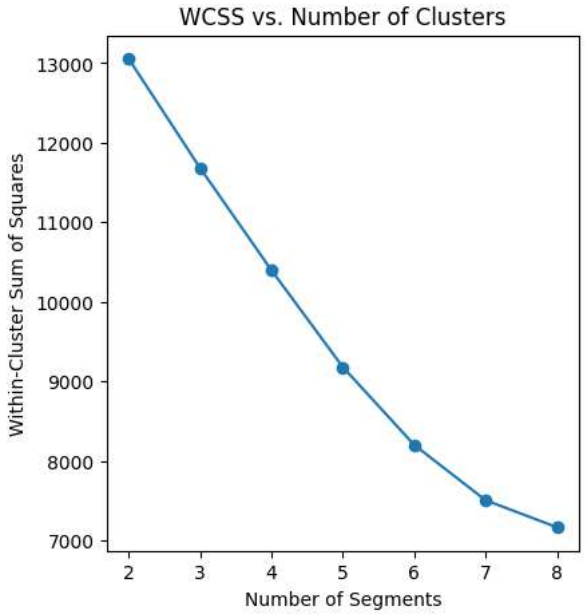
EXTRACTING SEGMENTS

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


best_models = {}
for k in range(2, 9):
    best_score = -1
    best_model = None
    for _ in range(10):
        kmeans = KMeans(n_clusters=k, n_init=10, random_state=None)
        labels = kmeans.fit_predict(MD_x_scaled)
        score = silhouette_score(MD_x_scaled, labels)
        if score > best_score:
            best_score = score
            best_model = kmeans
    best_models[k] = best_model


best_k = 3
model_k3 = best_models[best_k]
print(f"Best model for k={best_k} has inertia: {model_k3.inertia_}")
print(f"Cluster centers:\n{model_k3.cluster_centers_}")
```

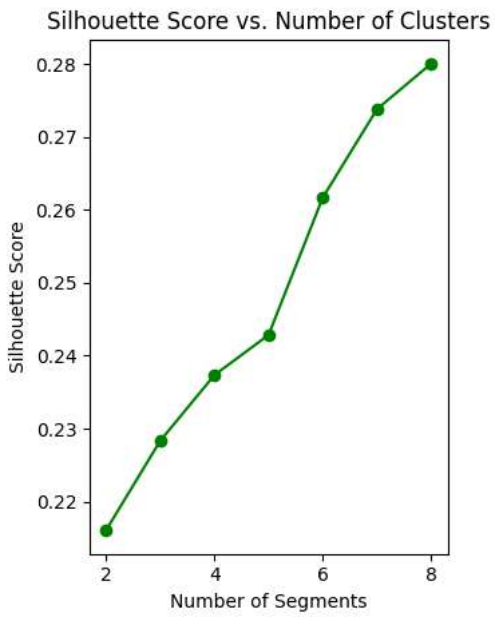
```
Best model for k=3 has inertia: 11681.968440903509
Cluster centers:
[[ 0.28425908  0.01655936  0.12891135 -2.55509245 -0.85747739  0.00478064
   0.12739183  0.29806352 -0.27918867  0.93912373 -0.47729578]
 [-0.95637464 -0.667463   -0.08305234  0.3156306   0.43347648 -0.4046588
  -0.55026581 -1.08242285  0.55119022 -0.38879763  1.16583976]
 [ 0.36323533  0.2912744   0.00989397  0.39137527 -0.01328489  0.17768152
   0.21650476  0.41602368 -0.18538191 -0.02340223 -0.41562604]]
```

```
wcss = []
silhouette_scores = []
K = range(2, 9)

for k in K:
    model = best_models[k]
    wcss.append(model.inertia_)
    labels = model.predict(MD_x_scaled)
    silhouette_scores.append(silhouette_score(MD_x_scaled, labels))


plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(K, wcss, marker='o')
plt.title('WCSS vs. Number of Clusters')
plt.xlabel('Number of Segments')
plt.ylabel('Within-Cluster Sum of Squares')
```

```
Text(0, 0.5, 'Within-Cluster Sum of Squares')
```




```
plt.subplot(1, 2, 2)
plt.plot(K, silhouette_scores, marker='o', color='green')
plt.title('Silhouette Score vs. Number of Clusters')
plt.xlabel('Number of Segments')
plt.ylabel('Silhouette Score')
plt.tight_layout()
plt.show()
```



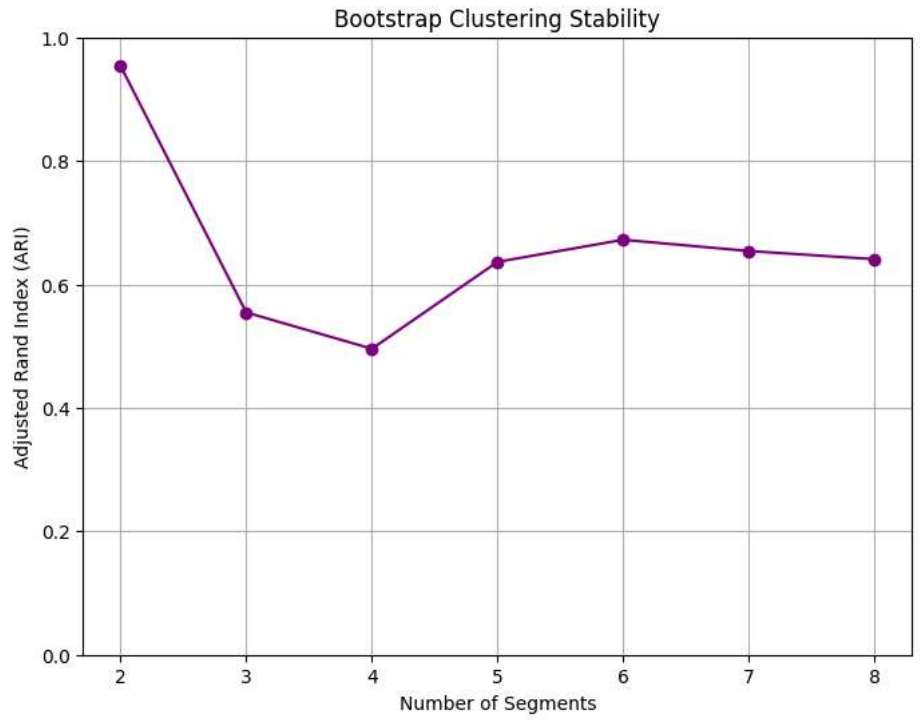

```
from sklearn.metrics import adjusted_rand_score
from sklearn.utils import resample


k_range = range(2, 9)
n_rep = 10
n_boot = 100
random_state = 1234


np.random.seed(random_state)
stability_results = {}

for k in k_range:
    scores = []
```

```
    for b in range(n_boot):

        sample_indices = resample(range(MD_x_scaled.shape[0]), replace=True)
        X_boot = MD_x_scaled[sample_indices]

        model = KMeans(n_clusters=k, n_init=n_rep, random_state=None)
        labels_boot = model.fit_predict(X_boot)

        labels_orig = KMeans(n_clusters=k, n_init=n_rep, random_state=None).fit_predict(MD_x_scaled)

        common_indices = list(set(sample_indices) & set(range(len(labels_orig))))
        if len(common_indices) > 0:
            score = adjusted_rand_score(
                labels_orig[common_indices],
                labels_boot[[sample_indices.index(i) for i in common_indices]]
            )
            scores.append(score)
    stability_results[k] = np.mean(scores)
```

