Travel Review Clustering

Context:

This data set is populated by capturing user ratings from Google reviews. Reviews on attractions from 24 categories across Europe are considered. Google user rating ranges from 1 to 5 and average user rating per category is calculated.

Data Definition:

Input variables:

1) **User** Unique user id

2) **Attribute 1:** Average ratings on churches

3) **Attribute 2:** Average ratings on resorts

4) **Attribute 3:** Average ratings on beaches

5) **Attribute 4:** Average ratings on parks

6) **Attribute 5:** Average ratings on theatres

7) **Attribute 6:** Average ratings on museums

8) **Attribute 7:** Average ratings on malls

9) **Attribute 8:** Average ratings on zoo

10) **Attribute 9:** Average ratings on restaurants

11) **Attribute 10:** Average ratings on pubs/bars

12) **Attribute 11:** Average ratings on local services

13) **Attribute 12:** Average ratings on burger/pizza shops

14) **Attribute 13:** Average ratings on hotels/other lodgings

15) **Attribute 14:** Average ratings on juice bars

16) **Attribute 15:** Average ratings on art galleries

17) **Attribute 16:** Average ratings on dance clubs

18) **Attribute 17:** Average ratings on swimming pools

19) **Attribute 18:** Average ratings on gyms

20) **Attribute 19:** Average ratings on bakeries

21) **Attribute 20:** Average ratings on beauty & spas

22) **Attribute 21:** Average ratings on cafes

23) **Attribute 22:** Average ratings on view points

24) **Attribute 23:** Average ratings on monuments

25) **Attribute 24:** Average ratings on gardens

1. Import Packages

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
# Set default setting of seaborn
sns.set()
```

2. Read the Data

In [2]:
```
# read the data
raw_data = pd.read_csv(r"C:\Users\hp\Downloads\google_review_ratings.csv")

# print the first five rows of the data
raw_data.head()
```

Out[2]:

	User	Category 1	Category 2	Category 3	Category 4	Category 5	Category 6	Category 7	Category 8	Cate
0	User 1	0.0	0.0	3.63	3.65	5.0	2.92	5.0	2.35	
1	User 2	0.0	0.0	3.63	3.65	5.0	2.92	5.0	2.64	
2	User 3	0.0	0.0	3.63	3.63	5.0	2.92	5.0	2.64	
3	User 4	0.0	0.5	3.63	3.63	5.0	2.92	5.0	2.35	
4	User 5	0.0	0.0	3.63	3.63	5.0	2.92	5.0	2.64	

5 rows × 26 columns

3. Data Types and Dimensions

In [3]:
```
# check the data types for variables
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5456 entries, 0 to 5455
Data columns (total 26 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   User         5456 non-null   object
 1   Category 1   5456 non-null   float64
 2   Category 2   5456 non-null   float64
 3   Category 3   5456 non-null   float64
 4   Category 4   5456 non-null   float64
 5   Category 5   5456 non-null   float64
 6   Category 6   5456 non-null   float64
 7   Category 7   5456 non-null   float64
 8   Category 8   5456 non-null   float64
 9   Category 9   5456 non-null   float64
 10  Category 10  5456 non-null   float64
 11  Category 11  5456 non-null   object
 12  Category 12  5455 non-null   float64
 13  Category 13  5456 non-null   float64
 14  Category 14  5456 non-null   float64
 15  Category 15  5456 non-null   float64
 16  Category 16  5456 non-null   float64
 17  Category 17  5456 non-null   float64
 18  Category 18  5456 non-null   float64
 19  Category 19  5456 non-null   float64
 20  Category 20  5456 non-null   float64
 21  Category 21  5456 non-null   float64
 22  Category 22  5456 non-null   float64
 23  Category 23  5456 non-null   float64
 24  Category 24  5455 non-null   float64
 25  Unnamed: 25  2 non-null      float64
dtypes: float64(24), object(2)
memory usage: 1.1+ MB
```

Before manipulation of data, let's check dimension of dataset

In [4]:
```
# get the shape
print(raw_data.shape)
```

```
(5456, 26)
```

We see that dataframe has 5456 instances and 26 features

4. Data Manipulation

Here we will change features names and remove redundant feature

Droping redundant feature

In [5]:
```
# Using drop() function to remove redundant feature
data = raw_data.drop(['Unnamed: 25'] ,axis = 1)
data.head()
```

Out[5]:

	User	Category 1	Category 2	Category 3	Category 4	Category 5	Category 6	Category 7	Category 8	Cate
0	User 1	0.0	0.0	3.63	3.65	5.0	2.92	5.0	2.35	
1	User 2	0.0	0.0	3.63	3.65	5.0	2.92	5.0	2.64	
2	User 3	0.0	0.0	3.63	3.63	5.0	2.92	5.0	2.64	
3	User 4	0.0	0.5	3.63	3.63	5.0	2.92	5.0	2.35	
4	User 5	0.0	0.0	3.63	3.63	5.0	2.92	5.0	2.64	

5 rows × 25 columns

Renaming features

To perform the action we will create list containing actual names of features

In [6]:
```
column_names = ['user_id', 'churches', 'resorts', 'beaches', 'parks',
                'theatres', 'museums', 'malls', 'zoo', 'restaurants',
                'pubs_bars', 'local_services', 'burger_pizza_shops',
                'hotels_other_lodgings', 'juice_bars', 'art_galleries',
                'dance_clubs', 'swimming_pools', 'gyms', 'bakeries',
                'beauty_spas', 'cafes', 'view_points', 'monuments', 'gardens']
```

Applying above columns names to dataframe columns

In [7]:
```
data.columns = column_names
```

Checking if changes are applied or not

In [8]:
```
data.head()
```

Out[8]:

	user_id	churches	resorts	beaches	parks	theatres	museums	malls	zoo	restaurants	...	ar
0	User 1	0.0	0.0	3.63	3.65	5.0	2.92	5.0	2.35	2.33	...	
1	User 2	0.0	0.0	3.63	3.65	5.0	2.92	5.0	2.64	2.33	...	
2	User 3	0.0	0.0	3.63	3.63	5.0	2.92	5.0	2.64	2.33	...	
3	User 4	0.0	0.5	3.63	3.63	5.0	2.92	5.0	2.35	2.33	...	
4	User 5	0.0	0.0	3.63	3.63	5.0	2.92	5.0	2.64	2.33	...	

5 rows × 25 columns

Columns are renamed

In [9]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5456 entries, 0 to 5455
Data columns (total 25 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   user_id                5456 non-null   object
 1   churches               5456 non-null   float64
 2   resorts                5456 non-null   float64
 3   beaches                5456 non-null   float64
 4   parks                  5456 non-null   float64
 5   theatres               5456 non-null   float64
 6   museums                5456 non-null   float64
 7   malls                  5456 non-null   float64
 8   zoo                    5456 non-null   float64
 9   restaurants            5456 non-null   float64
 10  pubs_bars              5456 non-null   float64
 11  local_services         5456 non-null   object
 12  burger_pizza_shops     5455 non-null   float64
 13  hotels_other_lodgings  5456 non-null   float64
 14  juice_bars             5456 non-null   float64
 15  art_galleries          5456 non-null   float64
 16  dance_clubs            5456 non-null   float64
 17  swimming_pools         5456 non-null   float64
 18  gyms                   5456 non-null   float64
 19  bakeries               5456 non-null   float64
 20  beauty_spas            5456 non-null   float64
 21  cafes                  5456 non-null   float64
 22  view_points            5456 non-null   float64
 23  monuments              5456 non-null   float64
 24  gardens                5455 non-null   float64
dtypes: float64(23), object(2)
memory usage: 1.0+ MB
```

