In [2]:

```
#importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [4]:

```
#reading file
df = pd.read_csv("mcdonalds.csv")
df.head()
```

Out[4]:

	yummy	convenient	spicy	fattening	greasy	fast	cheap	tasty	expensive	healthy	disgusting	Like	Age	VisitFrequency
0	No	Yes	No	Yes	No	Yes	Yes	No	Yes	No	No	-3	61	Every three months
1	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	No	No	+2	51	Every three months
2	No	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	+1	62	Every three months
3	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No	No	Yes	+4	69	Once a week
4	No	Yes	No	Yes	Yes	Yes	Yes	No	No	Yes	No	+2	49	Once a month

In [5]:

```
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
```

In [6]:

```
df.shape
```

Out[6]:

```
(1453, 15)
```

In [7]:

```
df.describe()
```

Out[7]:

	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 [8]:

```
df.isna().sum()
```

Out[8]:

```
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 [9]:

```
#variable counts
df['Gender'].value_counts()
```

Out[9]:

```
Gender
Female    788
Male      665
Name: count, dtype: int64
```

In [10]:

```
df['VisitFrequency'].value_counts()
```

Out[10]:

```
VisitFrequency
Once a month             439
Every three months       342
Once a year              252
Once a week              235
Never                    131
More than once a week     54
Name: count, dtype: int64
```

In [11]:

```
df['Like'].value_counts()
```

Out[11]:

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

Step1: Deciding not to segment

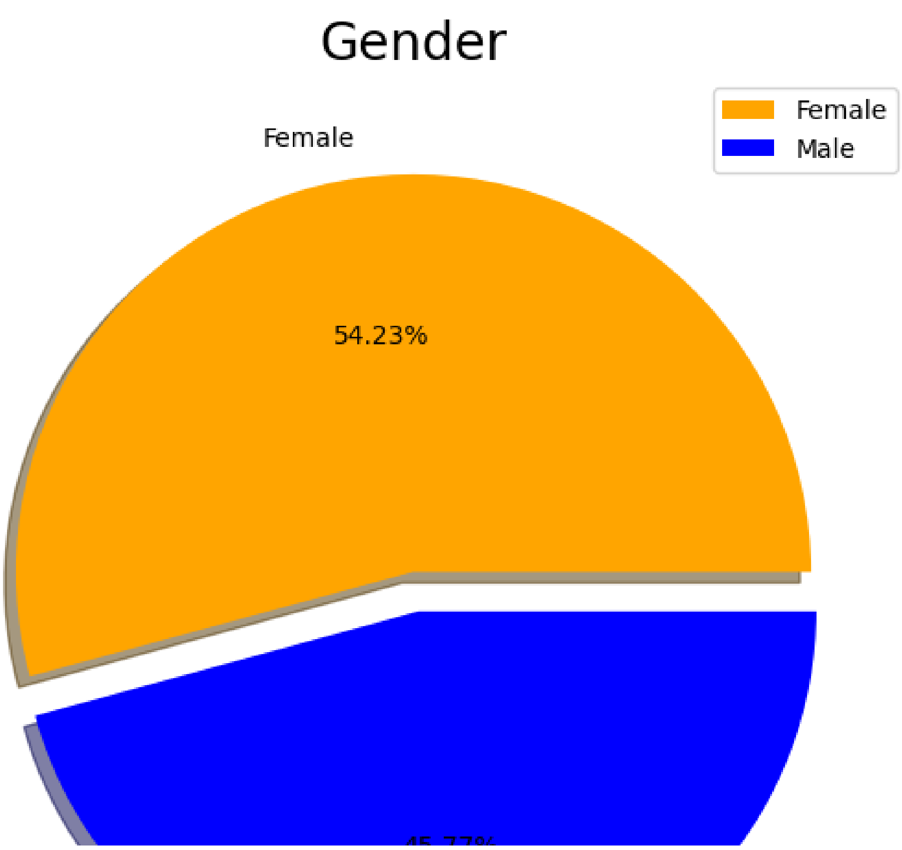
Step2: Specifying ideal target segment

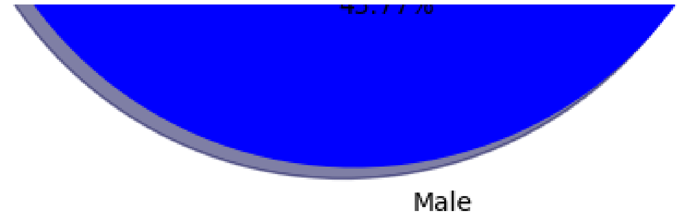
sStep3: Collecting data

Step 4: DATA EXPLORATION

In [12]:

```
#target segmentation
#1 segmenting based on age and gender (sociodemographic segmentation)
labels = ['Female', 'Male']
size = df['Gender'].value_counts()
colors = ['orange', 'blue']
explode = [0, 0.1]
plt.rcParams['figure.figsize'] = (7, 7)
plt.pie(size, colors = colors, explode = explode, labels = labels, shadow = True, autopc
t = '%.2f%%')
plt.title('Gender', fontsize = 20)
plt.axis('off')
plt.legend()
plt.show()
#percentage proporation of female is customers in more than male customers
```



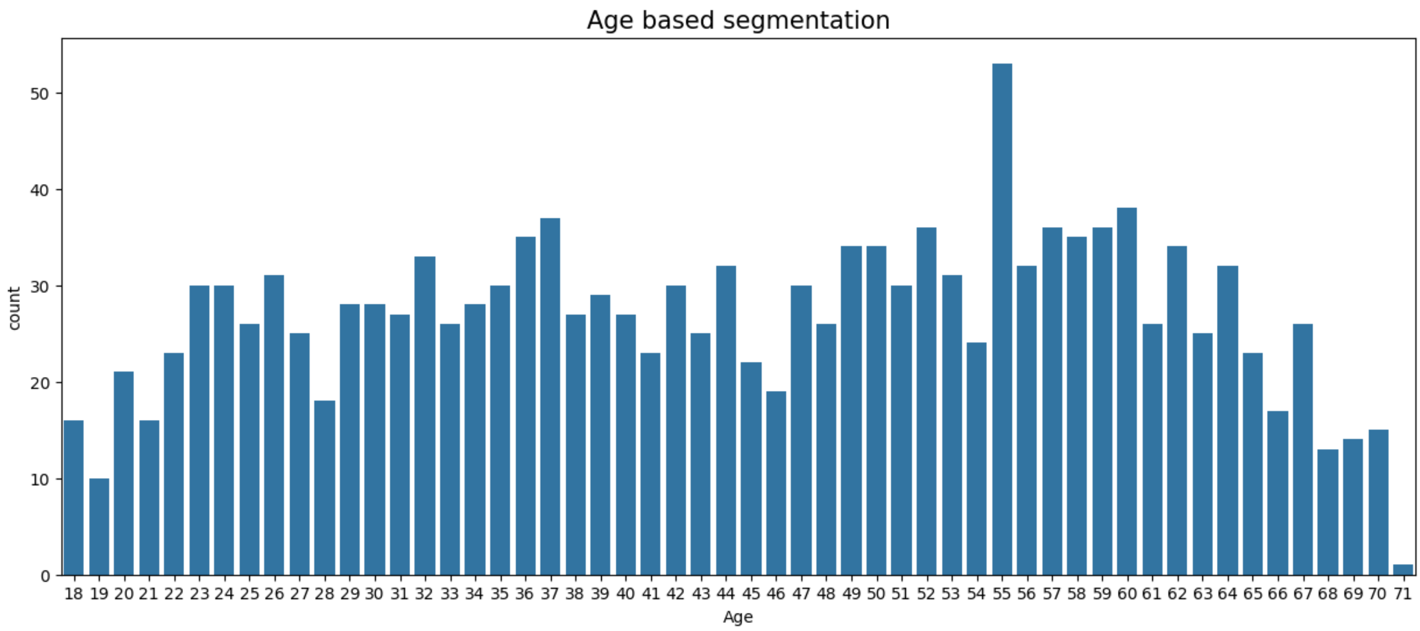


In [13]:

```
# Age based Segmentation
plt.figure(figsize=(15,6))
a = sns.countplot(x = df['Age'])
plt.title('Age based segmentation', fontsize=15)
#maximum customers of mcdonalds belongs to age group 30-40 and 50-60
```