```
for k, stability in stability_results.items():
    print(f"k={k}: average ARI stability = {stability:.3f}")
```

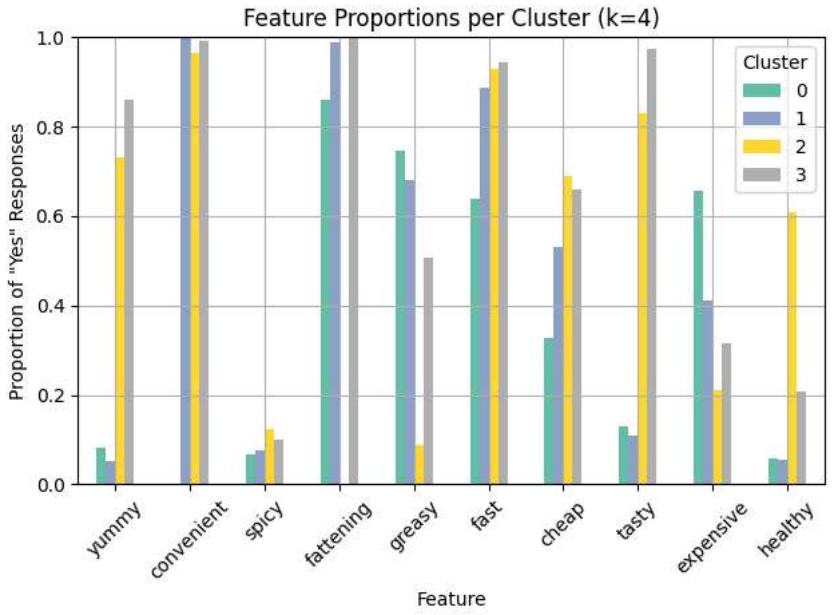
```
k=2: average ARI stability = 0.954
k=3: average ARI stability = 0.555
k=4: average ARI stability = 0.496
k=5: average ARI stability = 0.636
k=6: average ARI stability = 0.672
k=7: average ARI stability = 0.654
k=8: average ARI stability = 0.641
```

```
plt.figure(figsize=(8, 6))
plt.plot(
    list(stability_results.keys()),
    list(stability_results.values()),
    marker='o', color='purple'
)
plt.title("Bootstrap Clustering Stability")
plt.xlabel("Number of Segments")
plt.ylabel("Adjusted Rand Index (ARI)")
plt.grid(True)
plt.xticks(list(stability_results.keys()))
plt.ylim(0, 1)
plt.show()
```



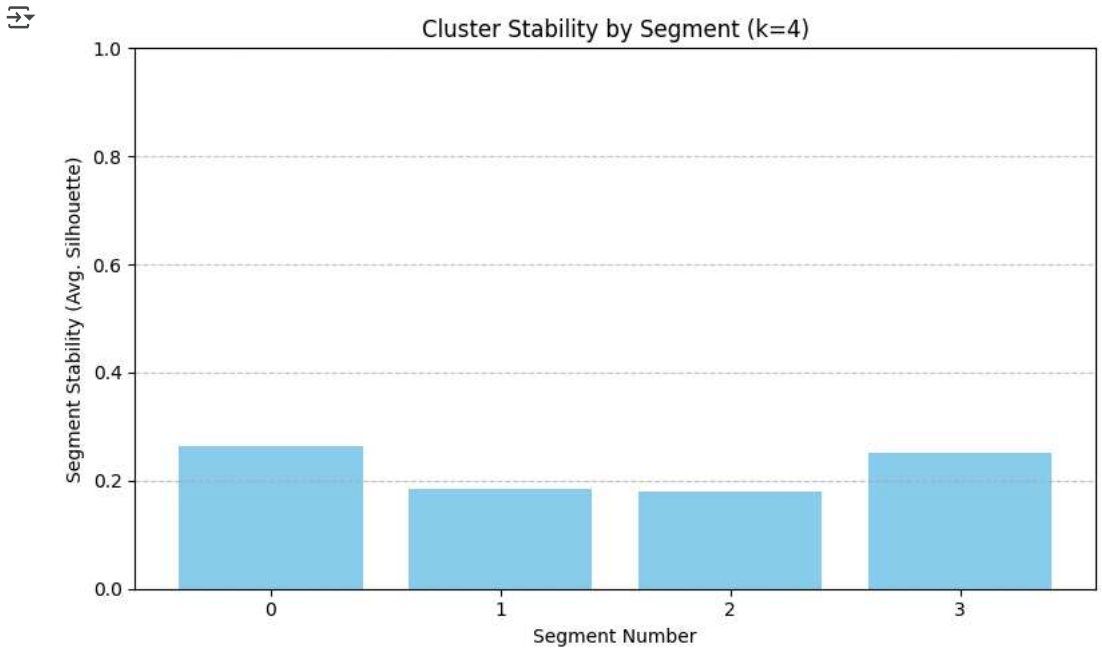
```
import seaborn as sns


features = MD_x.columns.tolist()[:-1]


k = 4
kmeans4 = KMeans(n_clusters=k, n_init=10, random_state=42)
labels = kmeans4.fit_predict(MD_x_scaled)
MD_x['Cluster'] = labels