Changing data type of local_services

In [10]:
```
data['local_services'] = pd.to_numeric(data['local_services'],errors = 'coerce')
```

In [11]:
```
data['local_services'].unique() # Getting all unique values
```

Out[11]:
```
array([1.7 , 1.73, 1.71, 1.69, 1.67, 1.65, 1.66, 1.64, 1.63, 5.  , 1.56,
       1.55, 1.53, 1.52, 1.51, 1.5 , 1.49, 1.48, 1.46, 2.13, 2.12, 2.1 ,
       2.09, 2.08, 2.07, 2.06, 2.05, 2.04, 2.03, 2.01, 2.  , 1.98, 1.97,
       1.95, 1.93, 1.91, 1.9 , 1.88, 1.86, 1.84, 1.83, 1.81, 1.79, 1.77,
       1.74, 1.72, 1.68, 1.61, 1.6 , 1.59, 1.58, 1.99, 2.47, 2.35, 2.48,
       2.59, 3.39, 2.31, 2.78, 2.79, 2.77, 2.76, 2.75, 2.74, 2.72, 2.71,
       2.69, 2.68, 2.66, 2.65, 2.63, 2.61, 2.57, 2.55, 2.53, 2.51, 2.49,
       1.94, 1.92, 1.82, 1.76, 1.62, 4.08, 4.04, 4.02, 4.01, 4.  , 3.99,
       3.98, 3.96, 3.95, 3.94, 3.93, 3.91, 3.9 , 3.88, 3.87, 3.86, 3.84,
       3.85, 2.17, 3.  , 2.99, 2.82, 2.83, 2.81, 2.8 , 2.73, 2.67, 2.64,
       2.62, 2.54, 2.52, 2.5 , 2.02, 1.57, 1.54, 1.47, 1.87, 1.85, 1.78,
       2.6 , 1.45, 1.43, 1.41, 1.39, 1.37, 1.36, 1.34, 1.32, 1.3 , 1.28,
       1.26, 1.24, 1.22, 1.2 , 1.18, 1.16, 1.14, 1.12, 1.1 , 1.08, 1.07,
       1.05, 1.03, 1.01, 0.99, 0.97, 0.96, 0.94, 0.92, 0.9 , 0.88, 0.87,
       0.85, 0.83, 0.82, 0.84, 0.86, 3.35, 3.34, 3.33, 3.32, 3.31, 3.3 ,
       3.29, 3.28, 3.27, 3.26, 3.25, 3.24, 3.22, 3.21, 3.19, 3.17, 3.16,
       3.14, 3.08, 3.06, 3.04, 3.02, 2.98, 2.97, 2.94, 2.92, 2.91, 2.89,
       2.87, 2.85, 2.7 , 2.28, 2.26, 2.25, 2.45, 2.43, 2.41, 2.39, 2.37,
       2.34, 2.32, 2.3 , 2.24, 2.23, 2.21, 2.19, 2.18, 2.15, 2.14, 2.11,
       1.89, 1.75, 1.8 , 1.44, 1.42, 1.17, 1.19, 1.35, 1.38, 1.33, 1.31,
       1.29, 1.27, 1.25, 1.23, 1.21, 1.06, 1.04, 1.02, 1.  , 0.98, 0.95,
       0.93, 0.91, 4.95, 4.96, 3.07, 2.9 , 2.88, 2.84, 2.22, 2.29, 2.27,
       2.2 , 1.96, 3.82, 3.11, 2.93, 2.16, 3.74, 3.72, 2.58, 2.44, 2.42,
       2.4 , 2.33, 1.4 , 1.09, 3.15, 3.13, 3.12, 3.1 , 3.09, 2.96, 2.95,
       2.86, 2.38, 2.36, 3.56, 0.89, 0.81, 4.98, 3.4 , 4.06, 4.05, 4.03,
       3.97, 3.03, 3.01, 3.18, 1.15, 1.13, 1.11, 3.23, 2.46,  nan, 0.8 ,
       4.94, 3.92, 3.2 , 3.05, 0.79, 0.78, 4.09, 4.97, 2.56, 3.54])
```

Drop rows where local_services is equal to '2\t2.'