Out[13]:

```
Text(0.5, 1.0, 'Age based segmentation')
```

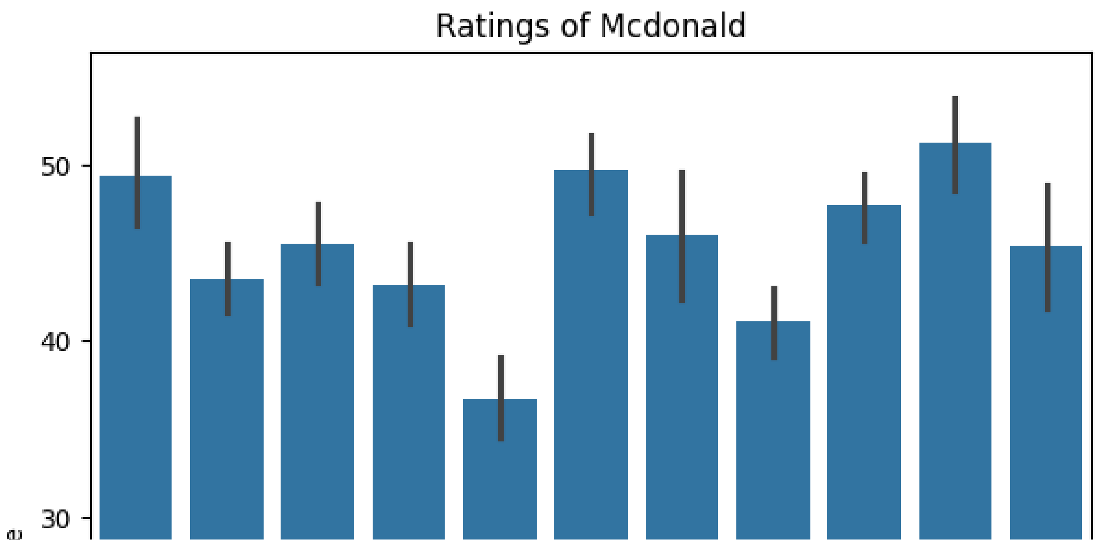


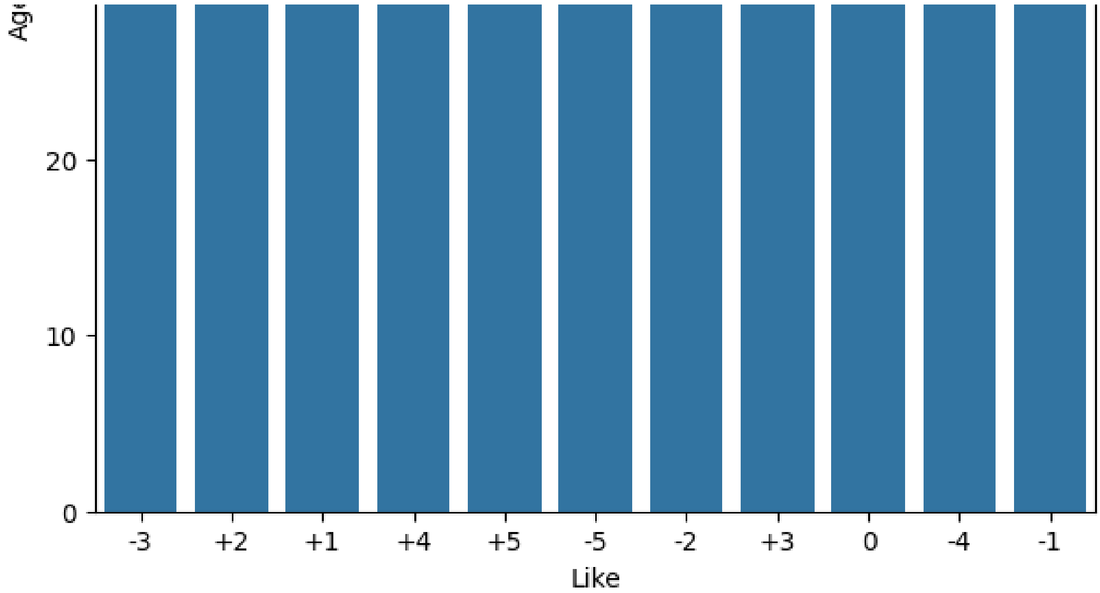
In [14]:

```
# psychographic segmentation
#replacing text in Like column with numbered values for convience
df['Like'] = df['Like'].replace({'I hate it!-5': '-5','I love it!+5':'+5'})

sns.barplot(x='Like', y='Age', data=df)
plt.title('Ratings of Mcdonald')
```

Out[14]:

```
Text(0.5, 1.0, 'Ratings of Mcdonald')
```





In [15]:

```
#Label encoding for categorical varibales
#first 11 columns are categorical in nature
from sklearn.preprocessing import LabelEncoder
```

In [16]:

```
df_cat = df.iloc[:,0:11]
df_cat
```

Out[16]:

	yummy	convenient	spicy	fattening	greasy	fast	cheap	tasty	expensive	healthy	disgusting
0	No	Yes	No	Yes	No	Yes	Yes	No	Yes	No	No
1	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	No	No
2	No	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No
3	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No	No	Yes
4	No	Yes	No	Yes	Yes	Yes	Yes	No	No	Yes	No
...	...	...	...	...	...	...	...	...	...	...	...
1448	No	Yes	No	Yes	Yes	No	No	No	Yes	No	Yes
1449	Yes	Yes	No	Yes	No	No	Yes	Yes	No	Yes	No
1450	Yes	Yes	No	Yes	No	Yes	No	Yes	Yes	No	No
1451	Yes	Yes	No	No	No	Yes	Yes	Yes	No	Yes	No
1452	No	Yes	No	Yes	Yes	No	No	No	Yes	No	Yes

1453 rows × 11 columns

In [17]:

```
def labelling(x):
    df[x] = LabelEncoder().fit_transform(df[x])
    return df

df_cat = df.iloc[:,0:11]
for i in df_cat:
    labelling(i)

df
```

Out[17]:

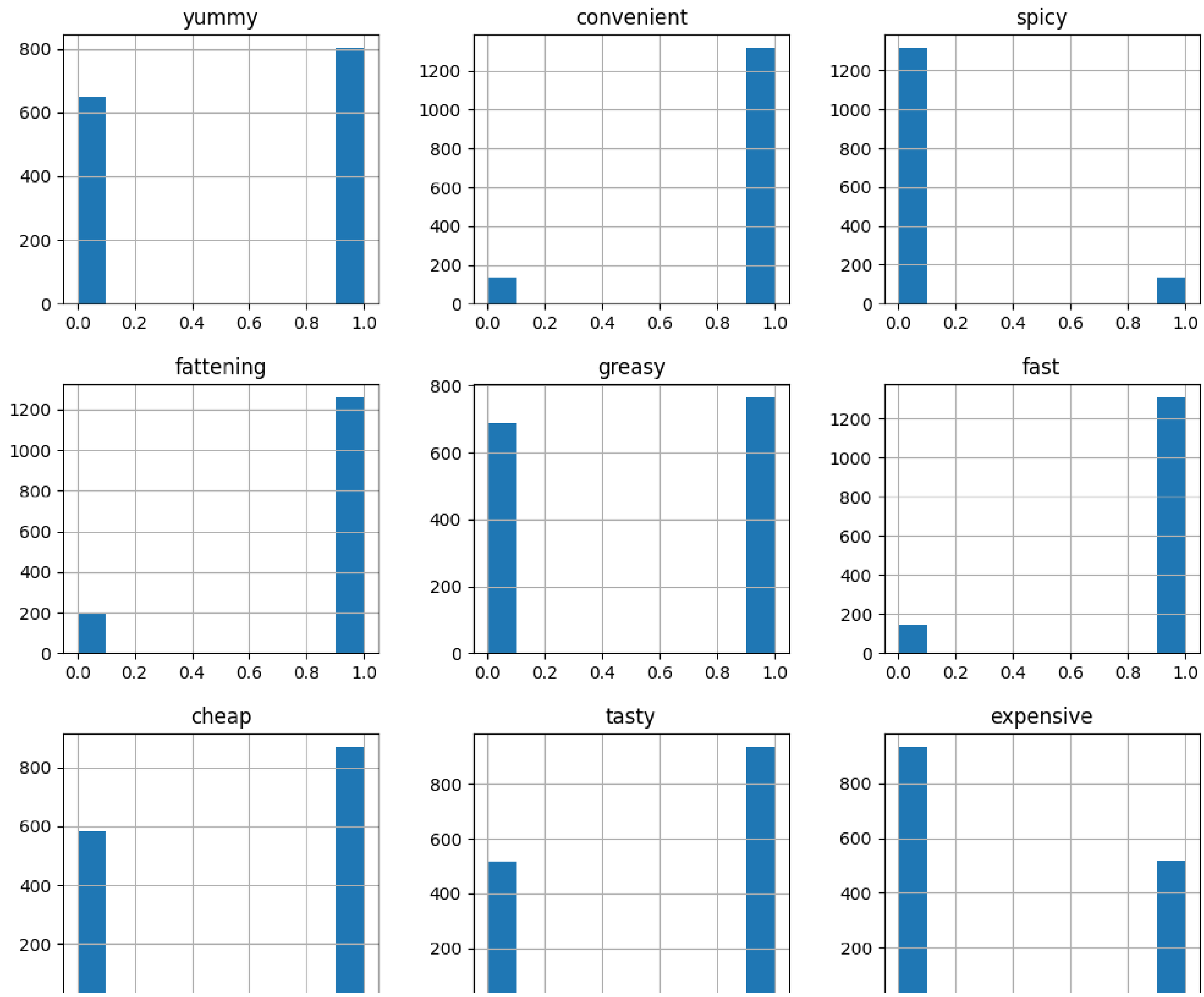
	yummy	convenient	spicy	fattening	greasy	fast	cheap	tasty	expensive	healthy	disgusting	Like	Age	VisitFrequer