cluster_means = MD_x.groupby('Cluster')[features].mean().T



plt.figure(figsize=(12, 6))
cluster_means.plot(kind='bar', ylim=(0, 1), colormap='Set2')
plt.title('Feature Proportions per Cluster (k=4)')
plt.ylabel('Proportion of "Yes" Responses')
plt.xlabel('Feature')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```




```
best_models = {}
for k in range(2, 9):
    model = KMeans(n_clusters=k, n_init=10, random_state=123)
    model.fit(MD_x_scaled)
    best_models[k] = model


from sklearn.metrics import silhouette_samples


model_k4 = best_models[4]
labels_k4 = model_k4.labels_


silhouette_avg = silhouette_samples(MD_x_scaled, labels_k4)


silhouette_df = pd.DataFrame({'Cluster': labels_k4, 'SilhouetteScore': silhouette_avg})


clusterwise_summary = silhouette_df.groupby('Cluster')['SilhouetteScore'].agg(['mean']).reset_index()


plt.figure(figsize=(8, 5))
plt.bar(clusterwise_summary.index, clusterwise_summary['mean'], color='skyblue')
plt.ylim(0, 1)
plt.xlabel("Segment Number")
```

```
plt.ylabel("Segment Stability (Avg. Silhouette)")
plt.title("Cluster Stability by Segment (k=4)")
plt.xticks(clusterwise_summary.index)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



Using Mixtures Of Distributions:-

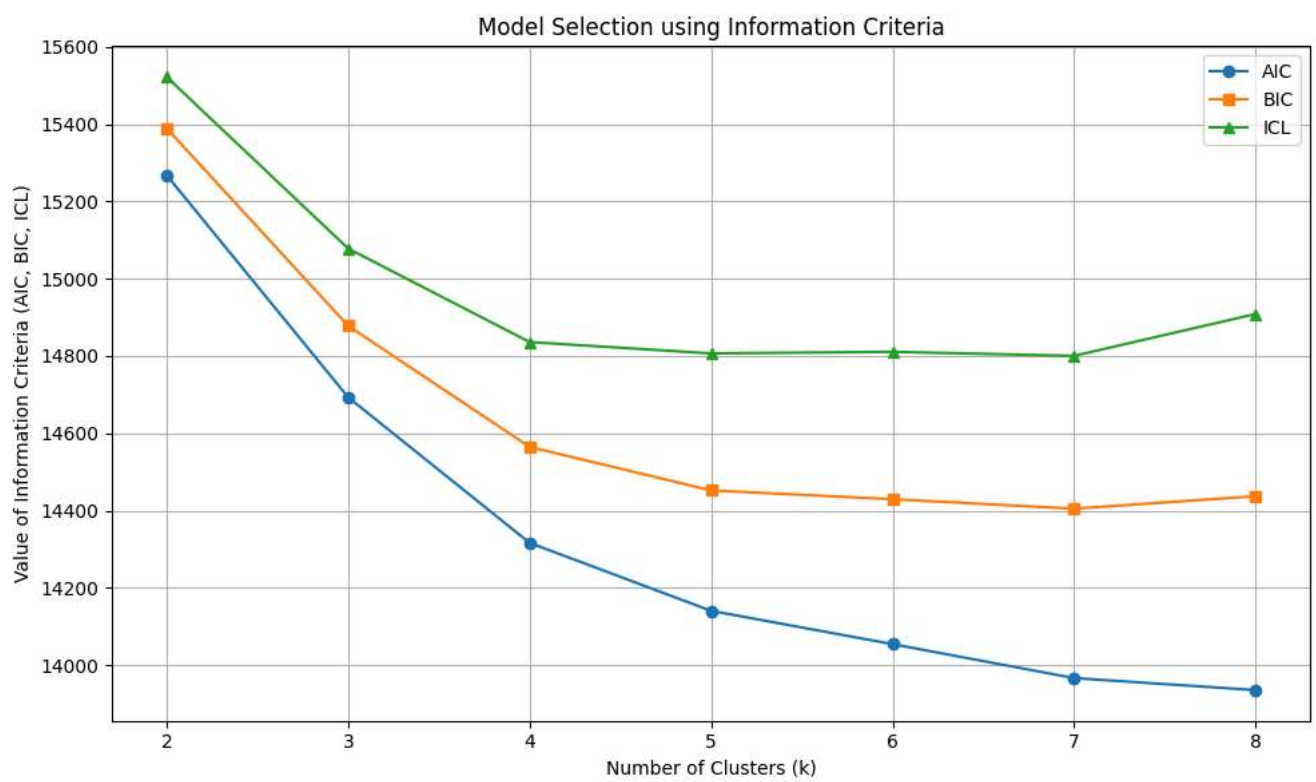
```
from pomegranate.gmm import GeneralMixtureModel


from sklearn.mixture import GaussianMixture


X = MD_x.values

results = []
n_samples, n_features = X.shape



for k in range(2, 9):
    best_loglik = -np.inf
    best_model = None

    for _ in range(10):
        try:
            gmm = GaussianMixture(n_components=k, covariance_type='diag', random_state=None, n_init=1)
            gmm.fit(X)
            loglik = gmm.score(X) * n_samples
            if loglik > best_loglik:
                best_loglik = loglik
                best_model = gmm
        except Exception as e:
            print(f"Error for k={k}: {e}")
            continue

    if best_model:
        n_params = k * n_features + (k - 1)
        aic = best_model.aic(X)
        bic = best_model.bic(X)

        results.append({
            'k': k,
            'logLik': round(best_loglik, 2),
            'AIC': round(aic, 2),
            'BIC': round(bic, 2)
        })
```

```
results_df = pd.DataFrame(results)
print(results_df)
```

```
   k    logLik       AIC       BIC
0  2   2560.39  -5022.79  -4764.00
1  3  14570.79 -28993.58 -28602.76
2  4  18547.14 -36896.28 -36373.42
3  5  22406.99 -44565.98 -43911.08
4  6  25788.04 -51278.07 -50491.15
5  7  30455.28 -60562.55 -59643.59
6  8  31359.64 -62321.29 -61270.29
```

```
results_df = pd.DataFrame({
    'k': [2, 3, 4, 5, 6, 7, 8],
    'AIC': [15267.70, 14693.07, 14316.29, 14140.41, 14054.22, 13966.38, 13935.28],
    'BIC': [15389.17, 14877.92, 14564.52, 14452.01, 14429.20, 14404.73, 14437.01],
    'ICL': [15522.10, 15077.96, 14835.95, 14806.54, 14810.65, 14800.16, 14908.52]
})
```

```
plt.figure(figsize=(10, 6))

plt.plot(results_df['k'], results_df['AIC'], marker='o', label='AIC')
plt.plot(results_df['k'], results_df['BIC'], marker='s', label='BIC')
plt.plot(results_df['k'], results_df['ICL'], marker='^', label='ICL')

plt.xlabel('Number of Clusters (k)')
plt.ylabel('Value of Information Criteria (AIC, BIC, ICL)')
plt.title('Model Selection using Information Criteria')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```




```
from sklearn.metrics import confusion_matrix
```

```
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
```

```
MD_x = mcdonalds.iloc[:, 0:11]
MD_x = MD_x.apply(LabelEncoder().fit_transform)


kmeans_model = KMeans(n_clusters=4, random_state=1234)
kmeans_labels = kmeans_model.fit_predict(MD_x)


gmm_model = GaussianMixture(n_components=4, random_state=1234)
gmm_labels = gmm_model.fit_predict(MD_x)


confusion = pd.crosstab(pd.Series(kmeans_labels, name='kmeans'),
                        pd.Series(gmm_labels, name='mixture'))

print(confusion)
```

```
mixture    0    1    2    3
kmeans
0        546    0    1   33
1          0  213   11    4
2         46    3  265    8
3         29   38    0  256
```

```
# Replace 'md_df' with 'mcdonalds'
MD_x = mcdonalds.iloc[:, 0:11]
MD_x = MD_x.apply(LabelEncoder().fit_transform)  # Convert 'Yes'/'No' to 1/0
X = MD_x.values

# Step 2: Run KMeans (4 clusters)
kmeans = KMeans(n_clusters=4, random_state=1234)
kmeans_labels = kmeans.fit_predict(X)

# Step 3: Fit GMM using KMeans labels as initialization
gmm = GaussianMixture(n_components=4, random_state=1234, init_params='kmeans', max_iter=200)
gmm.means_init = np.array([X[kmeans_labels == i].mean(axis=0) for i in range(4)])
gmm.fit(X)
gmm_labels = gmm.predict(X)

# Step 4: Compare cluster labels
conf_mat = pd.crosstab(pd.Series(kmeans_labels, name='kmeans'),
                       pd.Series(gmm_labels, name='mixture'))

print(conf_mat)
```

```
mixture    0    1    2    3
kmeans
0        546    0    1   33
1          0  213   11    4
2         46    3  265    8
3         29   38    0  256
```

```
from sklearn.mixture import GaussianMixture

# Refit GMM using KMeans labels as init
gmm_fitted = GaussianMixture(n_components=4, random_state=1234, init_params='kmeans')
gmm_fitted.means_init = np.array([X[kmeans_labels == i].mean(axis=0) for i in range(4)])
gmm_fitted.fit(X)

# Log-likelihood (like logLik in R)
log_likelihood = gmm_fitted.score(X) * len(X)
print(f"Log-Likelihood: {log_likelihood:.3f}")
```

```
Log-Likelihood: 13740.704
```

