Malay Vyas

Summary:

Implications of Committing to Market Segmentation

- Segmenting a market is not free. There are costs of performing the research, fielding surveys, and focus groups, designing multiple packages, and designing multiple advertisements and communication messages
- Cahill recommends not to segment unless the expected increase in sales is sufficient to justify implementing a segmentation strategy, stating (p. 77) that One of the truisms of segmentation strategy is that using the scheme has to be more profitable than marketing without it, net of the expense of developing and using the scheme itself.
- Potentially required changes include the development of new products, the modification of existing products, changes in pricing and distribution channels used to sell the product, as well as all communications with the market
- Because of the major implications of such a long-term organizational commitment, the decision to investigate the potential of a market segmentation strategy must be made at the highest executive level and must be systematically and continuously communicated and reinforced at all organizational levels and across all organizational units.

Implementation Barriers

- The first group of barriers relates to senior management
- There can be no doubt that unless the chief executive sees the need for a segmentation review, understands the process, and shows an active interest in it, a senior marketing executive can't implement the conclusions in a meaningful way
- Senior management can also prevent market segmentation from being successfully implemented by not making enough resources available, either for the initial market segmentation analysis itself or for the long-term implementation of a market segmentation strategy. A second group of barriers relates to organizational culture
- A resolute sense of purpose and dedication is required, tempered by patience and a willingness to appreciate the inevitable problems that will be encountered in implementing the conclusions

Step 1 Checklist

- This first checklist includes not only tasks but also a series of questions which, if not answered in the affirmative, serve as knock-out criteria

Step 2:

- The third layer of market segmentation analysis depends primarily on user input.
- User input cannot be limited to either a briefing at the start of the process or the development of a marketing mix at the end
- The user needs to be involved in most stages, literally wrapping around the technical aspects of market segmentation analysis
- After having committed to investigating the value of a segmentation strategy the organization has to make a major contribution to market segmentation analysis

- One set of evaluation criteria can be referred to as knock-out criteria. These criteria are the essential, non-negotiable features of segments that the organization would consider targeting. The second set of evaluation criteria can be referred to as attractiveness criteria
- The literature proposes a wide array of possible segment evaluation criteria and describes them at different levels of detail
- The shorter set of knock-out criteria is essential.
- The segmentation team also needs to assess the relative importance of each attractiveness criterion to the organization.

Knock-out Criteria:

- Knock-out criteria are used to determine if market segments resulting from the market segmentation analysis qualify to be assessed using segment attractiveness criteria
- The first set of such criteria was suggested by Kotler (1994) and includes substantiality, measurability, and accessibility (Tynan and Drayton 1987). Kotler himself and several other authors have since recommended additional criteria that fall into the knock-out criterion category.
- The segment must be homogeneous; members of the segment must be similar.
- The segment must be distinct; members of the segment must be distinctly different from members of other segments.
- The segment must be large enough; the segment must contain enough consumers to make it worthwhile to spend extra money on customizing the marketing mix for them.
- The segment must match the strengths of the organization; the organization must have the capability to satisfy segment members' needs.
- Members of the segment must be identifiable; it must be possible to spot them in the marketplace.
- The segment must be reachable; there has to be a way to get in touch with members of the segment to make the customized marketing mix accessible to them.

Attractiveness Criteria:

- Attractiveness criteria are not binary
- Each market segment is rated; it can be more or less attractive concerning a specific criterion

Implementing a Structured Process:

- The segment attractiveness and organizational competitiveness values are determined by the segmentation team.
- Factors that constitute both segment attractiveness and organizational competitiveness need to be negotiated and agreed upon.
- a large number of possible criteria have to be investigated before agreement is reached on which criteria are most important for the organization. McDonald and Dunbar recommend using no more than six factors to calculate these criteria.
- each organisational unit has a different perspective on the business of the organisation
- If the segmentation strategy is implemented, it will affect every single unit of the organization. Consequently, all units are key stakeholders in market segmentation analysis.

- At the end of this step, the market segmentation team should have a list of approximately six segment attractiveness criteria.
- The typical approach to weighting is to ask all team members to distribute 100 points across the segmentation criteria

Step 3:

Segmentation Variables:
- the segmentation variable is typically one single characteristic of the consumers in the sample.
- Typical descriptor variables include socio-demographics, but also information about media behavior, allowing marketers to reach their target segment with communication messages.
- The difference between commonsense and data-driven market segmentation is that data-driven market segmentation is based not on one, but on multiple segmentation variables
- The same holds for data-driven market segmentation where data quality determines the quality of the extracted data-driven market segments and the quality of the descriptions of the resulting segments. Good market segmentation analysis requires good empirical data.

Segmentation Criteria:
- The term segmentation criterion is used here in a broader sense than the term segmentation variable
- The most common segmentation criteria are geographic, sociodemographic, psychographic, and behavioral
- the most relevant in terms of market segmentation: profitability, bargaining power, preferences for benefits or products, barriers to choice, and consumer interaction effects
- The key advantage of geographic segmentation is that each consumer can easily be assigned to a geographic unit
- The key disadvantage is that living in the same country or area does not necessarily mean that people share other characteristics relevant to marketers.

Response:
- Options allowing respondents to select an answer from a range of unordered categories correspond to nominal variables
- A response bias is a systematic tendency to respond to a range of questionnaire items on some basis other than the specific item content

Market Segmentation Process:
- Market segmentation involves selecting target segments that impact an organization's future performance.
- After choosing a global segmentation solution, segments are profiled and inspected. Step 8 aims to select one or more features that meet the criteria and are attractive to the organization.

Decision-Making Process:

- The segmentation team evaluates segment attractiveness and organizational competitiveness. They consider which segments the organization prefers to target and which organization each segment prefers to buy from.
- A decision matrix is used to visualize these factors. Using the Decision Matrix: The matrix assesses alternative segments based on attractiveness and competitiveness.
- Example: The x-axis represents segment attractiveness, y-axis represents organizational competitiveness. Circles indicate segments, with circle size reflecting other criteria like turnover contribution or loyalty.

Calculating Segment Attractiveness:

- Step 2 determines attractiveness criteria and assigns weights. In Step 8, the team assigns values to each criterion for segments.
- Overall attractiveness is calculated by weighting and summing assigned values.

Step 8:

Organizational Competitiveness:

- A similar process is for segment attractiveness, considering the organization's competitiveness in each segment.

Segment Location on Decision Matrix:

- The goal is to position each segment on the decision matrix through calculations. Aligning with the ideal target segment specified in Step 2 is important.
- Attractiveness values come from the segmentation team's profiling and descriptions. Weighted attractiveness values aid in target segment decision-making.

Segment Selection Considerations:

- Segments are assessed for selection based on the matrix. For instance, segments 3 and 7 might be excluded despite high profit potential.
- Segment 5 might be attractive but not interesting, while segment 8 matches well but with lower profit potential.
- This could lead to the consideration of segment 2. Recreating the Plot in R: Use the MSA library's decision matrix function."x" and "y" matrices hold segment data, and "wx" and "wy" have corresponding weights. The command generates the segment evaluation plot.