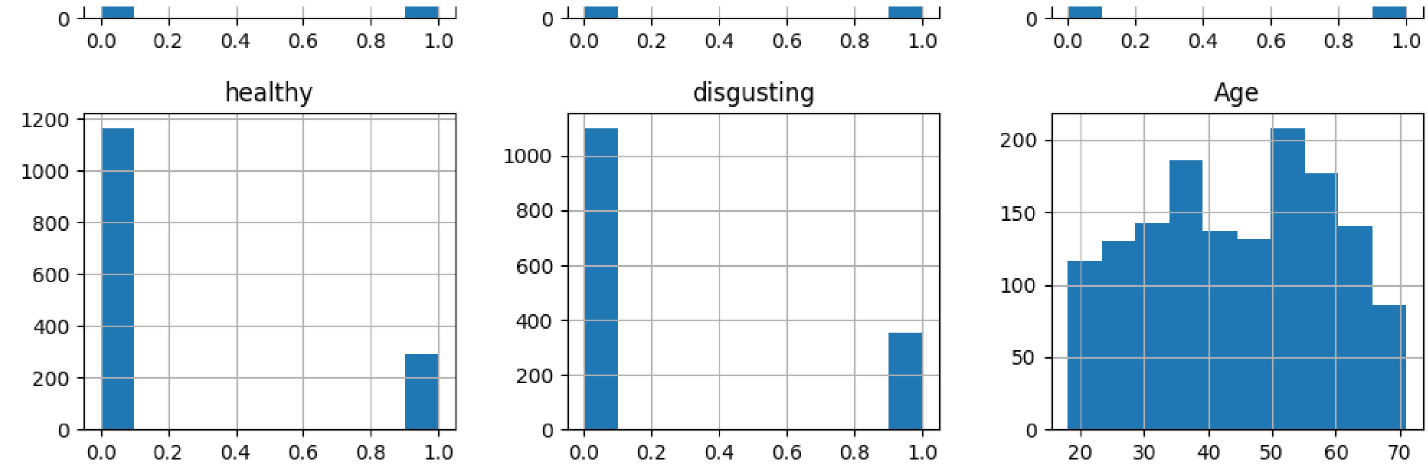
	yummy	convenient	spicy	fattening	greasy	fast	cheap	tasty	expensive	healthy	disgusting	Like	Age	VisitFrequer
0	0	1	0	1	0	1	1	0	1	0	0	-3	61	Every th mon
1	1	1	0	1	1	1	1	1	1	0	0	+2	51	Every th mon
2	0	1	1	1	1	1	0	1	1	1	0	+1	62	Every th mon
3	1	1	0	1	1	1	1	1	0	0	1	+4	69	Once a we
4	0	1	0	1	1	1	1	0	0	1	0	+2	49	Once a mo
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1448	0	1	0	1	1	0	0	0	1	0	1	-5	47	Once a y
1449	1	1	0	1	0	0	1	1	0	1	0	+2	36	Once a we
1450	1	1	0	1	0	1	0	1	1	0	0	+3	52	Once a mo
1451	1	1	0	0	0	1	1	1	0	1	0	+4	41	Every th mon
1452	0	1	0	1	1	0	0	0	1	0	1	-3	30	Every th mon

1453 rows × 15 columns

In [18]:

```
#Histogram of the each attributes
plt.rcParams['figure.figsize'] = (12,14)
df.hist()
plt.show()
```





In [19]:

```
# converting 11 categorical columns into array
# converting 11 categorical columns into array
x = df_cat.values
x
```

Out[19]:

```
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 [20]:

```
#calculating mean of 11 columns
round(df.iloc[:,0:11].mean(),2)
```

Out[20]:

```
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
```

PRINCIPAL COMPONENT ANALYSIS

In [23]:

```
from sklearn.decomposition import PCA
from sklearn import preprocessing
import pandas as pd # Import pandas for DataFrame manipulation

# Assuming 'df_cat' contains categorical columns
# Convert categorical columns to numerical using one-hot encoding
df_encoded = pd.get_dummies(df_cat)

# Extract values from the encoded DataFrame
x = df_encoded.values

pca_data = preprocessing.scale(x)
```

```
pca = PCA(n_components=11)
pc = pca.fit_transform(x)
names = ['pc1','pc2','pc3','pc4','pc5','pc6','pc7','pc8','pc9','pc10','pc11']
pf= pd.DataFrame(data = pc, columns = names)
pf
```

Out[23]:

	pc1	pc2	pc3	pc4	pc5	pc6	pc7	pc8	pc9	pc10	pc11
0	0.601560	-0.309824	0.937985	-0.567524	0.285254	-0.551214	-0.299787	0.230850	0.255982	0.729319	-0.801964
1	-0.309200	0.548984	-1.033545	-0.133961	0.063172	-0.122466	-0.135591	-0.049153	0.157651	0.697650	-0.707730
2	0.530917	1.032991	-0.172590	0.979007	1.187434	-0.972140	0.824645	0.515310	-0.455784	0.087340	0.343288
3	-0.244555	-0.498867	-1.193306	0.292740	-0.963666	-0.051099	-0.076769	-0.327357	-0.039602	-0.354513	-0.072173
4	0.264539	-1.142133	0.040357	0.775459	1.207843	-0.137609	-0.646357	0.242902	-0.105230	0.045109	0.116311
...	...	...	...	...	...	...	...	...	...	...	...
1448	2.192373	0.388953	-0.019427	0.283696	-0.205149	0.433562	-0.106502	0.488684	-0.193166	-0.612069	-0.644988
1449	-1.353882	0.020235	0.429699	0.628406	-0.189066	0.539953	-0.461645	1.241746	-0.430545	-0.349937	-0.273892
1450	-0.262894	1.502830	0.312339	-0.661347	-0.265528	-0.272523	-0.129538	-0.051726	0.054100	0.079929	-0.018102
1451	-1.671692	-0.054546	0.794167	0.991542	0.067381	0.273915	-0.038658	-0.479948	0.031491	-0.003638	-0.148940
1452	2.192373	0.388953	-0.019427	0.283696	-0.205149	0.433562	-0.106502	0.488684	-0.193166	-0.612069	-0.644988

1453 rows × 11 columns

In [24]:

```
pf.head()
```

Out[24]:

	pc1	pc2	pc3	pc4	pc5	pc6	pc7	pc8	pc9	pc10	pc11
0	0.601560	-0.309824	0.937985	-0.567524	0.285254	-0.551214	-0.299787	0.230850	0.255982	0.729319	-0.801964
1	-0.309200	0.548984	-1.033545	-0.133961	0.063172	-0.122466	-0.135591	-0.049153	0.157651	0.697650	-0.707730
2	0.530917	1.032991	-0.172590	0.979007	1.187434	-0.972140	0.824645	0.515310	-0.455784	0.087340	0.343288
3	-0.244555	-0.498867	-1.193306	0.292740	-0.963666	-0.051099	-0.076769	-0.327357	-0.039602	-0.354513	-0.072173
4	0.264539	-1.142133	0.040357	0.775459	1.207843	-0.137609	-0.646357	0.242902	-0.105230	0.045109	0.116311

In [25]:

```
pf.describe()
```

Out[25]:

	pc1	pc2	pc3	pc4	pc5	pc6	pc7	pc8	
count	1.453000e+03	1.453000e+03	1.453000e+03	1.453000e+03	1.453000e+03	1.453000e+03	1.453000e+03	1.453000e+03	1.
mean	-3.423124e-17	-3.178615e-17	-3.728760e-17	1.222544e-17	2.934106e-17	4.217778e-17	-2.353398e-17	-1.075839e-16	
std	1.070630e+00	8.590719e-01	7.136397e-01	5.639864e-01	4.771627e-01	4.387946e-01	4.096939e-01	3.890812e-01	3.
min	-1.680681e+00	-1.471169e+00	-1.245658e+00	-8.352641e-01	-1.479180e+00	-1.205604e+00	-1.139780e+00	-1.317510e+00	1.
25%	-7.745357e-01	-5.046595e-01	-6.374982e-01	-3.980992e-01	-3.441131e-01	-2.725227e-01	-1.898144e-01	-2.428570e-01	
50%	-1.000498e-01	-1.628654e-01	3.801030e-02	-1.645013e-01	4.735897e-02	-4.893440e-02	-9.003195e-02	-4.915267e-02	5.
75%	6.976332e-01	5.537409e-01	4.972289e-01	3.821626e-01	2.896724e-01	2.471271e-01	3.278651e-02	1.832773e-01	1.
max	2.412045e+00	1.808963e+00	1.876628e+00	1.916653e+00	1.489629e+00	1.872125e+00	1.910441e+00	1.585883e+00	1.

In [26]:

```
#Proportion of Variance (from PC1 to PC11)
pca.explained_variance_ratio_
```

Out[26]:

```
array([0.29944723, 0.19279721, 0.13304535, 0.08309578, 0.05948052,
       0.05029956, 0.0438491 , 0.03954779, 0.0367609 , 0.03235329,
       0.02932326])
```

In [27]:

```
np.cumsum(pca.explained_variance_ratio_)
```

Out[27]:

```
array([0.29944723, 0.49224445, 0.6252898 , 0.70838558, 0.7678661 ,
       0.81816566, 0.86201476, 0.90156255, 0.93832345, 0.97067674,
       1.        ])
```

In [29]:

```
# correlation coefficient between original variables and the component

loadings = pca.components_
num_pc = pca.n_features_
pc_list = ["PC"+str(i) for i in list(range(1, num_pc+1))]
loadings_df = pd.DataFrame.from_dict(dict(zip(pc_list, loadings)))
# Transpose the DataFrame so that the PCs become the index and the variables become the c
olumns
loadings_df = loadings_df.T
loadings_df['variable'] = df_cat.columns.values
loadings_df = loadings_df.set_index('variable')
loadings_df
```

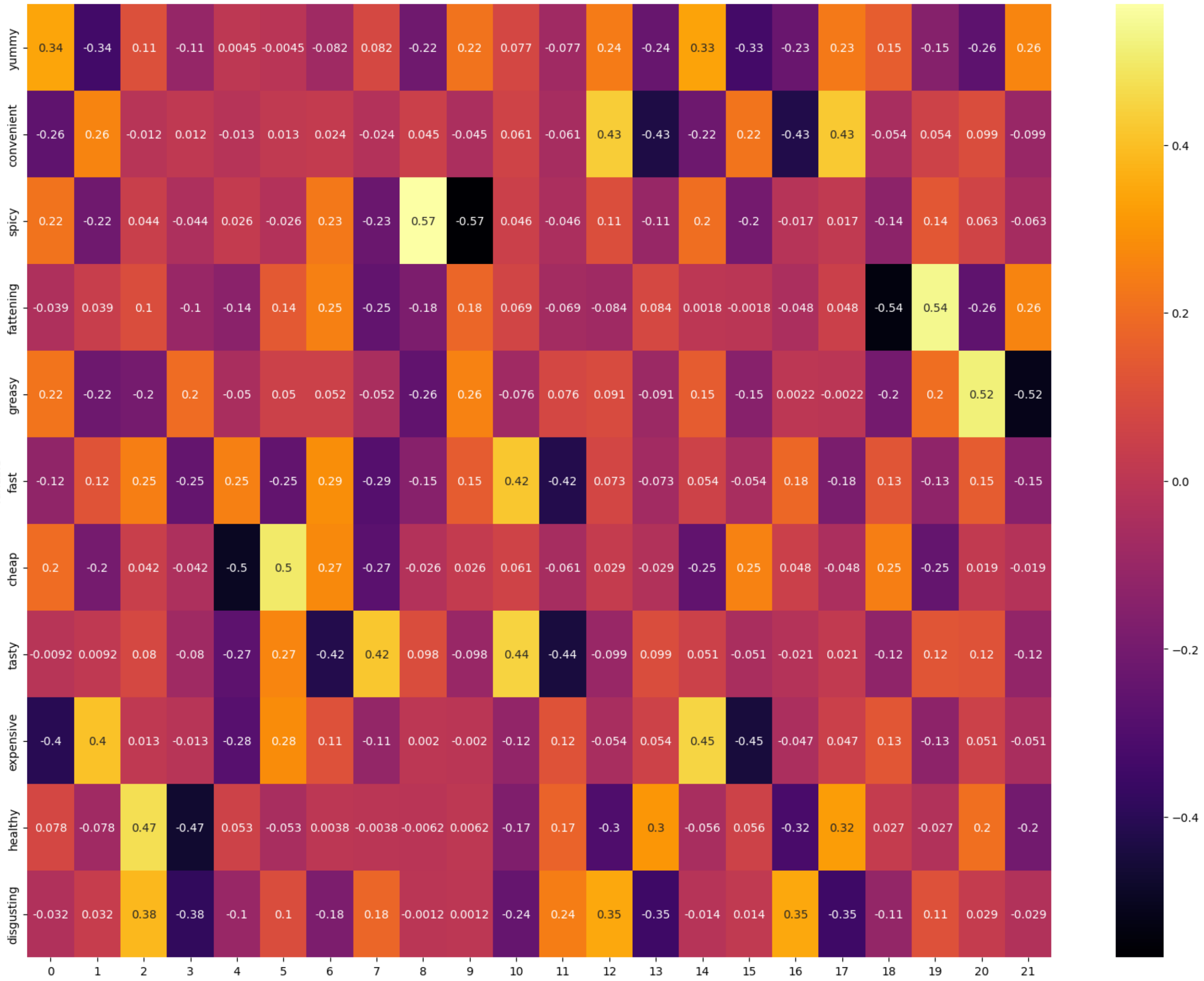
Out[29]:

	0	1	2	3	4	5	6	7	8	9	...	12
variable												
yummy	0.337243	-0.337243	0.109836	-0.109836	0.004495	-0.004495	-0.082188	0.082188	-0.215273	0.215273	...	0.238426
convenient	-0.257238	0.257238	-0.011606	0.011606	-0.013300	0.013300	0.024108	-0.024108	0.045141	-0.045141	...	0.431783
spicy	0.215274	-0.215274	0.044205	-0.044205	0.026176	-0.026176	0.227943	-0.227943	0.567364	-0.567364	...	0.105578
fattening	-0.039006	0.039006	0.100710	-0.100710	-0.139738	0.139738	0.250414	-0.250414	-0.179577	0.179577	...	-0.084116
greasy	0.217460	-0.217460	-0.196299	0.196299	-0.049936	0.049936	0.051905	-0.051905	-0.255548	0.255548	...	0.091197
fast	-0.120730	0.120730	0.245953	-0.245953	0.251084	-0.251084	0.287450	-0.287450	-0.148031	0.148031	...	0.073002
cheap	0.198357	-0.198357	0.042241	-0.042241	-0.500375	0.500375	0.272903	-0.272903	-0.025576	0.025576	...	0.028602
tasty	-0.009221	0.009221	0.079959	-0.079959	-0.265825	0.265825	-0.416926	0.416926	0.097751	-0.097751	...	-0.099038
expensive	-0.404750	0.404750	0.013057	-0.013057	-0.283041	0.283041	0.113499	-0.113499	0.002013	-0.002013	...	-0.053789
healthy	0.077983	-0.077983	0.470804	-0.470804	0.053481	-0.053481	0.003775	-0.003775	-0.006157	0.006157	...	-0.302703
disgusting	-0.032130	0.032130	0.382981	-0.382981	-0.100218	0.100218	-0.177420	0.177420	-0.001161	0.001161	...	0.345975

11 rows × 22 columns

In [30]:

```
#Correlation matrix plot for loadings
plt.rcParams['figure.figsize'] = (20,15)
ax = sns.heatmap(loadings_df, annot=True, cmap='inferno')
plt.show()
```



In [32]:

```
pip install bioinfokit
```

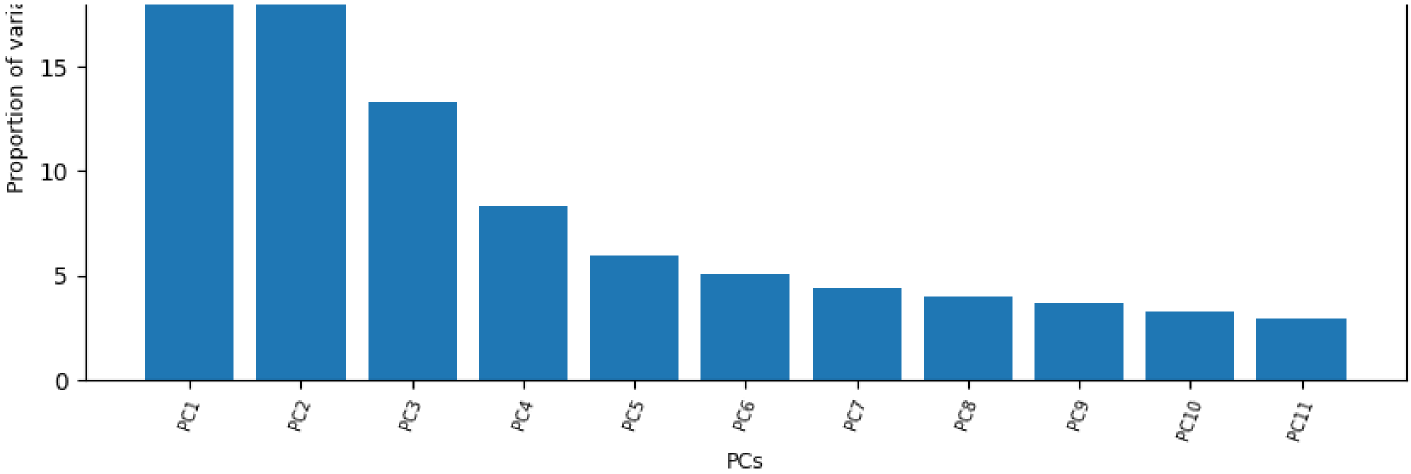
```
Collecting bioinfokit
  Downloading bioinfokit-2.1.4.tar.gz (88 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 88.1/88.1 kB 1.1 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from bi
oinfokit) (2.0.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from bio
infokit) (1.25.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (fro
m bioinfokit) (3.7.1)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from bio
infokit) (1.11.4)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (f
rom bioinfokit) (1.2.2)
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-packages (from b
ioinfokit) (0.13.1)
Requirement already satisfied: matplotlib-venn in /usr/local/lib/python3.10/dist-packages
(from bioinfokit) (0.11.10)
Requirement already satisfied: tabulate in /usr/local/lib/python3.10/dist-packages (from
bioinfokit) (0.9.0)
```

```
bioinfokit) (0.9.0)
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-packages (fr
om bioinfokit) (0.14.2)
Collecting textwrap3 (from bioinfokit)
  Downloading textwrap3-0.9.2-py2.py3-none-any.whl (12 kB)
Collecting adjustText (from bioinfokit)
  Downloading adjustText-1.1.1-py3-none-any.whl (11 kB)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-package
s (from matplotlib->bioinfokit) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (f
rom matplotlib->bioinfokit) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packag
es (from matplotlib->bioinfokit) (4.53.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packag
es (from matplotlib->bioinfokit) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages
(from matplotlib->bioinfokit) (24.0)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (
from matplotlib->bioinfokit) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-package
s (from matplotlib->bioinfokit) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-pac
kages (from matplotlib->bioinfokit) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (f
rom pandas->bioinfokit) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages
(from pandas->bioinfokit) (2024.1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (
from scikit-learn->bioinfokit) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-pac
kages (from scikit-learn->bioinfokit) (3.5.0)
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/dist-packages (f
rom statsmodels->bioinfokit) (0.5.6)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy
>=0.5.6->statsmodels->bioinfokit) (1.16.0)
Building wheels for collected packages: bioinfokit
  Building wheel for bioinfokit (setup.py) ... done
  Created wheel for bioinfokit: filename=bioinfokit-2.1.4-py3-none-any.whl size=59221 sha
256=c5484e8f0d2cbc7e8d7b67945d70526e77d145bf7948250db94970dcb926805b
  Stored in directory: /root/.cache/pip/wheels/45/b1/91/212510cab723ee76a25180836e8897f92
6820382374184b017
Successfully built bioinfokit
Installing collected packages: textwrap3, adjustText, bioinfokit
Successfully installed adjustText-1.1.1 bioinfokit-2.1.4 textwrap3-0.9.2
```

In [34]:

```python
#Scree plot (Elbow test)- PCA
from bioinfokit.visuz import cluster
num_pc = pca.n_components_
pc_list = ["PC"+str(i) for i in list(range(1, num_pc+1))] # Recalculate pc_list to match
the number of components
cluster.screeplot(obj=[pc_list, pca.explained_variance_ratio_],show=True,dim=(10,5))
```

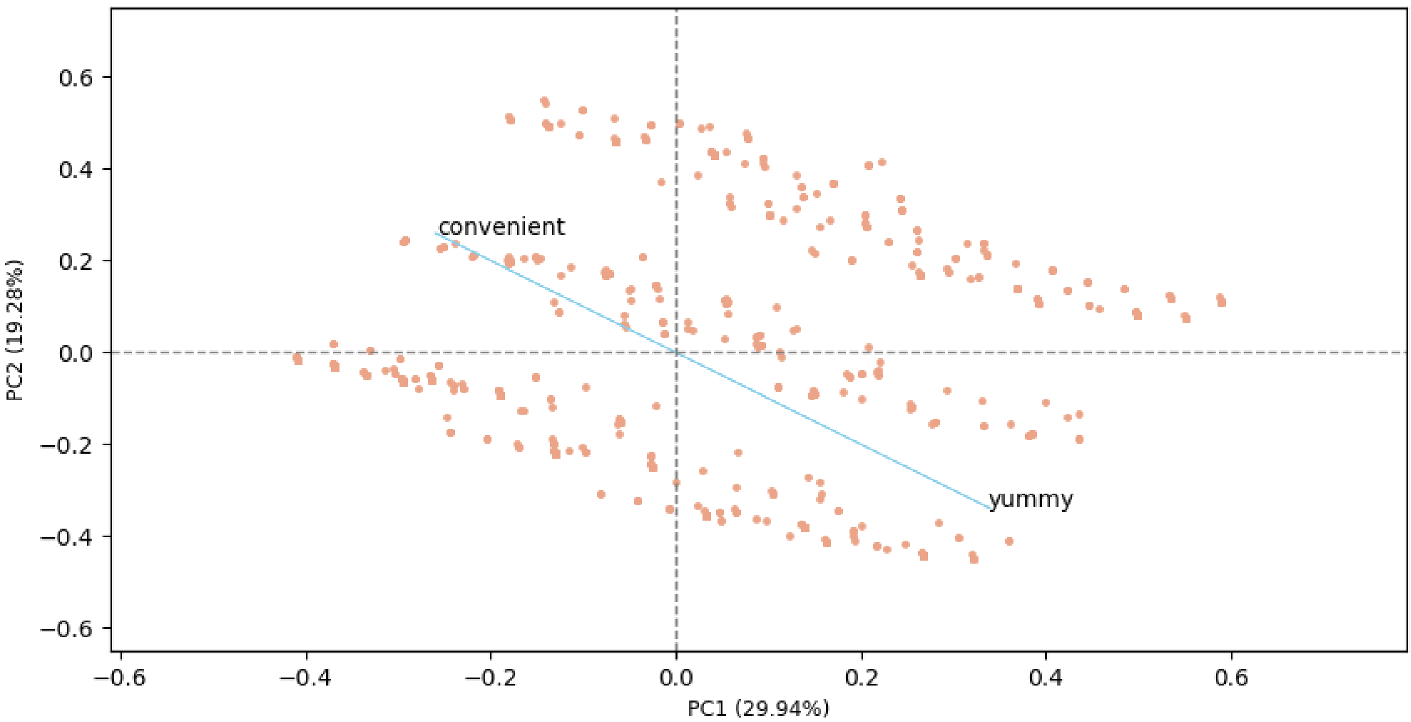
```
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
```



In [40]:

```
# get PC scores
pca_scores = PCA().fit_transform(x)

# get 2D biplot
cluster.biplot(
    cscore=pca_scores,
    loadings=loadings[0:2,:].T,   # Select the first two principal components and transpos
e
    labels=df.columns.values,
    var1=round(pca.explained_variance_ratio_[0]*100, 2),
    var2=round(pca.explained_variance_ratio_[1]*100, 2),
    show=True,
    dim=(10, 5)
)
```

```
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
WARNING:matplotlib.font_manager:findfont: Font family 'Arial' not found.
```



Step-5 Extracting Segments

In [41]:

```
!pip install yellowbrick
```

```
Requirement already satisfied: yellowbrick in /usr/local/lib/python3.10/dist-packages (1.
5)
Requirement already satisfied: matplotlib!=3.0.0,>=2.0.2 in /usr/local/lib/python3.10/dis
t-packages (from yellowbrick) (3.7.1)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages (f
rom yellowbrick) (1.11.4)
Requirement already satisfied: scikit-learn>=1.0.0 in /usr/local/lib/python3.10/dist-pack
ages (from yellowbrick) (1.2.2)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.10/dist-packages (
from yellowbrick) (1.25.2)
Requirement already satisfied: cycler>=0.10.0 in /usr/local/lib/python3.10/dist-packages
(from yellowbrick) (0.12.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-package
s (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (1.2.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packag
es (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (4.53.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packag
es (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages
(from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (24.0)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (
from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-package
s (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-pac
kages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (2.8.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (
from scikit-learn>=1.0.0->yellowbrick) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-pac
kages (from scikit-learn>=1.0.0->yellowbrick) (3.5.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from
python-dateutil>=2.7->matplotlib!=3.0.0,>=2.0.2->yellowbrick) (1.16.0)
```

In [46]:

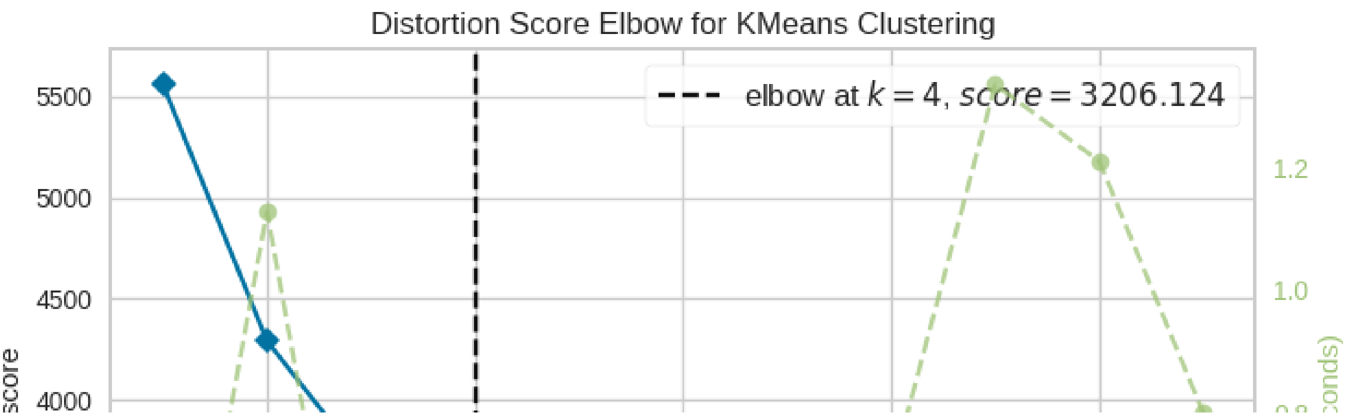
```
#Using k-means clustering analysis
from sklearn.utils.metaestimators import available_if
from sklearn.cluster import KMeans
from yellowbrick.cluster import KElbowVisualizer
model = KMeans()
visualizer = KElbowVisualizer(model, k=(1,12)).fit(df_cat)
visualizer.show()
```

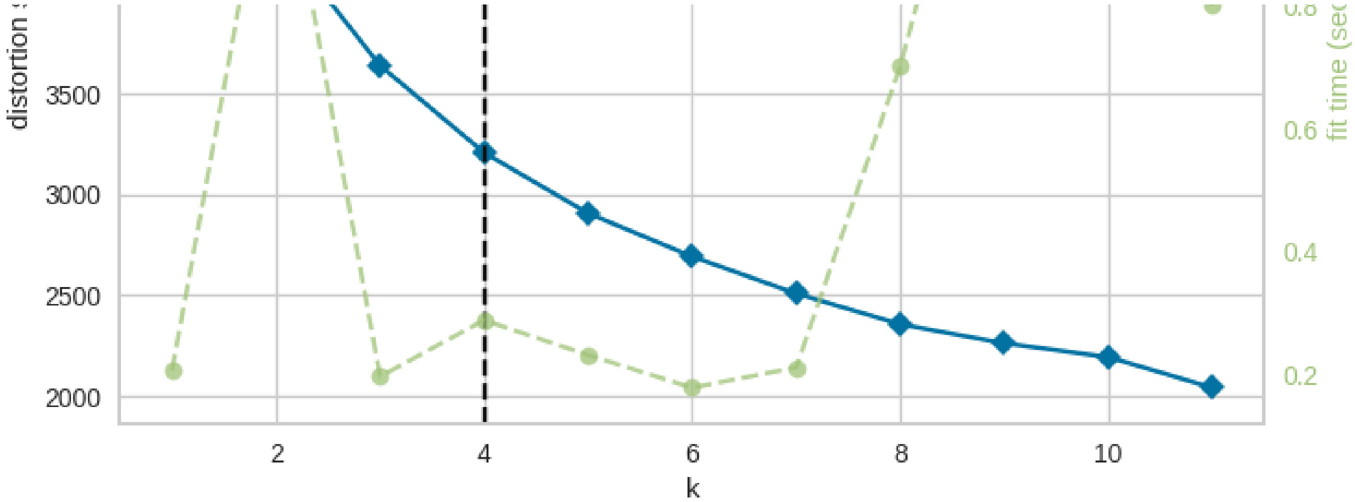
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-46-93343fc30455> in <cell line: 6>()
      4 from yellowbrick.cluster import KElbowVisualizer
      5 model = KMeans()
----> 6 visualizer = KElbowVisualizer(model, k=(1,12)).fit(df_cat)
      7 visualizer.show()

/usr/local/lib/python3.10/dist-packages/yellowbrick/cluster/elbow.py in fit(self, X, y, *
*kwargs)
    337             # Set the k value and fit the model
    338             self.estimator.set_params(n_clusters=k)
--> 339             self.estimator.fit(X, **kwargs)
    340
    341             # Append the time and score to our plottable metrics

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py in fit(self, X, y, sam
ple_weight)
   1415         self._validate_params()
   1416
-> 1417         X = self._validate_data(
   1418             X,
   1419             accept_sparse="csr",

/usr/local/lib/python3.10/dist-packages/sklearn/base.py in _validate_data(self, X, y, res
et, validate_separately, **check_params)
```

```
et, validate_separately, check_params)
    563                 raise ValueError("Validation should be done on X, y or both.")
    564             elif not no_val_X and no_val_y:
--> 565                 X = check_array(X, input_name="X", **check_params)
    566                 out = X
    567             elif no_val_X and not no_val_y:

/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator, input_name)
    877                     array = xp.astype(array, dtype, copy=False)
    878                 else:
--> 879                     array = _asarray_with_order(array, order=order, dtype=dtype, xp=xp)
    880             except ComplexWarning as complex_warning:
    881                 raise ValueError(

/usr/local/lib/python3.10/dist-packages/sklearn/utils/_array_api.py in _asarray_with_order(array, dtype, order, copy, xp)
    183     if xp.__name__ in {"numpy", "numpy.array_api"}:
    184         # Use NumPy API to support order
--> 185         array = numpy.asarray(array, order=order, dtype=dtype)
    186         return xp.asarray(array, copy=copy)
    187     else:

/usr/local/lib/python3.10/dist-packages/pandas/core/generic.py in __array__(self, dtype)
   1996     def __array__(self, dtype: npt.DTypeLike | None = None) -> np.ndarray:
   1997         values = self._values
-> 1998         arr = np.asarray(values, dtype=dtype)
   1999         if (
   2000             astype_is_view(values.dtype, arr.dtype)

ValueError: could not convert string to float: 'No'
```