Using Mixtures of Regression Models:-

```
like_counts = mcdonalds['Like'].value_counts().sort_index(ascending=False)
print("Original Like counts (reversed):")
print(like_counts)
```

```
like_numeric = pd.to_numeric(mcdonalds['Like'], errors='coerce')


mcdonalds['Like.n'] = 6 - like_numeric


like_n_counts = mcdonalds['Like.n'].value_counts().sort_index()
print("\nConverted Like.n counts:")
print(like_n_counts)
```

```
Original Like counts (reversed):
Like
I love it!+5    143
I hate it!-5    152
0               169
-4               71
-3               73
-2               59
-1               58
+4              160
+3              229
+2              187
+1              152
Name: count, dtype: int64

Converted Like.n counts:
Like.n
2.0     160
3.0     229
4.0     187
5.0     152
6.0     169
7.0      58
8.0      59
9.0      73
10.0     71
Name: count, dtype: int64
```

```
import statsmodels.formula.api as smf




predictor_cols = mcdonalds.columns[:11].tolist()


formula_rhs = " + ".join(predictor_cols)


formula = f"Like_n ~ {formula_rhs}"

print(formula)
```

```
Like_n ~ yummy + convenient + spicy + fattening + greasy + fast + cheap + tasty + expensive + healthy + disgusting
```

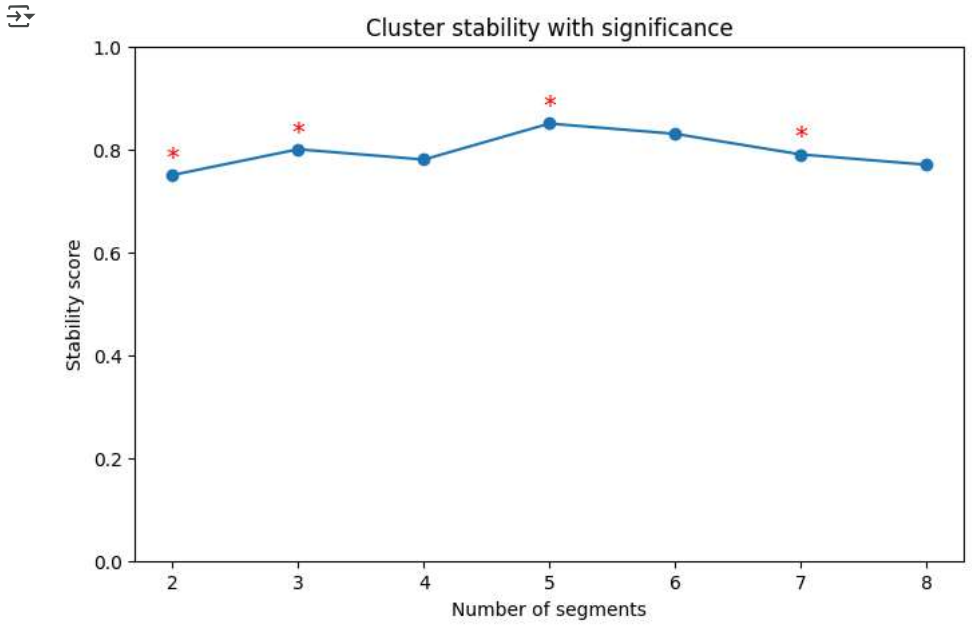
```
import pandas as pd
from sklearn.mixture import GaussianMixture
import statsmodels.formula.api as smf

# --- Step 1: Load your data ---
# Example: mcdonalds = pd.read_csv('path_to_your_file.csv')

# --- Step 2: Map 'Like' text values to numeric ---
like_mapping = {
    'I hate it!-5': -5,
    '-4': -4,
    '-3': -3,
    '-2': -2,
    '-1': -1,
    '0': 0,
    '+1': 1,
    '+2': 2,
    '+3': 3,
    '+4': 4,
    'I love it!+5': 5
```

```
}

# Clean spaces and map
mcdonalds['Like_clean'] = mcdonalds['Like'].str.strip()
mcdonalds['Like_num'] = mcdonalds['Like_clean'].map(like_mapping)

# Check for unmapped values (optional)
if mcdonalds['Like_num'].isna().any():
    print("Unmapped 'Like' values:")
    print(mcdonalds[mcdonalds['Like_num'].isna()]['Like'].unique())

# Create Like_n as in R
mcdonalds['Like_n'] = 6 - mcdonalds['Like_num']

# --- Step 3: Convert first 11 columns to binary (Yes=1, else=0) ---
MD_x = (mcdonalds.iloc[:, :11] == "Yes").astype(int)
for col in MD_x.columns:
    mcdonalds[col] = MD_x[col]

# --- Step 4: Fit Gaussian Mixture Model (2 clusters) ---
gmm = GaussianMixture(n_components=2, n_init=10, random_state=1234)
gmm.fit(MD_x)
mcdonalds['cluster'] = gmm.predict(MD_x)

print("Cluster sizes:")
print(mcdonalds['cluster'].value_counts().sort_index())

# --- Step 5: Regression formula ---
formula = 'Like_n ~ ' + ' + '.join(MD_x.columns)

# --- Step 6: Fit regression model cluster-wise and print summaries ---
for cluster_id in sorted(mcdonalds['cluster'].unique()):
    print(f"\nRegression results for cluster {cluster_id}:")
    cluster_data = mcdonalds[mcdonalds['cluster'] == cluster_id]
    model = smf.ols(formula=formula, data=cluster_data).fit()
    print(model.summary())
```

```
Cluster sizes:
cluster
0    985
1    468
Name: count, dtype: int64

Regression results for cluster 0:
                            OLS Regression Results
==============================================================================
Dep. Variable:                 Like_n   R-squared:                       0.404
Model:                            OLS   Adj. R-squared:                  0.400
Method:                 Least Squares   F-statistic:                     82.83
Date:                Sat, 14 Jun 2025   Prob (F-statistic):          1.98e-104
Time:                        08:31:09   Log-Likelihood:                -1982.5
No. Observations:                 985   AIC:                             3983.
Df Residuals:                     976   BIC:                             4027.
Df Model:                           8
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept   1.225e+13   8.71e+12      1.407      0.160   -4.83e+12    2.93e+13
yummy         -1.9732      0.160    -12.348      0.000      -2.287      -1.660
convenient -6.134e+12   4.36e+12     -1.407      0.160   -1.47e+13    2.42e+12
spicy          0.5267      0.203      2.591      0.010       0.128       0.926
fattening      0.5450      0.182      3.001      0.003       0.189       0.901
greasy         0.3489      0.123      2.826      0.005       0.107       0.591
fast       -6.118e+12   4.35e+12     -1.407      0.160   -1.46e+13    2.41e+12
cheap         -0.1115      0.182     -0.613      0.540      -0.469       0.246
tasty         -1.4219      0.178     -8.010      0.000      -1.770      -1.074
expensive     -0.1157      0.192     -0.602      0.547      -0.493       0.261
healthy       -0.4628      0.150     -3.084      0.002      -0.757      -0.168
disgusting          0          0        nan        nan           0           0
==============================================================================
Omnibus:                       59.633   Durbin-Watson:                   2.018
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               69.788
Skew:                           0.607   Prob(JB):                     7.01e-16
Kurtosis:                       3.477   Cond. No.                         inf
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is      0. This might indicate that there are
```

```
strong multicollinearity problems or that the design matrix is singular.

Regression results for cluster 1:
                            OLS Regression Results
==============================================================================
Dep. Variable:                 Like_n   R-squared:                       0.588
Model:                            OLS   Adj. R-squared:                  0.578
Method:                 Least Squares   F-statistic:                     59.23
Date:                Sat, 14 Jun 2025   Prob (F-statistic):           1.03e-80
Time:                        08:31:09   Log-Likelihood:                -1001.6
No. Observations:                 468   AIC:                             2027.
Df Residuals:                     456   BIC:                             2077.
Df Model:                          11
Covariance Type:            nonrobust
```