In [12]:
```
# Get row number where local services is invalid
data[data['local_services'] == '2\t2.']['local_services']
```

Out[12]:
```
Series([], Name: local_services, dtype: float64)
```

Droping row 2712

In [13]:
```
data = data.drop(data[data['local_services'] == '2\t2.'].index)
```

Now, we can changing data type of local_services

In [14]:
```
data[['local_services']] = data[['local_services']].apply(pd.to_numeric)
```

In [15]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5456 entries, 0 to 5455
Data columns (total 25 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   user_id                5456 non-null   object 
 1   churches               5456 non-null   float64
 2   resorts                5456 non-null   float64
 3   beaches                5456 non-null   float64
 4   parks                  5456 non-null   float64
 5   theatres               5456 non-null   float64
 6   museums                5456 non-null   float64
 7   malls                  5456 non-null   float64
 8   zoo                    5456 non-null   float64
 9   restaurants            5456 non-null   float64
 10  pubs_bars              5456 non-null   float64
 11  local_services         5455 non-null   float64
 12  burger_pizza_shops     5455 non-null   float64
 13  hotels_other_lodgings  5456 non-null   float64
 14  juice_bars             5456 non-null   float64
 15  art_galleries          5456 non-null   float64
 16  dance_clubs            5456 non-null   float64
 17  swimming_pools         5456 non-null   float64
 18  gyms                   5456 non-null   float64
 19  bakeries               5456 non-null   float64
 20  beauty_spas            5456 non-null   float64
 21  cafes                  5456 non-null   float64
 22  view_points            5456 non-null   float64
 23  monuments              5456 non-null   float64
 24  gardens                5455 non-null   float64
dtypes: float64(24), object(1)
memory usage: 1.0+ MB
```

All the features are numeric except the user_id as it is categorical in nature

In [16]:
```
data_manipulated = data.copy(deep =True ) # Creating a copy of dataframe
```

5. Statistical Summary

In [17]:
```
# data frame with numerical features
data_manipulated.describe()
```

Out[17]:

	churches	resorts	beaches	parks	theatres	museums	malls
count	5456.000000	5456.000000	5456.000000	5456.000000	5456.000000	5456.00000	5456.000000
mean	1.455720	2.319707	2.489331	2.796886	2.958941	2.89349	3.351395
std	0.827604	1.421438	1.247815	1.309159	1.339056	1.28240	1.413492
min	0.000000	0.000000	0.000000	0.830000	1.120000	1.11000	1.120000
25%	0.920000	1.360000	1.540000	1.730000	1.770000	1.79000	1.930000
50%	1.340000	1.905000	2.060000	2.460000	2.670000	2.68000	3.230000
75%	1.810000	2.682500	2.740000	4.092500	4.312500	3.84000	5.000000
max	5.000000	5.000000	5.000000	5.000000	5.000000	5.00000	5.000000

8 rows × 24 columns

In [18]:
```
# data frame with categorical features
data.describe(include='object')
```

Out[18]:

	user_id
count	5456
unique	5456
top	User 1
freq	1

6. Handling Missing Values

If the missing values are not handled properly we may end up drawing an inaccurate inference about the data. Due to improper handling, the result obtained will differ from the ones where the missing values are present.

In [19]:
```
# get the count of missing values
missing_values = data_manipulated.isnull().sum()

# print the count of missing values
print(missing_values)
```

```
user_id                  0
churches                 0
resorts                  0
beaches                  0
parks                    0
theatres                 0
museums                  0
malls                    0
zoo                      0
restaurants              0
pubs_bars                0
local_services           1
burger_pizza_shops       1
hotels_other_lodgings    0
juice_bars               0
art_galleries            0
dance_clubs              0
swimming_pools           0
gyms                     0
bakeries                 0
beauty_spas              0
cafes                    0
view_points              0
monuments                0
gardens                  1
dtype: int64
```

In [21]:
```
data_manipulated.drop('user_id',axis=1,inplace=True)
```

In [22]:
```
data_no_missing = data_manipulated.fillna(data_manipulated.mean())
```

In [23]:
```
# Rechecking missing values
missing_values = data_no_missing.isnull().sum()