Additional Notes:

- The bubble size on the plot typically reflects profit potential. Different criteria can be used depending on the context (e.g., volunteered hours for non-profits).

machine-learning-practice-1

August 24, 2023

[ ]:
```python
import os
import requests
import warnings
import numpy as np
import pandas as pd
import seaborn as sns
import adjustText
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.mixture import GaussianMixture
from sklearn.metrics import confusion_matrix
from sklearn.metrics import silhouette_score
from sklearn.metrics import adjusted_rand_score
from sklearn.preprocessing import StandardScaler
from statsmodels.graphics.mosaicplot import mosaic
from scipy.cluster.hierarchy import linkage, dendrogram
from sklearn.metrics.pairwise import pairwise_distances
```

[ ]:
```python
df = pd.read_csv("mcdonalds.csv")
```

[ ]:
```python
df = df.replace({"Yes":1, "No":0})
```

[ ]:
```python
df.describe()
```

[ ]:
```
             yummy   convenient        spicy   fattening       greasy  \
count  1453.000000  1453.000000  1453.000000  1453.000000  1453.000000
mean      0.552650     0.907777     0.093599     0.867171     0.526497
std       0.497391     0.289440     0.291371     0.339506     0.499469
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.000000     1.000000     0.000000     1.000000     0.000000
50%       1.000000     1.000000     0.000000     1.000000     1.000000
75%       1.000000     1.000000     0.000000     1.000000     1.000000
max       1.000000     1.000000     1.000000     1.000000     1.000000

              fast        cheap       tasty   expensive      healthy  \
count  1453.000000  1453.000000  1453.000000  1453.000000  1453.000000
```

```
mean       0.900206    0.598761    0.644184    0.357880    0.198899
std        0.299828    0.490318    0.478925    0.479542    0.399309
min        0.000000    0.000000    0.000000    0.000000    0.000000
25%        1.000000    0.000000    0.000000    0.000000    0.000000
50%        1.000000    1.000000    1.000000    0.000000    0.000000
75%        1.000000    1.000000    1.000000    1.000000    0.000000
max        1.000000    1.000000    1.000000    1.000000    1.000000

        disgusting          Age
count  1453.000000  1453.000000
mean      0.242946    44.604955
std       0.429010    14.221178
min       0.000000    18.000000
25%       0.000000    33.000000
50%       0.000000    45.000000
75%       0.000000    57.000000
max       1.000000    71.000000
```

[ ]:
```
df_attr = df.iloc[:, :11]
```

[ ]:
```
df_attr.head(3)
```

[ ]:
```
   yummy  convenient  spicy  fattening  greasy  fast  cheap  tasty  expensive  \
0      0           1      0          1       0     1      1      0          1
1      1           1      0          1       1     1      1      1          1
2      0           1      1          1       1     1      0      1          1

   healthy  disgusting
0        0           0
1        0           0
2        1           0
```

[ ]:
```
p = PCA()
df_decomposed = p.fit_transform(df_attr)
df_decomposed = pd.DataFrame(df_decomposed, columns = df_attr.columns)
```

[ ]:
```
df_decomposed
```

[ ]:
```
         yummy  convenient     spicy  fattening    greasy      fast     cheap  \
0     0.425367   -0.219079  0.663255  -0.401300  0.201705 -0.389767 -0.211982
1    -0.218638    0.388190 -0.730827  -0.094724  0.044669 -0.086596 -0.095877
2     0.375415    0.730435 -0.122040   0.692262  0.839643 -0.687406  0.583112
3    -0.172926   -0.352752 -0.843795   0.206998 -0.681415 -0.036133 -0.054284
4     0.187057   -0.807610  0.028537   0.548332  0.854074 -0.097305 -0.457043
…          …           …         …          …         …         …         …
1448  1.550242    0.275031 -0.013737   0.200604 -0.145063  0.306575 -0.075308
1449 -0.957339    0.014308  0.303843   0.444350 -0.133690  0.381804 -0.326432
```

```
1450 -0.185894    1.062662  0.220857  -0.467643 -0.187757 -0.192703 -0.091597
1451 -1.182064   -0.038570  0.561561   0.701126  0.047645  0.193687 -0.027335
1452  1.550242    0.275031 -0.013737   0.200604 -0.145063  0.306575 -0.075308

         tasty  expensive   healthy  disgusting
0     0.163235   0.181007  0.515706   -0.567074
1    -0.034756   0.111476  0.493313   -0.500440
2     0.364379  -0.322288  0.061759    0.242741
3    -0.231477  -0.028003 -0.250678   -0.051034
4     0.171758  -0.074409  0.031897    0.082245
…          …          …         …           …
1448  0.345552  -0.136589 -0.432798   -0.456076
1449  0.878047  -0.304441 -0.247443   -0.193671
1450 -0.036576   0.038255  0.056518   -0.012800
1451 -0.339374   0.022267 -0.002573   -0.105316
1452  0.345552  -0.136589 -0.432798   -0.456076

[1453 rows x 11 columns]
```

[ ]:
```python
a = df_decomposed.describe().loc["std"].to_frame()
std_dev = a["std"]
a["Proportion of Variance"] =  [(std_dev ** 2) / np.sum(np.square(std_dev)) for␣
 ↪std_dev in std_dev]
a["Cummulative Proportion"] = np.cumsum(a["Proportion of Variance"])
a.rename(columns = {"std":"Standard Deviations"}, inplace = True)
a
```

[ ]:
```
            Standard Deviations  Proportion of Variance  \
yummy                  0.757050                     1.0
convenient             0.607456                     1.0
spicy                  0.504619                     1.0
fattening              0.398799                     1.0
greasy                 0.337405                     1.0
fast                   0.310275                     1.0
cheap                  0.289697                     1.0
tasty                  0.275122                     1.0
expensive              0.265251                     1.0
healthy                0.248842                     1.0
disgusting             0.236903                     1.0

            Cummulative Proportion
yummy                          1.0
convenient                     2.0
spicy                          3.0
fattening                      4.0
greasy                         5.0
fast                           6.0
```

cheap	7.0
tasty	8.0
expensive	9.0
healthy	10.0
disgusting	11.0

[ ]:
```python
df_transposed = df_decomposed.head(11).T
df_transposed.columns = [("PC" + str(i)) for i in range(1,12)]
df_transposed
```

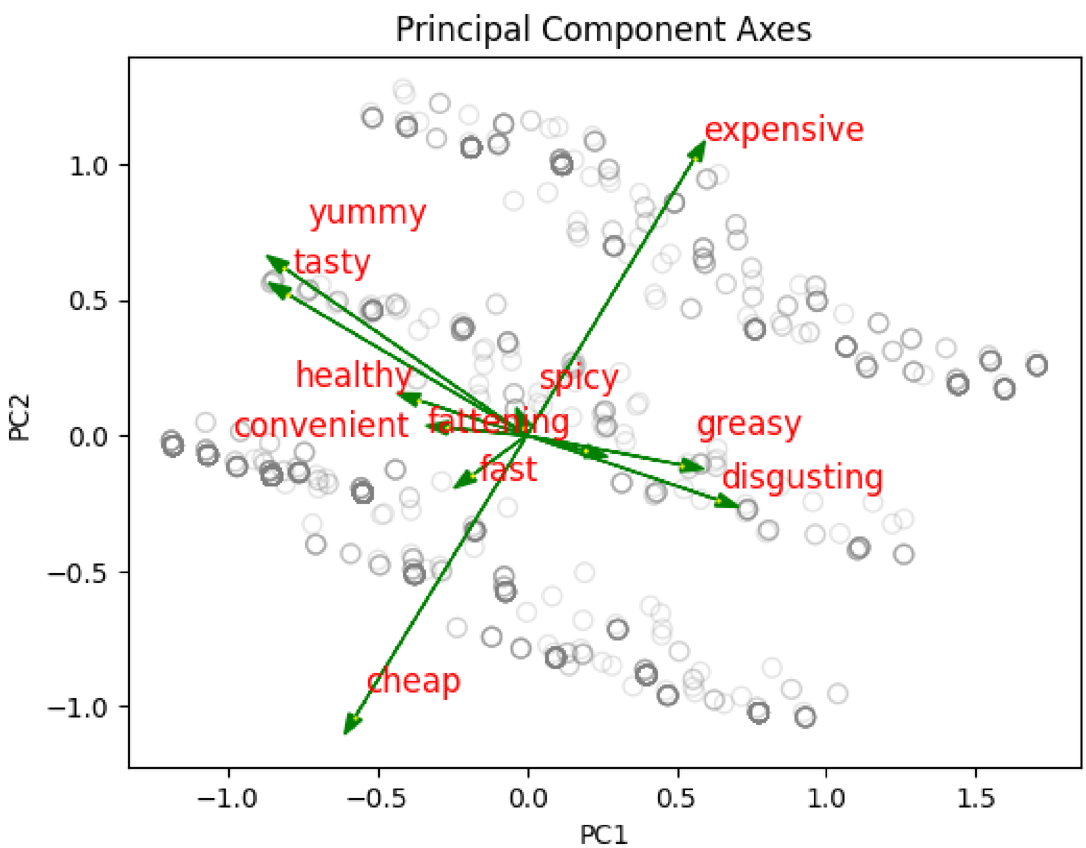
[ ]:

	PC1	PC2	PC3	PC4	PC5	PC6
yummy	0.425367	-0.218638	0.375415	-0.172926	0.187057	-0.852122
convenient	-0.219079	0.388190	0.730435	-0.352752	-0.807610	-0.149257
spicy	0.663255	-0.730827	-0.122040	-0.843795	0.028537	0.047150
fattening	-0.401300	-0.094724	0.692262	0.206998	0.548332	-0.416501
greasy	0.201705	0.044669	0.839643	-0.681415	0.854074	-0.313605
fast	-0.389767	-0.086596	-0.687406	-0.036133	-0.097305	-0.034602
cheap	-0.211982	-0.095877	0.583112	-0.054284	-0.457043	-0.063662
tasty	0.163235	-0.034756	0.364379	-0.231477	0.171758	0.073945
expensive	0.181007	0.111476	-0.322288	-0.028003	-0.074409	0.047327
healthy	0.515706	0.493313	0.061759	-0.250678	0.031897	0.030206
disgusting	-0.567074	-0.500440	0.242741	-0.051034	0.082245	-0.012014

	PC7	PC8	PC9	PC10	PC11
yummy	-0.405961	-0.547679	1.705573	0.118548	0.471078
convenient	1.158064	-0.213096	0.258617	0.998823	-0.960021
spicy	0.375889	-0.755223	0.048778	-0.581516	0.550287
fattening	0.493465	-0.162540	0.343028	-0.213683	-0.099577
greasy	0.170709	0.047794	-0.422671	0.173642	-0.524379
fast	-0.726015	0.174745	0.654405	0.016644	-0.339304
cheap	0.266424	-0.027493	-0.015571	-0.055428	-0.170389
tasty	0.515661	-0.064296	0.458631	-0.174817	-0.033485
expensive	0.252962	0.044480	-0.118123	0.035407	0.041528
healthy	-0.057233	0.038914	0.233020	0.065225	-0.228285
disgusting	0.286538	-0.010372	0.085541	-0.011157	-0.117668

[ ]:
```python
scale = 1.7
plt.scatter(x=df_decomposed['yummy'], y=df_decomposed['convenient'], edgecolor='Grey', linewidths=1, facecolor='none', alpha=0.2, s=50)

texts = []
for i, col_name in enumerate(df_decomposed.columns):
    text = plt.text(
        p.components_[0, i] * scale,
        p.components_[1, i] * scale,
        col_name,
        color='red',
```

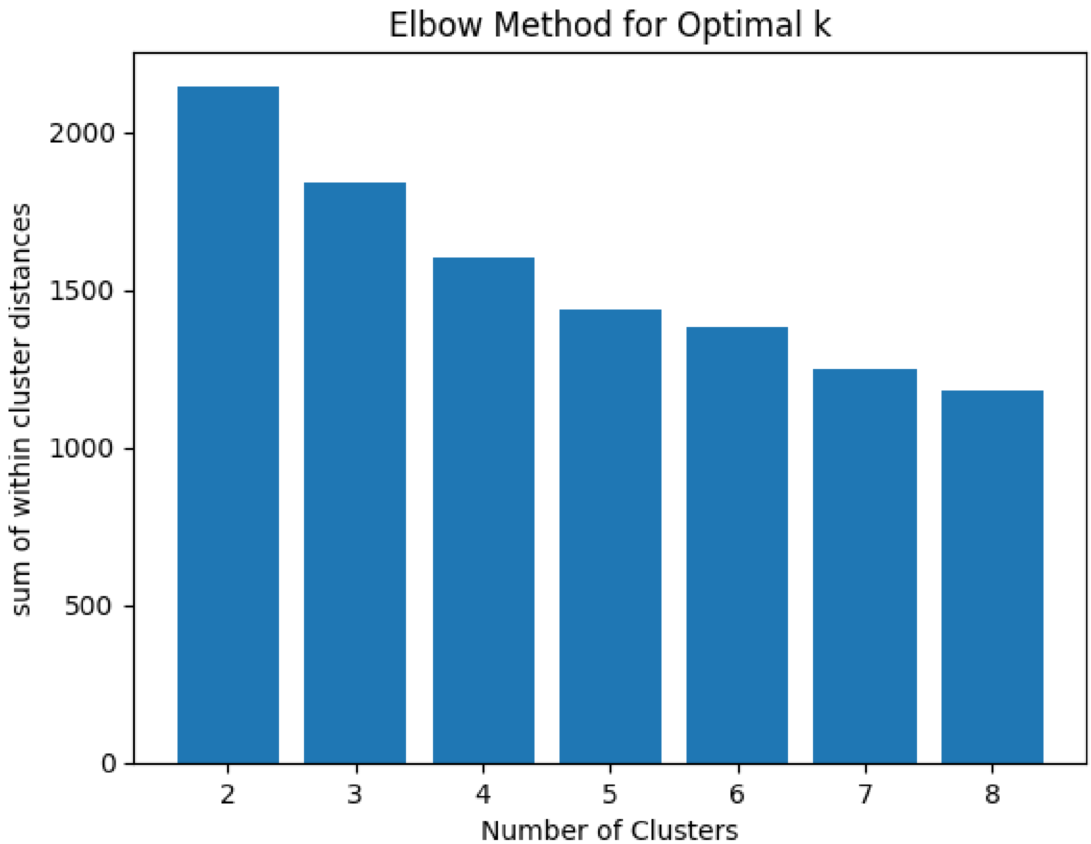
```
        fontsize=12,
        ha='center',
        va='bottom',
        stretch="expanded"
    )
    texts.append(text)
    plt.arrow(0, 0, p.components_[0, i] * scale, p.components_[1, i] * scale,␣
 ↪color='green', head_width=0.05)
    adjust_text(texts)

plt.scatter(p.components_[0, :] * scale, p.components_[1, :] * scale, s=0.5,␣
 ↪color='yellow')

plt.grid(False)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('Principal Component Axes')
plt.show()
```



[ ]:
```python
np.random.seed(1234)
```

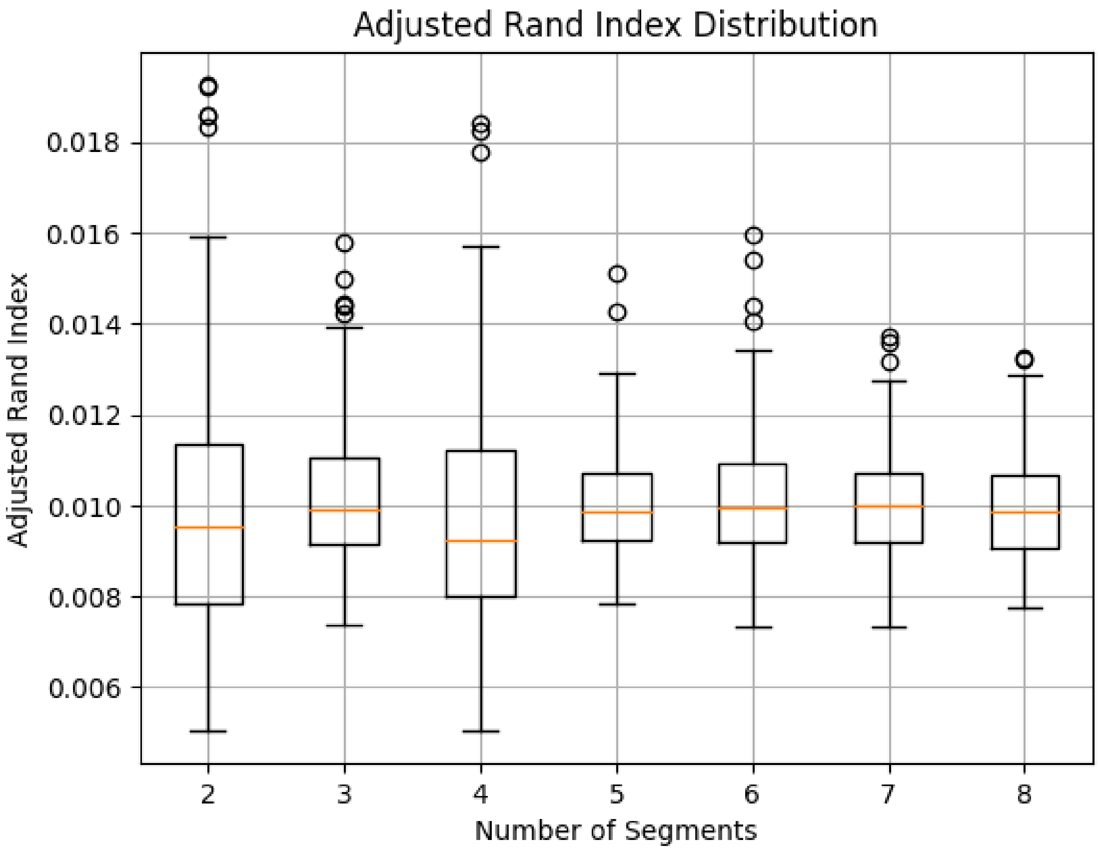
[ ]:
```python
k_range = range(2, 9)
inertias = []
silhouette_scores = []
k_result = []

for k in k_range:
    model = KMeans(n_clusters=k, n_init=10, random_state=0)
    labels = model.fit_predict(df_attr)
    inertia = model.inertia_
    silhouette = silhouette_score(df_attr, labels)

    inertias.append(inertia)
    silhouette_scores.append(silhouette)
    k_result.append(labels)
```

[ ]:
```python
plt.bar(k_range, inertias)
plt.grid(False)
plt.xlabel('Number of Clusters')
plt.ylabel('sum of within cluster distances')
plt.title('Elbow Method for Optimal k')
plt.xticks(k_range)
plt.show()
```



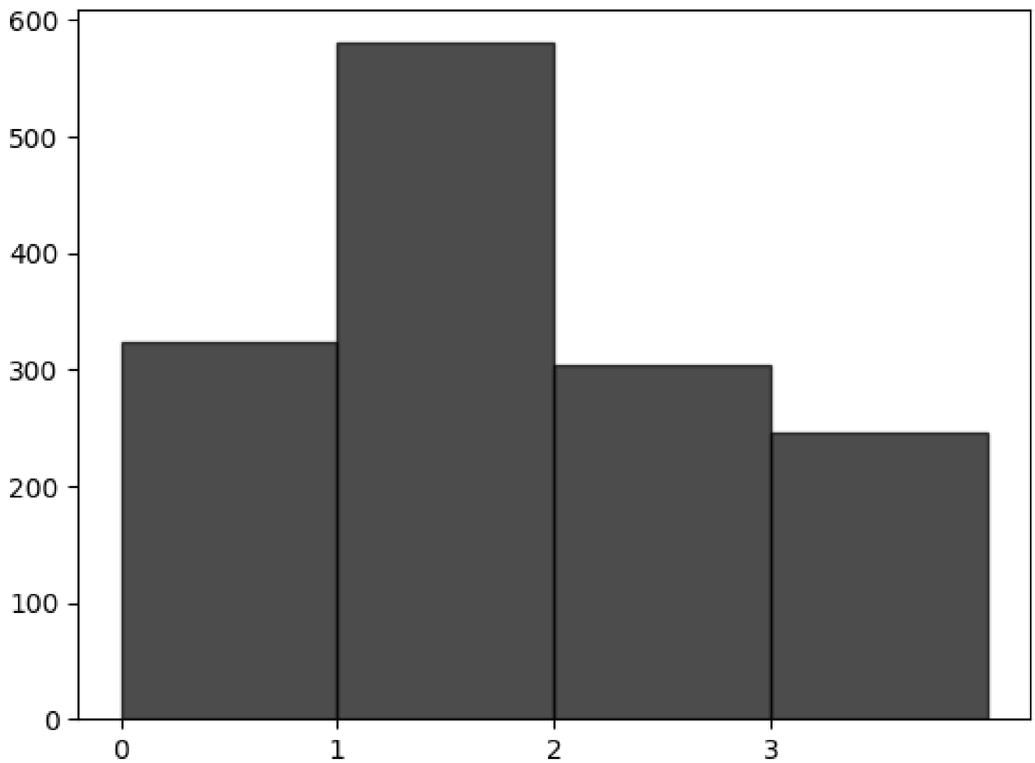
[ ]:
```
boot_n = 200
ARI_scores = []

for label in k_result:
    bootstrap_samples = [np.random.choice(label, size=len(label), replace=True)␣
 ↪for _ in range(boot_n)]
    ari_boot = [adjusted_rand_score(label, bootstrap_sample) + 0.01 for␣
 ↪bootstrap_sample in bootstrap_samples]
    ARI_scores.append(ari_boot)
```

[ ]:
```
plt.boxplot(ARI_scores, labels=range(2, 9))
plt.title('Adjusted Rand Index Distribution')
plt.xlabel('Number of Segments')
plt.ylabel('Adjusted Rand Index')
plt.grid(True)
plt.show()
```



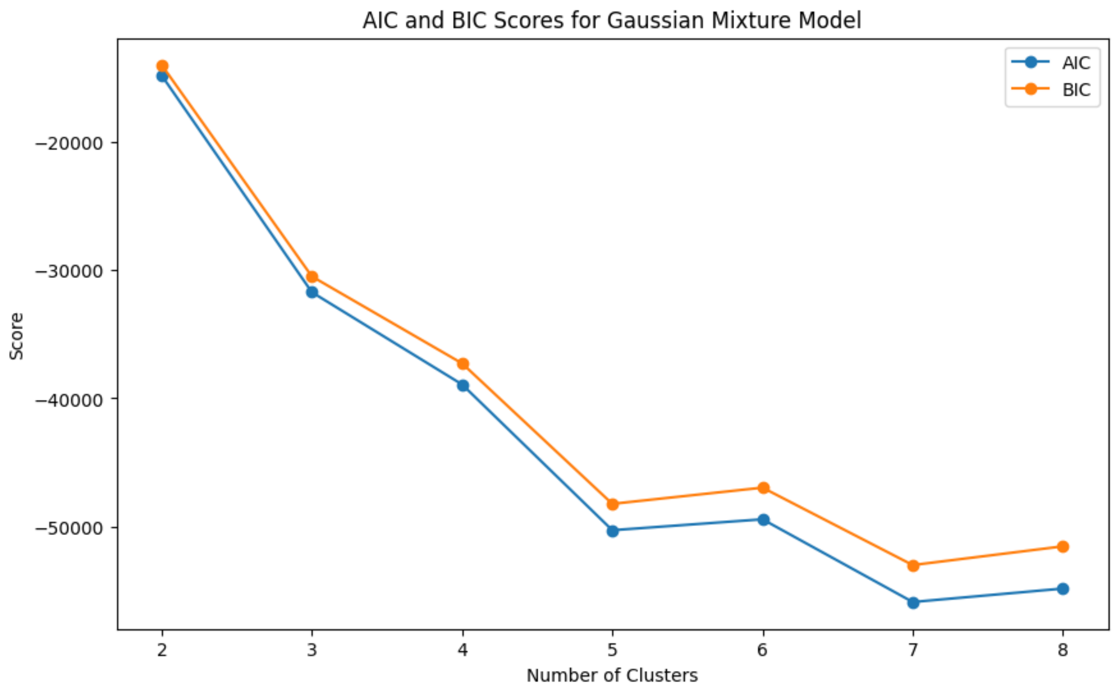
[ ]:
```
model = KMeans(n_clusters=4, n_init=10, random_state=0)
model.fit(df_attr)
labels = model.predict(df_attr)
```

[ ]:
```
plt.hist(labels, bins=range(5), color='black', edgecolor='black', alpha=0.7)
plt.xticks(range(4))
plt.grid(False)
plt.show()
```



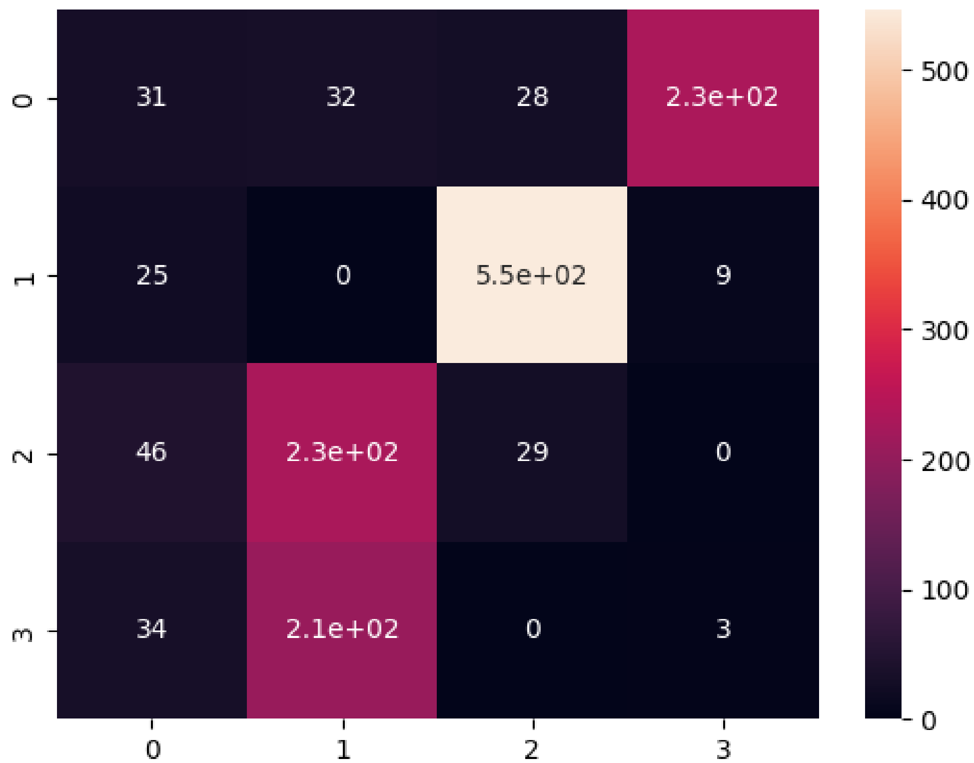
[ ]:
```
data = df_attr
scores = {'AIC': [], 'BIC': []}
models = []

for k in range(2, 9):
    model = GaussianMixture(n_components=k, n_init=10).fit(data)
    models.append(model)
    scores['AIC'].append(model.aic(data))
    scores['BIC'].append(model.bic(data))

results_df = pd.DataFrame(scores, index=range(2, 9))
print(results_df)

plt.figure(figsize=(10, 6))
for score_name in scores:
    plt.plot(range(2, 9), scores[score_name], marker='o', label=score_name)
plt.xlabel('Number of Clusters')
plt.ylabel('Score')
plt.title('AIC and BIC Scores for Gaussian Mixture Model')
plt.legend()
plt.show()
```

```
          AIC          BIC
2 -14834.648124 -14016.033346
3 -31728.350134 -30497.787275
4 -38915.066521 -37272.555579
5 -50274.623178 -48220.164155
6 -49422.845787 -46956.438682
7 -55868.524608 -52990.169421
8 -54818.919033 -51528.615764
```



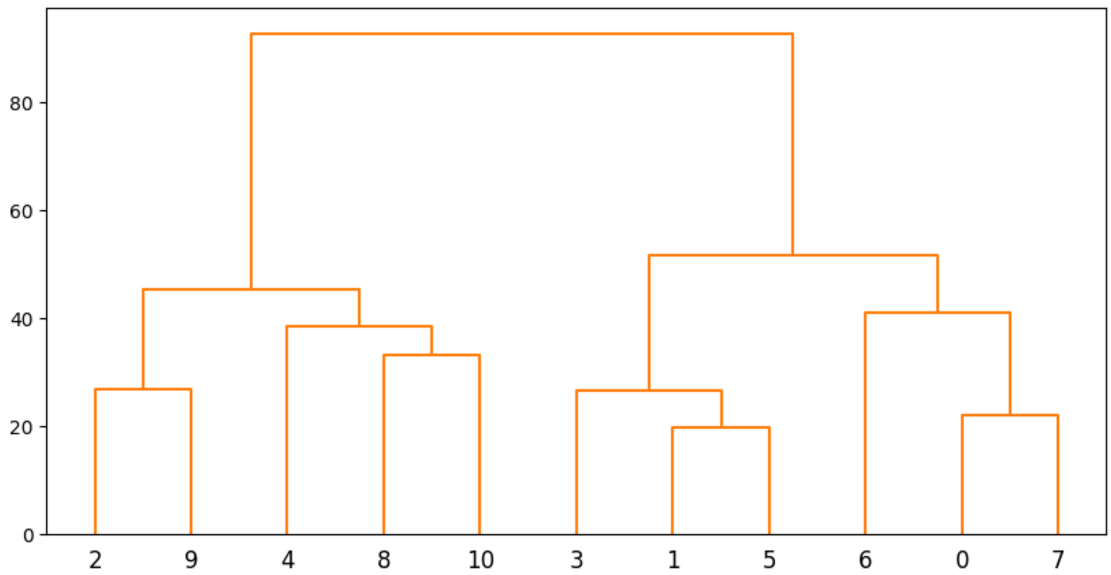
[ ]:
```
gmm = GaussianMixture(n_components=4, n_init=10, random_state=0).fit(df_attr)
cluster_assignments_gmm = gmm.predict(df_attr)

# KMeans Model
model = KMeans(n_clusters=4, n_init=10, random_state=0).fit(df_attr)
labels = model.labels_

# Confusion Matrix
conf_matrix = confusion_matrix(labels, cluster_assignments_gmm)

sns.heatmap(data=conf_matrix, annot = True)
```

[ ]: <Axes: >



[ ]:
```
distances = pairwise_distances(df_attr.T, metric='euclidean')
linkage_matrix = linkage(distances, method='ward')

plt.figure(figsize=(10, 5))
dendrogram(linkage_matrix, color_threshold=100)
plt.show()
```

```
<ipython-input-146-d0a9c6bc2b75>:2: ClusterWarning: scipy.cluster: The symmetric
non-negative hollow observation matrix looks suspiciously like an uncondensed
distance matrix
  linkage_matrix = linkage(distances, method='ward')
```



[ ]:
```
df_cluster = df_attr.copy()
df_cluster["cluster"]= labels
df_cluster["cluster"].value_counts()
```

[ ]:
```
1    580
0    323
2    304
3    246
Name: cluster, dtype: int64
```

[ ]:
```
data = {
    'Attribute': df_attr.columns,
    **{f'Cluster{i}': (df_cluster[df_cluster.cluster == i].
 ↪drop(columns="cluster").sum() / value).values
       for i, value in enumerate(df_cluster["cluster"].value_counts().
 ↪sort_index())}
}
cluster_data = pd.DataFrame(data)
```

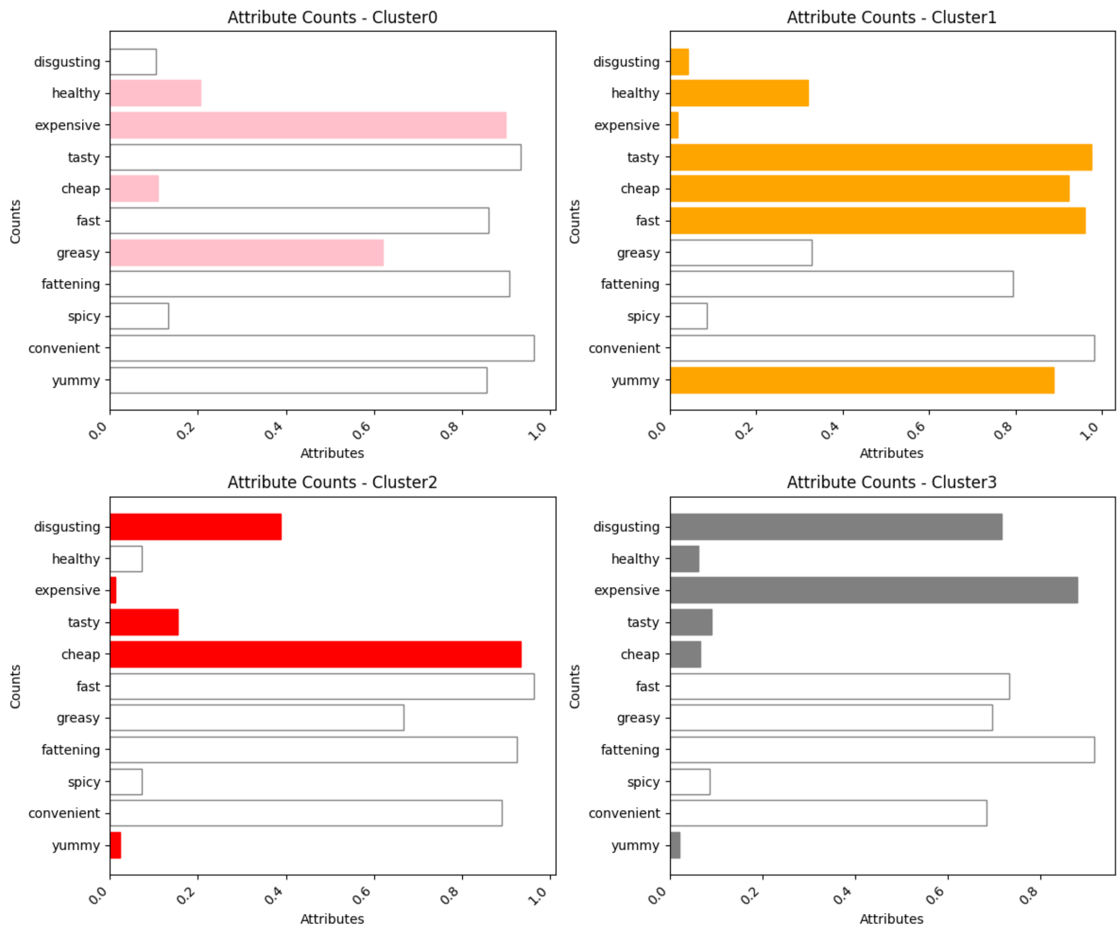
[ ]:
```
cluster_data
```

[ ]:

	Attribute	Cluster0	Cluster1	Cluster2	Cluster3
0	yummy	0.854489	0.887931	0.023026	0.020325
1	convenient	0.962848	0.981034	0.891447	0.682927
2	spicy	0.133127	0.086207	0.072368	0.085366
3	fattening	0.907121	0.794828	0.924342	0.914634
4	greasy	0.619195	0.329310	0.667763	0.695122

5	fast	0.860681	0.960345	0.963816	0.731707
6	cheap	0.108359	0.922414	0.934211	0.065041
7	tasty	0.931889	0.975862	0.154605	0.089431
8	expensive	0.897833	0.017241	0.013158	0.878049
9	healthy	0.204334	0.320690	0.072368	0.060976
10	disgusting	0.105263	0.043103	0.388158	0.715447

[ ]:
```python
df_data= pd.DataFrame(data)

clusters = ['Cluster0', 'Cluster1', 'Cluster2', 'Cluster3']

plt.figure(figsize=(12, 10))
for idx, cluster in enumerate(clusters, 1):
    if idx==1:
        highlighted_attributes = [6, 4, 8,9]
        color="pink"
    elif idx==2:
        highlighted_attributes = [0,5,6,7,8,9,10]
        color="orange"
    elif idx==3:
        highlighted_attributes = [0,6,7,8,10]
        color="red"
    else:
        highlighted_attributes = [0, 6,7,8,9,10]
        color="grey"
    plt.subplot(2, 2, idx)
    bars = plt.barh(df_data['Attribute'], df_data[cluster],edgecolor='Grey', facecolor='none',)
    for i in highlighted_attributes:
        bars[i].set_color(color)
    plt.grid(False)
    plt.xlabel('Attributes')
    plt.ylabel('Counts')
    plt.title(f'Attribute Counts - {cluster}')
    plt.xticks(rotation=45, ha='right')


plt.tight_layout()

plt.show()
```

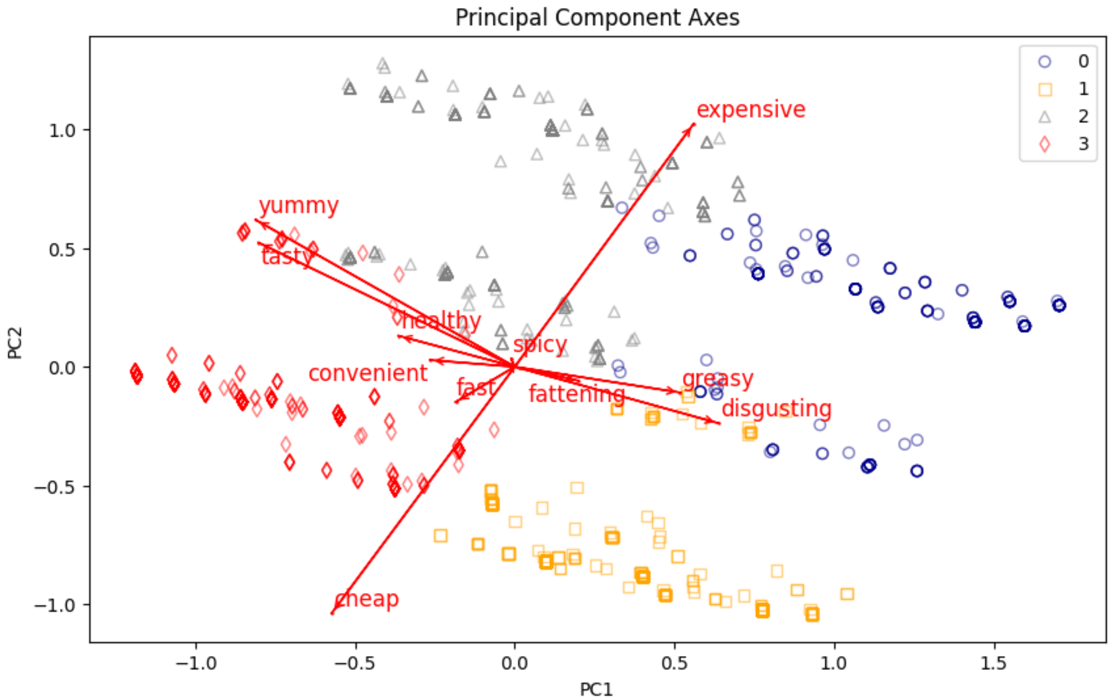


[ ]:
```
k_4 = KMeans(n_clusters=4, random_state=0)
k_4.fit(df_decomposed)
l_4 = k_4.labels_

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

markers = ['o', 's', '^', 'd']
edge_colors = ['darkblue', 'orange', 'grey', 'red']

for i in range(4):
    temp = df_decomposed[l_4 == i]
    plt.scatter(
        temp['yummy'],
        temp['convenient'],
        label=i,
        edgecolor=edge_colors[i],
        marker=markers[i],
        facecolor='none',
```

```python
        alpha=0.5
    )

scale = 1.7

texts = []
for i, j in enumerate(df_decomposed.columns):
    text = plt.text(
        p.components_[0, i] * scale,
        p.components_[1, i] * scale,
        j,
        color='red',
        fontsize=12,
        ha='center',
        va='bottom',
        stretch="expanded"
    )
    texts.append(text)
    plt.arrow(0, 0, p.components_[0, i] * scale, p.components_[1, i] * scale,␣
 ↪color='red')
    plt.annotate(
        '', xytext=(0, 0),
        xy=(p.components_[0, i] * scale, p.components_[1, i] * scale),
        arrowprops=dict(
            arrowstyle="->",
            color='red'
        )
    )
adjust_text(texts)

plt.scatter(
    p.components_[0, :] * scale,
    p.components_[1, :] * scale,
    s=0.5,
    color='red'
)

plt.grid(False)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('Principal Component Axes')
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
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
```

```
warnings.warn(
```



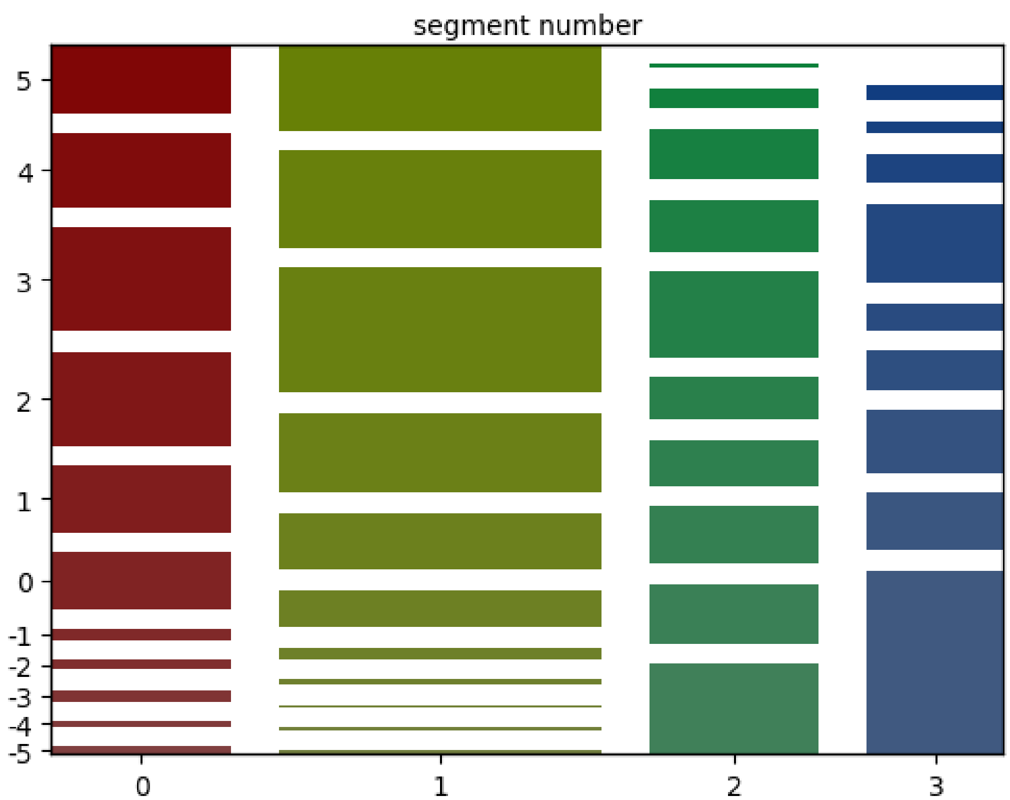
[ ]:
```python
l = df['Like'].replace({
    f'I love it!+{i}': i for i in range(5, 0, -1)
    }).replace({
    f'I hate it!-{i}': -i for i in range(5, 0, -1)
    }).astype(int)
```

[ ]:
```python
l
```

[ ]:
```
0       -3
1        2
2        1
3        4
4        2
        ..
1448    -5
1449     2
1450     3
1451     4
1452    -3
Name: Like, Length: 1453, dtype: int64
```

[ ]:
```
data = {
    'segment': labels,
    'lovehate': l
}
ds = pd.DataFrame(data)
crosstab = pd.crosstab(ds['segment'], ds['lovehate'])
plt.figure(figsize=(10, 6))
mosaic_data = crosstab.stack()
mosaic(mosaic_data, title='', axes_label=True, gap=0.06, labelizer=lambda k: '')
plt.grid(False)
plt.title('')
plt.xlabel('segment number')
plt.ylabel('Count')
plt.show()
```

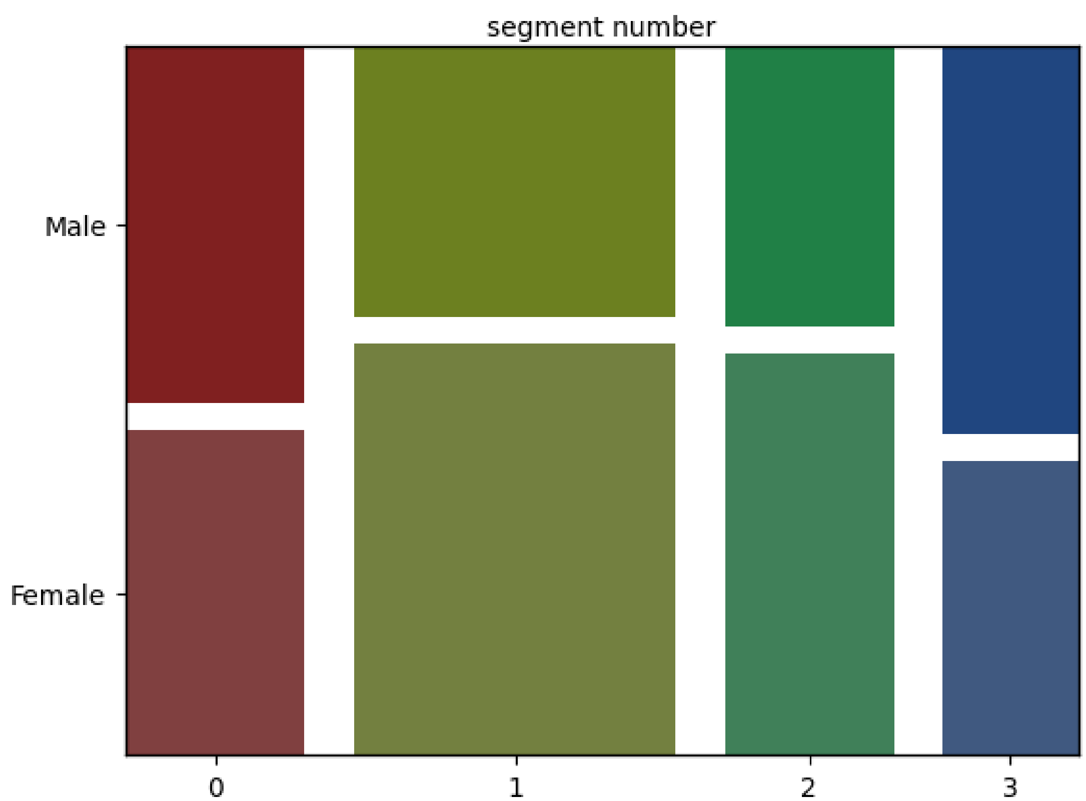
```
<Figure size 1000x600 with 0 Axes>
```



[ ]:
```
data = {
    'segment': labels,
    'lovehate': df.Gender
}
```

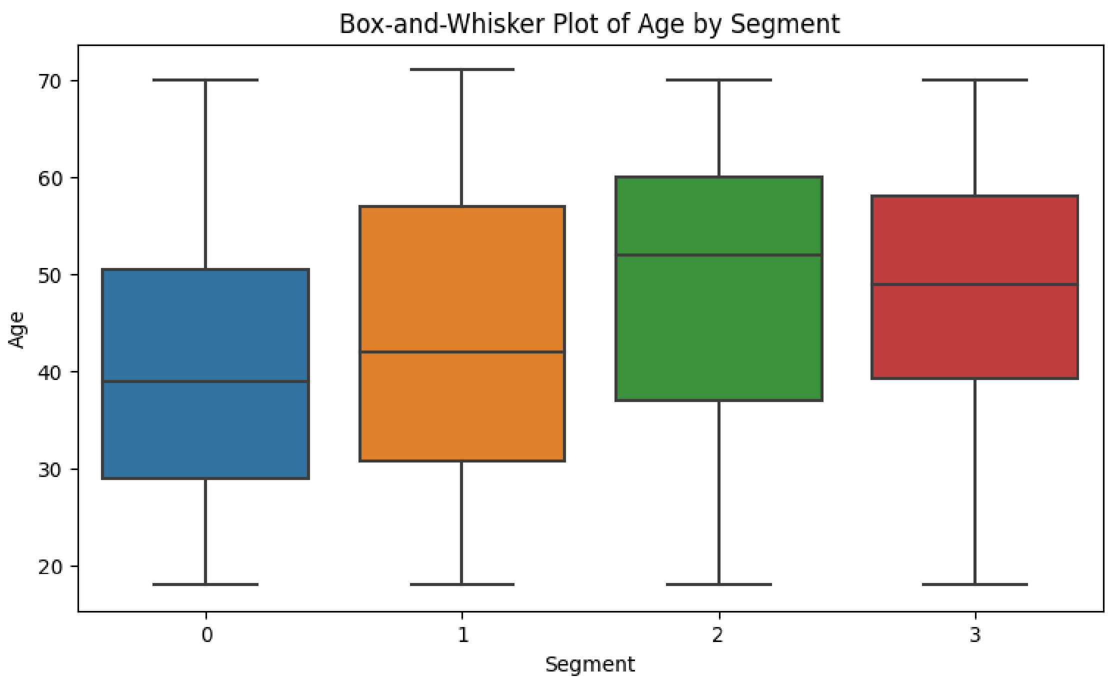
```
ds = pd.DataFrame(data)
crosstab = pd.crosstab(ds['segment'], ds['lovehate'])
plt.figure(figsize=(10, 6))
mosaic_data = crosstab.stack()
mosaic(mosaic_data, title='', axes_label=True, gap=0.06, labelizer=lambda k: '')
plt.title('')
plt.xlabel('segment number')
plt.ylabel('Count')
plt.grid(False)
plt.show()
```

<Figure size 1000x600 with 0 Axes>



[ ]:
```
data = {
    'segment': labels,
    'Age': df.Age
}
ds = pd.DataFrame(data)

plt.figure(figsize=(9, 5))
sns.boxplot(x='segment', y='Age', data=ds)
```

```
plt.xlabel('Segment')
plt.ylabel('Age')
plt.title('Box-and-Whisker Plot of Age by Segment')

plt.show()
```



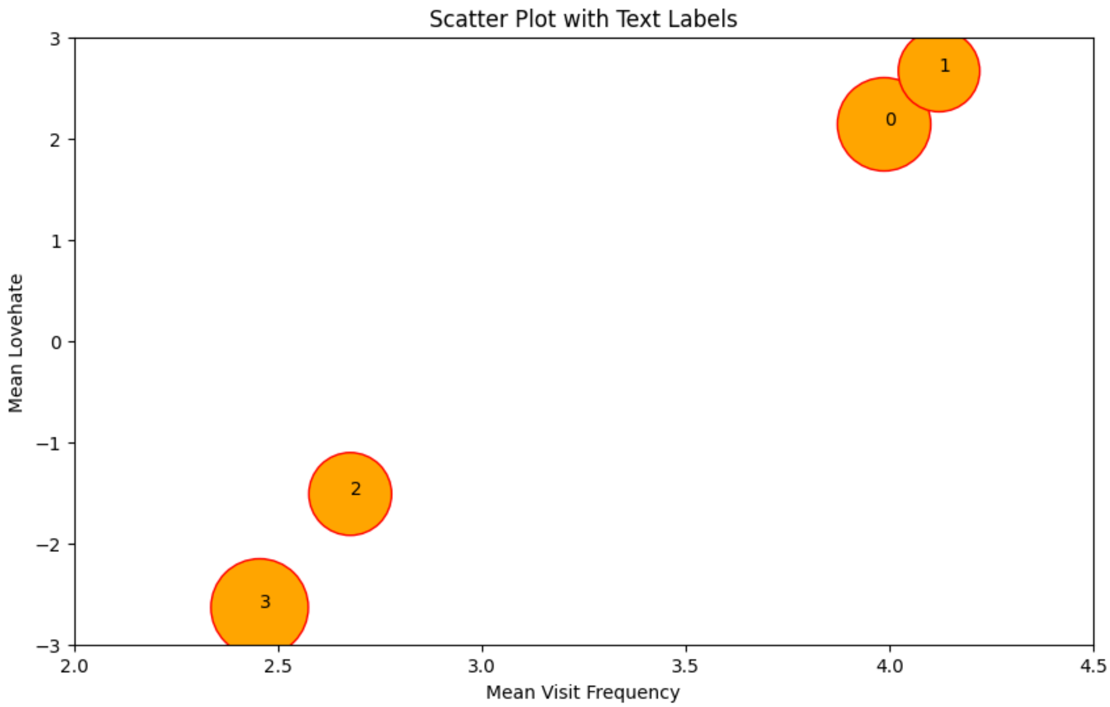
[ ]:

```
data = {
    'segment': labels,
    'lovehate': l,
    'Age': df.Age,
    'VisitFrequency': df.VisitFrequency,
    'Gender': df.Gender
}
ds = pd.DataFrame(data)

visit_frequency_mapping = {
    'Never': 1,
    'Once a year': 2,
    'Every three months': 3,
    'Once a month': 4,
    'More than once a week': 5,
    'Once a week': 6
}
gender_mapping = {
    "Female":0,
```

```
    "Male":1
}
ds['VisitFrequencyNumeric'] = ds['VisitFrequency'].map(visit_frequency_mapping)
ds["GenderNumeric"] = ds["Gender"].map(gender_mapping)
visit = ds.groupby('segment')['VisitFrequencyNumeric'].mean()
like = ds.groupby('segment')['lovehate'].mean()
female = ds.groupby('segment')["GenderNumeric"].mean()
print(visit)
print(like)
print(female)
```

```
segment
0    3.987616
1    4.122414
2    2.677632
3    2.455285
Name: VisitFrequencyNumeric, dtype: float64
segment
0    2.139319
1    2.665517
2   -1.513158
3   -2.634146
Name: lovehate, dtype: float64
segment
0    0.523220
1    0.398276
2    0.411184
3    0.569106
Name: GenderNumeric, dtype: float64
```

[ ]:
```
plt.figure(figsize=(10, 6))
plt.scatter(visit, like, s=5000 * female, edgecolor="red", facecolor="orange")

for i, segment in enumerate(visit.index):
    plt.text(visit[segment], like[segment], str(i))

plt.xlim(2, 4.5)
plt.ylim(-3, 3)
plt.xlabel('Mean Visit Frequency')
plt.ylabel('Mean Lovehate')
plt.title('Scatter Plot with Text Labels')

plt.grid(False)
plt.show()
```


Scatter Plot with Text Labels
Mean Lovehate
Mean Visit Frequency
3
2
1
0
−1
−2
−3
2.0
2.5
3.0
3.5
4.0
4.5
0
1
2
3

```
[ ]:
```