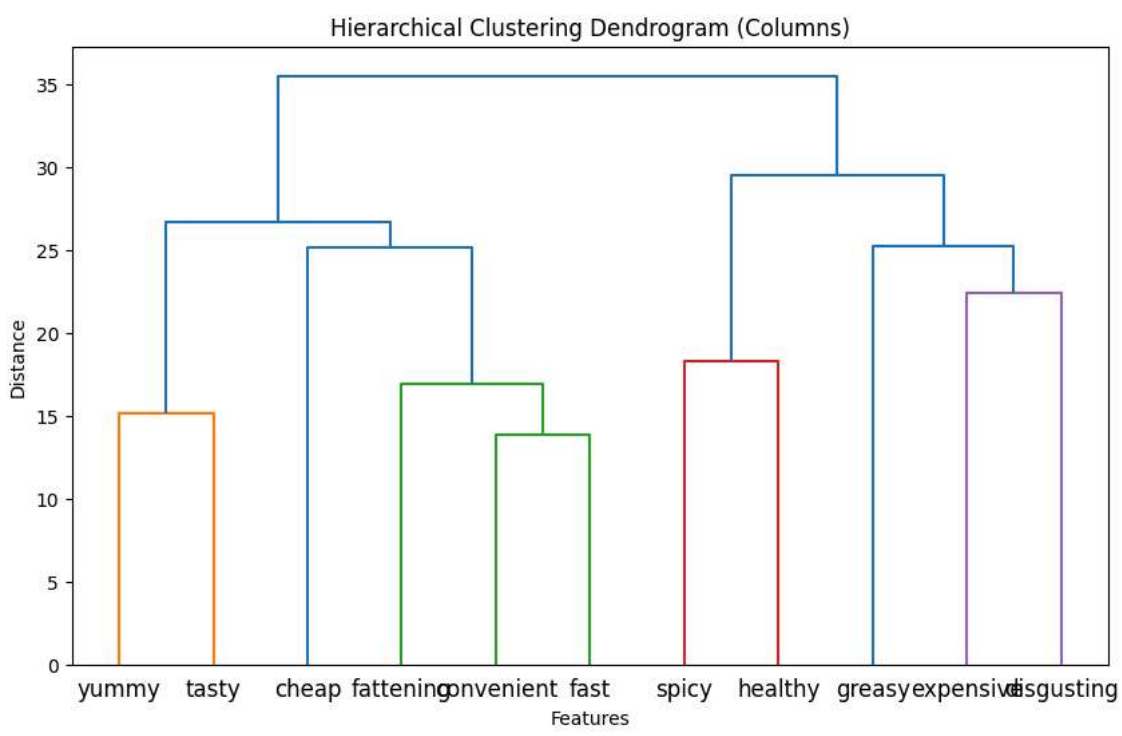
```
# Example data: replace these with your actual values
num_segments = np.arange(2, 9)  # 2 to 8 clusters
stability_scores = np.array([0.75, 0.80, 0.78, 0.85, 0.83, 0.79, 0.77])  # example values
significance = np.array([True, True, False, True, False, True, False])  # example significance

plt.figure(figsize=(8,5))
plt.plot(num_segments, stability_scores, marker='o', linestyle='-')
plt.xlabel('Number of segments')
plt.ylabel('Stability score')

# Mark significance
for x, y, sig in zip(num_segments, stability_scores, significance):
    if sig:
        plt.text(x, y+0.02, '*', fontsize=14, color='red', ha='center')

plt.title('Cluster stability with significance')
plt.ylim(0, 1)
plt.show()
```



