DATA MINING & ANALYTICS (2023)

Make sure you fill in any place that says `YOUR CODE HERE` or `YOUR ANSWER HERE`, as well as your name below:

NAME = "Jeonghyun Lee"

Lab 2: Clustering

Please read the following instructions very carefully.

About the Dataset

The dataset for this lab has been created from some custom features from Lab 1. The columns are named as q1, q2....etc. A description of the features can be found at this link: https://docs.google.com/spreadsheets/d/18wwyjGku2HYfgDX9Vez64lGHz31E_PfbpmAdfb7ly6M/edit?usp=sharing

Working on the assignment / FAQs

- **Always use the seed/random_state as *42* wherever applicable** (This is to ensure repeatability in answers, across students and coding environments).
    - This can typically look like taking in another argument `random_state = 42` when applicable.
- The points allotted per question is listed.
- To avoid any ambiguity, each question also specifies what *value* the function must return. Note that these are dummy values and not the answers themselves.
- If a question has multiple answers (due to differences in handling NaNs, zeros etc.), all answers will be considered.
- Most assignments have bonus questions for extra credit, do try them out!
- You can delete the `raise NotImplementedError()` when you are attempting the question.
- **Submitting the assignment** : Save your work as a PDF (Print -> Save as PDF), download the `.ipynb` file from Colab (Download -> Download as .ipynb), and upload these two files to Gradescope. **Run all cells before submitting.**
- **MAKE A COPY OF THIS FILE FOR YOURSELF TO EDIT/SAVE.**
- That's about it. Happy coding!

```
import pandas as pd
import collections
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import numpy as np
from sklearn.preprocessing import normalize

import matplotlib
import matplotlib.pyplot as plt

%matplotlib inline
matplotlib.style.use('ggplot')

#DOWNLOADING DATASET
!wget -nc http://quadro.ist.berkeley.edu:1331/yelp_reviewers.csv
# !unzip -u yelp_reviewers.zip
print('Dataset Downloaded: yelp_reviewers.csv')
df = pd.read_csv('yelp_reviewers.csv', delimiter= ',')
df = df.sample(frac=0.3, random_state=42)
print(df.dropna().describe())

print('....SETUP COMPLETE....')
```

```
File ‘yelp_reviewers.csv’ already there; not retrieving.

Dataset Downloaded: yelp_reviewers.csv
                q3          q4          q5          q6          q7  ₩
count  7177.000000  7177.000000  7177.000000  7177.000000  7177.000000
mean      6.838651     5.281455     4.750871     8.808973     1.539160
std       7.597977    16.208703    13.866352    19.980443     0.885421
min       1.000000     1.000000     1.000000     1.000000     0.000000
25%       3.000000     1.000000     1.000000     2.000000     1.100000
50%       5.000000     2.000000     2.000000     5.000000     1.610000
75%       9.000000     4.000000     4.000000     9.000000     2.200000
max     252.000000   607.000000   474.000000   773.000000     5.530000

                q8          q9         q10         q11         q12  ...  ₩
```

```
       count  7177.000000  7177.000000  7177.000000  7177.000000  7177.000000  ...
       mean      0.934928     0.870281     1.549898    26.732782    25.660616  ...
       std       0.976816     0.950066     1.024145    10.226302    11.451583  ...
       min       0.000000     0.000000     0.000000     2.900000     1.410000  ...
       25%       0.000000     0.000000     0.690000    20.000000    16.670000  ...
       50%       0.690000     0.690000     1.610000    25.710000    25.000000  ...
       75%       1.390000     1.390000     2.200000    33.330000    33.330000  ...
       max       6.410000     6.160000     6.650000    77.780000    75.000000  ...

                 q16r         q16u         q16v         q16w         q16x  ₩
       count  7177.000000  7177.000000  7177.000000  7177.000000  7177.000000
       mean      3.641912     0.462843    22.503414    25.665180     0.003744
       std       1.483358     0.507827    14.350555    29.021007     0.006019
       min       1.000000     0.000000     1.000000     1.000000     0.000000
       25%       3.000000     0.000000    10.000000     9.000000     0.000491
       50%       4.000000     0.333333    21.000000    18.000000     0.001967
       75%       5.000000     0.666667    33.000000    33.000000     0.004666
       max       5.000000     6.000000    53.000000   868.000000     0.150618

                 q16y         q16z         q16aa        q16ab        q16ac
       count  7177.000000  7177.000000  7177.000000  7177.000000  7177.000000
       mean     74.046169     0.675212     0.552041     1.127751     3.649254
       std      50.031941     1.503059     2.042566     4.652206     0.977100
       min       1.333333     0.000000     0.000000     0.000000     1.000000
       25%      39.666667     0.000000     0.000000     0.000000     3.200000
       50%      62.900000     0.000000     0.000000     0.500000     3.777778
       75%      95.687500     1.000000     0.000000     1.307692     4.333333
       max     507.200000    44.000000   106.000000   342.300000     5.000000

       [8 rows x 40 columns]
       ....SETUP COMPLETE....
```

```
df.head().T
```

