In [158]:
```
import pandas as pd
#importing data set
mcdonalds = pd.read_csv("C:/Users/raghu/Downloads/mcdonalds.csv")

#displaying first 5 rows of data
print(mcdonalds.head())
```

```
  yummy convenient spicy fattening greasy fast cheap tasty expensive healthy
\
0    No        Yes    No       Yes     No  Yes   Yes    No       Yes      No
1   Yes        Yes    No       Yes    Yes  Yes   Yes   Yes       Yes      No
2    No        Yes   Yes       Yes    Yes  Yes    No   Yes       Yes     Yes
3   Yes        Yes    No       Yes    Yes  Yes   Yes   Yes        No      No
4    No        Yes    No       Yes    Yes  Yes   Yes    No        No     Yes

  disgusting Like  Age      VisitFrequency  Gender
0         No   -3   61  Every three months  Female
1         No   +2   51  Every three months  Female
2         No   +1   62  Every three months  Female
3        Yes   +4   69         Once a week  Female
4         No   +2   49        Once a month    Male
```

In [159]:
```
mcdonalds.shape #shape of the data
```

Out[159]:
```
(1453, 15)
```

In [160]:
```
print(mcdonalds.columns.values) #columns or attributes
```

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

In [161]:
```
1 mcdonalds.describe() #describing with statistics
```

Out[161]:

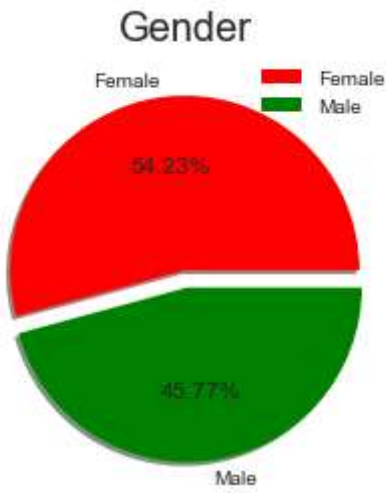
	Age
count	1453.000000
mean	44.604955
std	14.221178
min	18.000000
25%	33.000000
50%	45.000000
75%	57.000000
max	71.000000

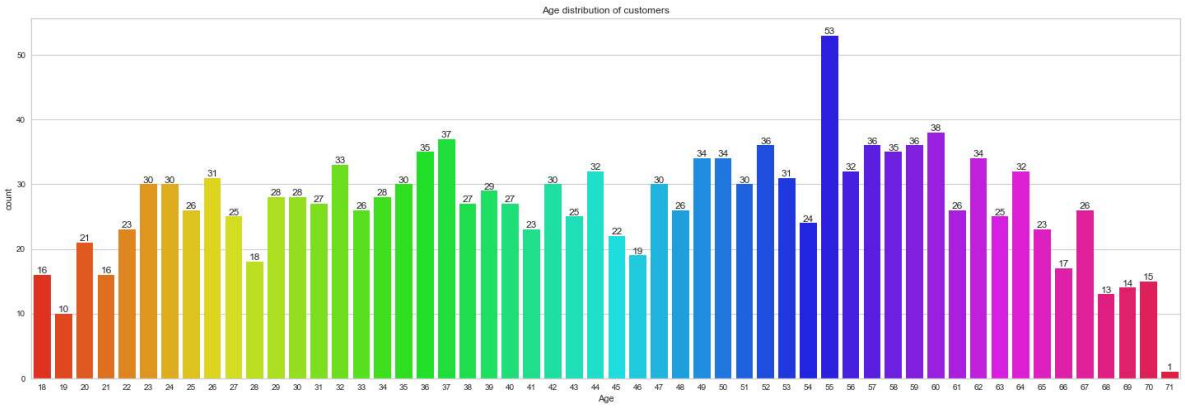
mcdonalds.info()

In [162]:
```
mcdonalds.info() #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 [163]:
```
#piechart related to gender
labels = ['Female', 'Male']
size = mcdonalds['Gender'].value_counts()
colors = ['red', 'green']
explode = [0, 0.1]
plt.rcParams['figure.figsize'] = (4, 4)
plt.pie(size, colors = colors, explode = explode, labels = labels, shadow = Tr
plt.title('Gender', fontsize = 20)
plt.axis('off')
plt.legend()
plt.show()


#counterplot for age
plt.rcParams['figure.figsize'] = (25, 8)
f = sns.countplot(x=mcdonalds['Age'],palette = 'hsv')
f.bar_label(f.containers[0])
plt.title('Age distribution of customers')
plt.show()
```





In [164]:
```
import matplotlib.pyplot as plt #importing matplotlib
import seaborn as sns #seaborn
```

In [165]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

In [166]:
```
import numpy as np