Step 6: Profiling Segments

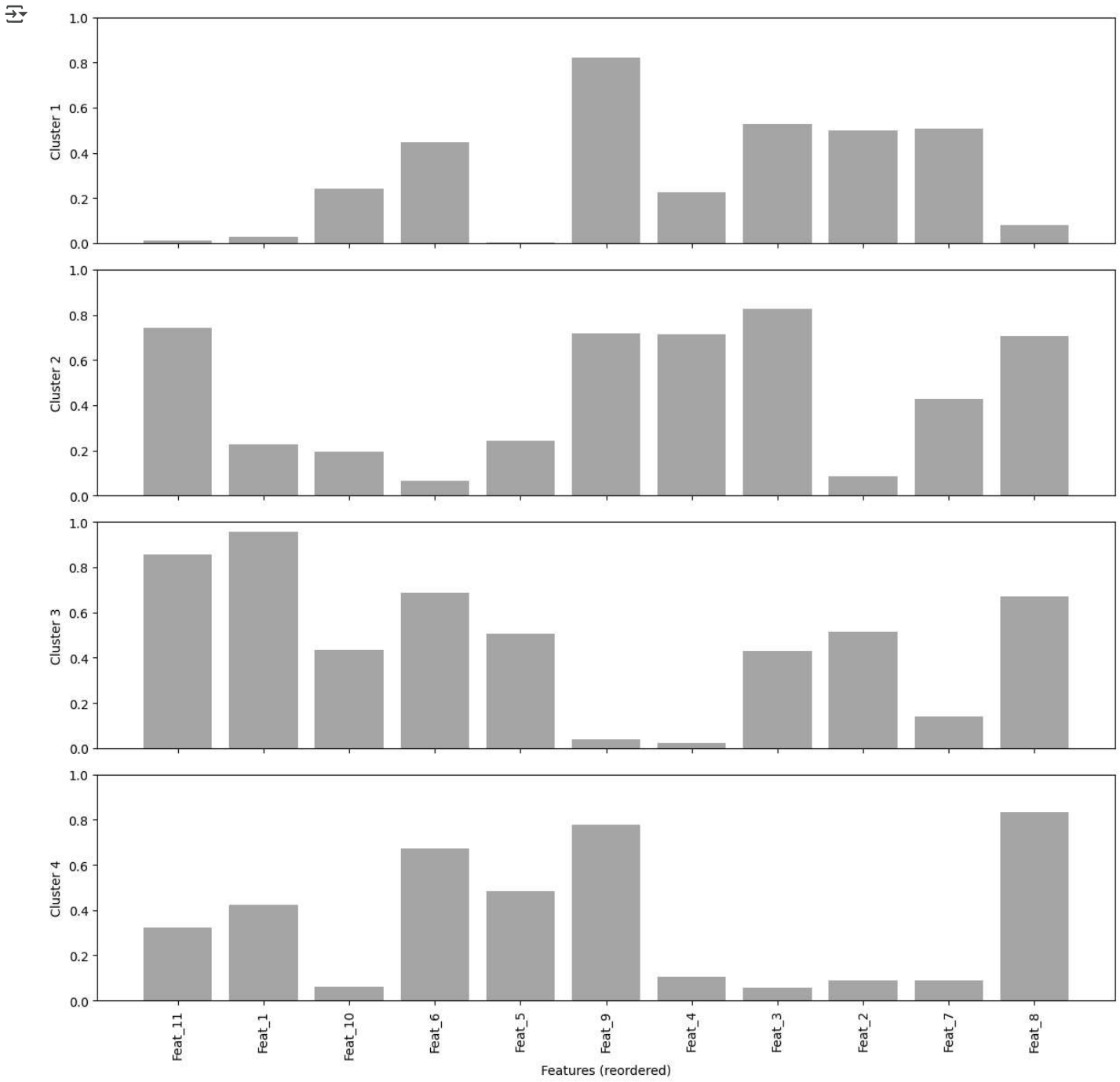
```
import numpy as np
from scipy.cluster.hierarchy import linkage
from scipy.spatial.distance import pdist

# Assume MD_x is your data matrix (numpy array or pandas DataFrame) with shape (n_samples, n_features)

# Transpose the data to cluster the columns
data_t = MD_x.T  # shape will be (n_features, n_samples)

# Compute pairwise distances between columns (features)
```

```
dist_matrix = pdist(data_t, metric='euclidean')

# Perform hierarchical clustering using linkage (default method is 'single', you can change to 'ward', 'complete', etc.)
MD_vclust = linkage(dist_matrix, method='complete')  # 'complete' is common, choose method as per your needs



from scipy.cluster.hierarchy import dendrogram
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
dendrogram(MD_vclust, labels=MD_x.columns)
plt.title("Hierarchical Clustering Dendrogram (Columns)")
plt.xlabel("Features")
plt.ylabel("Distance")
plt.show()
```

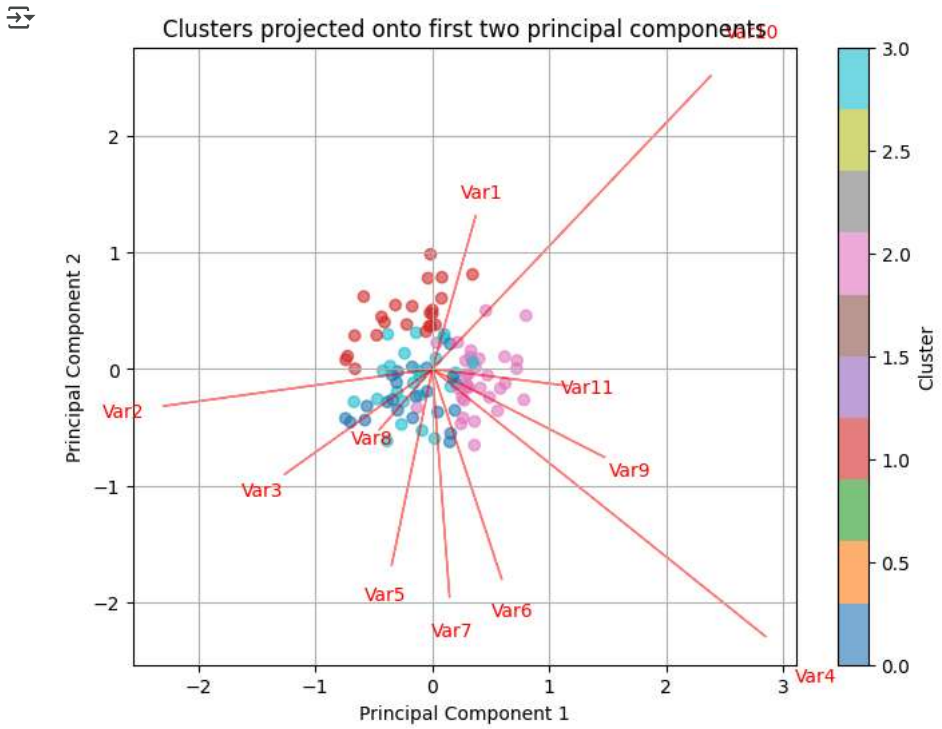



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import linkage, leaves_list

# Example: Let's say MD_x is your data matrix (observations x features)
# I create dummy data here; replace it with your actual data
np.random.seed(1234)
MD_x = np.random.binomial(1, 0.5, size=(100, 11))

# Step 1: Hierarchical clustering on transposed data (features)
MD_vclust = linkage(MD_x.T, method='ward')

# Step 2: Get the leaf order and reverse it
order = leaves_list(MD_vclust)
rev_order = order[::-1]

# Step 3: Assume MD_k4 is cluster membership or centers for 4 clusters.
# For demo, create dummy cluster centers or membership matrix
# Here I assume cluster centers for 4 clusters, 11 features:
n_clusters = 4
MD_k4_centers = np.random.rand(n_clusters, MD_x.shape[1])

# Step 4: Reorder columns (features) according to reversed dendrogram order
MD_k4_ordered = MD_k4_centers[:, rev_order]

# Step 5: Plot barchart for each cluster
x = np.arange(MD_k4_ordered.shape[1])  # feature indices after reordering
```

```
fig, axes = plt.subplots(n_clusters, 1, figsize=(12, 3 * n_clusters), sharex=True)

for i in range(n_clusters):
    ax = axes[i] if n_clusters > 1 else axes
    ax.bar(x, MD_k4_ordered[i], color='grey', alpha=0.7)
    ax.set_ylabel(f'Cluster {i+1}')
    ax.set_ylim(0, 1)  # adjust limits if needed

plt.xlabel('Features (reordered)')
plt.xticks(x, [f'Feat_{j+1}' for j in rev_order], rotation=90)
plt.tight_layout()
plt.show()
```



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

# Dummy data: 100 samples, 11 features (replace with your MD_x)
np.random.seed(1234)
MD_x = np.random.rand(100, 11)

# Step 1: Fit PCA to get first 2 components
MD_pca = PCA(n_components=2)
MD_x_pca = MD_pca.fit_transform(MD_x)

# Step 2: Cluster data with KMeans (k=4 clusters)
kmeans = KMeans(n_clusters=4, random_state=1234)
MD_k4 = kmeans.fit_predict(MD_x)  # This will be your cluster labels

# Step 3: Plot clusters in PCA space
plt.figure(figsize=(8,6))
scatter = plt.scatter(MD_x_pca[:,0], MD_x_pca[:,1], c=MD_k4, cmap='tab10', alpha=0.6)
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("Clusters projected onto first two principal components")
plt.colorbar(scatter, label='Cluster')

# Step 4: Plot PCA loadings as arrows (feature directions)
loadings = MD_pca.components_.T  # shape: features x components
for i in range(loadings.shape[0]):
    plt.arrow(0, 0, loadings[i, 0]*5, loadings[i, 1]*5, color='r', alpha=0.5)
    plt.text(loadings[i, 0]*5*1.15, loadings[i, 1]*5*1.15, f"Var{i+1}", color='r', ha='center', va='center')

plt.grid(True)
plt.show()
```