# print the count of missing values
print(missing_values)
```

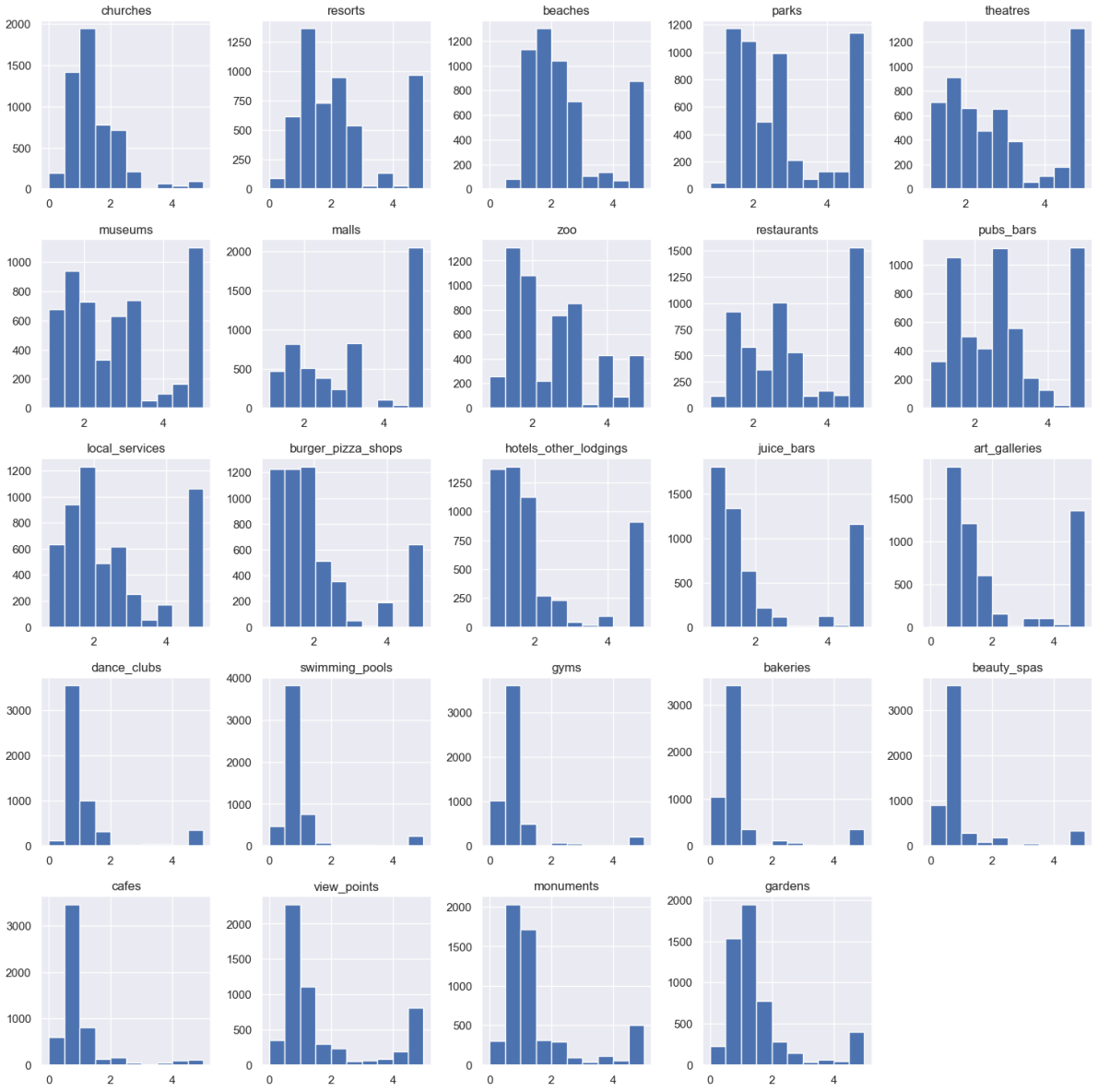
```
churches                 0
resorts                  0
beaches                  0
parks                    0
theatres                 0
museums                  0
malls                    0
zoo                      0
restaurants              0
pubs_bars                0
local_services           0
burger_pizza_shops       0
hotels_other_lodgings    0
juice_bars               0
art_galleries            0
dance_clubs              0
swimming_pools           0
gyms                     0
bakeries                 0
beauty_spas              0
cafes                    0
view_points              0
monuments                0
gardens                  0
dtype: int64
```

There are no missing values present in the data.

7. Exploratory Data Analysis

PDF's of features

In [24]:
```
fig = data_no_missing.hist(figsize = (18,18))
```



Lets check if users have rated all the features

In [25]:
```
data_description = data_no_missing.describe()
rated = data_description.loc['min'] > 0
rated[rated]
```

Out[25]:
```
parks                 True
theatres              True
museums               True
malls                 True
zoo                   True
restaurants           True
pubs_bars             True
local_services        True
burger_pizza_shops    True
hotels_other_lodgings True
juice_bars            True
Name: min, dtype: bool
```

Visualizing number of reviews for each category

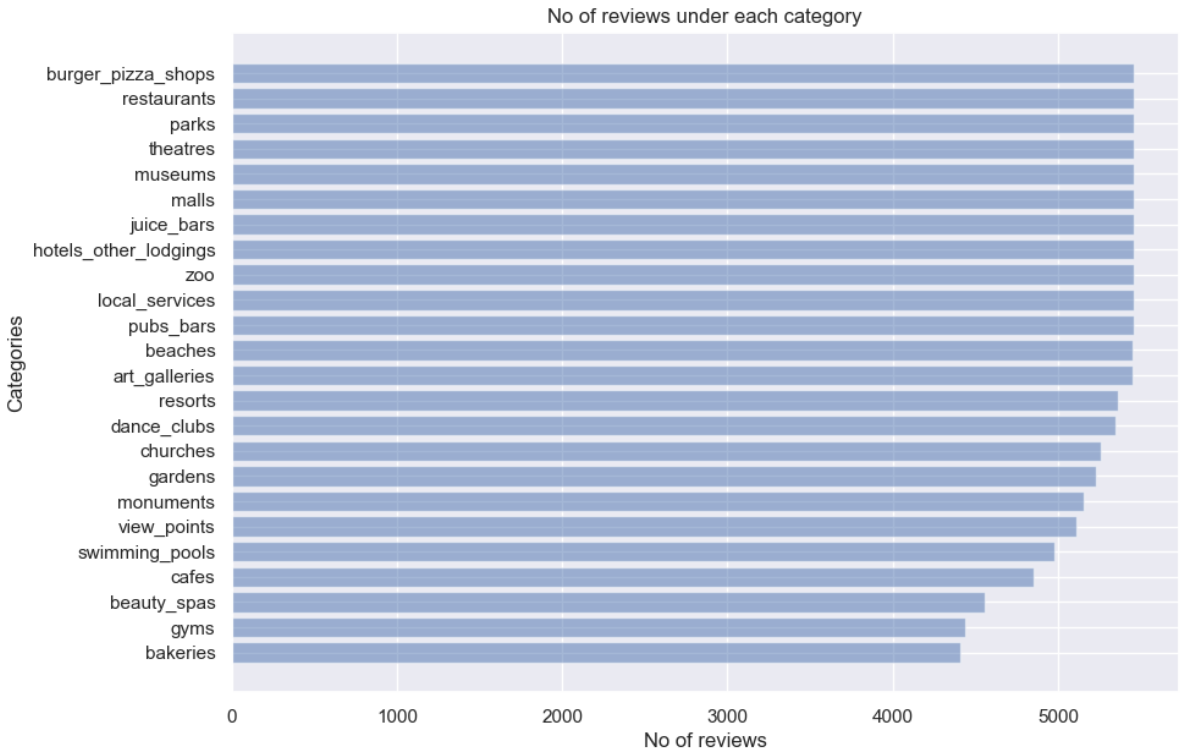
In [26]:
```
# Creating the dataframe containg number of review for each feature
reviews = data_no_missing[column_names[1:]].astype(bool).sum(axis=0).sort_values()
reviews
```

Out[26]:
```
bakeries              4410
gyms                  4439
beauty_spas           4560
cafes                 4852
swimming_pools        4977
view_points           5111
monuments             5154
gardens               5231
churches              5261
dance_clubs           5344
resorts               5366
art_galleries         5452
beaches               5452
pubs_bars             5456
local_services        5456
zoo                   5456
hotels_other_lodgings 5456
juice_bars            5456
malls                 5456
museums               5456
theatres              5456
parks                 5456
restaurants           5456
burger_pizza_shops    5456
dtype: int64
```

In [27]:
```
column_names = data_no_missing.columns.values
```

In [29]:
```
plt.figure(figsize=(10,7))
plt.barh(np.arange(len(column_names[0:])), reviews.values, align='center', alpha=0.
plt.yticks(np.arange(len(column_names[0:])), reviews.index)
plt.xlabel('No of reviews')
plt.ylabel('Categories')
plt.title('No of reviews under each category')
```

Out[29]:
```
Text(0.5, 1.0, 'No of reviews under each category')
```



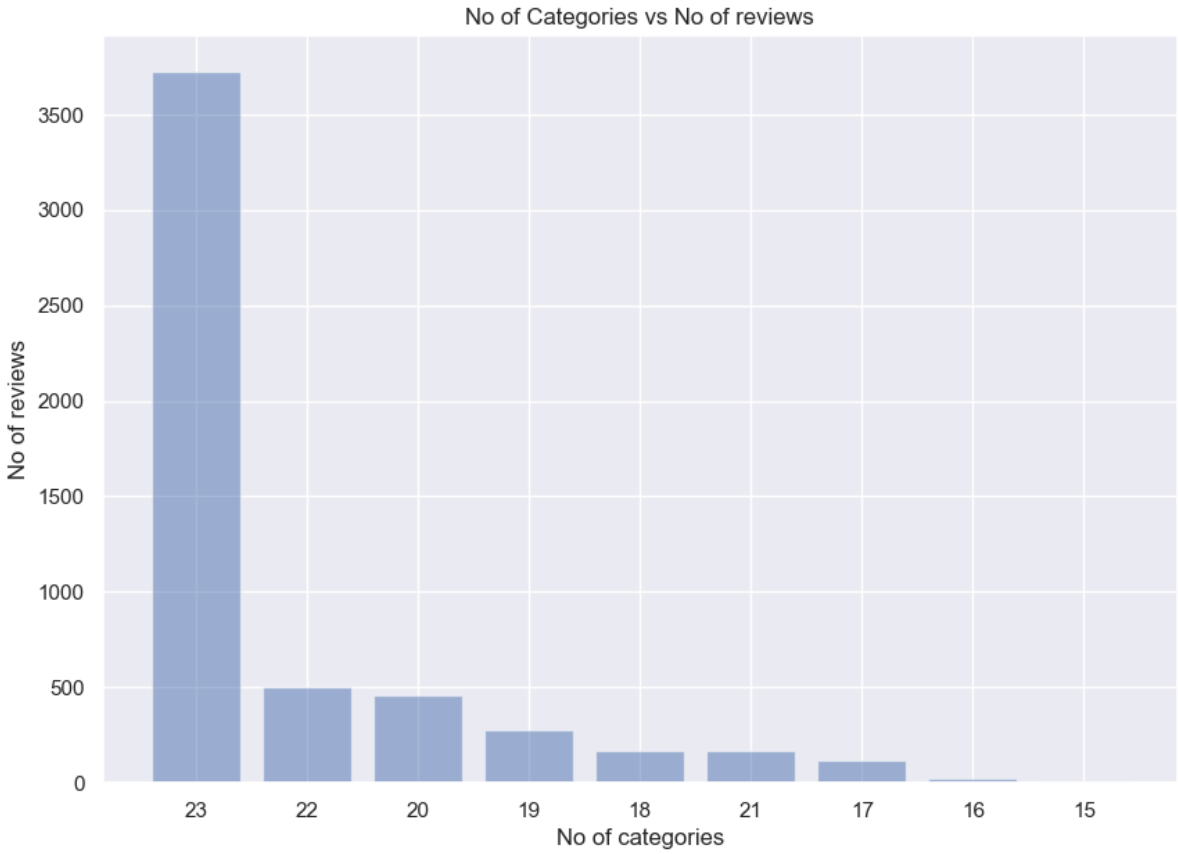
Now let's check how many users have given reviews to the features

In [30]:
```
# Creating a dataframe to store number of reviews by users
no_of_reviews = data_no_missing[column_names[1:]].astype(bool).sum(axis=1).value_cc
no_of_reviews
```

Out[30]:
```
23    3725
22     505
20     461
19     277
18     170
21     167
17     119
16      26
15       6
Name: count, dtype: int64
```

In [31]:
```
# Plotting the number of customers vs numbe of review
plt.figure(figsize=(10,7))
plt.bar(np.arange(len(no_of_reviews)), no_of_reviews.values, align='center', alpha=
plt.xticks(np.arange(len(no_of_reviews)), no_of_reviews.index)
plt.ylabel('No of reviews')
plt.xlabel('No of categories')
plt.title('No of Categories vs No of reviews')
```

Out[31]:
```
Text(0.5, 1.0, 'No of Categories vs No of reviews')
```



Conclusion

Around 3500 users have given a rating for all the 24 categories and the least no of rating given by a user is 15. So for users with lesser number of ratings a recommender system can be built

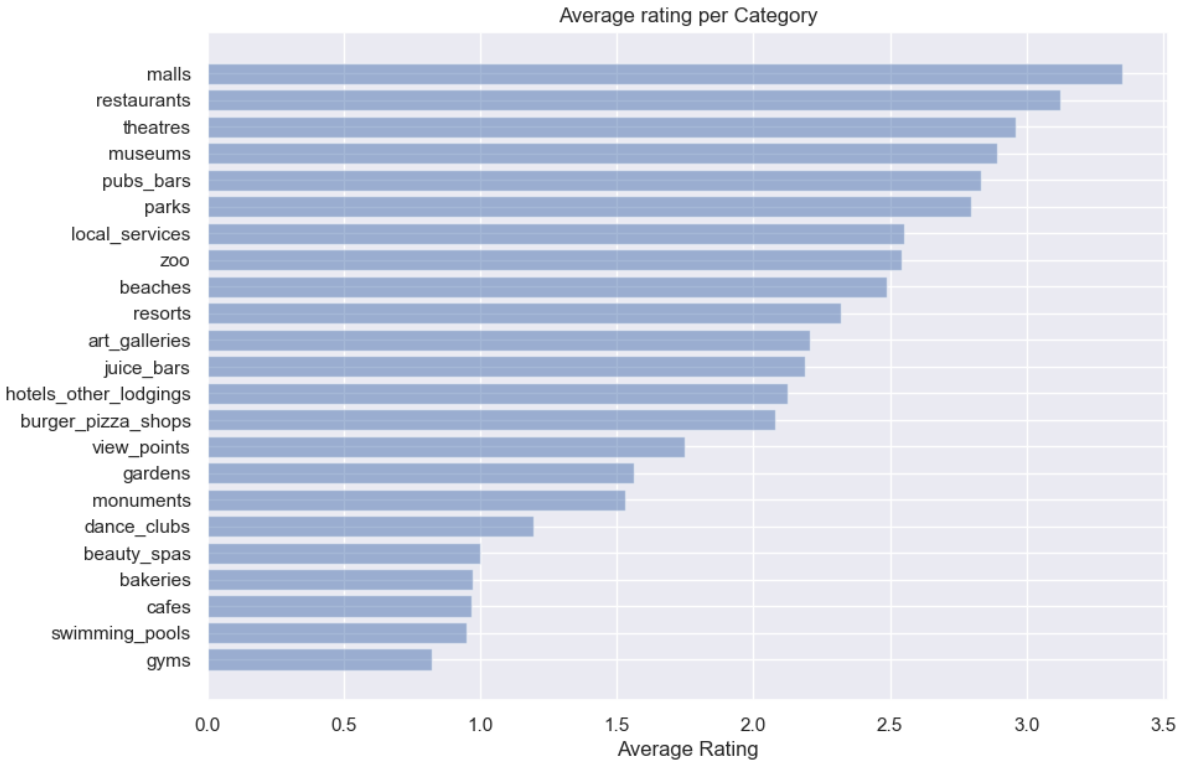
Now let's check Average rating per feature

In [32]:
```
# Creating a dataframe to store average rating for each feature
avg_rating = data_no_missing[column_names[1:]].mean() # average rating is calculate
avg_rating = avg_rating.sort_values() # sorting the rating in increasing order
avg_rating
```

Out[32]:
```
gyms                    0.822414
swimming_pools          0.949203
cafes                   0.965838
bakeries                0.969811
beauty_spas             1.000071
dance_clubs             1.192801
monuments               1.531453
gardens                 1.560755
view_points             1.750537
burger_pizza_shops      2.078339
hotels_other_lodgings   2.125511
juice_bars              2.190861
art_galleries           2.206573
resorts                 2.319707
beaches                 2.489331
zoo                     2.540795
local_services          2.550071
parks                   2.796886
pubs_bars               2.832729
museums                 2.893490
theatres                2.958941
restaurants             3.126019
malls                   3.351395
dtype: float64
```

In [33]:
```
# Plotting average rating plots
plt.figure(figsize=(10,7))
plt.barh(np.arange(len(column_names[1:])), avg_rating.values, align='center', alpha
plt.yticks(np.arange(len(column_names[1:])), avg_rating.index)
plt.xlabel('Average Rating')
plt.title('Average rating per Category')
```

Out[33]:
```
Text(0.5, 1.0, 'Average rating per Category')
```



Creating a id column

In [35]:
```
data_final = data_1.copy(deep = True)
data_final.head()
```

Out[35]:

	churches	resorts	beaches	parks	theatres	museums	malls	zoo	restaurants	pubs_bars	...
0	0.0	0.0	3.63	3.65	5.0	2.92	5.0	2.35	2.33	2.64	...
1	0.0	0.0	3.63	3.65	5.0	2.92	5.0	2.64	2.33	2.65	...
2	0.0	0.0	3.63	3.63	5.0	2.92	5.0	2.64	2.33	2.64	...
3	0.0	0.5	3.63	3.63	5.0	2.92	5.0	2.35	2.33	2.64	...
4	0.0	0.0	3.63	3.63	5.0	2.92	5.0	2.64	2.33	2.64	...

5 rows × 24 columns

8. Hierarchical Clustering

import relevant packages

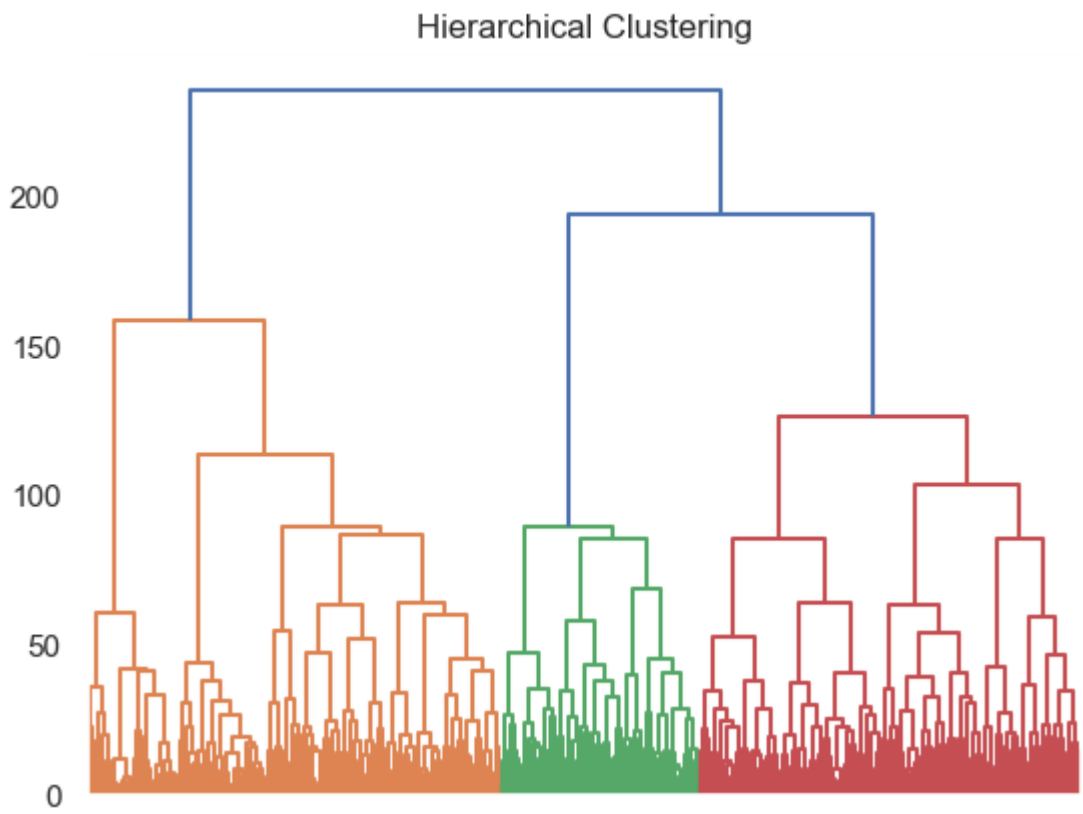
In [36]:
```
import scipy.cluster.hierarchy as sch
from sklearn.preprocessing import scale as s
from scipy.cluster.hierarchy import dendrogram, linkage
```

In [37]:
```
Z = sch.linkage(data_final,method='ward')
Z
```

Out[37]:
```
array([[2.00000000e+00, 4.00000000e+00, 0.00000000e+00, 2.00000000e+00],
       [1.33800000e+03, 1.34600000e+03, 0.00000000e+00, 2.00000000e+00],
       [6.70000000e+02, 6.74000000e+02, 0.00000000e+00, 2.00000000e+00],
       ...,
       [1.08930000e+04, 1.09060000e+04, 1.58239564e+02, 2.27800000e+03],
       [1.09030000e+04, 1.09070000e+04, 1.93830853e+02, 3.17800000e+03],
       [1.09080000e+04, 1.09090000e+04, 2.35727279e+02, 5.45600000e+03]])
```

In [38]:
```
# Creating and plotting a dendogram
den = sch.dendrogram(Z)
plt.tick_params(
    axis='x',
    which='both',
    bottom=False,
    top=False,
    labelbottom=False)
plt.title('Hierarchical Clustering')
```

Out[38]:
```
Text(0.5, 1.0, 'Hierarchical Clustering')
```



Creating a function to determine the cutting line

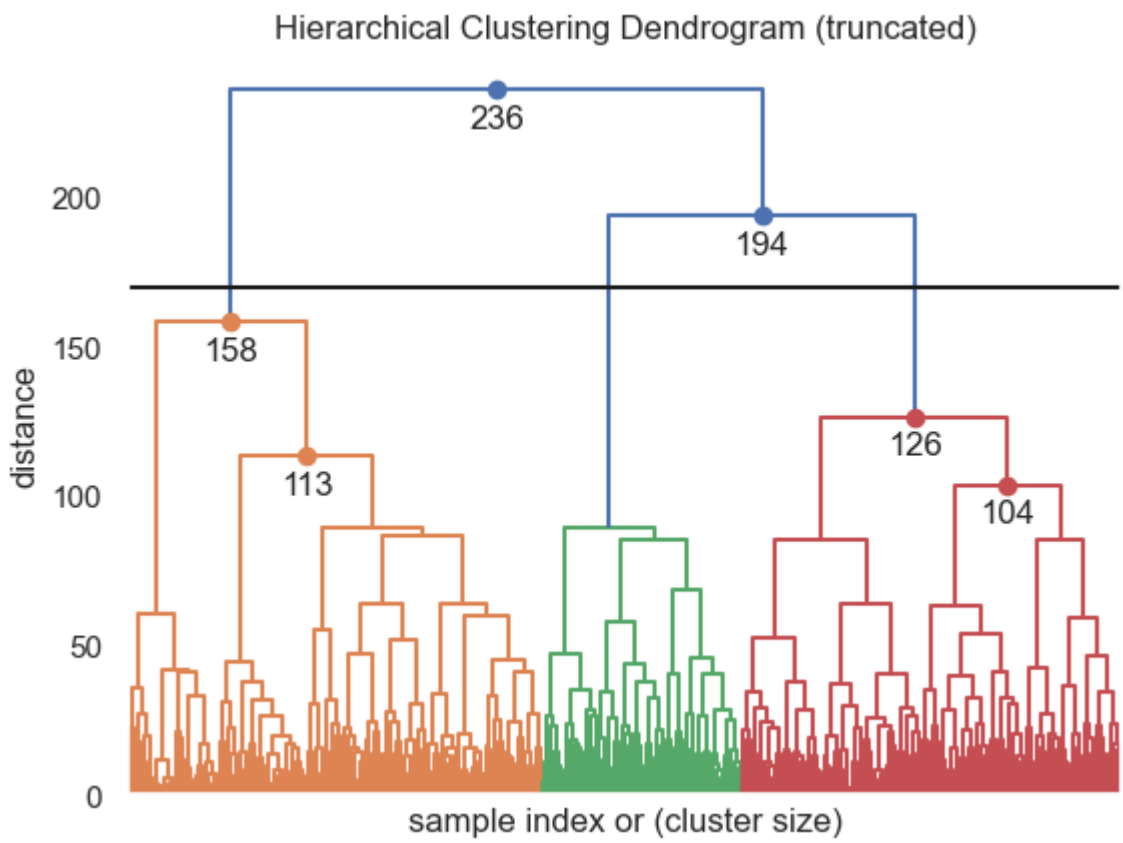
In [39]:
```
def fd(*args, **kwargs):
    max_d = kwargs.pop('max_d', None)
    if max_d and 'color_threshold' not in kwargs:
        kwargs['color_threshold'] = max_d
    annotate_above = kwargs.pop('annotate_above', 0)

    ddata = dendrogram(*args, **kwargs)

    if not kwargs.get('no_plot', False):
        plt.title('Hierarchical Clustering Dendrogram (truncated)')
        plt.xlabel('sample index or (cluster size)')
        plt.ylabel('distance')
        for i, d, c in zip(ddata['icoord'], ddata['dcoord'], ddata['color_list']):
            x = 0.5 * sum(i[1:3])
            y = d[1]
            if y > annotate_above:
                plt.plot(x, y, 'o', c=c)
                plt.annotate("%.3g" % y, (x, y), xytext=(0, -5),
                             textcoords='offset points',
                             va='top', ha='center')
        if max_d:
            plt.axhline(y=max_d, c='k')
    return ddata
```

Creating a dendogram with cutting line

Observing the height of each dendrogram division we decided to go with 80000 where the line would be drawn and 30000 to determine the dendrogram nodes

In [56]:
```
fd(Z,leaf_rotation=90.,show_contracted=True,annotate_above=100,max_d=170)
plt.tick_params(
    axis='x',
    which='both',
```

```
    bottom=False,
    top=False,
    labelbottom=False)
```



Creating a hierarchical clustering model

In [41]:
```
# Importing packages
from sklearn.cluster import AgglomerativeClustering
```

In [42]:
```
# Creating a Agglomerative Clustering
hc_model = AgglomerativeClustering(n_clusters = 2, affinity = 'euclidean', linkage
```

In [43]:
```
# Fitting the model
y_cluster = hc_model.fit_predict(data_final)
```

```
C:\Users\hp\anaconda3\anaconda\Lib\site-packages\sklearn\cluster\_agglomerative.p
y:1005: FutureWarning: Attribute `affinity` was deprecated in version 1.2 and will
be removed in 1.4. Use `metric` instead
  warnings.warn(
```

Adding the cluster column

In [44]:
```
data_clustered = data_final.copy()
```

In [45]:
```
data_clustered["Cluster"] = y_cluster.astype('object')
```

In [46]:
```
data_clustered.head()
```

Out[46]:

	churches	resorts	beaches	parks	theatres	museums	malls	zoo	restaurants	pubs_bars	...
0	0.0	0.0	3.63	3.65	5.0	2.92	5.0	2.35	2.33	2.64	...
1	0.0	0.0	3.63	3.65	5.0	2.92	5.0	2.64	2.33	2.65	...
2	0.0	0.0	3.63	3.63	5.0	2.92	5.0	2.64	2.33	2.64	...
3	0.0	0.5	3.63	3.63	5.0	2.92	5.0	2.35	2.33	2.64	...
4	0.0	0.0	3.63	3.63	5.0	2.92	5.0	2.64	2.33	2.64	...

5 rows × 25 columns

Visualizing the clusters

In [47]:
```
cols = list(data_final.columns)
#cols.remove("user_id")
cols
```

Out[47]:
```
['churches',
 'resorts',
 'beaches',
 'parks',
 'theatres',
 'museums',
 'malls',
 'zoo',
 'restaurants',
 'pubs_bars',
 'local_services',
 'burger_pizza_shops',
 'hotels_other_lodgings',
 'juice_bars',
 'art_galleries',
 'dance_clubs',
 'swimming_pools',
 'gyms',
 'bakeries',
 'beauty_spas',
 'cafes',
 'view_points',
 'monuments',
 'gardens']
```

In [48]:
```
sns.pairplot(data_clustered, hue="Cluster", diag_kind="hist")
```

```
C:\Users\hp\anaconda3\anaconda\Lib\site-packages\seaborn\axisgrid.py:118: UserWarn
ing: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[48]:
```
<seaborn.axisgrid.PairGrid at 0x249f6b6a210>
```

9. Conclusion

By using heirarchical clustering we clustered review into two category, positive review and a negeative review

In [49]:
```
result = data_clustered.copy()
```

In [50]:
```
result.replace({'Cluster' : 1} , 'Positive' , inplace=True)
result.replace({'Cluster' : 0} , 'Negative' , inplace= True)
```

In [51]:
```
result.head()
```

Out[51]:

	churches	resorts	beaches	parks	theatres	museums	malls	zoo	restaurants	pubs_bars	...
0	0.0	0.0	3.63	3.65	5.0	2.92	5.0	2.35	2.33	2.64	...
1	0.0	0.0	3.63	3.65	5.0	2.92	5.0	2.64	2.33	2.65	...
2	0.0	0.0	3.63	3.63	5.0	2.92	5.0	2.64	2.33	2.64	...
3	0.0	0.5	3.63	3.63	5.0	2.92	5.0	2.35	2.33	2.64	...
4	0.0	0.0	3.63	3.63	5.0	2.92	5.0	2.64	2.33	2.64	...

5 rows × 25 columns