In [11]:
```python
#importing libraries and packages
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

First we explore the key characteristics of the data set by loading the data set and inspecting basic features such as the variable names, the sample size.

In [2]:
```python
df=pd.read_csv("https://homepage.boku.ac.at/leisch/MSA/datasets/mcdonalds.csv")
```

In [3]:
```python
df.head()
```

Out[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
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

In [8]:
```python
df.columns
```

Out[8]:
```
Index(['yummy', 'convenient', 'spicy', 'fattening', 'greasy', 'fast', 'cheap',
       'tasty', 'expensive', 'healthy', 'disgusting', 'Like', 'Age',
       'VisitFrequency', 'Gender'],
      dtype='object')
```

In [9]:
```python
df.shape
```

Out[9]:
```
(1453, 15)
```

In [10]:
```python
df.head(3)
```

Out[10]:

	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

In [13]:
```python
matx = np.array(df.iloc[:, 0:11])
matx
```

Out[13]:
```
array([['No', 'Yes', 'No', ..., 'Yes', 'No', 'No'],
       ['Yes', 'Yes', 'No', ..., 'Yes', 'No', 'No'],
       ['No', 'Yes', 'Yes', ..., 'Yes', 'Yes', 'No'],
       ...,
       ['Yes', 'Yes', 'No', ..., 'Yes', 'No', 'No'],
       ['Yes', 'Yes', 'No', ..., 'No', 'Yes', 'No'],
       ['No', 'Yes', 'No', ..., 'Yes', 'No', 'Yes']], dtype=object)
```

In [14]:
```python
matx = np.where(matx == "Yes", 1, 0)
col_means = np.mean(matx, axis=0)

# Round the calculated means to two decimal places
rounded_col_means = np.round(col_means, 2)

print(rounded_col_means)
```

```
[0.55 0.91 0.09 0.87 0.53 0.9  0.6  0.64 0.36 0.2  0.24]
```

In [15]:
```python
list(df.columns)
```

Out[15]:
```
['yummy',
 'convenient',
 'spicy',
 'fattening',
 'greasy',
 'fast',
 'cheap',
 'tasty',
 'expensive',
 'healthy',
 'disgusting',
 'Like',
 'Age',
 'VisitFrequency',
 'Gender']
```

In [16]:
```python
mapped_dict = {key: value for key, value in zip(df.columns, rounded_col_means)}

print(mapped_dict)
```

```
{'yummy': 0.55, 'convenient': 0.91, 'spicy': 0.09, 'fattening': 0.87, 'greasy': 0.53, 'fast': 0.9, 'cheap': 0.6, 'tasty': 0.64, 'expensive': 0.36, 'healthy': 0.2, 'disg
usting': 0.24}
```

principal components analysis

In [18]:
```python
pca = PCA()
MD_pca = pca.fit_transform(matx)

# The transformed data is now stored in MD_pca

MD_pca
```

Out[18]:
```
array([[ 0.42536706, -0.21907878,  0.6632553 , ...,  0.18100693,
         0.51570617, -0.56707389],
       [-0.21863768,  0.38818996, -0.73082668, ...,  0.11147641,
         0.49331285, -0.50044033],
       [ 0.37541475,  0.73043507, -0.12203978, ..., -0.32228786,
         0.06175857,  0.24274108],
       ...,
       [-0.18589445,  1.06266156,  0.22085675, ...,  0.03825472,
         0.05651822, -0.01279977],
       [-1.18206441, -0.03856977,  0.56156066, ...,  0.02226748,
        -0.00257265, -0.10531631],
       [ 1.55024186,  0.27503101, -0.01373727, ..., -0.13658866,
        -0.43279782, -0.45607556]])
```

In [21]:
```python
pca = PCA()
MD_pca = pca.fit_transform(matx)

def print_pca_summary(pca_result, num_digits=1):
    fmt_str = "{:." + str(num_digits) + "f}"
    print("Standard deviations of principal components:")
    print([fmt_str.format(val) for val in np.sqrt(pca_result.explained_variance_)])

    print("\nProportion of variance explained by each principal component:")
    print([fmt_str.format(val) for val in pca_result.explained_variance_ratio_])

    print("\nCumulative proportion of variance explained:")
    cumulative_variance_explained = np.cumsum(pca_result.explained_variance_ratio_)
    print([fmt_str.format(val) for val in cumulative_variance_explained])

print_pca_summary(pca, num_digits=1)
```

```
Standard deviations of principal components:
['0.8', '0.6', '0.5', '0.4', '0.3', '0.3', '0.3', '0.3', '0.3', '0.2', '0.2']

Proportion of variance explained by each principal component:
['0.3', '0.2', '0.1', '0.1', '0.1', '0.1', '0.0', '0.0', '0.0', '0.0', '0.0']

Cumulative proportion of variance explained:
['0.3', '0.5', '0.6', '0.7', '0.8', '0.8', '0.9', '0.9', '0.9', '1.0', '1.0']
```

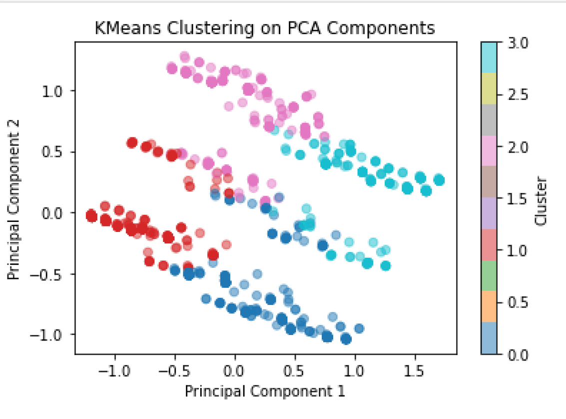
In [26]:
```python
data=matx

# Step 1: Standardization
data_std = (data - np.mean(data, axis=0)) / np.std(data, axis=0)

# Step 2 and 3: PCA and Eigendecomposition
pca = PCA()
pca.fit(data)

# Step 4: Explained variance ratio
explained_variance_ratio = pca.explained_variance_ratio_

# Step 5: Formation of Principal Components
principal_components = pca.transform(data)

# Display the results
print("Explained Variance Ratio:")
print(explained_variance_ratio)

print("\nPrincipal Components:")
print(principal_components)
```

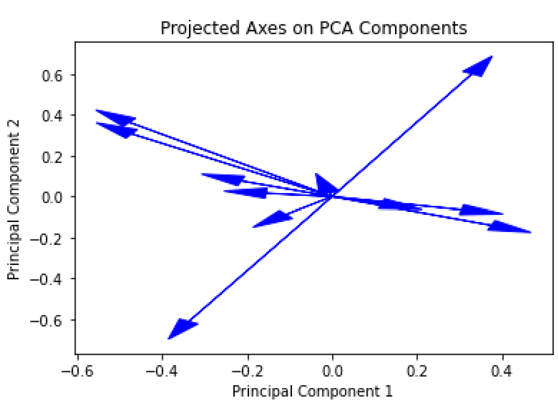
```
Explained Variance Ratio:
[0.29944723 0.19279721 0.13304535 0.08309578 0.05948052 0.05029956
 0.0438491  0.03954779 0.0367609  0.03235329 0.02932326]

Principal Components:
[[ 0.42536706 -0.21907878  0.6632553  ...  0.18100693  0.51570617
  -0.56707389]
 [-0.21863768  0.38818996 -0.73082668 ...  0.11147641  0.49331285
  -0.50044033]
 [ 0.37541475  0.73043507 -0.12203978 ... -0.32228786  0.06175857
   0.24274108]
 ...
 [-0.18589445  1.06266156  0.22085675 ...  0.03825472  0.05651822
  -0.01279977]
 [-1.18206441 -0.03856977  0.56156066 ...  0.02226748 -0.00257265
  -0.10531631]
 [ 1.55024186  0.27503101 -0.01373727 ... -0.13658866 -0.43279782
  -0.45607556]]
```

Extracting Segments

Using k-Means

In [27]:
```python
# Performing KMeans clustering on the principal components
num_clusters = 4
kmeans = KMeans(n_clusters=num_clusters)
cluster_labels = kmeans.fit_predict(MD_pca)

# Plot the predicted clusters
plt.scatter(MD_pca[:, 0], MD_pca[:, 1], c=cluster_labels, cmap='tab10', alpha=0.5)
plt.colorbar(label='Cluster')
plt.title('KMeans Clustering on PCA Components')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()


def plot_projected_axes(pca_result):

    loadings = pca_result.components_.T


    for i in range(len(loadings)):
        x0, y0 = 0, 0  # Origin
        x1, y1 = loadings[i, 0], loadings[i, 1]
        plt.arrow(x0, y0, x1, y1, head_width=0.05, head_length=0.1, fc='blue', ec='blue')

    plt.title('Projected Axes on PCA Components')
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.show()

plot_projected_axes(pca)
```

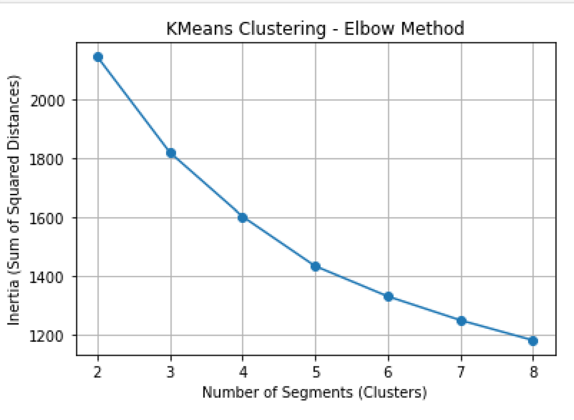




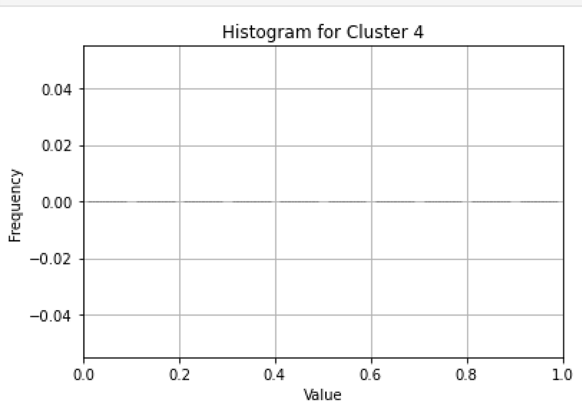
In [28]:
```python
np.random.seed(1234)
n_clusters_range = range(2, 9)
best_kmeans = None
best_score = float('-inf')

for n_clusters in n_clusters_range:
    kmeans = KMeans(n_clusters=n_clusters, n_init=10, random_state=1234)
    kmeans.fit(matx)
    score = kmeans.score(matx)

    # Select the best model based on the highest score (inertia)
    if score > best_score:
        best_kmeans = kmeans
        best_score = score

# Retrieve the labels from the best clustering model
cluster_labels = best_kmeans.labels_

# Print the cluster labels
print("Cluster Labels:")
print(cluster_labels)
```

```
Cluster Labels:
[4 2 5 ... 5 3 0]
```

In [29]:
```python
np.random.seed(1234)
n_clusters_range = range(2, 9)
inertia_values = []

for n_clusters in n_clusters_range:
    kmeans = KMeans(n_clusters=n_clusters, n_init=10, random_state=1234)
    kmeans.fit(matx)
    inertia_values.append(kmeans.inertia_)

plt.plot(n_clusters_range, inertia_values, marker='o')
plt.xlabel('Number of Segments (Clusters)')
plt.ylabel('Inertia (Sum of Squared Distances)')
plt.title('KMeans Clustering - Elbow Method')
plt.grid(True)
plt.show()
```



In [31]:
```python
cluster_number = 4
indices_cluster_4 = np.where(cluster_labels == cluster_number)[0]
data_cluster_4 = matx[indices_cluster_4]
plt.hist(data_cluster_4, bins=np.linspace(0, 1, 11), edgecolor='black')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram for Cluster 4')
plt.xlim(0, 1)
plt.grid(True)
plt.show()
```



Using Mixtures of Distributions

In [ ]:
```python
from sklearn.mixture import BayesianGaussianMixture

np.random.seed(1234)

n_components_range = range(2, 9)
aic_values = []
bic_values = []
icl_values = []

for n_components in n_components_range:
    gmm = BayesianGaussianMixture(n_components=n_components, n_init=10, random_state=1234)
    gmm.fit(matx)

    aic_values.append(gmm.aic(matx))
    bic_values.append(gmm.bic(matx))
    icl_values.append(gmm.lower_bound_)

for n_components, aic, bic, icl in zip(n_components_range, aic_values, bic_values, icl_values):
    print(f"Number of Components: {n_components}, AIC: {aic}, BIC: {bic}, ICL: {icl}")

# Plot the information criteria (AIC, BIC, ICL)
plt.plot(n_components_range, aic_values, marker='o', label='AIC')
plt.plot(n_components_range, bic_values, marker='o', label='BIC')
plt.plot(n_components_range, icl_values, marker='o', label='ICL')
plt.xlabel('Number of Clusters')
plt.ylabel('Value of Information Criteria')
plt.title('Model Selection - Bayesian Gaussian Mixture')
plt.legend()
plt.grid(True)
plt.show()
```

Using Mixtures of Regression Models

In [ ]:
```python
import statsmodels.api as sm

columns_for_regression = df.columns[1:12]
formula_string = "Like.n ~ " + " + ".join(columns_for_regression)
formula = sm.formula.ols(formula_string, data=df)

# Fit the linear regression model
model = sm.OLS.from_formula(formula, data=df)
results = model.fit()
print(results.summary())

# Plot
plt.figure()
plt.scatter(results.fittedvalues, results.resid, alpha=0.5)
plt.xlabel("Fitted Values")
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.axhline(y=0, color='r', linestyle='--')
plt.grid(True)
plt.show()
```