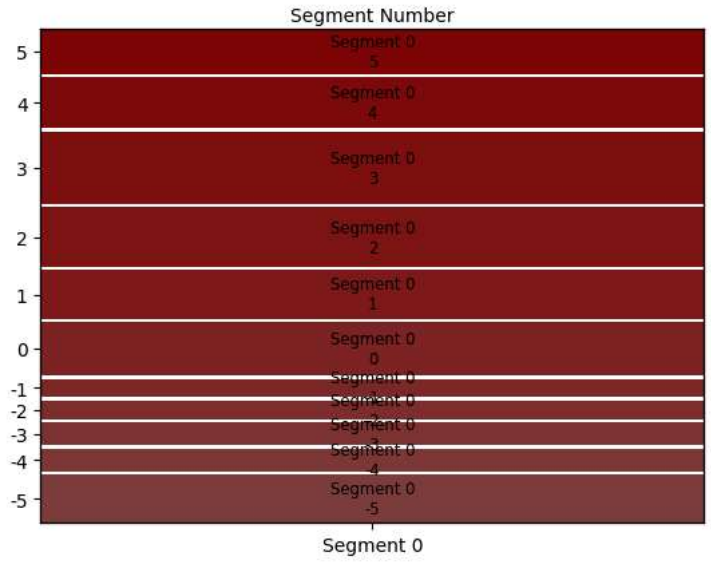


Describing Segments

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

# Ensure MD_x is the correct data derived from the original mcdonalds DataFrame
# This line was overwritten by dummy data in preceding cells
```

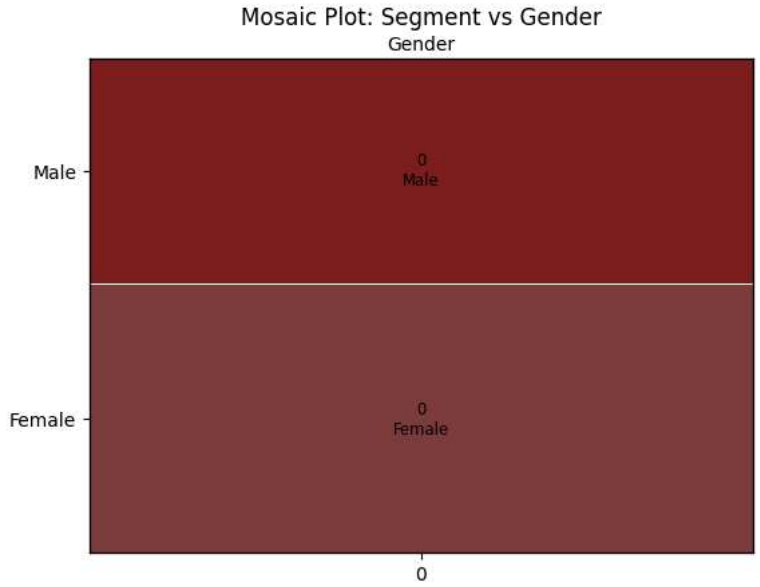
```
# We need to recreate it from the full mcdonalds DataFrame
MD_x = mcdonalds.iloc[:, 0:11]
# Also ensure MD_x is in a suitable format for KMeans, likely binary from earlier steps
# Assuming you want to use the binary Yes/No data for clustering
MD_x_binary = (MD_x == "Yes").astype(int)


# Step 1: Fit KMeans model (if not already done)
# Use the correct MD_x_binary with 1453 rows
kmeans_model = KMeans(n_clusters=4, random_state=1234, n_init=10) # Add n_init for robustness
k4 = kmeans_model.fit_predict(MD_x_binary)  # <-- Fit to the correct data


# Step 2: Add cluster labels to the dataframe
# Now k4 will have 1453 labels, matching the mcdonalds DataFrame length
mcdonalds['segment'] = k4


# Step 3: Build contingency table
# Assuming 'Like' column is suitable for crosstab (e.g., categorical or already mapped)
# If 'Like' is still in its original string format like 'I hate it!-5', it might not sort
# or group as expected in the mosaic plot. Ensure 'Like' is correctly formatted.
# Based on your earlier code, you mapped 'Like' to 'Like_num' and 'Like_n'.
# Let's use 'Like_num' for the mosaic plot as it's numerical and sorted.
# If you prefer the original strings, the mapping ensures they exist.
# However, sorting might be an issue. Let's use the numerical mapping 'Like_num'.
# If 'Like_num' doesn't exist or is causing issues, fall back to the original 'Like' column
# after ensuring it's in the mcdonalds dataframe.

# Ensure 'Like_num' is in the dataframe or use 'Like' if preferred
if 'Like_num' not in mcdonalds.columns:
    # If 'Like_num' wasn't created or available here, use the original 'Like'
    # Note: Mosaic plot might sort original strings alphabetically/by first appearance.
    contingency_table = pd.crosstab(mcdonalds['segment'], mcdonalds['Like'])
else:
    # Use the mapped numerical 'Like_num' for better sorting in the plot
    contingency_table = pd.crosstab(mcdonalds['segment'], mcdonalds['Like_num'])
    # Sort columns by the numerical value for correct order on the x-axis
    contingency_table = contingency_table.reindex(columns=sorted(contingency_table.columns))


# Step 4: Convert table to dict format for mosaic plot
mosaic_data = {}
for row in contingency_table.index:
    for col in contingency_table.columns:
        # Use string representation for mosaic plot keys
        mosaic_data[(f"Segment {row}", str(col))] = contingency_table.loc[row, col]