#extracting first 11 features
MD_x = pd.DataFrame(mcdonalds.iloc[:, 0:11])
MD_x
```

Out[166]:

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

1453 rows × 11 columns

In [167]:
```
#mean calculation of dat with yes as an attribute value
MD_x = (MD_x == "Yes").astype(int)
np.round(np.mean(MD_x, axis=0), 2)
```

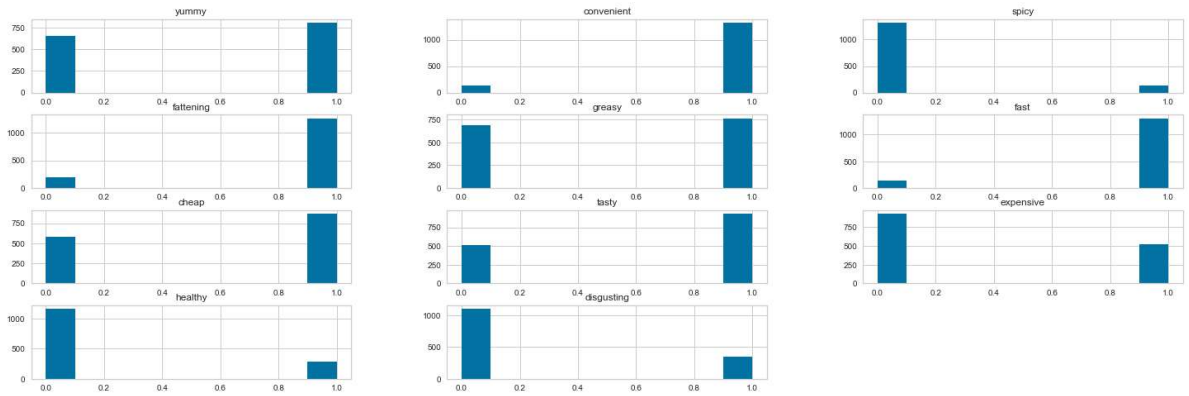
Out[167]:
```
yummy         0.55
convenient    0.91
spicy         0.09
fattening     0.87
greasy        0.53
fast          0.90
cheap         0.60
tasty         0.64
expensive     0.36
healthy       0.20
disgusting    0.24
dtype: float64
```

In [168]:
```
#converting categorical values using labelencoder
MD_x["yummy"]=le.fit_transform(MD_x["yummy"])
MD_x['convenient']=le.fit_transform(MD_x['convenient'])
MD_x['spicy']=le.fit_transform(MD_x['spicy'])
MD_x['fattening']=le.fit_transform(MD_x['fattening'])
MD_x['greasy']=le.fit_transform(MD_x['greasy'])
MD_x['fast']=le.fit_transform(MD_x['fast'])
MD_x['cheap']=le.fit_transform(MD_x['cheap'])
MD_x['tasty']=le.fit_transform(MD_x['tasty'])
MD_x['expensive']=le.fit_transform(MD_x['expensive'])
MD_x['healthy']=le.fit_transform(MD_x['healthy'])
MD_x['disgusting']=le.fit_transform(MD_x['disgusting'])
print(MD_x)
```

```
      yummy  convenient  spicy  fattening  greasy  fast  cheap  tasty  \
0         0           1      0          1       0     1      1      0
1         1           1      0          1       1     1      1      1
2         0           1      1          1       1     1      0      1
3         1           1      0          1       1     1      1      1
4         0           1      0          1       1     1      1      0
...     ...         ...    ...        ...     ...   ...    ...    ...
1448      0           1      0          1       1     0      0      0
1449      1           1      0          1       0     0      1      1
1450      1           1      0          1       0     1      0      1
1451      1           1      0          0       0     1      1      1
1452      0           1      0          1       1     0      0      0

      expensive  healthy  disgusting
0             1        0           0
1             1        0           0
2             1        1           0
3             0        0           1
4             0        1           0
...         ...      ...         ...
1448          1        0           1
1449          0        1           0
1450          1        0           0
1451          0        1           0
1452          1        0           1