q13	NaN	NaN	
q14	7	10	
q15	510.0	132.0	
q16a	0	0	
q16b	0.0	0.0	
q16c	0.0	0.0	
q16d	3.0	1.0	
q16e	0.013725	0.045455	C
q16f	0.0	0.0	
q16g	0	1	
q16h	0	1	
q16i	0	0	
q16j	0.0	0.0	
q16k	0	0	

Question 1 (1 point)

What is the best choice of k according to the silhouette metric for clustering q4-q6? Only consider 2 <= k <= 8. (hint: take a look at `silhouette_score`).

NOTE: For features with high variance, empty clusters can occur. There are several ways of dealing with empty clusters. A common approach is to drop empty clusters. The preferred approach for this lab is to treat the empty clusters as “singletons”, leaving them empty with single point placeholders (so no need to drop anything for the purposes of the lab).

```
#Make sure you return the answer value in this function.
#The return value should be an integer.
def q1(df):

 # YOUR CODE HERE
 q4_q6 = df[['q4', 'q5', 'q6']]
 best_score = -1

 for i in range (2, 9):
  kmeans = KMeans(n_clusters=i,  n_init = 10, random_state=42)
  cluster = kmeans.fit(q4_q6)
  score = silhouette_score(q4_q6, cluster.labels_)

  if score > best_score:
   best_score = score
   k = i

 return k

 # For KMeans use argument n_init = 10 when applicable.
 raise NotImplementedError()

print(q1(df))
```

```
2
```

What is the best choice of k?

```
# YOUR ANSWER HERE
2
```

```
2
```

Question 2 (1 point)

What is the best choice of k according to the silhouette metric for clustering q7-q10? Only consider 2 <= k <= 8.

Note: Keep in mind, there may be missing values in this part of the dataset! For these missing values, first find the subset of data specified for this question (q7-q10), then replace the missing values with 0. We do this since the missing values from q7-q10 are most commonly because of taking the log's of values of 0 from q3-q6.

```
#Make sure you return the answer value in this function.
#The return value should be an integer.
def q2(df):

 # YOUR CODE HERE

 q7_q10 = df[['q7', 'q8', 'q9', 'q10']].fillna(0)
 best_score = -1

 for i in range (2, 9):
  kmeans = KMeans(n_clusters=i,  n_init = 10, random_state=42)
  cluster = kmeans.fit(q7_q10)
  score = silhouette_score(q7_q10, cluster.labels_)

  if score > best_score:
   best_score = score
   k = i

 return k
 # For KMeans use argument n_init = 10 when applicable.
 raise NotImplementedError()

print(q2(df))
```

```
2
```

What is the best choice of k?

```
# YOUR ANSWER HERE
2
```

```
2
```

Question 3 (1 point)

What is the best choice of k according to the silhouette metric for clustering q11-q13? Only consider 2 <= k <= 8.

Note: Keep in mind, there may be missing values in this part of the dataset! For these missing values, first find the subset of data specified for this question (q11-q13), then drop rows that have missing values.

```
#Make sure you return the answer value in this function.
#The return value should be an integer.
def q3(df):

 # YOUR CODE HERE
 q11_q13 = df[['q11', 'q12', 'q13']].dropna()
 best_score = -1

 for i in range (2, 9):
  kmeans = KMeans(n_clusters=i,  n_init = 10, random_state=42)
  cluster = kmeans.fit(q11_q13)
  score = silhouette_score(q11_q13, cluster.labels_)

  if score > best_score:
   best_score = score
   k = i

 return k
 # For KMeans use argument n_init = 10 when applicable.
 raise NotImplementedError()

print(q3(df))
```

```
8
```