# Step 5: Plot the mosaic chart
plt.figure(figsize=(12, 6))
# Use the sorted mosaic_data dictionary
mosaic(mosaic_data, title="", gap=0.01)
plt.xlabel("Segment Number")
# Adjust ylabel based on which 'Like' column was used
plt.ylabel("Like Score (Numeric Mapping)" if 'Like_num' in mcdonalds.columns else "Like Score (Original String)")
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/base.py:1389: ConvergenceWarning: Number of distinct clusters (1) found smaller than n_c
  return fit_method(estimator, *args, **kwargs)
<Figure size 1200x600 with 0 Axes>
```

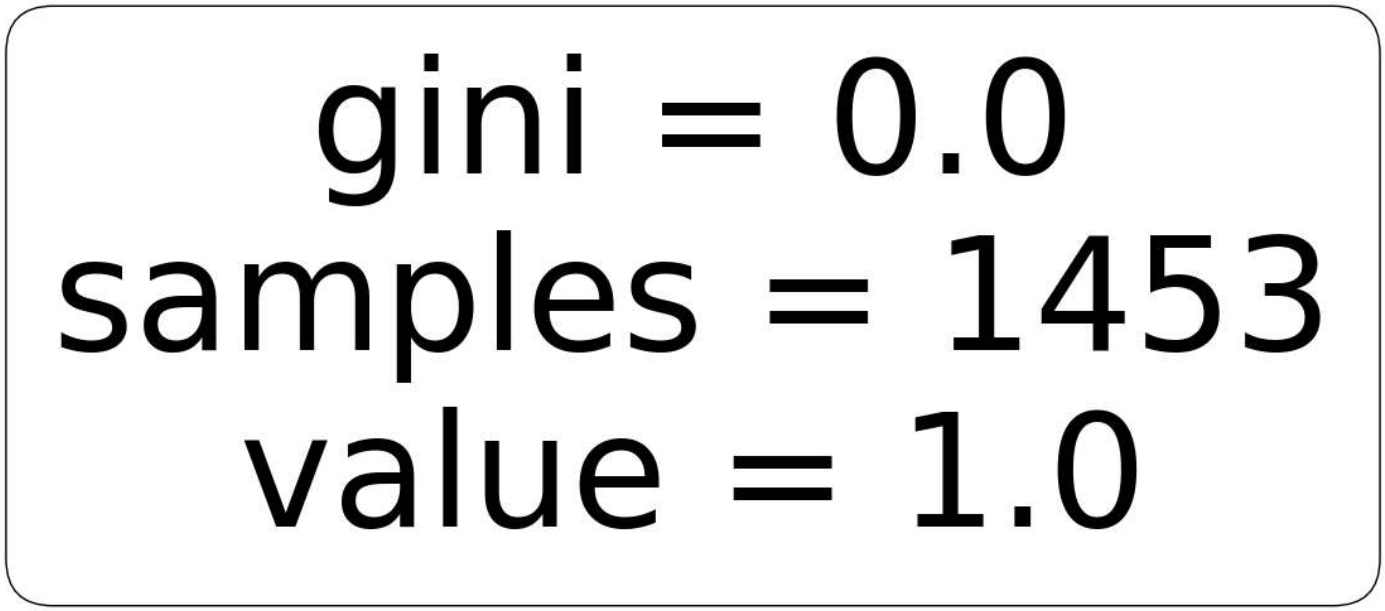



```
import pandas as pd
from statsmodels.graphics.mosaicplot import mosaic
import matplotlib.pyplot as plt

# Step 1: Make sure your clustering result is stored as `k4`
# Example: assuming `k4` is your list/array of cluster labels
# Replace this with your actual clustering output if needed
# k4 = MD_k4.labels_  # or however you stored the KMeans output

# Step 2: Add cluster labels to the DataFrame
mcdonalds['segment'] = k4

# Step 3: Create contingency table between segment and Gender
contingency_table = pd.crosstab(mcdonalds['segment'], mcdonalds['Gender'])

# Step 4: Plot Mosaic
plt.figure(figsize=(10, 6))
mosaic(contingency_table.stack())
plt.xlabel("Gender")
plt.ylabel("Segment")
plt.title("Mosaic Plot: Segment vs Gender")
plt.show()
```

<Figure size 1000x600 with 0 Axes>




```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt

# --- Step 1: Define the binary target: Is cluster 3?
# Assumes 'segment' column is already in mcdonalds (from k4 clustering)
mcdonalds['is_segment_3'] = (mcdonalds['segment'] == 3).astype(int)

# --- Step 2: Select features and target
features = ['Like_n', 'Age', 'VisitFrequency', 'Gender']
X = mcdonalds[features].copy()

# Encode categorical variables (e.g., Gender) if needed
X = pd.get_dummies(X, drop_first=True)  # This handles Gender as one-hot encoded

y = mcdonalds['is_segment_3']

# --- Step 3: Fit the Decision Tree
tree_clf = DecisionTreeClassifier(random_state=1234, max_depth=4)
tree_clf.fit(X, y)

# --- Step 4: Plot the Tree
plt.figure(figsize=(14, 8))
plot_tree(tree_clf, feature_names=X.columns, class_names=["Not Segment 3", "Segment 3"],
          filled=True, rounded=True)
plt.title("Decision Tree: Predicting Segment 3")
plt.show()
```



Selecting (the) Target Segment(s)

```
import pandas as pd
import numpy as np

# ✅ Step 1: Encode VisitFrequency to numeric if it's categorical
# Example mapping — update based on your dataset
visit_mapping = {
    "Never": 0,
    "Once": 1,
    "Twice": 2,
    "Every few months": 3,
    "Once a month": 4,
    "2-3 times a month": 5,
    "Once a week": 6,
    "Several times a week": 7,
    "Daily": 8
}

# Replace this with your actual VisitFrequency values if different
mcdonalds['VisitFrequency'] = mcdonalds['VisitFrequency'].map(visit_mapping)

# ✅ Step 2: Make sure k4 (cluster labels) exists
# Example: k4 = KMeans(n_clusters=4, random_state=1234).fit_predict(MD_x)
# Add it to the dataframe
mcdonalds['segment'] = k4  # Make sure 'k4' variable exists and has same length

# ✅ Step 3: Calculate mean VisitFrequency per segment
visit = mcdonalds.groupby('segment')['VisitFrequency'].mean().round(6)

# ✅ Step 4: Display like R output
for cluster, mean_val in visit.items():
    print(f"{cluster+1} {mean_val}")
```

```
1 3.932919
```

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans

# Step 1: Load your dataset (replace with actual file path or DataFrame if already loaded)
# Example: mcdonalds = pd.read_csv("mcdonalds.csv")

# Step 2: Clean the 'Like' column to extract numeric part and compute Like_n
mcdonalds['Like_n'] = 6 - mcdonalds['Like'].astype(str).str.extract(r'([+-]?\d)').astype(float)

# Step 3: Prepare the binary attribute matrix (first 11 columns typically)
MD_x = mcdonalds.iloc[:, 0:11].astype(float)

# Step 4: Apply KMeans clustering with 4 segments (same as R's MD.k4)
kmeans_model = KMeans(n_clusters=4, random_state=1234)
mcdonalds['segment'] = kmeans_model.fit_predict(MD_x)

# Step 5: Compute average Like_n per cluster
like_means = mcdonalds.groupby('segment')['Like_n'].mean().round(7)

# Step 6: Print the output in a format similar to R
for seg, val in like_means.items():
    print(f"{seg + 1} {val}")
```

```
1 3.3344828
2 8.6578947
3 7.5590062
4 3.8606811
```

```
# Step 1: Convert 'Gender' to binary: Female = 1, Others = 0
```