In [47]:

```python
#Using k-means clustering analysis
from sklearn.cluster import KMeans
from yellowbrick.cluster import KElbowVisualizer
from sklearn.preprocessing import OneHotEncoder

# Assuming 'df_cat' contains categorical columns, let's one-hot encode them
ohe = OneHotEncoder()
encoded_data = ohe.fit_transform(df_cat.select_dtypes(include=['object']))

# Convert the encoded data back to a DataFrame for easier handling
encoded_df = pd.DataFrame(encoded_data.toarray(), columns=ohe.get_feature_names_out(df_cat.select_dtypes(include=['object']).columns))

# Drop the original categorical columns from 'df_cat' and concatenate the encoded DataFrame
df_cat_numeric = df_cat.drop(df_cat.select_dtypes(include=['object']).columns, axis=1)
df_cat_processed = pd.concat([df_cat_numeric, encoded_df], axis=1)

model = KMeans()
visualizer = KElbowVisualizer(model, k=(1,12)).fit(df_cat_processed) # Use the processed DataFrame here
visualizer.show()
```





Out[47]:

```
<Axes: title={'center': 'Distortion Score Elbow for KMeans Clustering'}, xlabel='k', ylab
el='distortion score'>
```

In [49]:

```
#K-means clustering
from sklearn.preprocessing import OneHotEncoder

# Assuming 'df_cat' contains categorical columns, let's one-hot encode them
ohe = OneHotEncoder()
encoded_data = ohe.fit_transform(df_cat.select_dtypes(include=['object']))

# Convert the encoded data back to a DataFrame for easier handling
encoded_df = pd.DataFrame(encoded_data.toarray(), columns=ohe.get_feature_names_out(df_c
at.select_dtypes(include=['object']).columns))

# Drop the original categorical columns from 'df_cat' and concatenate the encoded DataFra
me
df_cat_numeric = df_cat.drop(df_cat.select_dtypes(include=['object']).columns, axis=1)
df_cat_processed = pd.concat([df_cat_numeric, encoded_df], axis=1)

kmeans = KMeans(n_clusters=4, init='k-means++', random_state=0).fit(df_cat_processed) #
Use the processed DataFrame here
df['cluster_num'] = kmeans.labels_ #adding to df
print (kmeans.labels_) #Label assigned for each data point
print (kmeans.inertia_) #gives within-cluster sum of squares.
print(kmeans.n_iter_) #number of iterations that k-means algorithm runs to get a minimum
within-cluster sum of squares
print(kmeans.cluster_centers_) #Location of the centroids on each cluster.
```

```
[2 0 0 ... 0 1 3]
3206.1208881117846
8
[[0.14551084 0.85448916 0.0371517  0.9628483  0.86687307 0.13312693
  0.09287926 0.90712074 0.38080495 0.61919505 0.13931889 0.86068111
  0.89164087 0.10835913 0.06811146 0.93188854 0.10216718 0.89783282
  0.79566563 0.20433437 0.89473684 0.10526316]
 [0.11206897 0.88793103 0.01896552 0.98103448 0.9137931  0.0862069
  0.20517241 0.79482759 0.67068966 0.32931034 0.03965517 0.96034483
  0.07758621 0.92241379 0.02413793 0.97586207 0.98275862 0.01724138
  0.67931034 0.32068966 0.95689655 0.04310345]
 [0.97697368 0.02302632 0.10855263 0.89144737 0.92763158 0.07236842
  0.07565789 0.92434211 0.33223684 0.66776316 0.03618421 0.96381579
  0.06578947 0.93421053 0.84539474 0.15460526 0.98684211 0.01315789
  0.92763158 0.07236842 0.61184211 0.38815789]
 [0.9796748  0.0203252  0.31707317 0.68292683 0.91463415 0.08536585
  0.08536585 0.91463415 0.30487805 0.69512195 0.26829268 0.73170732
  0.93495935 0.06504065 0.91056911 0.08943089 0.12195122 0.87804878
  0.93902439 0.06097561 0.28455285 0.71544715]]
```

In [50]:

```
#To see each cluster size
```

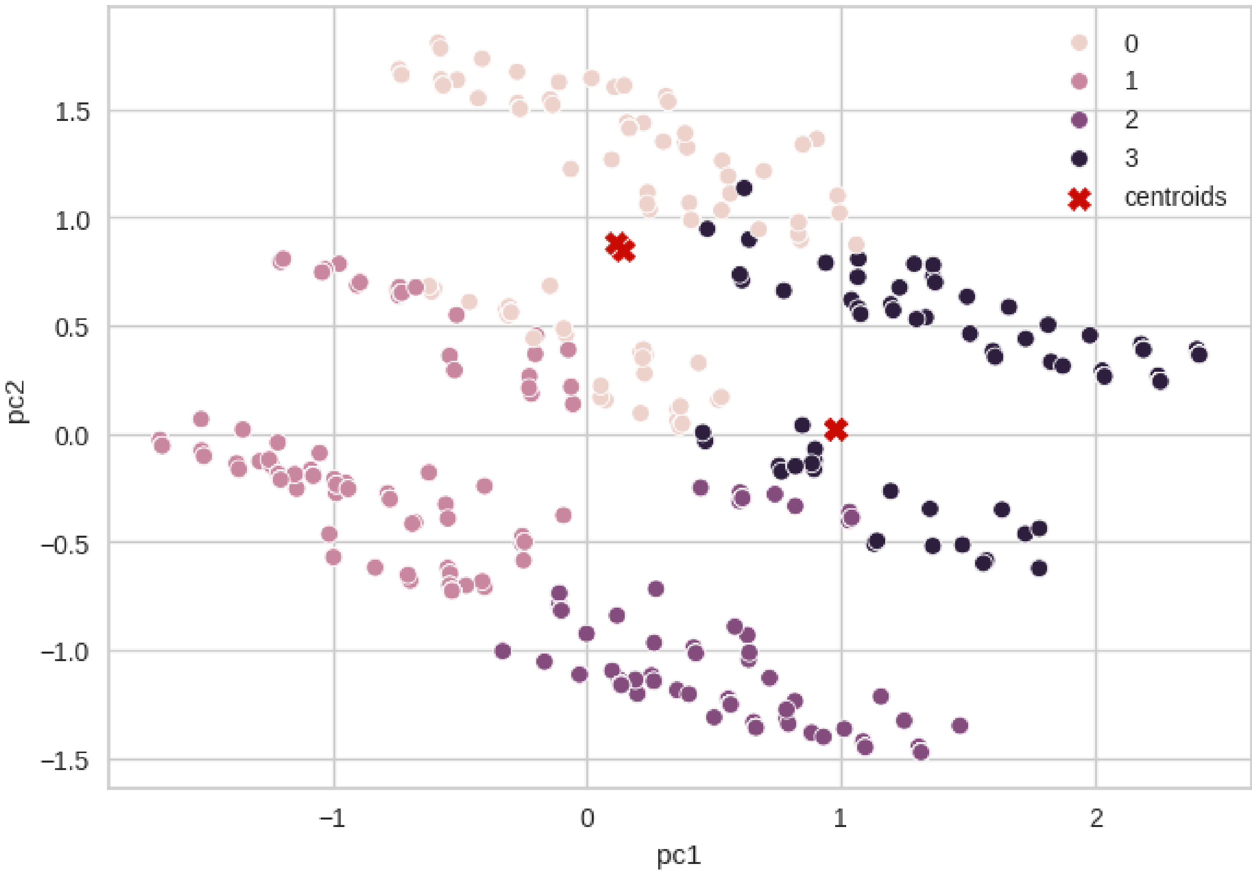
```
#To see each cluster size
from collections import Counter
Counter(kmeans.labels_)
```