What is the best choice of k?

```
# YOUR ANSWER HERE
8
```

```
8
```

Question 4 (1 point)

Take the best clustering (i.e., best value of K) from Question 3 and using the same subset of data from q11-q13, list the number of data points in each cluster. Return your answer in dictionary form (i.e. `ans = {0: 100, 1: 200, ...}`).

```
from typing import Counter

#Make sure you return the answer value in this function.
#The return value should be an dictionary. Eg : {0:1000,1:500,2:1460}.
def q4(df):

 # YOUR CODE HERE
 q11_q13 = df[['q11', 'q12', 'q13']].dropna()


 kmeans = KMeans(n_clusters = 8,  n_init = 10, random_state=42)
 cluster = kmeans.fit(q11_q13)

 cluster_dict = dict(Counter(sorted(cluster.labels_)))

 return cluster_dict

 # For KMeans use argument n_init = 10 when applicable.
 raise NotImplementedError()


#This is an graded cell, do not edit
print(q4(df))
```

```
{0: 9831, 1: 2140, 2: 1632, 3: 1228, 4: 3037, 5: 3216, 6: 5724, 7: 3301}
```

Question 5 (1 point)

Consider the best clustering from Question 3. Were there clusters that represented very funny but useless reviewers (check column definitions for columns corresponding to funny, useless, etc.)? If so, print the center of that cluster.

```
#Make sure you return the answer value in this function.
#The return value should be a list. Eg : [10, 30, 54].
def q5(df):

 # YOUR CODE HERE
 drop_df = df[["q11","q12","q13"]].dropna()
 km = KMeans(n_clusters=8, random_state=42)
 km.fit(drop_df)
 centers=km.cluster_centers_
 return centers[np.argmax(centers[:,1])]
 # For KMeans use argument n_init = 10 when applicable.
 raise NotImplementedError()


#This is a graded cell, do not edit
print(np.round_(q5(df), decimals=1, out=None))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from
  warnings.warn(
[ 1.1 98.3  0.6]
```

Question 6 (1 point)

Consider the best clustering from Question 3. What was the centroid of the cluster that represented relatively uniform strength in all voting categories?

```
#Make sure you return the answer value in this function.
#The return value should be a centroid in list form. Eg : [10, 10.5, 13].
def q6(df):

 # YOUR CODE HERE
 q11_q13 = df[['q11', 'q12', 'q13']].dropna()
 kmeans = KMeans(n_clusters=8, n_init=10, random_state=42)
 kmeans.fit(q11_q13)
 center = kmeans.cluster_centers_

 output = None
 best_score = float('inf')

 for i in center:
  cool = i[0]
  funny = i[1]
  useful = i[2]

  cluster_dist = max( abs(cool-funny),  abs(funny - useful), abs(cool - useful))

  if cluster_dist < best_score:
   best_score = cluster_dist
   output = i


 return output.tolist()
 # For KMeans use argument n_init = 10 when applicable.
 raise NotImplementedError()


#This is a graded cell, do not edit
print(q6(df))
```

```
[31.45574099965031, 30.37879937084976, 38.16278049632923]
```

Question 7 (1 point)

Cluster the dataset using $k = 7$ and using features q7-q15 (refer to the column descriptions if needed). What is the silhouette metric for this clustering? For a more in-depth understanding of cluster analysis with silhouette, look here.

As before, fill NaN values in q7-q10 with 0, but drop rows that have NaN values from q11-q15.

```
#Make sure you return the answer value in this function.
#The return value should be a float.
def q7(df):

 # YOUR CODE HERE
 new_df = df[['q7','q8','q9','q10','q11','q12','q13','q14','q15']].dropna()
 km = KMeans(n_clusters=5, random_state=42)
 km.fit(new_df)
 return silhouette_score(new_df, km.labels_)
 # For KMeans use argument n_init = 10 when applicable.
 raise NotImplementedError()


#This is a graded cell, do not edit
print(q7(df))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from
  warnings.warn(
0.5481158706623568
```

Question 8 (1 point)

Cluster the dataset using $k = 7$ and using features q7-q15 (refer to the column descriptions if needed).