[1453 rows x 11 columns]
```

In [169]:
```
MD_x.hist() #histplot
```

Out[169]:
```
array([[<AxesSubplot:title={'center':'yummy'}>,
        <AxesSubplot:title={'center':'convenient'}>,
        <AxesSubplot:title={'center':'spicy'}>],
       [<AxesSubplot:title={'center':'fattening'}>,
        <AxesSubplot:title={'center':'greasy'}>,
        <AxesSubplot:title={'center':'fast'}>],
       [<AxesSubplot:title={'center':'cheap'}>,
        <AxesSubplot:title={'center':'tasty'}>,
        <AxesSubplot:title={'center':'expensive'}>],
       [<AxesSubplot:title={'center':'healthy'}>,
        <AxesSubplot:title={'center':'disgusting'}>, <AxesSubplot:>]],
      dtype=object)
```



In [170]:
```
#fitting data with princial componnet analysis
from sklearn.decomposition import PCA
MD_pca = PCA().fit(MD_x)
```

In [171]:
```
#dataframe of 11 pca components
MD_pcadata=pd.DataFrame(MD_pca.components_,columns=["pca1","pca2","pca3","pca4
MD_pcadata
```

Out[171]:

	pca1	pca2	pca3	pca4	pca5	pca6	pca7	pca8	pc
0	-0.476933	-0.155332	-0.006356	0.116232	0.304443	-0.108493	-0.337186	-0.471514	0.329(
1	0.363790	0.016414	0.018809	-0.034094	-0.063839	-0.086972	-0.610633	0.307318	0.6012
2	-0.304444	-0.062515	-0.037019	-0.322359	-0.802373	-0.064642	-0.149310	-0.287265	0.0243
3	0.055162	-0.142425	0.197619	-0.354139	0.253960	-0.097363	0.118958	-0.002547	0.0678
4	-0.307535	0.277608	0.070620	-0.073405	0.361399	0.107930	-0.128973	-0.210899	-0.0031
5	0.170738	-0.347830	-0.355087	-0.406515	0.209347	-0.594632	-0.103241	-0.076914	-0.2613
6	-0.280519	-0.059738	0.707637	-0.385943	0.036170	-0.086846	-0.040449	0.360453	-0.0683
7	0.013041	-0.113079	0.375934	0.589622	-0.138241	-0.627799	0.140060	-0.072792	0.0295
8	0.572403	-0.018465	0.400280	-0.160512	-0.002847	0.166197	0.076069	-0.639086	0.0669
9	-0.110284	-0.665818	-0.075634	-0.005338	0.008707	0.239532	0.428087	0.079184	0.4543
10	0.045439	-0.541616	0.141730	0.250910	0.001642	0.339265	-0.489283	0.019552	-0.490(

In [172]:
```
#measures

print("Standard deviations")
print(np.round(MD_pca.explained_variance_ ** 0.5, 1))
print("variance ratio")
print(MD_pca.explained_variance_ratio_)
print("cummulative ratio")
print(np.cumsum(MD_pca.explained_variance_ratio_))
```

```
Standard deviations
[0.8 0.6 0.5 0.4 0.3 0.3 0.3 0.3 0.3 0.2 0.2]
variance ratio
[0.29944723 0.19279721 0.13304535 0.08309578 0.05948052 0.05029956
 0.0438491  0.03954779 0.0367609  0.03235329 0.02932326]
cummulative ratio
[0.29944723 0.49224445 0.6252898  0.70838558 0.7678661  0.81816566
 0.86201476 0.90156255 0.93832345 0.97067674 1.        ]
```

In [173]:
```
print("rotation :",MD_pca.components_.shape)
```

```
rotation : (11, 11)
```

comuns=MD_x.columns

In [174]:
```
#dataframe with relation between pca components and attributes
MD_pca.components_=np.array(MD_pca.components_)*-1
rotated_df=pd.DataFrame(MD_pca.components_,columns=["pca1","pca2","pca3","pca4
 'expensive', 'healthy', 'disgusting'])
print(rotated_df)
```

```
                pca1      pca2      pca3      pca4      pca5      pca6  \
yummy       0.476933  0.155332  0.006356 -0.116232 -0.304443  0.108493
convenient -0.363790 -0.016414 -0.018809  0.034094  0.063839  0.086972
spicy       0.304444  0.062515  0.037019  0.322359  0.802373  0.064642
fattening  -0.055162  0.142425 -0.197619  0.354139 -0.253960  0.097363
greasy      0.307535 -0.277608 -0.070620  0.073405 -0.361399 -0.107930
fast       -0.170738  0.347830  0.355087  0.406515 -0.209347  0.594632
cheap       0.280519  0.059738 -0.707637  0.385943 -0.036170  0.086846
tasty      -0.013041  0.113079 -0.375934 -0.589622  0.138241  0.627799
expensive  -0.572403  0.018465 -0.400280  0.160512  0.002847 -0.166197
healthy     0.110284  0.665818  0.075634  0.005338 -0.008707 -0.239532
disgusting -0.045439  0.541616 -0.141730 -0.250910 -0.001642 -0.339265