Out[50]:

```
Counter({2: 304, 0: 323, 1: 580, 3: 246})
```

In [51]:

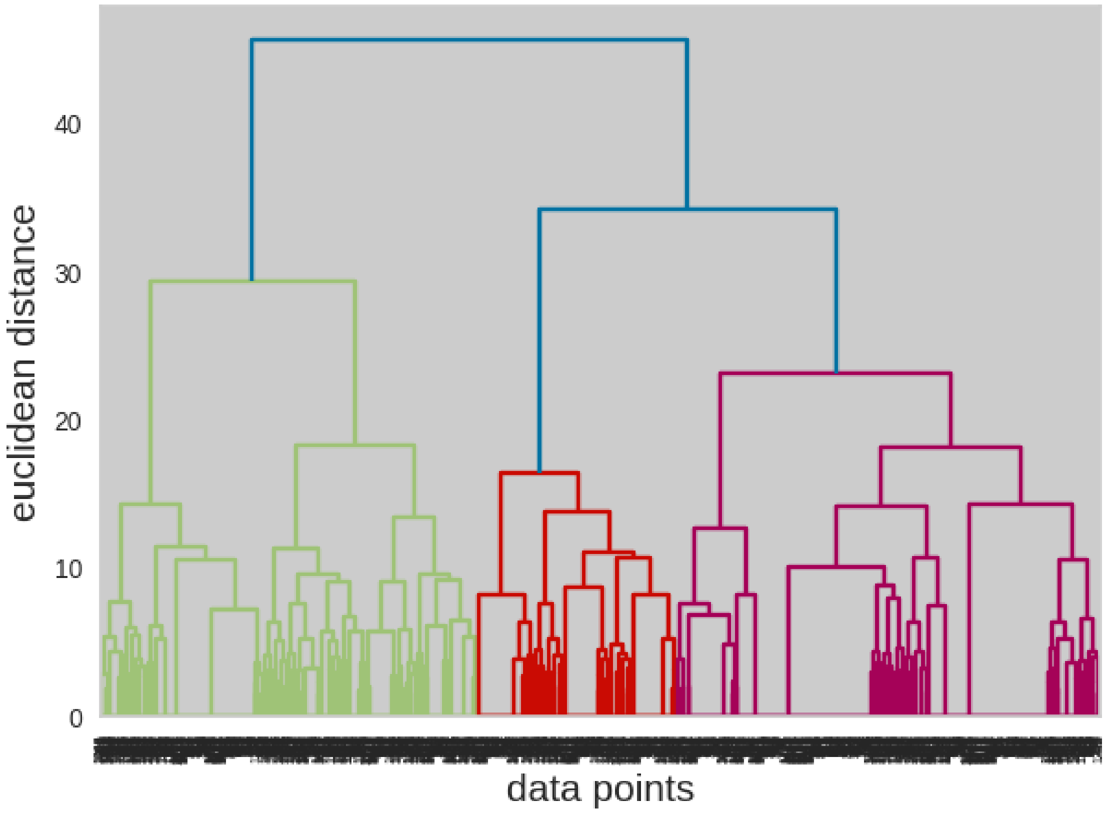
```
#Visulazing clusters
sns.scatterplot(data=pf, x="pc1", y="pc2", hue=kmeans.labels_)
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1],
            marker="X", c="r", s=80, label="centroids")
plt.legend()
plt.show()
```



In [53]:

```
# Hierarchical Clustering Algotithm
#create demogram and find the best clustering value
from scipy.cluster.hierarchy import linkage, dendrogram
from sklearn.preprocessing import OneHotEncoder

# Assuming 'df_cat' contains categorical columns, let's one-hot encode them
ohe = OneHotEncoder()
encoded_data = ohe.fit_transform(df_cat.select_dtypes(include=['object']))

# Convert the encoded data back to a DataFrame for easier handling
encoded_df = pd.DataFrame(encoded_data.toarray(), columns=ohe.get_feature_names_out(df_c
at.select_dtypes(include=['object']).columns))

# Drop the original categorical columns from 'df_cat' and concatenate the encoded DataFra
me
df_cat_numeric = df_cat.drop(df_cat.select_dtypes(include=['object']).columns, axis=1)
df_cat_processed = pd.concat([df_cat_numeric, encoded_df], axis=1)

# Now use the processed DataFrame for hierarchical clustering
merg = linkage(df_cat_processed, method='ward') # Use the processed DataFrame here
plt.rcParams['figure.figsize'] = (7,5)
dendrogram(merg,leaf_rotation = 90)
plt.xlabel("data points", fontsize = 15)
```

```
plt.ylabel("euclidean distance", fontsize=15)
plt.show()
```



Step 7: Describing Segment

In [54]:

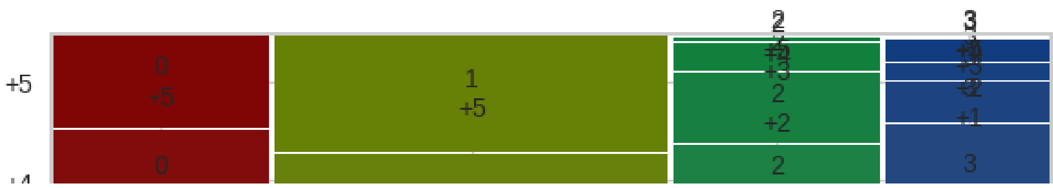
```
#DESCRIBING SEGMENTS

from statsmodels.graphics.mosaicplot import mosaic
from itertools import product

crosstab =pd.crosstab(df['cluster_num'],df['Like'])
#Reordering cols
crosstab = crosstab[['-5','-4','-3','-2','-1','0','+1','+2','+3','+4','+5']]
crosstab
```

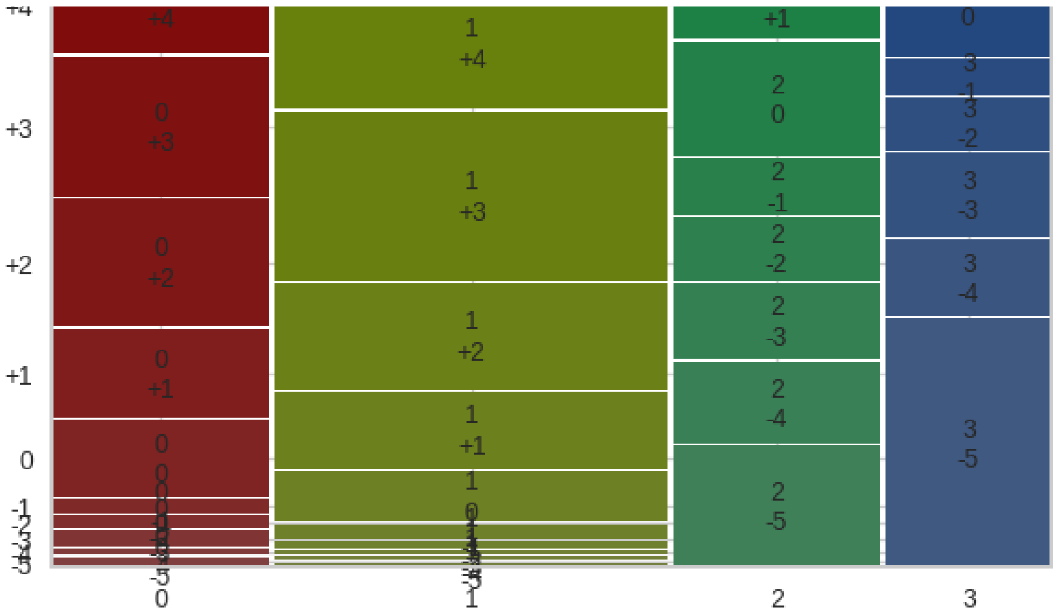
Out[54]:

Like	-5	-4	-3	-2	-1	0	+1	+2	+3	+4	+5
cluster_num											
0	5	3	7	6	7	36	42	60	66	47	44
1	4	4	2	6	13	43	65	90	143	111	99
2	54	36	34	28	25	51	31	31	12	2	0
3	89	28	30	19	13	39	14	6	8	0	0

In [55]:

```
plt.rcParams['figure.figsize'] = (7,5)
mosaic(crosstab.stack())
plt.show()
```





In [56]:

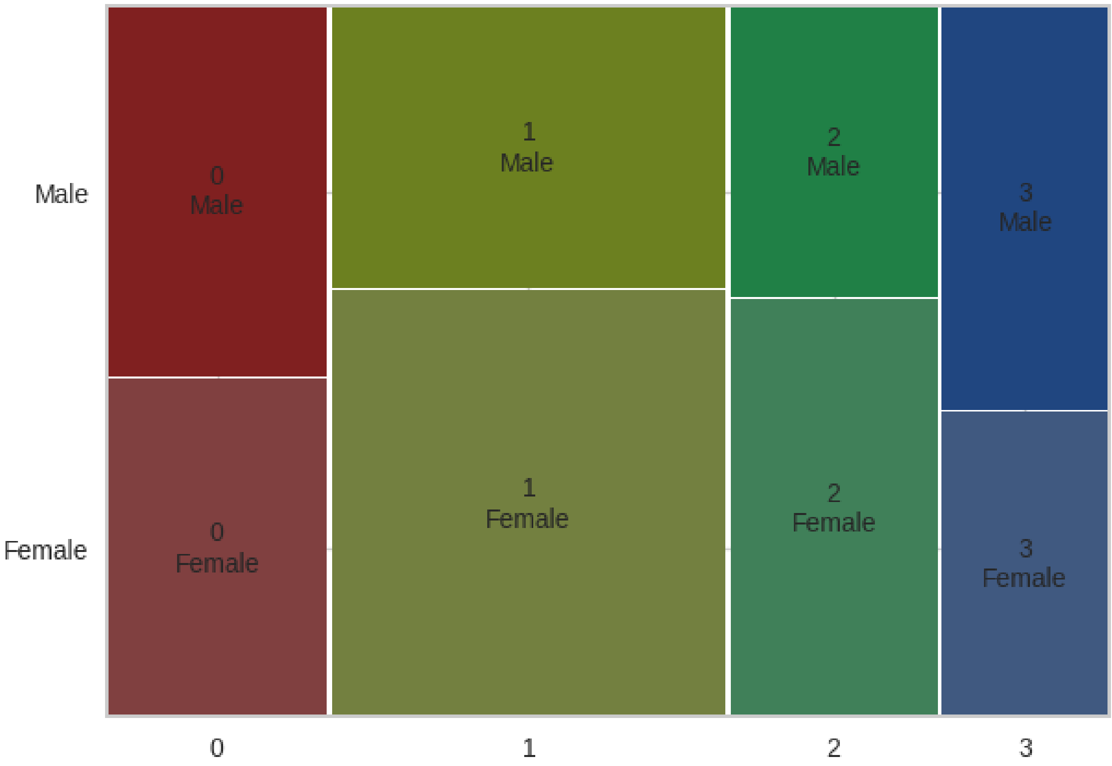
```
#Mosaic plot gender vs segment
crosstab_gender =pd.crosstab(df['cluster_num'],df['Gender'])
crosstab_gender
```