What is the average q3 value in each of the clusters? Replace/fill NaN values for q7-q15 as you have done for previous questions.

```
#Make sure you return the answer value in this function.
#The return value should be an Array. Eg : [10, 30, 54].
def q8(df):

 # YOUR CODE HERE
 q7_q15 = df[['q7', 'q8', 'q9', 'q10', 'q11', 'q12', 'q12', 'q13', 'q14', 'q15']].dropna()

 kmeans = KMeans(n_clusters = 7, n_init=10, random_state=42)
 cluster = kmeans.fit(q7_q15)

 q7_q15['label'] = cluster.labels_
 q7_q15['q3 value'] = np.exp(q7_q15['q7'])

 mean = q7_q15.groupby('label')['q3 value'].mean().to_numpy()

 return mean
 # For KMeans use argument n_init = 10 when applicable.
 raise NotImplementedError()


#This is a graded cell, do not edit
print(np.round_(q8(df), decimals=1, out=None))
```

```
[7.2 4.1 7.1 7.4 5.6 6.3 1.6]
```

Question 9 (2 points)

We will now cluster the dataset using all features in the dataset.

We can drop features with high incidents of `-inf` / `NaN` / blank values. We will also perform some form of normalization on these features so as not to over bias the clustering towards the larger magnitude features.

Let's go ahead and get started.

Data Cleansing and Normalization

Check how many null values there are in each column.

```
# YOUR CODE HERE
df.isna().sum()
```

```
user_id      0
q3           0
q4           0
q5           0
q6           0
q7           0
q8       35280
q9       36743
q10      24338
q11      21383
q12      21383
q13      21383
q14          0
q15          0
q16a         0
q16b         0
q16c         0
q16d         0
q16e         0
q16f         0
q16g         0
q16h         0
q16i         0
q16j         0
q16k         0
q16l         0
q16m         0
q16n         0
q16o         0
q16p         0
q16q         0
q16r         0
q16s         0
q16t         0
q16u         0
q16v         0
q16w         0
q16x         0
q16y         0
q16z         0
q16aa        0
q16ab    14469
```

```
q16ac    0
dtype: int64
```

It looks like q8 - q13 and q16ab have a lot of null values. Let's see what the impact is of removing the two columns with the most null values.

Drop the two columns with the most NaN values, and then remove all rows with NaN values remaining.

```
# YOUR CODE HERE
null_count = df.isnull().sum()
drop_column = null_count.nlargest(2).index

remove_two_column = df.drop(columns=drop_column).dropna()

remove_two_column
```

	user_id	q3	q4	q5	q6	q7	q10	q11	q12	q13	...	q16t	q16u	q16v	q16w	q1
47453	Gd_IGX3BmRYbPD84ovLEoA	8	2	1	8	2.08	2.08	18.18	9.09	72.73	...	no	0.375000	8	39	0.0017
53000	Ihx1EQHDTIoXM35Cc08r2Q	2	1	1	2	0.69	0.69	25.00	25.00	50.00	...	no	1.000000	22	6	0.0000
64580	N22hkNXzJdz_v_KocOy6vA	1	0	0	1	0.00	0.00	0.00	0.00	100.00	...	no	1.000000	37	5	0.0004
84662	UZ2TflixHLqkCL9G6ykCNw	5	0	0	4	1.61	1.39	0.00	0.00	100.00	...	no	1.400000	14	18	0.0015
50079	HcL7R7ingTW8nenpD3X2cg	8	8	5	13	2.08	2.56	30.77	19.23	50.00	...	no	0.500000	3	30	0.0098
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
3090	09cpNEc8L-jr9R8-e7cJuA	6	1	2	2	1.79	0.69	20.00	40.00	40.00	...	no	1.166667	10	16	0.0012
69511	OrtDTPj1J2injmWcHyTyWw	3	1	2	8	1.10	2.08	9.09	18.18	72.73	...	no	0.666667	25	13	0.0030
77193	RjjsMfDoxbwMVPi-DLvftQ	19	2	2	7	2.94	1.95	18.18	18.18	63.64	...	yes	0.315789	12	62	0.0188
88687	W21PBCWu59Bo5LRv9-sYNg	8	0	1	5	2.08	1.61	0.00	16.67	83.33	...	no	0.250000	34	31	0.0000
107905	cD9d9XFoC_bETPzjpnRj9g	9	14	11	15	2.20	2.71	35.00	27.50	37.50	...	no	0.555556	28	35	0.0045

19582 rows × 41 columns

By removing two features, we have effectively doubled the number of rows remaining than if we just removed all rows with a NaN value. That's pretty good.