                pca7      pca8      pca9     pca10     pca11
yummy       0.337186  0.471514 -0.329042  0.213711 -0.374753
convenient  0.610633 -0.307318 -0.601286 -0.076593  0.139656
spicy       0.149310  0.287265 -0.024397 -0.192051  0.088571
fattening  -0.118958  0.002547 -0.067816 -0.763488 -0.369539
greasy      0.128973  0.210899  0.003125 -0.287846  0.729209
fast        0.103241  0.076914  0.261342  0.178226  0.210878
cheap       0.040449 -0.360453  0.068385  0.349616  0.026792
tasty      -0.140060  0.072792 -0.029539 -0.176303  0.167181
expensive  -0.076069  0.639086 -0.066996  0.185572  0.072483
healthy    -0.428087 -0.079184 -0.454399  0.038117  0.289592
disgusting  0.489283 -0.019552  0.490069 -0.157608  0.040662
```

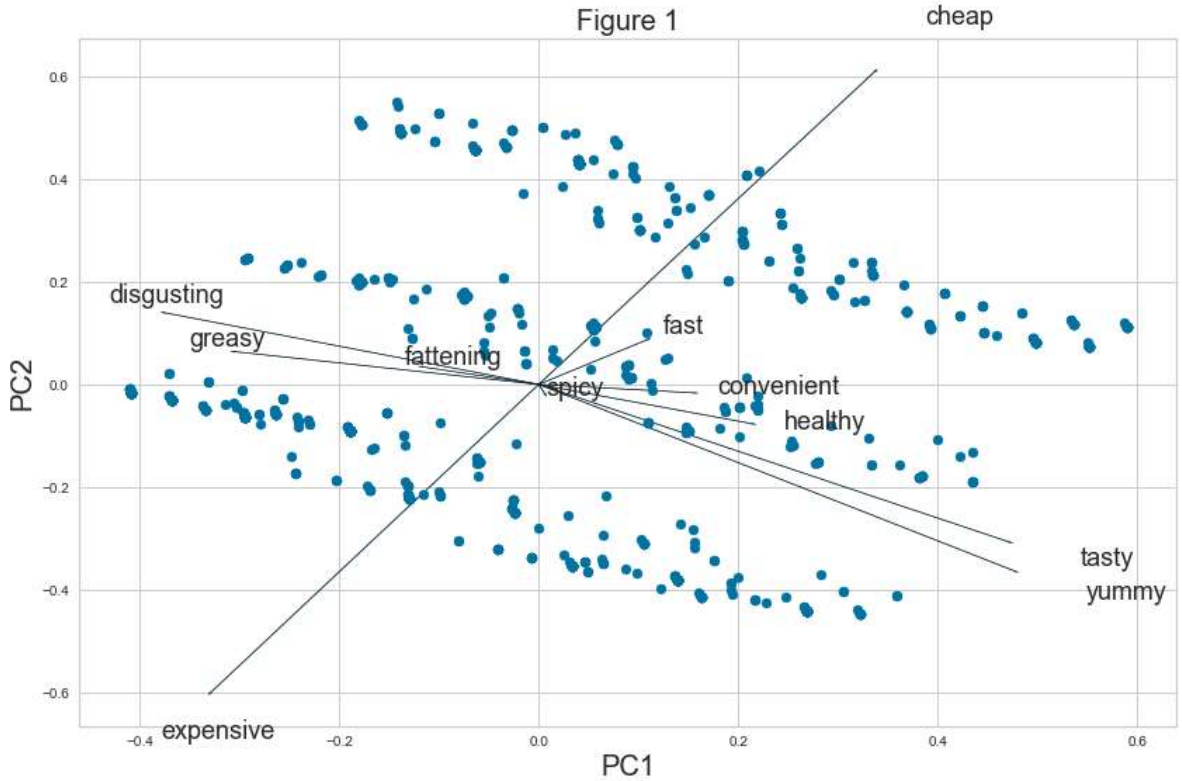
In [175]:
```
sns.heatmap(rotated_df,annot=True) #heatmap
```

Out[175]: <AxesSubplot:>

In [176]:
```
#for biplot
PC1 = PCA().fit_transform(MD_x)[:,0]
PC2 = PCA().fit_transform(MD_x)[:,1]
ldngs = MD_pca.components_
scalePC1 = 1.0/(PC1.max() - PC1.min())
scalePC2 = 1.0/(PC2.max() - PC2.min())
features = ['yummy' ,'convenient', 'spicy', 'fattening', 'greasy' ,'fast', 'ch
 'expensive', 'healthy', 'disgusting']
```

In [177]:
```
fig, ax = plt.subplots(figsize=(14, 9))

