In [57]:

```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
```

In [58]:

```
#Loading Dataset
df_trvl_rvw = pd.read_csv('tripadvisor_review.csv')
#df_trvl_rvw = df_trvl_rvw.set_index('User ID')
df_trvl_rvw.head()
```

Out[58]:

	User ID	Category 1	Category 2	Category 3	Category 4	Category 5	Category 6	Category 7	Category 8	Ca
0	User 1	0.93	1.8	2.29	0.62	0.80	2.42	3.19	2.79	
1	User 2	1.02	2.2	2.66	0.64	1.42	3.18	3.21	2.63	
2	User 3	1.22	0.8	0.54	0.53	0.24	1.54	3.18	2.80	
3	User 4	0.45	1.8	0.29	0.57	0.46	1.52	3.18	2.96	
4	User 5	0.51	1.2	1.18	0.57	1.54	2.02	3.18	2.78	

In [59]:

```
#rename the columns for easier understanding
re_name = ['User ID','art galleries'
,'dance clubs'
,'juice bars'
,'restaurants'
,'museums'
,'resorts'
,'parks'
,'beaches'
,'theaters'
,'religious']

#rename
df_trvl_rvw.columns = re_name
df_trvl_rvw.head()
```

Out[59]:

	User ID	art galleries	dance clubs	juice bars	restaurants	museums	resorts	parks	beaches	theaters	reli
0	User 1	0.93	1.8	2.29	0.62	0.80	2.42	3.19	2.79	1.82	
1	User 2	1.02	2.2	2.66	0.64	1.42	3.18	3.21	2.63	1.86	
2	User 3	1.22	0.8	0.54	0.53	0.24	1.54	3.18	2.80	1.31	
3	User 4	0.45	1.8	0.29	0.57	0.46	1.52	3.18	2.96	1.57	
4	User 5	0.51	1.2	1.18	0.57	1.54	2.02	3.18	2.78	1.18	

In [60]:

```
df_trvl_rvw.isnull().sum()
```

Out[60]:

```
User ID          0
art galleries    0
dance clubs      0
juice bars       0
restaurants      0
museums          0
resorts          0
parks            0
beaches          0
theaters         0
religious        0
dtype: int64
```

In [61]:

```
df_trvl_rvw.drop('User ID', axis = 1, inplace = True)
df_trvl_rvw = df_trvl_rvw.fillna(0)
```

In [62]:

```
df_trvl_rvw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 980 entries, 0 to 979
Data columns (total 10 columns):
 #   Column         Dtype  
---  ------         -----  
 0   art galleries  float64
 1   dance clubs    float64
 2   juice bars     float64
 3   restaurants    float64
 4   museums        float64
 5   resorts        float64
 6   parks          float64
 7   beaches        float64
 8   theaters       float64
 9   religious      float64
dtypes: float64(10)
memory usage: 76.7 KB
```

In [63]:

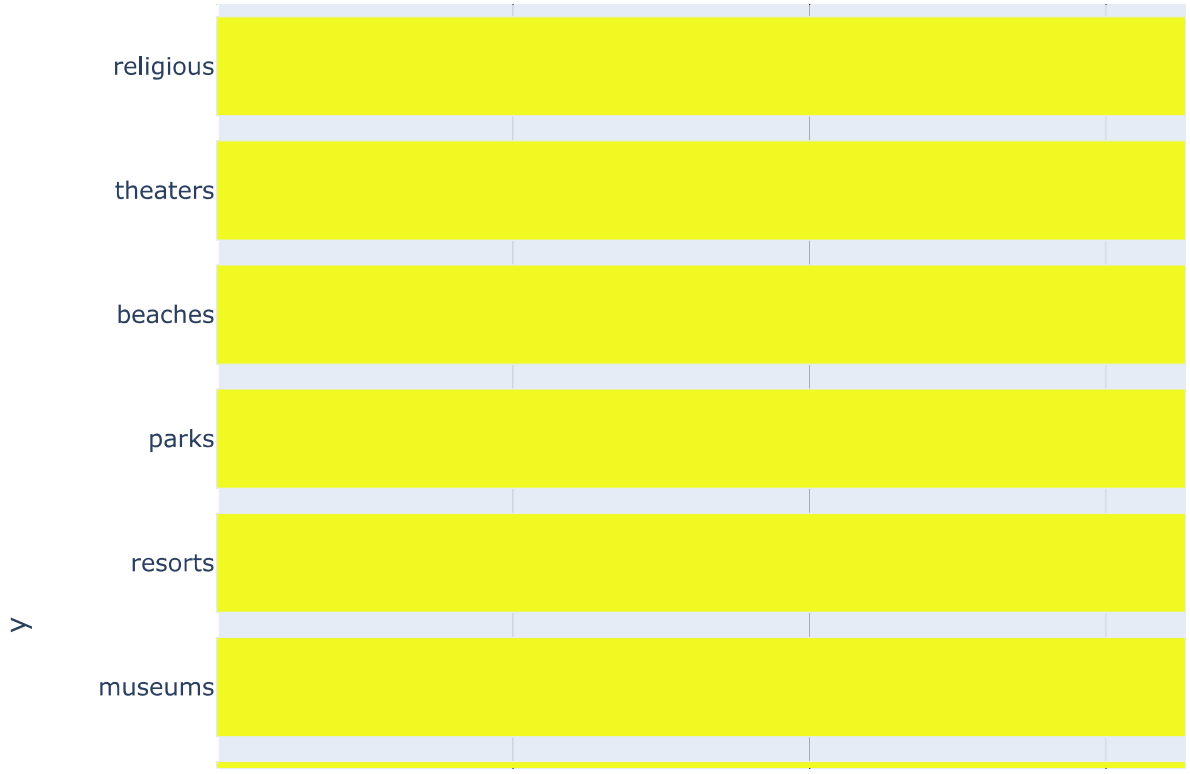
```
#All of our data is type 'float'
pd.set_option('display.max_columns', 30)
df_trvl_rvw.describe()
```

Out[63]:

	art galleries	dance clubs	juice bars	restaurants	museums	resorts	parks	
count	980.000000	980.000000	980.000000	980.000000	980.000000	980.000000	980.000000	98
mean	0.893194	1.352612	1.013306	0.532500	0.939735	1.842898	3.180939	
std	0.326912	0.478280	0.788607	0.279731	0.437430	0.539538	0.007824	
min	0.340000	0.000000	0.130000	0.150000	0.060000	0.140000	3.160000	
25%	0.670000	1.080000	0.270000	0.410000	0.640000	1.460000	3.180000	
50%	0.830000	1.280000	0.820000	0.500000	0.900000	1.800000	3.180000	
75%	1.020000	1.560000	1.572500	0.580000	1.200000	2.200000	3.180000	
max	3.220000	3.640000	3.620000	3.440000	3.300000	3.760000	3.210000	

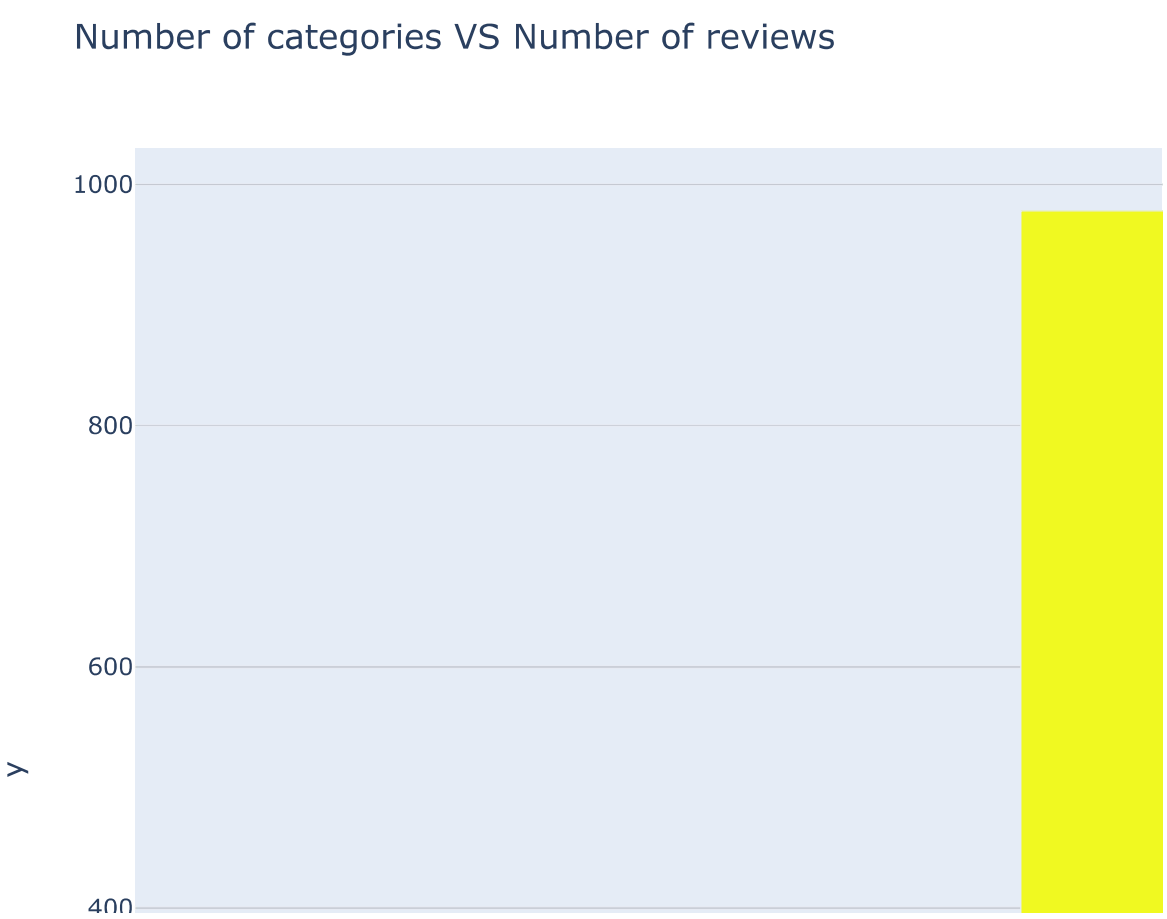
In [64]:

```
# visualize our first plot: we will examine the number of reviews under each category
import matplotlib.pyplot as plt
import warnings
%matplotlib inline
import plotly.express as px

# Plotting pretty figures and avoid blurry images
%config InlineBackend.figure_format = 'retina'
# Larger scale for plots in notebooks


# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Enable multiple cell outputs
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = 'all'

column_names = ['art galleries'
,'dance clubs'
,'juice bars'
,'restaurants'
,'museums'
,'resorts'
,'parks'
,'beaches'
,'theaters'
,'religious']

counts = df_trvl_rvw[column_names[:]].astype(bool).sum(axis=0).sort_values()

test = []
for i in range(len(counts.index)):
    test.append(counts.index[i])




fig = px.bar(counts,
             x=counts,
             y=test,
             color=counts,
                             labels={
                    "Total Ratings": "this is x",
                    "Categories": "this is y)"
                },
                height = 800,
                title="Number of reviews under each category")

fig.show()
```


religious
theaters
beaches
parks
resorts
museums
y

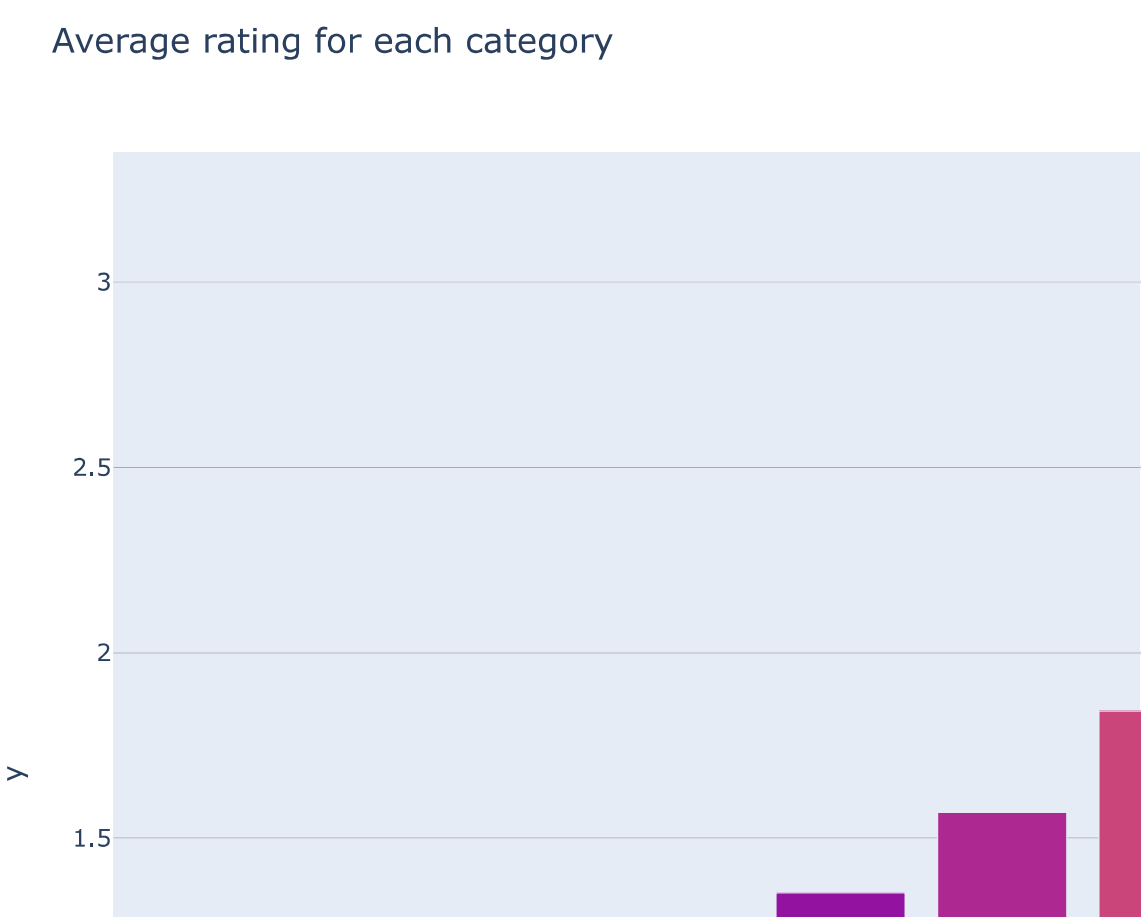
In [65]:

```
#reviews were given for each category.
reviews = df_trvl_rvw[column_names[:]].astype(bool).sum(axis=1).value_counts()


fig = px.bar(reviews,
             x=reviews.index,
             y=reviews.values,
             color=reviews.values,
                height = 800,
                title="Number of categories VS Number of reviews")

fig.show()
```



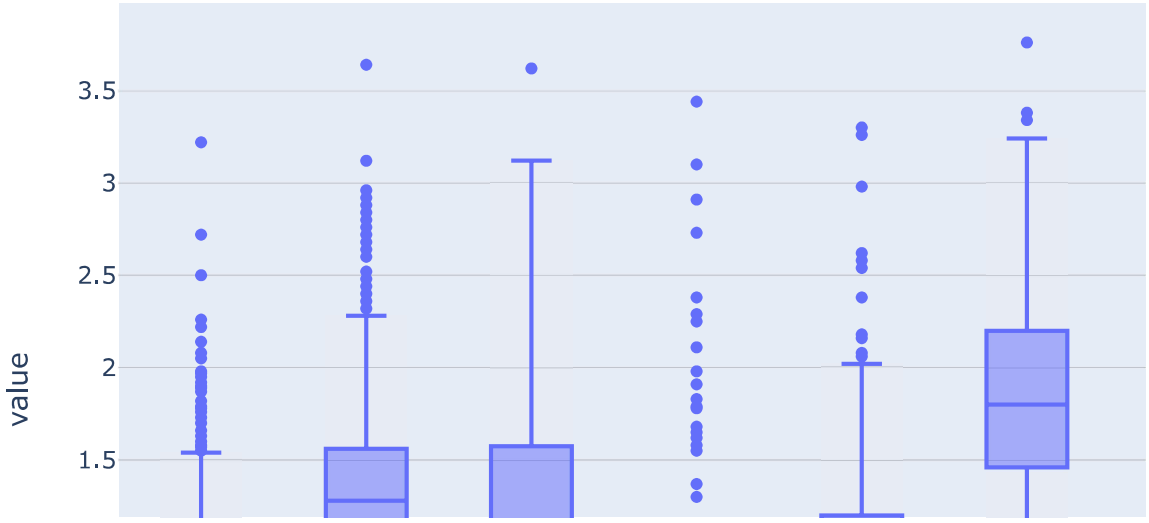
In [66]:

```
#average rating for each category
avg_rating = df_trvl_rvw[column_names[:]].mean().sort_values()



fig = px.bar(avg_rating,
             x = avg_rating.index,
             y = avg_rating.values,
             color = avg_rating.values,
             height = 800,
             title = "Average rating for each category")

fig.show()
```



In [67]:

```
fig = px.box(df_trvl_rvw, y = ['art galleries'
,'dance clubs'
,'juice bars'
,'restaurants'
,'museums'
,'resorts'
,'parks'
,'beaches'
,'theaters'
,'religious'])
fig.show()
```



In [68]:

```
df_new = pd.DataFrame(df_trvl_rvw)

Q1 = df_new.quantile(0.25)
Q3 = df_new.quantile(0.75)
IQR = Q3 - Q1

pd.set_option('display.max_info_rows', 30)

((df_new < (Q1 - 1.5 * IQR)) | (df_new > (Q3 + 1.5 * IQR))).sum()
```

Out[68]:

```
art galleries     47
dance clubs       54
juice bars         1
restaurants       45
museums           13
resorts            4
parks            399
beaches           24
theaters          23
religious          0
dtype: int64
```

In [69]:

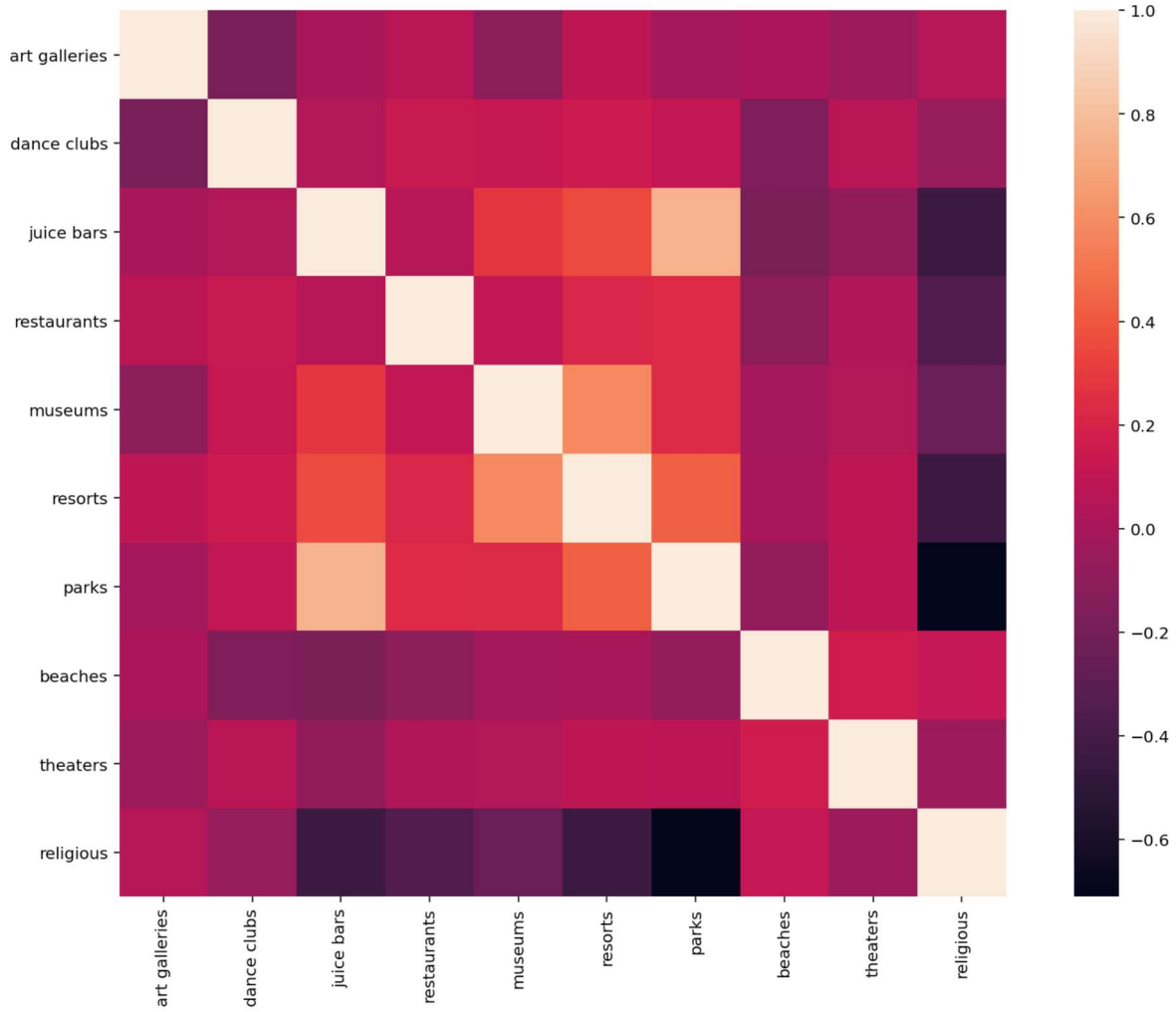
```
import seaborn as sns

plt.figure(figsize=(15,10))
cor = df_trvl_rvw.corr() #Calculate the correlation of the above variables
sns.heatmap(cor, square = True) #Plot the correlation as heat map
```

Out[69]:

```
<Figure size 1080x720 with 0 Axes>
```

Out[69]:

```
<AxesSubplot:>
```



In [70]:

```
df_trvl_rvw.shape
```

Out[70]:

```
(980, 10)
```

In [71]:

```
df_trvl_rvw = df_trvl_rvw.drop_duplicates()
```

In [72]:

```
df_trvl_rvw.shape
```

Out[72]:

```
(941, 10)
```

In [73]:

```
entertainment = ['theaters', 'dance clubs']
food_travel = ['restaurants', 'resorts','juice bars']
historical = ['museums', 'art galleries','religious']
nature = ['beaches', 'parks']
```

In [74]:

```
df_categories = pd.DataFrame(columns = ['entertainment', 'food_travel', 'historical', 'natu
```

In [75]:

```
df_categories['entertainment'] = df_trvl_rvw[entertainment].mean(axis=1)
df_categories['food_travel'] = df_trvl_rvw[food_travel].mean(axis = 1)
df_categories['historical'] = df_trvl_rvw[historical].mean(axis = 1)
df_categories['nature'] = df_trvl_rvw[nature].mean(axis = 1)
```

In [76]:

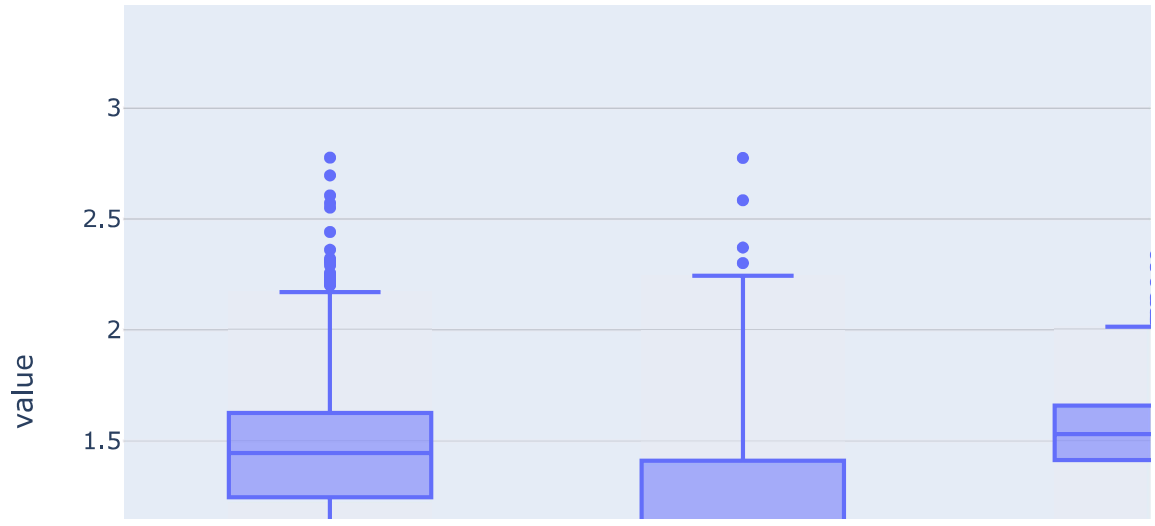
```
df_categories.describe()
```

Out[76]:

	entertainment	food_travel	historical	nature
count	941.000000	941.000000	941.000000	941.000000
mean	1.462912	1.121647	1.546213	3.008799
std	0.312203	0.388452	0.186365	0.068679
min	0.500000	0.156667	1.106667	2.810000
25%	1.245000	0.820000	1.413333	2.960000
50%	1.445000	1.060000	1.530000	3.005000
75%	1.625000	1.410000	1.656667	3.050000
max	2.775000	2.773333	2.336667	3.285000

In [77]:

```
fig = px.box(df_categories, y =['entertainment', 'food_travel', 'historical', 'nature'])
fig.show()
```



In [78]:

```
from pyclustertend import hopkins

hopkins(df_categories, df_categories.shape[0])
```

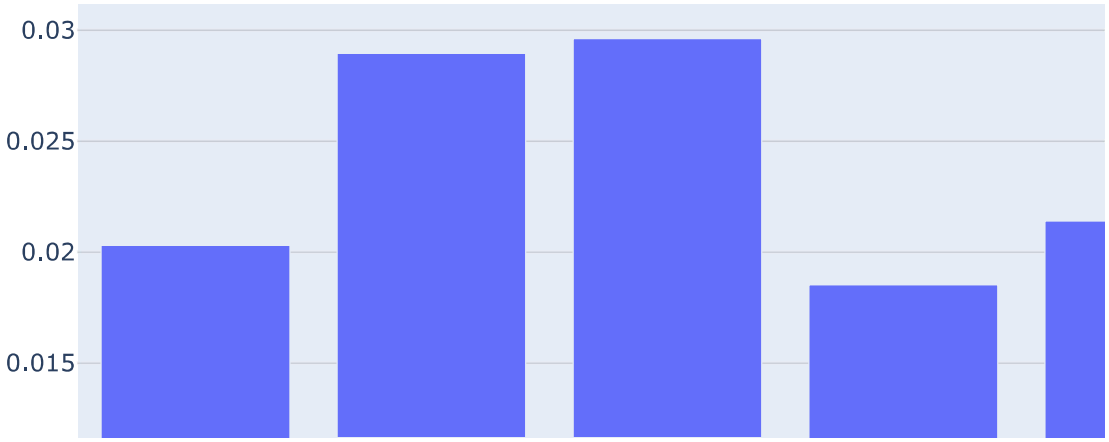
Out[78]:

```
0.21364958744784734
```

In [79]:

```
#Agglomerative Clustering
from scipy.cluster.hierarchy import dendrogram
from scipy.cluster import hierarchy
from sklearn.cluster import AgglomerativeClustering
from scipy.spatial.distance import pdist
import plotly.graph_objects as go

Z = hierarchy.linkage(df_categories, 'ward')

c, coph_dists = hierarchy.cophenet(Z, pdist(df_categories, 'hamming'))

ward = c


A = hierarchy.linkage(df_categories,'average')

c, coph_dists = hierarchy.cophenet(A, pdist(df_categories, 'hamming'))

average = c


B = hierarchy.linkage(df_categories,'single')

c, coph_dists = hierarchy.cophenet(B, pdist(df_categories, 'hamming'))

single = c


C = hierarchy.linkage(df_categories,'complete')

c, coph_dists = hierarchy.cophenet(C, pdist(df_categories, 'hamming'))

complete = c


D = hierarchy.linkage(df_categories,'weighted')

c, coph_dists = hierarchy.cophenet(D, pdist(df_categories, 'hamming'))

weighted = c


E = hierarchy.linkage(df_categories,'centroid')

c, coph_dists = hierarchy.cophenet(E, pdist(df_categories, 'hamming'))

centroid = c

F = hierarchy.linkage(df_categories,'median')

c, coph_dists = hierarchy.cophenet(F, pdist(df_categories, 'hamming'))

median = c


metrics=['ward', 'average', 'single', 'complete', 'weighted', 'centroid', 'median']


fig = go.Figure([go.Bar(x=metrics, y=[ward, average, single, complete, weighted, centroid,
```

```
fig.show()
```



In [80]:

```
from sklearn.metrics import silhouette_score
for n_clusters in range(2,10):
    clusterer = AgglomerativeClustering (n_clusters=n_clusters, distance_threshold = None)
    preds = clusterer.fit_predict(df_categories)


    score = silhouette_score (df_categories, preds)
    print ("For n_clusters = {}, silhouette score is {})".format(n_clusters, score))
```

```
For n_clusters = 2, silhouette score is 0.3418169738928389)
For n_clusters = 3, silhouette score is 0.2545822953304492)
For n_clusters = 4, silhouette score is 0.21919915815293894)
For n_clusters = 5, silhouette score is 0.1857103264800873)
For n_clusters = 6, silhouette score is 0.18638372714000376)
For n_clusters = 7, silhouette score is 0.19230432552941282)
For n_clusters = 8, silhouette score is 0.18314232248653736)
For n_clusters = 9, silhouette score is 0.18527826016980006)
```

In [81]:

```
#check the Davies-Bouldin score.We want a values as close to 0 as possible
from sklearn.metrics import davies_bouldin_score

for n_clusters in range(2,10):
    clusterer = AgglomerativeClustering (n_clusters=n_clusters, distance_threshold = None)
    preds = clusterer.fit_predict(df_categories)

    score = davies_bouldin_score (df_categories , preds)
    print ("For n_clusters = {}, the Davies-Bouldin score is {})".format(n_clusters, score)
```

```
For n_clusters = 2, the Davies-Bouldin score is 1.1389848758478498)
For n_clusters = 3, the Davies-Bouldin score is 1.2779640449892402)
For n_clusters = 4, the Davies-Bouldin score is 1.3811925249494243)
For n_clusters = 5, the Davies-Bouldin score is 1.4498363376528345)
For n_clusters = 6, the Davies-Bouldin score is 1.4539406040366332)
For n_clusters = 7, the Davies-Bouldin score is 1.2906552438897336)
For n_clusters = 8, the Davies-Bouldin score is 1.2849398213510637)
For n_clusters = 9, the Davies-Bouldin score is 1.2524152683655723)
```

In [82]:

```
#Calinski-Harabasz index.We want as high a score as possible
from sklearn.metrics import calinski_harabasz_score

for n_clusters in range(2,10):
    clusterer = AgglomerativeClustering (n_clusters=n_clusters, distance_threshold = None)
    preds = clusterer.fit_predict(df_categories)

    score = calinski_harabasz_score(df_categories, preds)
    print ("For n_clusters = {}, the Calinski-Harabasz score is {})".format(n_clusters, sco
```

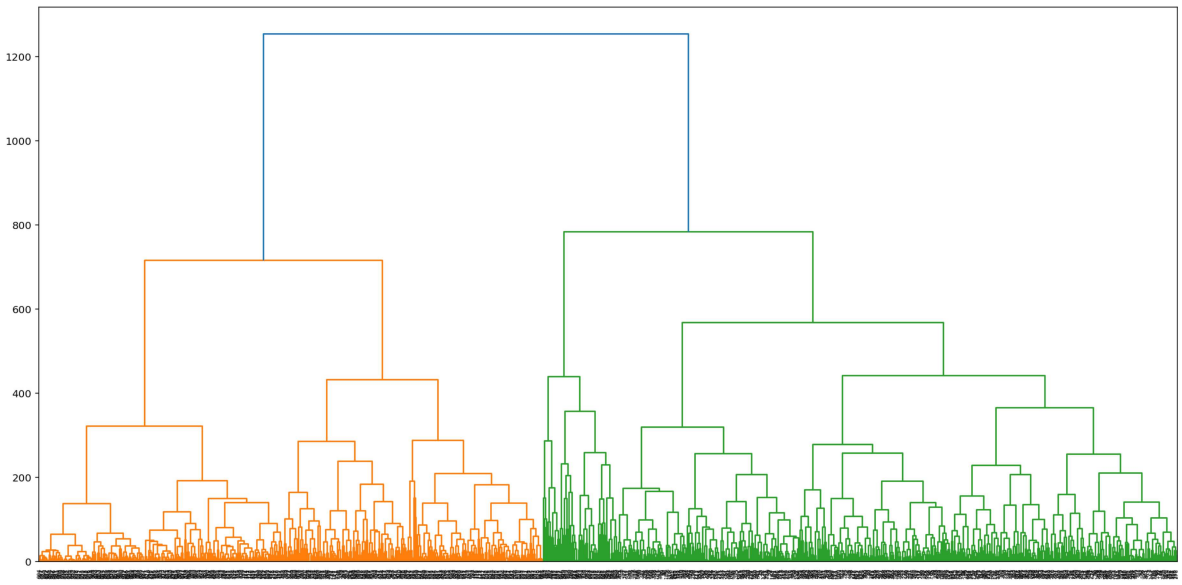
```
For n_clusters = 2, the Calinski-Harabasz score is 519.8935666574885)
For n_clusters = 3, the Calinski-Harabasz score is 428.99352073446323)
For n_clusters = 4, the Calinski-Harabasz score is 368.76673452400416)
For n_clusters = 5, the Calinski-Harabasz score is 336.3946416980188)
For n_clusters = 6, the Calinski-Harabasz score is 315.7454361879021)
For n_clusters = 7, the Calinski-Harabasz score is 300.9252851740537)
For n_clusters = 8, the Calinski-Harabasz score is 291.83934867479667)
For n_clusters = 9, the Calinski-Harabasz score is 285.91597760644333)
```

In [83]:

```
model = AgglomerativeClustering(distance_threshold=0, n_clusters = None)
model = model.fit(df_categories)

Z = hierarchy.linkage(model.children_, 'average')

plt.figure(figsize=(20,10))

dn = hierarchy.dendrogram(Z)
```

Out[83]:

```
<Figure size 1440x720 with 0 Axes>
```



In [84]:

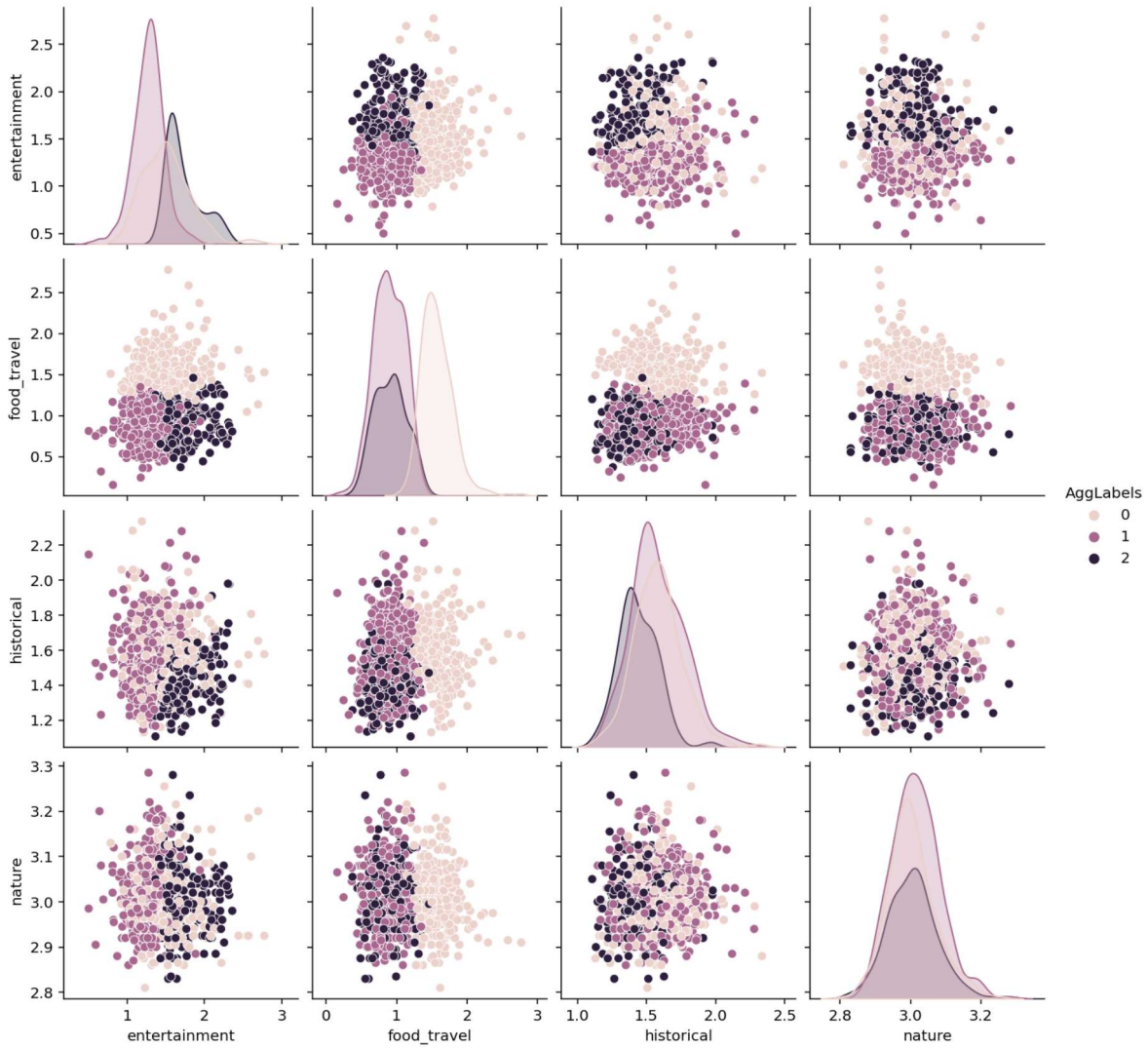
```
df = pd.DataFrame(df_trvl_rvw)

model = AgglomerativeClustering(distance_threshold=None, n_clusters = 3)
model = model.fit(df_categories)
y_agg=model.fit_predict(df_categories)

df_agg = df_categories.copy()
df_agg["AggLabels"] = y_agg

#uncomment the following line and let your machine explode!
sns.pairplot( df_agg, hue="AggLabels")

df_agg["AggLabels"].value_counts(0)
```

Out[84]:

```
<seaborn.axisgrid.PairGrid at 0x2284a80f7f0>
```

Out[84]:

```
1    403
0    311
2    227
Name: AggLabels, dtype: int64
```



In [85]:

```
#DBSCAN
from sklearn.neighbors import NearestNeighbors

nearest_neighbors = NearestNeighbors(n_neighbors=73) #sqrt(5456) = 73
nearest_neighbors.fit(df_categories)
distances, indices = nearest_neighbors.kneighbors(df_categories)
distances = np.sort(distances, axis=0)[:, 1]
#print(distances)
plt.figure(figsize=(20,10))
plt.plot(distances)
plt.show()
```

Out[85]:

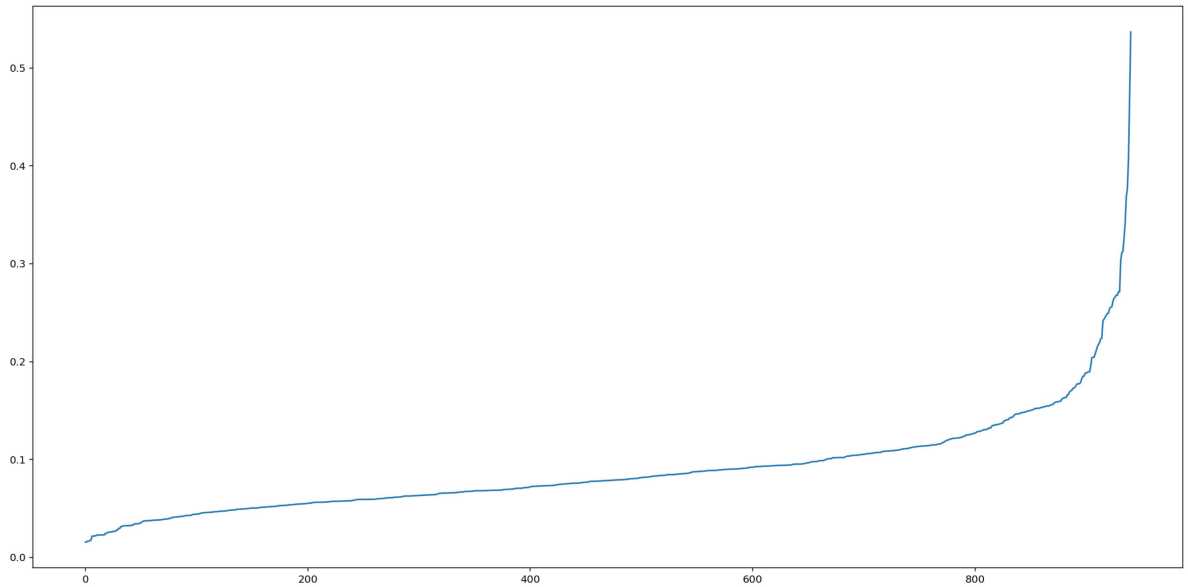
```
NearestNeighbors(n_neighbors=73)
```

Out[85]:

```
<Figure size 1440x720 with 0 Axes>
```

Out[85]:

```
[<matplotlib.lines.Line2D at 0x2285d323760>]
```



In [86]:

```
#Optimal value for eps where a 'knee' is formed i~ 0.2
from sklearn.cluster import DBSCAN

model2 = DBSCAN(eps=0.3, min_samples=10)

model2 = model2.fit(df_categories)

np.unique(model2.labels_)

y_db=model2.labels_

df_db = df_categories.copy()
df_db["DBLabels"] = y_db

df_db["DBLabels"].value_counts(0)


sns.pairplot( df_db, hue="DBLabels")
```

Out[86]:

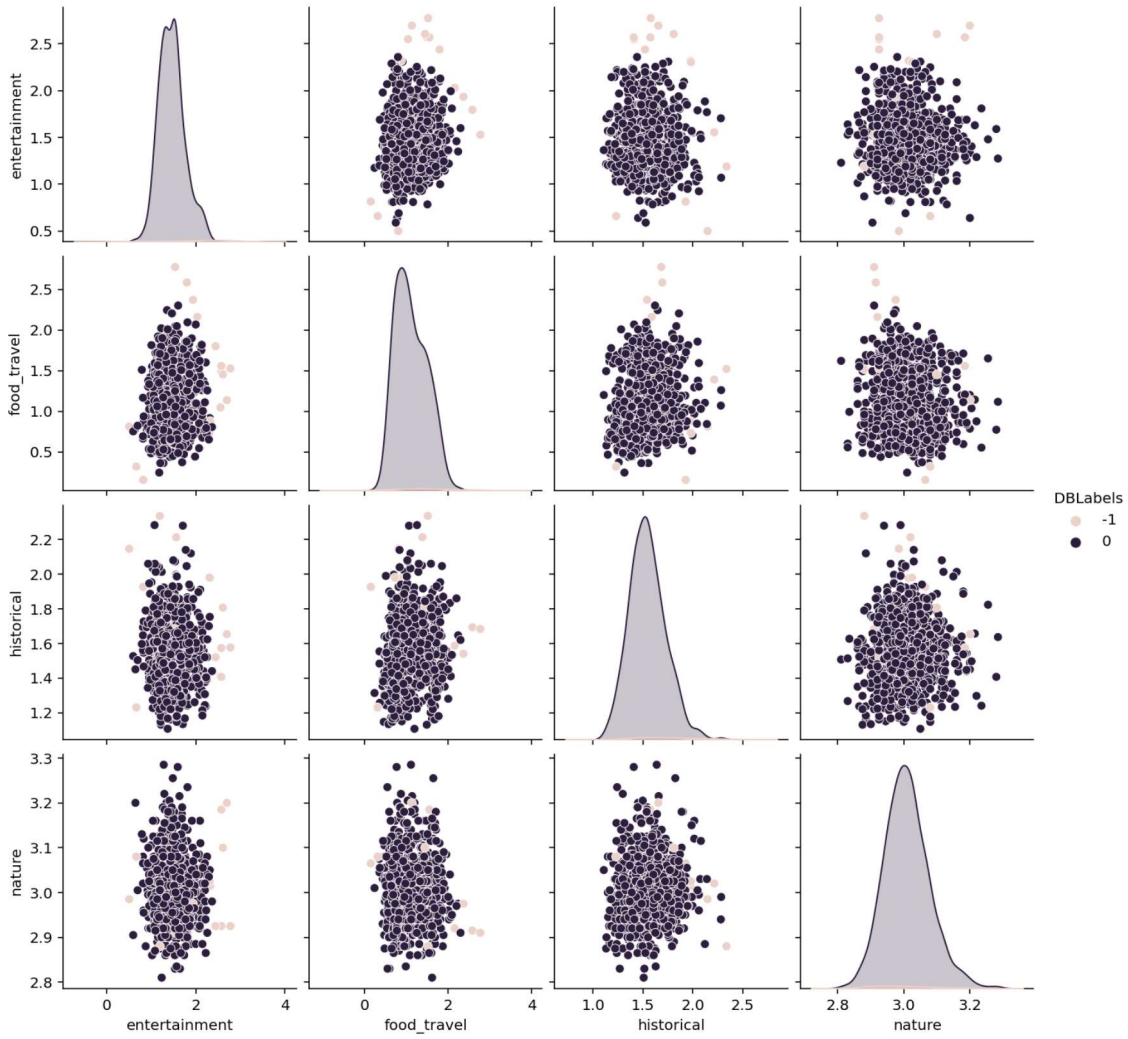
```
array([-1,  0], dtype=int64)
```

Out[86]:

```
 0    923
-1     18
Name: DBLabels, dtype: int64
```

Out[86]:

```
<seaborn.axisgrid.PairGrid at 0x2285d3359a0>
```



In [87]:

```
df_db
```

Out[87]:

	entertainment	food_travel	historical	nature	DBLabels
0	1.810	1.776667	1.383333	2.990	0
1	2.030	2.160000	1.586667	2.920	-1
2	1.055	0.870000	1.320000	2.990	0
3	1.685	0.793333	1.256667	3.070	0
4	1.190	1.256667	1.530000	2.980	0
...	...	...	...	...	...
975	1.055	0.736667	1.606667	2.975	0
976	1.130	1.060000	1.610000	2.985	0
977	1.330	0.960000	1.643333	2.990	0
978	0.660	0.320000	1.230000	3.080	-1
979	0.950	1.333333	1.556667	3.025	0

941 rows × 5 columns

In [88]:

```
# Get names of indexes for which column DB_Labels has value -1
indexNames = df_db[ df_db['DBLabels'] == -1 ].index
# Delete these row indexes from dataFrame
df_db.drop(indexNames , inplace=True)

df_db['DBLabels'].unique()
```

Out[88]:

```
array([0], dtype=int64)
```

In [89]:

```
df_categories
```

Out[89]:

	entertainment	food_travel	historical	nature
0	1.810	1.776667	1.383333	2.990
1	2.030	2.160000	1.586667	2.920
2	1.055	0.870000	1.320000	2.990
3	1.685	0.793333	1.256667	3.070
4	1.190	1.256667	1.530000	2.980
...	...	...	...	...
975	1.055	0.736667	1.606667	2.975
976	1.130	1.060000	1.610000	2.985
977	1.330	0.960000	1.643333	2.990
978	0.660	0.320000	1.230000	3.080
979	0.950	1.333333	1.556667	3.025

941 rows × 4 columns

In [90]:

```
import numpy as np
from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
```

In [91]:

```
# Generate sample data
centers = df_categories
X, labels_true = make_blobs(n_samples=750, centers=df_categories, cluster_std=0.4,
                            random_state=0)

X = StandardScaler().fit_transform(X)
```

In [92]:

```
# Compute DBSCAN
db = DBSCAN(eps=0.3, min_samples=10).fit(X)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_
```

In [93]:

```
# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) #- (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)

print('Estimated number of clusters: %d' % n_clusters_)
print('Estimated number of noise points: %d' % n_noise_)
print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels_true, labels))
print("Completeness: %0.3f" % metrics.completeness_score(labels_true, labels))
print("V-measure: %0.3f" % metrics.v_measure_score(labels_true, labels))
print("Adjusted Rand Index: %0.3f"
      % metrics.adjusted_rand_score(labels_true, labels))
print("Adjusted Mutual Information: %0.3f"
      % metrics.adjusted_mutual_info_score(labels_true, labels))
print("Silhouette Coefficient: %0.3f"
      % metrics.silhouette_score(X, labels))
```

```
Estimated number of clusters: 1
Estimated number of noise points: 750
Homogeneity: 0.000
Completeness: 1.000
V-measure: 0.000
Adjusted Rand Index: 0.000
Adjusted Mutual Information: 0.000

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-93-7a73db259374> in <module>
     13       % metrics.adjusted_mutual_info_score(labels_true, labels))
     14 print("Silhouette Coefficient: %0.3f"
---> 15       % metrics.silhouette_score(X, labels))

c:\users\shery\pycharmprojects\cw_ml\venv\lib\site-packages\sklearn\utils\va
lidation.py in inner_f(*args, **kwargs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64
     65             # extra_args > 0

c:\users\shery\pycharmprojects\cw_ml\venv\lib\site-packages\sklearn\metrics
\cluster\_unsupervised.py in silhouette_score(X, labels, metric, sample_siz
e, random_state, **kwds)
    115         else:
    116             X, labels = X[indices], labels[indices]
--> 117     return np.mean(silhouette_samples(X, labels, metric=metric, **kw
ds))
    118
    119

c:\users\shery\pycharmprojects\cw_ml\venv\lib\site-packages\sklearn\utils\va
lidation.py in inner_f(*args, **kwargs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64
     65             # extra_args > 0

c:\users\shery\pycharmprojects\cw_ml\venv\lib\site-packages\sklearn\metrics
```

```
\cluster\_unsupervised.py in silhouette_samples(X, labels, metric, **kwds)
    227     n_samples = len(labels)
    228     label_freqs = np.bincount(labels)
--> 229     check_number_of_labels(len(le.classes_), n_samples)
    230
    231     kwds['metric'] = metric

c:\users\shery\pycharmprojects\cw_ml\venv\lib\site-packages\sklearn\metrics
\cluster\_unsupervised.py in check_number_of_labels(n_labels, n_samples)
     32     """
     33     if not 1 < n_labels < n_samples:
---> 34         raise ValueError("Number of labels is %d. Valid values are 2
"
     35                          "to n_samples - 1 (inclusive)" % n_labels)
     36

ValueError: Number of labels is 1. Valid values are 2 to n_samples - 1 (incl
usive)
```

In [94]:

```
# Plot result
import matplotlib.pyplot as plt
%matplotlib inline
```

In [95]:

```
# Black removed and is used for noise instead.
unique_labels = set(labels)
colors = [plt.cm.Spectral(each)
          for each in np.linspace(0, 1, len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise.
        col = [0, 0, 0, 1]

    class_member_mask = (labels == k)

    xy = X[class_member_mask & core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=14)

    xy = X[class_member_mask & ~core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=6)

plt.title('Estimated number of clusters: %d' % n_clusters_)
plt.show()
```

Out[95]:

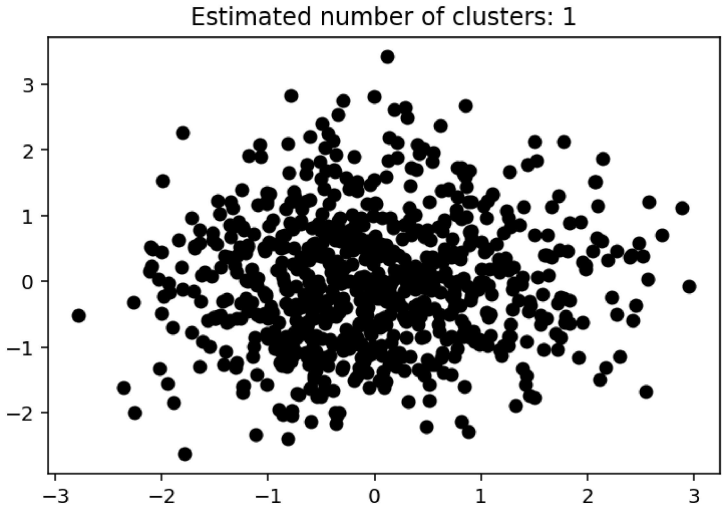
```
[<matplotlib.lines.Line2D at 0x22859f28640>]
```

Out[95]:

```
[<matplotlib.lines.Line2D at 0x22859f288e0>]
```

Out[95]:

```
Text(0.5, 1.0, 'Estimated number of clusters: 1')
```

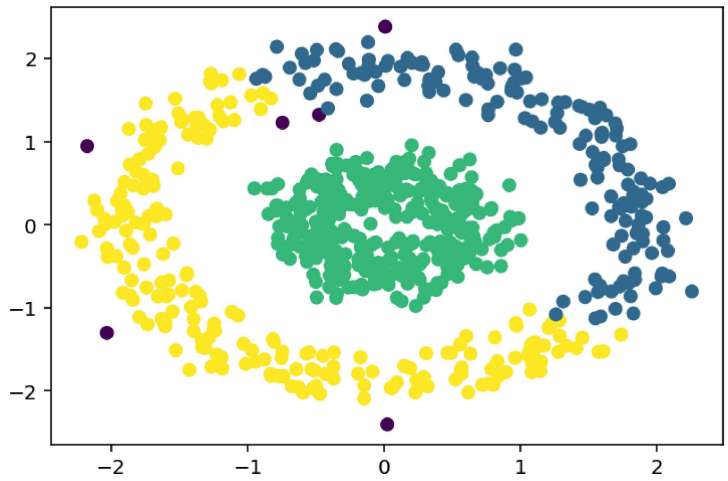


In [97]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.datasets import make_circles
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN
X, y = make_circles(n_samples=750, factor=0.3, noise=0.1)
X = StandardScaler().fit_transform(X)
y_pred = DBSCAN(eps=0.3, min_samples=10).fit_predict(X)

plt.scatter(X[:,0], X[:,1], c=y_pred)
print('Number of clusters: {}'.format(len(set(y_pred[np.where(y_pred != -1)]))))
print('Homogeneity: {}'.format(metrics.homogeneity_score(y, y_pred)))
print('Completeness: {}'.format(metrics.completeness_score(y, y_pred)))
print("V-measure: %0.3f" % metrics.v_measure_score(labels_true,  y_pred))#labels))
print("Adjusted Rand Index: %0.3f"
      % metrics.adjusted_rand_score(labels_true, y_pred))#labels))
print("Adjusted Mutual Information: %0.3f"
      % metrics.adjusted_mutual_info_score(labels_true, y_pred))#labels))
print("Silhouette Coefficient: %0.3f"
      % metrics.silhouette_score(X,y_pred))
```

Out[97]:

```
<matplotlib.collections.PathCollection at 0x228591ed9a0>

Number of clusters: 3
Homogeneity: 1.0
Completeness: 0.6477405612067773
V-measure: 0.278
Adjusted Rand Index: 0.000
Adjusted Mutual Information: 0.000
Silhouette Coefficient: 0.296
```



In [ ]:

```
```