Now, let's preprocess categorical variables into dummy variables. (hint: look at `pd.get_dummies`).

```
# YOUR CODE HERE
remove_two_column['q16t_yes'] = pd.get_dummies(remove_two_column, columns = ['q16t'], drop_first = True)['q16t_yes']
remove_two_column['q16s_freshman'] = pd.get_dummies(remove_two_column, columns = ['q16s'], drop_first = True)['q16s_freshman']
remove_two_column = remove_two_column.drop(columns=['q16t', 'q16s','user_id'])
remove_two_column
```

	q3	q4	q5	q6	q7	q10	q11	q12	q13	q14	...	q16v	q16w	q16x	q16y	q16z	q16aa	q16ab
47453	8	2	1	8	2.08	2.08	18.18	9.09	72.73	10	...	8	39	0.001755	91.072917	4	0	1.000000
53000	2	1	1	2	0.69	0.69	25.00	25.00	50.00	10	...	22	6	0.000000	46.500000	0	3	0.000000
64580	1	0	0	1	0.00	0.00	0.00	0.00	100.00	5	...	37	5	0.000498	197.000000	0	0	0.000000
84662	5	0	0	4	1.61	1.39	0.00	0.00	100.00	6	...	14	18	0.001578	167.000000	1	0	1.250000
50079	8	8	5	13	2.08	2.56	30.77	19.23	50.00	9	...	3	30	0.009861	91.552083	1	13	4.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3090	6	1	2	2	1.79	0.69	20.00	40.00	40.00	9	...	10	16	0.001286	362.916667	0	0	2.500000
69511	3	1	2	8	1.10	2.08	9.09	18.18	72.73	9	...	25	13	0.003016	60.111111	1	0	1.333333
77193	19	2	2	7	2.94	1.95	18.18	18.18	63.64	11	...	12	62	0.018841	41.166667	0	3	0.500000
88687	8	0	1	5	2.08	1.61	0.00	16.67	83.33	8	...	34	31	0.000000	36.041667	0	0	0.347826
107905	9	14	11	15	2.20	2.71	35.00	27.50	37.50	8	...	28	35	0.004566	62.851852	2	0	0.000000

19582 rows × 40 columns

Now, normalize the remaining values.

```
# YOUR CODE HERE
normalize_df = remove_two_column.copy()
for column in normalize_df.columns:
  if column == 'q16t_yes' or column == 'q16s_freshman':
    continue
  normalize_df[column] = normalize_df[column]  / normalize_df[column].abs().max()
normalize_df
```

	q3	q4	q5	q6	q7	q10	q11	q12	q13	q14	...	q16v	
47453	0.031746	0.003295	0.002110	0.010349	0.376130	0.312782	0.194793	0.102261	0.7273	0.909091	...	0.150943	0
53000	0.007937	0.001647	0.002110	0.002587	0.124774	0.103759	0.267867	0.281246	0.5000	0.909091	...	0.415094	0
64580	0.003968	0.000000	0.000000	0.001294	0.000000	0.000000	0.000000	0.000000	1.0000	0.454545	...	0.698113	0.
84662	0.019841	0.000000	0.000000	0.005175	0.291139	0.209023	0.000000	0.000000	1.0000	0.545455	...	0.264151	0.
50079	0.031746	0.013180	0.010549	0.016818	0.376130	0.384962	0.329690	0.216335	0.5000	0.818182	...	0.056604	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	
3090	0.023810	0.001647	0.004219	0.002587	0.323689	0.103759	0.214293	0.449994	0.4000	0.818182	...	0.188679	0
69511	0.011905	0.001647	0.004219	0.010349	0.198915	0.312782	0.097396	0.204522	0.7273	0.818182	...	0.471698	0
77193	0.075397	0.003295	0.004219	0.009056	0.531646	0.293233	0.194793	0.204522	0.6364	1.000000	...	0.226415	0
88687	0.031746	0.000000	0.002110	0.006468	0.376130	0.242105	0.000000	0.187535	0.8333	0.727273	...	0.641509	0
107905	0.035714	0.023064	0.023207	0.019405	0.397830	0.407519	0.375013	0.309371	0.3750	0.727273	...	0.528302	0.

19582 rows × 40 columns

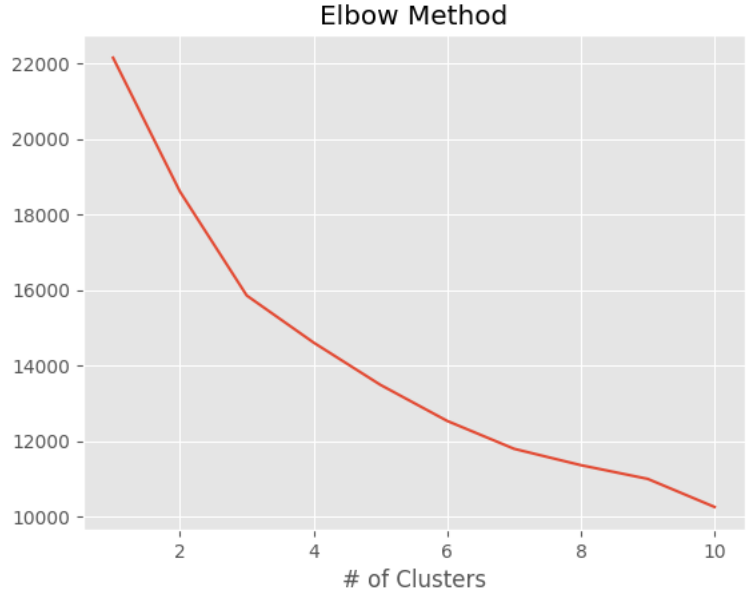
Using the the "sum of squared errors" metric along with the elbow method (make a graph and visually examine for the elbow), what is the best k to use for this dataset? (Hint: look at the `inertia_` attribute for k-means in sklearn).

The return value should be a graph to visualize the elbow method and the value of k determined from that graph.

```
# YOUR CODE HERE
ssd = []
for i in range(1,11):
  km = KMeans(n_clusters = i,n_init=10, random_state=42)
  km.fit_predict(normalize_df)
  ssd.append(km.inertia_)

plt.plot(range(1,11),ssd)
plt.xlabel('# of Clusters')
plt.title("Elbow Method")
# For KMeans use argument n_init = 10 when applicable.
```

```
Text(0.5, 1.0, 'Elbow Method')
```



Answer: The Best K value is 3

Question 10 (1 points)

For this question, please come up with your own question about this dataset and using a clustering technique as part of your method of answering it. Describe the question you propose and how clustering can answer that question. Feel free to use additional cells if needed.

Question: What is the optimal number of clusters (k) to identify distinct groups based on a combination of high review ratings and a high ratio of helpful votes among all votes?

```
# YOUR CODE HERE
def find_optimal_k(df):
    best_k = 2
    best_score = -1
    for i in range(2, 11):
        km = KMeans(n_clusters=i, random_state=42)
        labels = km.fit_predict(df)
        score = silhouette_score(df, labels)
        if score > best_score:
            best_score = score
            best_k = i
    return best_k

# Assuming 'df' is the DataFrame with the relevant columns
df = df[['q11','q12','q13','q16x']].dropna()
optimal_k = find_optimal_k(df)
print(f"The optimal number of clusters is: {optimal_k}")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from
  warnings.warn(
The optimal number of clusters is: 10
```

Written Answer

Answer: Using the KMeans clustering algorithm, we can analyze a dataset with columns q11, q12, q13, and q16x, which represent the ratio/percentages of different types of votes. By calculating the silhouette score for a range of cluster numbers (k) from 2 to 10, we can identify the best k value that maximizes cluster definition. This approach helps in discovering distinct groupings within the data based on the vote ratios, with the optimal k being the one with the highest silhouette score, indicating the clearest separation between clusters.

Bonus question (2 Points) - Reviewer overlap:

Now, let's take a look back at what we were doing last week, and use that in junction with what we've learned from above today.

For this bonus question, please:

- Download last week's dataset
- Aggregate cool, funny, and useful votes for each business id
- You may transform the aggregations (take %, log, or leave it as it is)
- Cluster this dataframe (you can choose k). Do you find any meaningful/interesting clusters?
- Assign the cluster label to each business id
- Merge this with users to show what clusters the reviewers have reviewed.

You should be returning a dataframe with the following structure in the end:

Rows: user IDs as indices.

Columns: boolean columns describing if the user ID has a review for each of the labels determined from the K-Means clustering, a boolean column describing if the user ID has a review for all of the given labels, and a column composing of lists of cluster IDs that the given user ID has written reviews for.

```
# YOUR CODE HERE
#DOWNLOADING DATASET IF NOT PRESENT
```