for i, feature in enumerate(features):
    ax.arrow(0, 0, ldngs[0, i],
             ldngs[1, i])
    ax.text(ldngs[0, i] * 1.15,
            ldngs[1, i] * 1.15,
            feature, fontsize=18)

ax.scatter(PC1 * scalePC1,PC2 * scalePC2)

ax.set_xlabel('PC1', fontsize=20)
ax.set_ylabel('PC2', fontsize=20)
ax.set_title('Figure 1', fontsize=20)
plt.figure()
```

Out[177]: <Figure size 1800x576 with 0 Axes>



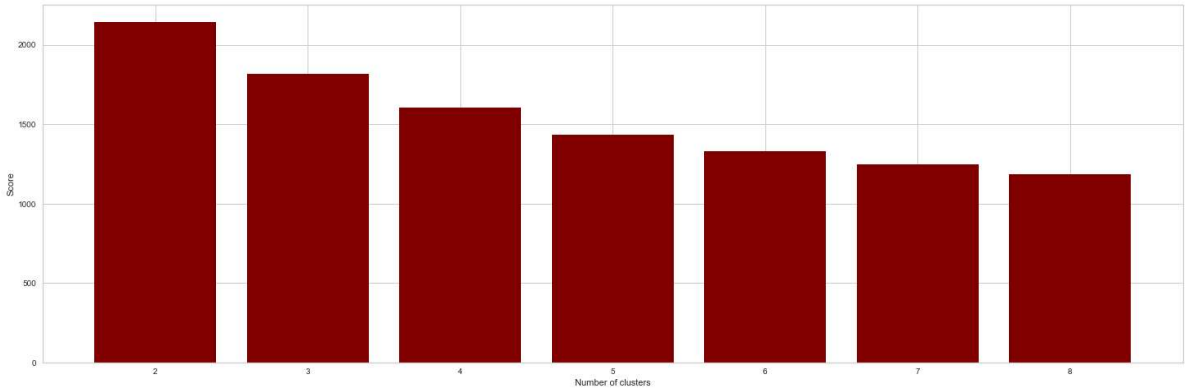
<Figure size 1800x576 with 0 Axes>

In [178]:
```
from sklearn.cluster import KMeans
```

In [179]:
```
#using kemans for clustering
k_range = range(2, 9)
scores_k=[]
kmlabels=[]
for k in k_range:
    km = KMeans(n_clusters=k, n_init=10, random_state=1234).fit(MD_x)
    kmlabels.append(km.labels_)
    scores_k.append(km.inertia_)
    print(km.labels_)
```