Out[56]:

Gender	Female	Male
cluster_num		
0	154	169
1	349	231
2	179	125
3	106	140

In [57]:

```
plt.rcParams['figure.figsize'] = (7,5)
mosaic(crosstab_gender.stack())
plt.show()
```

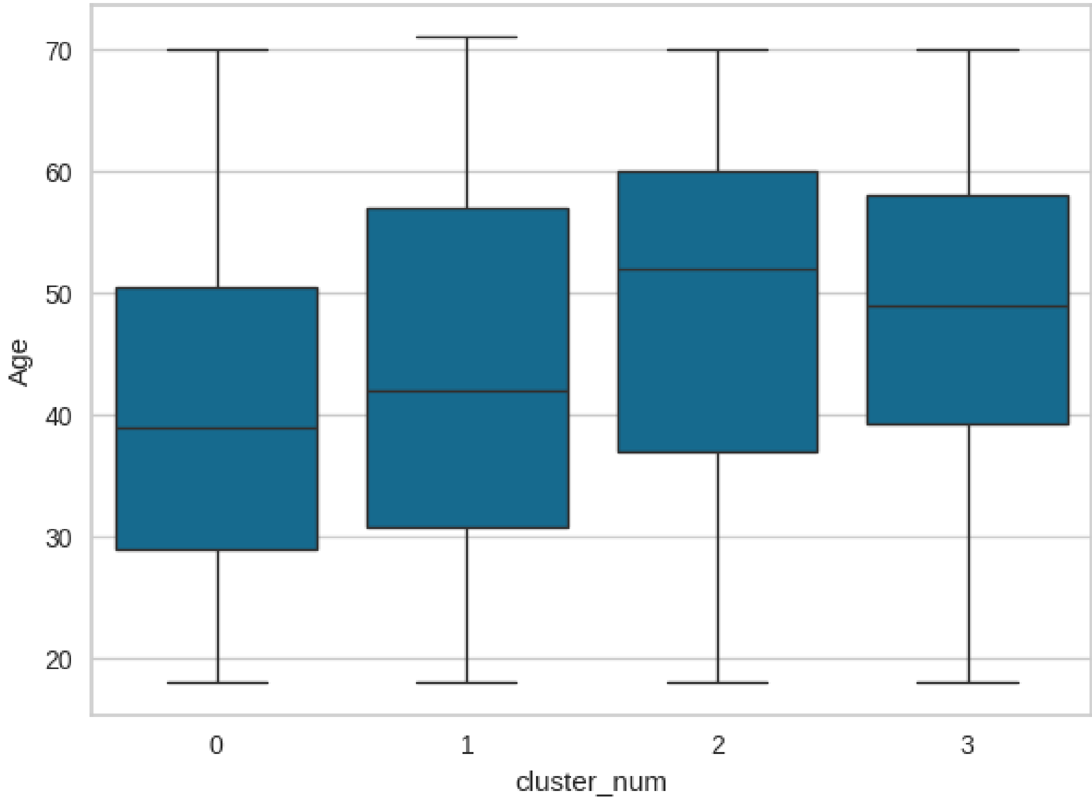


In [62]:

```
#box plot for age

sns.boxplot(x="cluster_num", y="Age", data=df)
```

Out[62]:

```
<Axes: xlabel='cluster_num', ylabel='Age'>
```



Step 8: Selecting Target Segment

In [63]:

```
#Calculating the mean
#Visit frequency
df['VisitFrequency'] = LabelEncoder().fit_transform(df['VisitFrequency'])
visit = df.groupby('cluster_num')['VisitFrequency'].mean()
visit = visit.to_frame().reset_index()
visit
```

Out[63]:

	cluster_num	VisitFrequency
0	0	2.547988
1	1	2.584483
2	2	2.822368
3	3	2.654472

In [64]:

```
#Like
df['Like'] = LabelEncoder().fit_transform(df['Like'])
Like = df.groupby('cluster_num')['Like'].mean()
Like = Like.to_frame().reset_index()
Like
```

Out[64]:

	cluster_num	Like
0	0	3.275542
1	1	2.962069
2	2	6.171053
3	3	7.422764

In [65]:

```
#Gender
df['Gender'] = LabelEncoder().fit_transform(df['Gender'])
Gender = df.groupby('cluster_num')['Gender'].mean()
Gender = Gender.to_frame().reset_index()
Gender
```

Out[65]:

	cluster_num	Gender
0	0	0.523220
1	1	0.398276
2	2	0.411184
3	3	0.569106

In [66]:

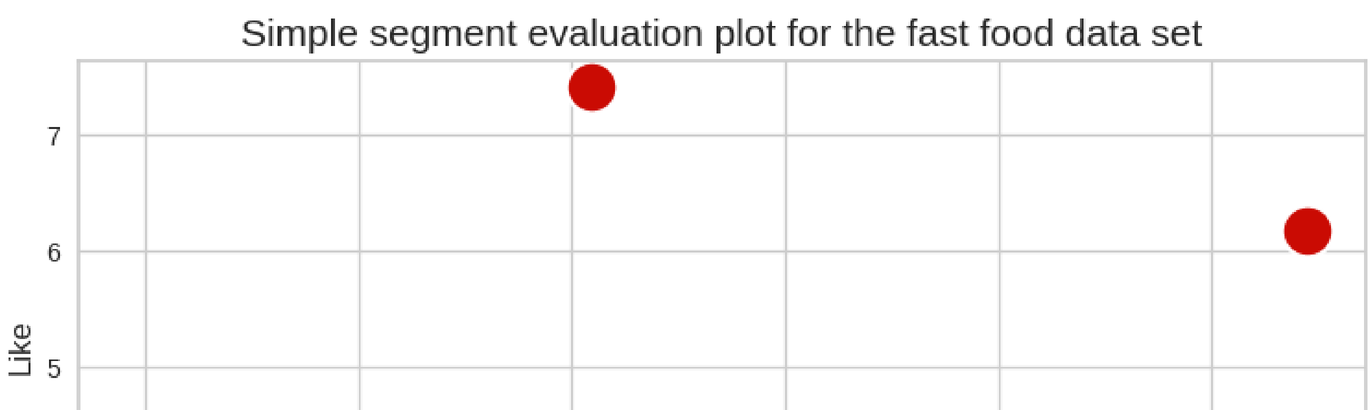
```
segment = Gender.merge(Like, on='cluster_num', how='left').merge(visit, on='cluster_num'
, how='left')
segment
```

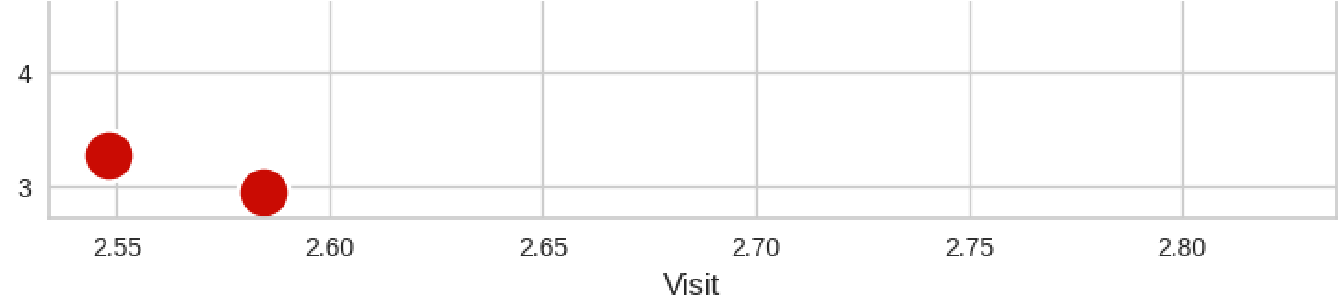
Out[66]:

	cluster_num	Gender	Like	VisitFrequency
0	0	0.523220	3.275542	2.547988
1	1	0.398276	2.962069	2.584483
2	2	0.411184	6.171053	2.822368
3	3	0.569106	7.422764	2.654472

In [67]:

```
#Target segments

plt.figure(figsize = (9,4))
sns.scatterplot(x = "VisitFrequency", y = "Like",data=segment,s=400, color="r")
plt.title("Simple segment evaluation plot for the fast food data set",
          fontsize = 15)
plt.xlabel("Visit", fontsize = 12)
plt.ylabel("Like", fontsize = 12)
plt.show()
```





In [ ]: