Name- Deepali Malik
Team- Pelluru Kushal Kumar

Code Conversion

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.mixture import GaussianMixture
from sklearn.ensemble import BaggingClassifier
from sklearn.metrics import silhouette_score, adjusted_rand_score
from sklearn.utils import resample
from scipy.cluster.hierarchy import dendrogram, linkage
from scipy.spatial.distance import pdist, squareform

print("\n1. Converting Ordinal Data to Numeric")
data = {'Like': ["I love it!+5", "+4", "+3", "0", "-1", "-2", "-3", "-4", "I hate it!-5"]}
df = pd.DataFrame(data)
like_mapping = {"I love it!+5": 5, "+4": 4, "+3": 3, "0": 0, "-1": -1, "-2": -2, "-3": -3, "-4": -4,
"I hate it!-5": -5}
df['Like_n'] = df['Like'].map(like_mapping)
print(df)
```

output:

```
1. Converting Ordinal Data to Numeric
           Like  Like_n
0  I love it!+5       5
1            +4       4
2            +3       3
3             0       0
4            -1      -1
5            -2      -2
6            -3      -3
7            -4      -4
8  I hate it!-5      -5
```

```
print("\n2. Creating Regression Formula Dynamically")
independent_vars = ["yummy", "convenient", "spicy", "fattening", "greasy",
                    "fast", "cheap", "tasty", "expensive", "healthy", "disgusting"]
dependent_var = "Like_n"
formula = f"{dependent_var} ~ " + " + ".join(independent_vars)
print("Regression Formula:", formula)
```

output:

```
2. Creating Regression Formula Dynamically
Regression Formula: Like_n ~ yummy + convenient + spicy + fattening + greasy + fast + cheap + tasty + expensive + healthy + disgusting

3. Mixtures of Regression Models (Latent Class Regression)
```

```
print("\n3. Mixtures of Regression Models (Latent Class Regression)")
np.random.seed(1234)
X = np.random.rand(100, len(independent_vars))
y = np.random.randint(-5, 6, 100)
gmm = GaussianMixture(n_components=2, random_state=1234)
gmm.fit(X)
clusters = gmm.predict(X)
print("Cluster Assignments:", clusters[:10])
print("Cluster Means:\n", gmm.means_)
```

Output:

```
3. Mixtures of Regression Models (Latent Class Regression)
Cluster Assignments: [0 0 0 1 0 0 1 0 0 0]
Cluster Means:
 [[0.52526462 0.47117622 0.49504997 0.75393923 0.5705316  0.58527726
  0.48365815 0.52497178 0.53276574 0.50128376 0.53450978]
 [0.52890113 0.6360627  0.52786452 0.21469533 0.48498877 0.40827456
  0.40549155 0.4552162  0.38973742 0.51362206 0.46534705]]
```

```
print("\n4. Data Transformation: Convert Yes/No to Numeric")
df_yes_no = pd.DataFrame({'feature1': ['Yes', 'No', 'Yes', 'No', 'Yes'],
                          'feature2': ['No', 'Yes', 'Yes', 'No', 'No'],
                          'feature3': ['Yes', 'Yes', 'No', 'No', 'Yes']})
df_numeric = df_yes_no.apply(lambda col: col.map(lambda x: 1 if x == "Yes" else 0))
print(df_numeric)
```

Output:

```
4. Data Transformation: Convert Yes/No to Numeric
   feature1  feature2  feature3
0         1         0         1
1         0         1         1
2         1         1         0
3         0         0         0
4         1         0         1
```

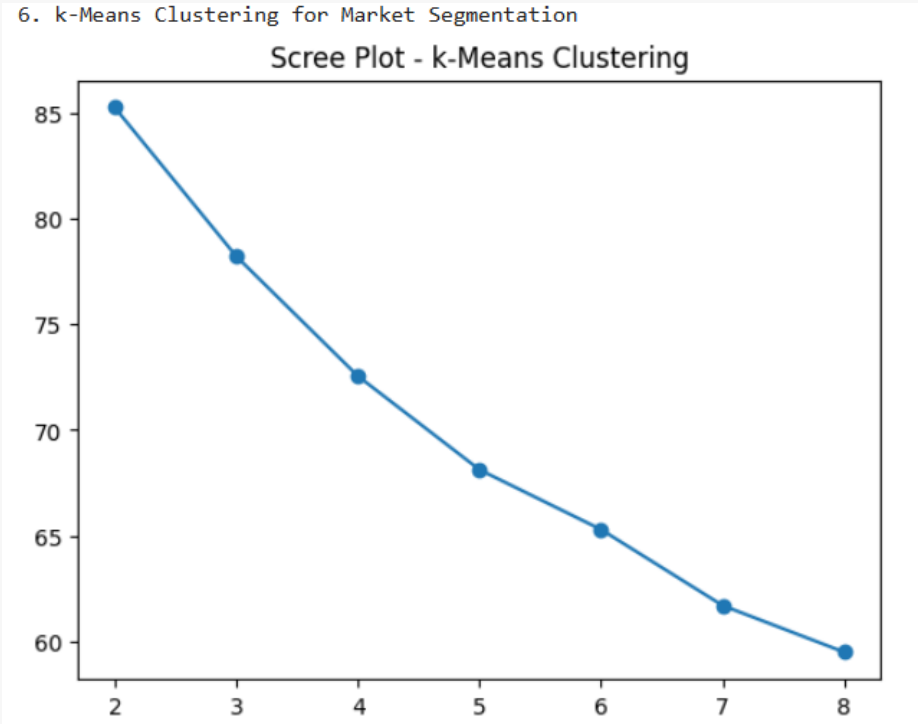
```
#print("\n5. PCA Explained Variance")
pca = PCA()
X_pca = pca.fit_transform(X)
print("\n5. PCA Explained Variance:", pca.explained_variance_ratio_.round(4))
```

Output:

```
5. PCA Explained Variance: [0.1458 0.1334 0.119  0.11   0.0968 0.0835 0.0765 0.0701 0.064  0.0556
 0.0453]
```

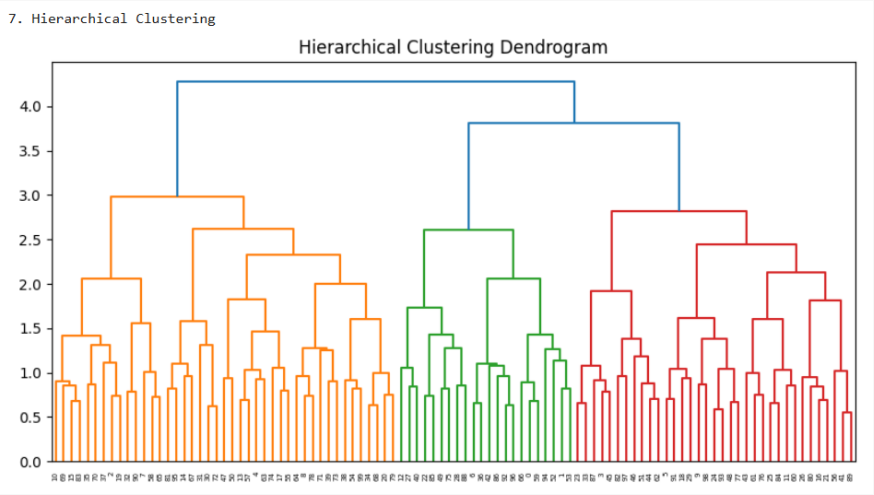
```
print("\n6. k-Means Clustering for Market Segmentation")
distortions = [KMeans(n_clusters=k, random_state=1234, n_init=10).fit(X).inertia_ for k in
range(2, 9)]
plt.figure()
plt.plot(range(2, 9), distortions, marker='o')
plt.title("Scree Plot - k-Means Clustering")
plt.show()
```

Output:



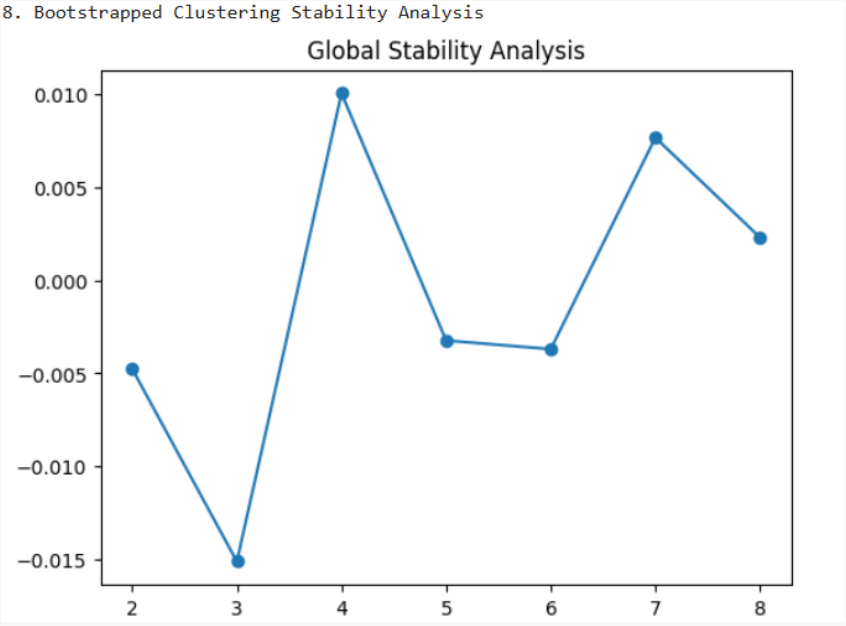

```
print("\n7. Hierarchical Clustering")
Z = linkage(X, method='ward')
plt.figure(figsize=(10, 5))
dendrogram(Z)
plt.title("Hierarchical Clustering Dendrogram")
plt.show()
```

Output:




```
print("\n8. Bootstrapped Clustering Stability Analysis")
stability_scores = [np.mean([adjusted_rand_score(
    KMeans(n_clusters=k, random_state=1234, n_init=10).fit(resample(X, y,
random_state=1234)[0]).labels_,
    KMeans(n_clusters=k, random_state=1234).fit(X).labels_)
    for _ in range(100)]) for k in range(2, 9)]
plt.figure()
plt.plot(range(2, 9), stability_scores, marker='o')
plt.title("Global Stability Analysis")
plt.show()
```

Output:



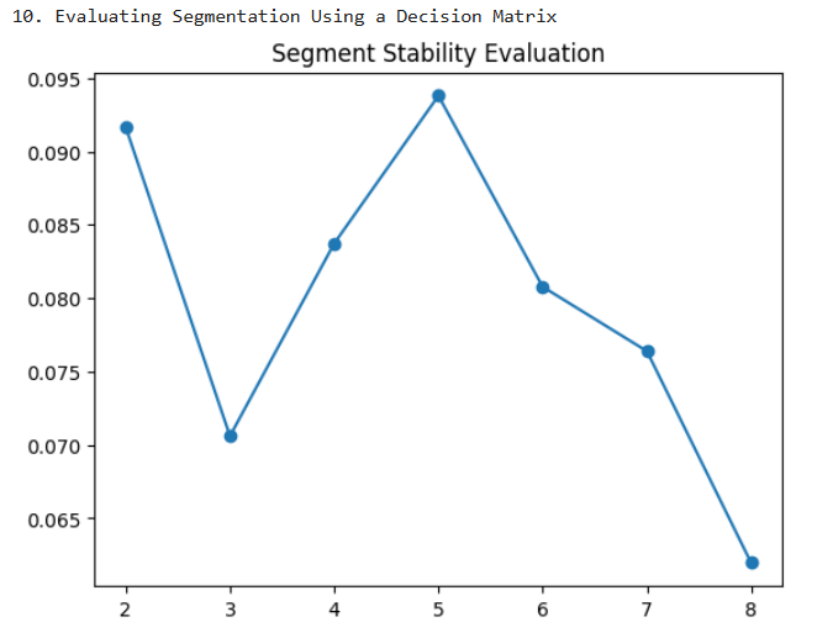
```
print("\n9. Two-Step Clustering")
agglo_cluster = AgglomerativeClustering(n_clusters=3, linkage="ward").fit_predict(X)
print(pd.Series(agglo_cluster).value_counts().rename("count"))
```

Output:

```
9. Two-Step Clustering
0    43
1    35
2    22
Name: count, dtype: int64
```

```
print("\n10. Evaluating Segmentation Using a Decision Matrix")
silhouette_scores = [silhouette_score(X, KMeans(n_clusters=k,
random_state=1234).fit_predict(X)) for k in range(2, 9)]
plt.figure()
plt.plot(range(2, 9), silhouette_scores, marker="o")
plt.title("Segment Stability Evaluation")
plt.show()
```

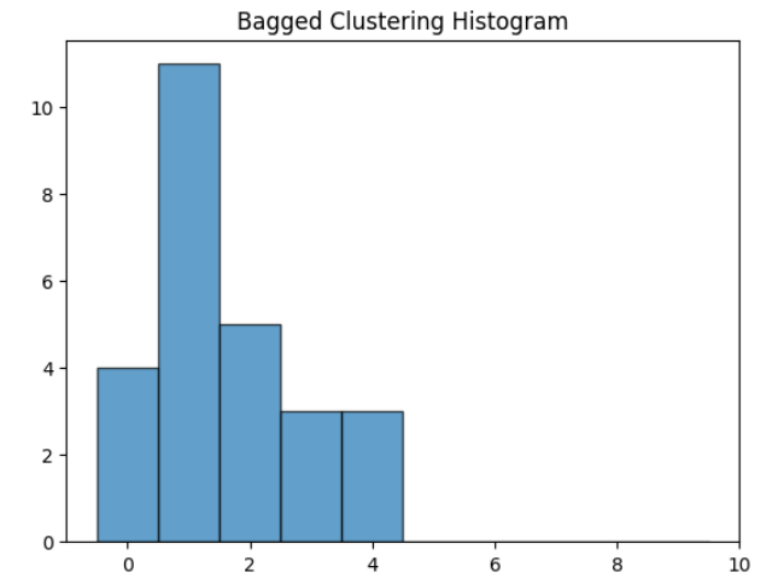
Output:




```
print("\n11. Bagged Clustering")
bagging_kmeans = BaggingClassifier(estimator=KMeans(n_clusters=10), n_estimators=50,
random_state=1234).fit(X, y)
plt.figure()
plt.hist(bagging_kmeans.predict(X), bins=np.arange(11)-0.5, edgecolor="black", alpha=0.7)
plt.title("Bagged Clustering Histogram")
plt.show()
```

Output:

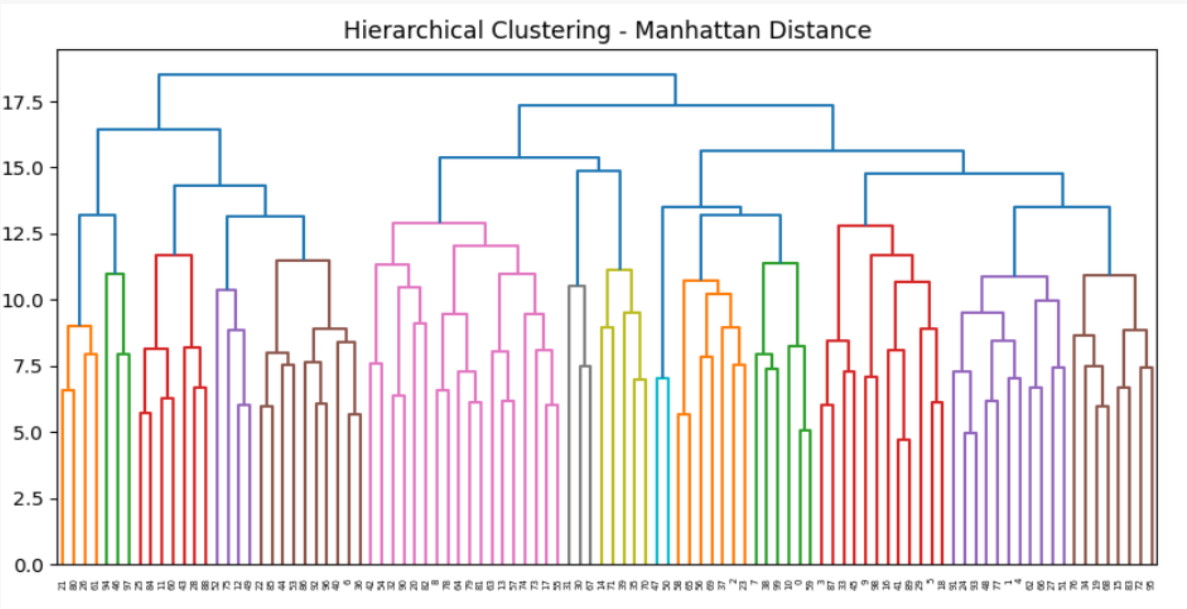
```
11. Bagged Clustering
```




```
print("\n12. Distance-Based Clustering")
risk_dist = squareform(pdist(X, metric="cityblock"))
Z = linkage(risk_dist, method="complete")
plt.figure(figsize=(10, 5))
dendrogram(Z)
plt.title("Hierarchical Clustering - Manhattan Distance")
plt.show()
```

Output:

```
12. Distance-Based Clustering
<ipython-input-9-f1d983b3cdec>:104: ClusterWarning: The symmetric non-negative hollow observation matrix looks suspiciously like an uncondensed
  Z = linkage(risk_dist, method="complete")
```



```
print("\n13. k-Means Clustering with Mixture Models")
gmm = GaussianMixture(n_components=4, random_state=1234).fit(X)
print("Cluster Assignments (Mixture Model):", np.bincount(gmm.predict(X)))
print("Log Likelihood:", gmm.score(X))
```

Output:

```
13. k-Means Clustering with Mixture Models
Cluster Assignments (Mixture Model): [22 24 33 21]
Log Likelihood: -0.06003475701589502
```

```
print("\n14. Mixture of Normal Distributions")
subset_data = X[:, :2]
counts = np.apply_along_axis(lambda x: np.bincount(x.astype(int), minlength=2), axis=0,
arr=subset_data)
print("Segment Counts:\n", counts)
print("Segment Probabilities:", np.round(counts.prod() * 100, 2)
```

Output:

```
14. Mixture of Normal Distributions
Segment Counts:
 [[100 100]
 [  0   0]]
Segment Probabilities: 0
```

```
print("\n15. Market Segmentation on Binary Data")
binary_data = (X > np.median(X)).astype(int)
counts = binary_data.mean(axis=0)
print("Feature Probabilities:", np.round(counts, 2))
print("Overall Probability:", np.round(np.prod(counts) * 100, 2))
```

Output:

```
15. Market Segmentation on Binary Data
Feature Probabilities: [0.53 0.53 0.49 0.52 0.59 0.46 0.42 0.49 0.46 0.51 0.5 ]
Overall Probability: 0.05
```