```
[1 0 0 ... 0 0 1]
[2 1 1 ... 1 0 2]
[1 3 3 ... 3 0 2]
[2 3 4 ... 4 0 1]
[2 5 0 ... 0 1 4]
[3 5 1 ... 1 6 4]
[4 2 5 ... 5 3 0]
```

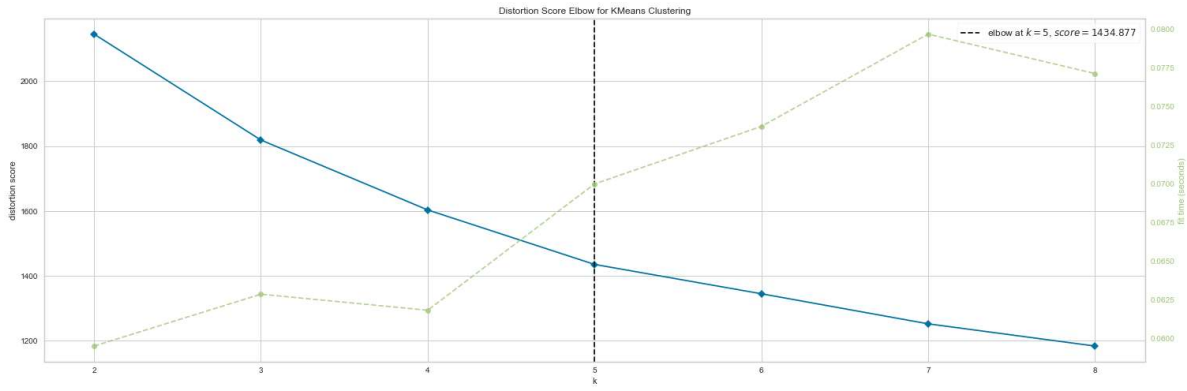
In [180]:
```
#barplot of segments and scores
plt.bar(list(range(2, 9)), scores_k,color='maroon')
plt.xlabel('Number of clusters')
plt.ylabel('Score')
plt.show()
```



In [181]:
```
! pip install yellowbrick
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: yellowbrick in c:\users\raghu\appdata\roaming
\python\python39\site-packages (1.5)
Requirement already satisfied: numpy>=1.16.0 in c:\programdata\anaconda3\lib
\site-packages (from yellowbrick) (1.21.5)
Requirement already satisfied: scipy>=1.0.0 in c:\programdata\anaconda3\lib\s
ite-packages (from yellowbrick) (1.7.3)
Requirement already satisfied: matplotlib!=3.0.0,>=2.0.2 in c:\programdata\an
aconda3\lib\site-packages (from yellowbrick) (3.5.1)
Requirement already satisfied: cycler>=0.10.0 in c:\programdata\anaconda3\lib
\site-packages (from yellowbrick) (0.11.0)
Requirement already satisfied: scikit-learn>=1.0.0 in c:\programdata\anaconda
3\lib\site-packages (from yellowbrick) (1.0.2)
Requirement already satisfied: packaging>=20.0 in c:\programdata\anaconda3\li
b\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (21.3)
Requirement already satisfied: python-dateutil>=2.7 in c:\programdata\anacond
a3\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (2.8.2)
Requirement already satisfied: pyparsing>=2.2.1 in c:\programdata\anaconda3\l
ib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (3.0.4)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\programdata\anaconda3
\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (1.3.2)
Requirement already satisfied: fonttools>=4.22.0 in c:\programdata\anaconda3
\lib\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (4.25.0)
Requirement already satisfied: pillow>=6.2.0 in c:\programdata\anaconda3\lib
\site-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (9.0.1)
Requirement already satisfied: six>=1.5 in c:\programdata\anaconda3\lib\site-
packages (from python-dateutil>=2.7->matplotlib!=3.0.0,>=2.0.2->yellowbrick)
(1.16.0)
Requirement already satisfied: joblib>=0.11 in c:\programdata\anaconda3\lib\s
ite-packages (from scikit-learn>=1.0.0->yellowbrick) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\programdata\anacond
a3\lib\site-packages (from scikit-learn>=1.0.0->yellowbrick) (2.2.0)
```

In [182]:
```
from yellowbrick.cluster import KElbowVisualizer
model = KMeans()
visualizer = KElbowVisualizer(model, k=(2, 9))

visualizer.fit(MD_x) # Fit the data to the visualizer
visualizer.show() # Finalize and render the figure
plt.show()
```

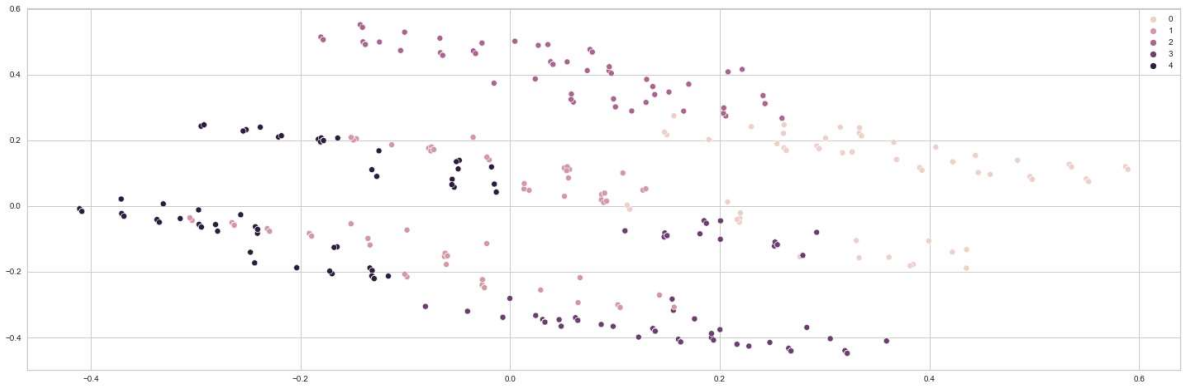


In [183]:
```
#selected 5 clusters as predicted from elbowmethod
kmeans = KMeans(n_clusters=5, init='k-means++', random_state=0).fit(MD_x)
MD_x['cluster_num'] = kmeans.labels_
print ('Labels:', kmeans.labels_)
print ('WCSS:', kmeans.inertia_)
```

```
Labels: [3 1 2 ... 2 4 0]
WCSS: 1434.6060971914783
```

In [184]:
```
#distribution of datapoints using pc1 and pc2

sns.scatterplot(x=PC1 * scalePC1 , y=PC2 * scalePC2, hue=kmeans.labels_)
```

Out[184]:
```
<AxesSubplot:>
```



In [185]:
```
#prediction
y_pred=kmeans.predict(MD_x[['yummy' ,'convenient', 'spicy', 'fattening', 'grea
 'expensive', 'healthy', 'disgusting']])
print(y_pred)
```

```
[3 1 2 ... 2 4 0]
```

In [186]:
```
# confusion matrix
from sklearn.metrics import confusion_matrix
cf=confusion_matrix(MD_x['cluster_num'],y_pred)
cf
```

Out[186]:
```
array([[232,   0,   0,   0,   0],
       [  0, 309,   0,   0,   0],
       [  0,   0, 257,   0,   0],
       [  0,   0,   0, 264,   0],
       [  0,   0,   0,   0, 391]], dtype=int64)
```

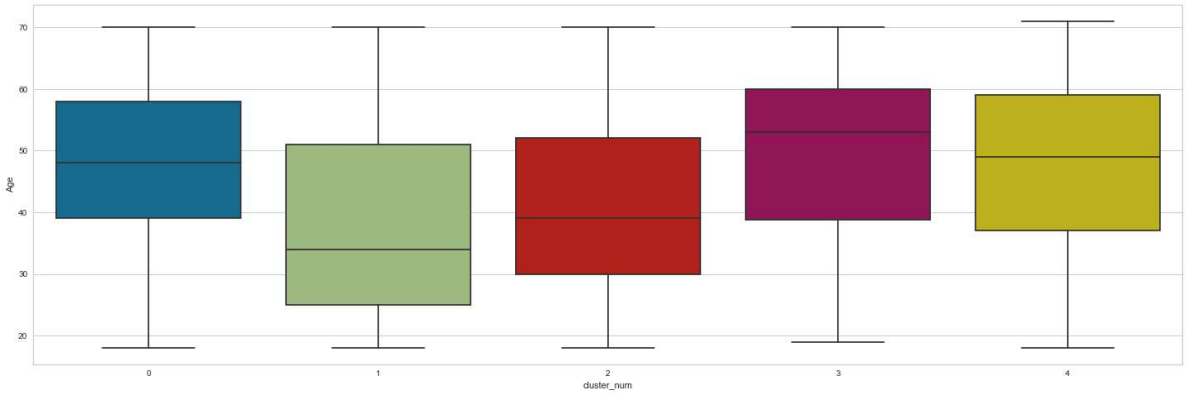
In [187]:
```
#calculating adjusted random score
from sklearn.metrics import adjusted_rand_score
score = adjusted_rand_score(MD_x['cluster_num'],y_pred)
```

In [188]:
```
print(score)
```

```
1.0
```

In [210]:
```
sns.boxplot(x=MD_x['cluster_num'],y=mcdonalds['Age'])
```

Out[210]:
```
<AxesSubplot:xlabel='cluster_num', ylabel='Age'>
```

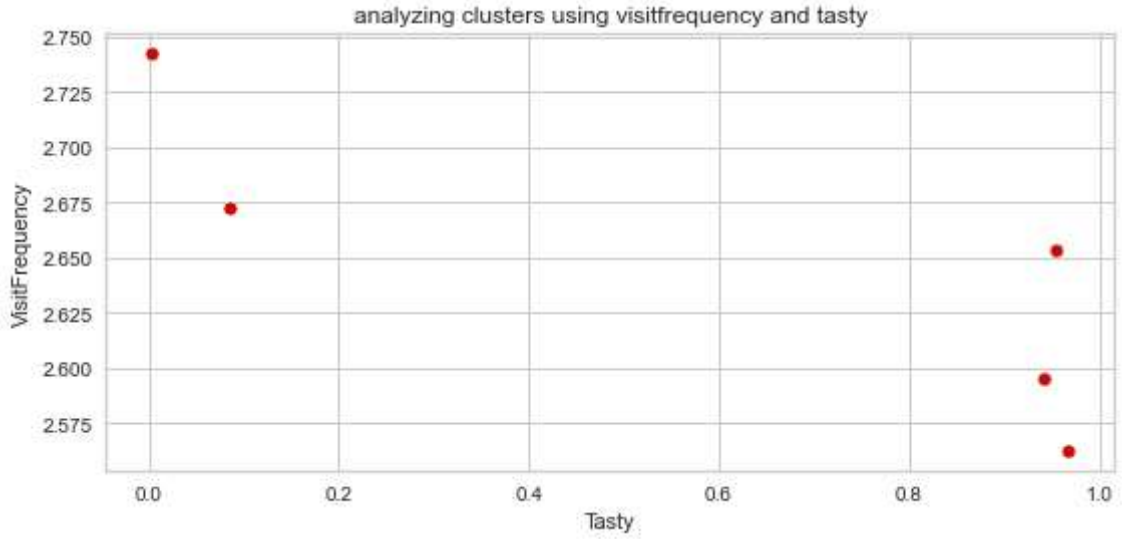


In [206]:
```
# groupby method

mcdonalds['cluster_num']=kmeans.labels_

mcdonalds['expensive'] = LabelEncoder().fit_transform(mcdonalds['expensive'])
expensive = mcdonalds.groupby('cluster_num')['expensive'].mean() #grouping exp
expensive=expensive.to_frame().reset_index()


mcdonalds['VisitFrequency'] = LabelEncoder().fit_transform(mcdonalds['VisitFre
frequency = mcdonalds.groupby('cluster_num')['VisitFrequency'].mean() #groupir
frequency=frequency.to_frame().reset_index()


mcdonalds['tasty'] = LabelEncoder().fit_transform(mcdonalds['tasty'])
tasty = mcdonalds.groupby('cluster_num')['tasty'].mean()  #grouping tatsy with
tasty=tasty.to_frame().reset_index()


criteria = expensive.merge(frequency, on='cluster_num', how='left')
criteria =criteria.merge(tasty, on='cluster_num', how='left')
criteria
```

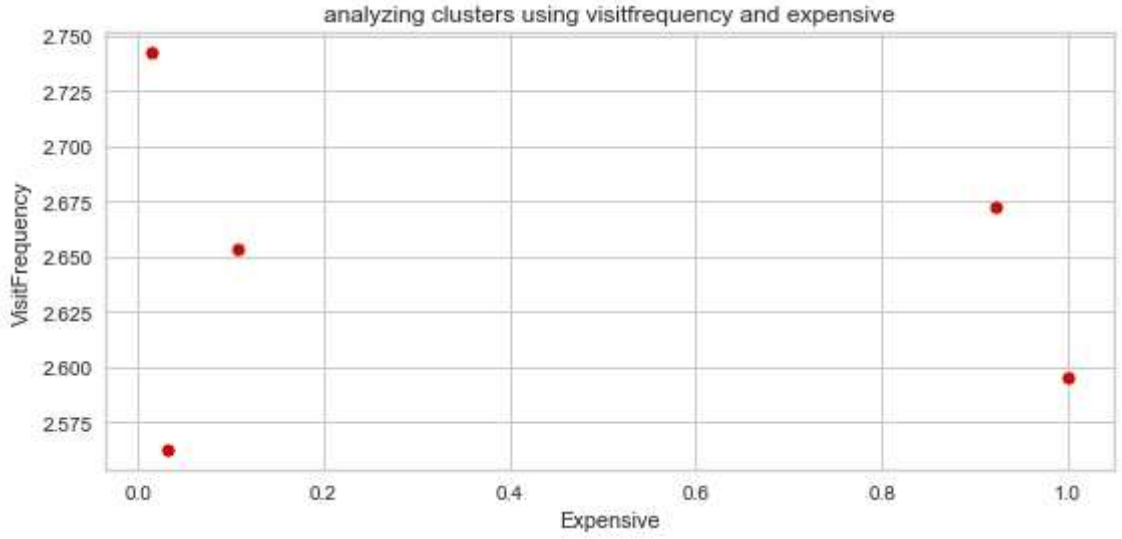
Out[206]:

	cluster_num	expensive	VisitFrequency	tasty
0	0	0.922414	2.672414	0.086207
1	1	0.106796	2.653722	0.954693
2	2	1.000000	2.595331	0.941634
3	3	0.015152	2.742424	0.003788
4	4	0.030691	2.562660	0.966752

In [207]:
```
#relation between expensive and visit
plt.figure(figsize = (9,4))
sns.scatterplot(y = "VisitFrequency", x = "tasty",data=criteria, color="r")
plt.title("analyzing clusters using visitfrequency and tasty")
plt.ylabel("VisitFrequency")
plt.xlabel("Tasty")
plt.show()
```



In [208]:
```
#reletion between tasty and visit

plt.figure(figsize = (9,4))
sns.scatterplot(y = "VisitFrequency", x = "expensive",data=criteria, color="r"
plt.title("analyzing clusters using visitfrequency and expensive")
plt.ylabel("VisitFrequency")
plt.xlabel("Expensive")
plt.show()
```




```
-----Therefore we can analyze that through the mcdonalds dataset.
The less expensive there is more visitFrequency,
```

```
More tasty there is average visitFrequency.This helps in market segmentation
of mcdonalds dataset.
```