DATA MINING & ANALYTICS (2022)

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

```
NAME = "Lilly Liu"
```

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://askoski.berkeley.edu/~zp/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....')
```

```
--2022-09-14 21:38:57--  http://askoski.berkeley.edu/~zp/yelp_reviewers.csv
Resolving askoski.berkeley.edu (askoski.berkeley.edu)... 169.229.192.179
Connecting to askoski.berkeley.edu (askoski.berkeley.edu)|169.229.192.179|:80...
HTTP request sent, awaiting response... 200 OK
Length: 35809479 (34M) [text/csv]
Saving to: ‘yelp_reviewers.csv’

yelp_reviewers.csv  100%[===================>]  34.15M  13.6MB/s    in 2.5s

2022-09-14 21:39:00 (13.6 MB/s) - ‘yelp_reviewers.csv’ saved [35809479/35809479]

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
```

	129451	116706	144394	
user_id	kIWQXgjmVdgEs9BOgr8G5A	fXU_-5DBmNlGhI8fbX-2vQ	prF_lbKywPnZhNqvJOOaDw	8
q3	1	1	1	
q4	0	0	0	
q5	0	0	0	
q6	0	0	0	
q7	0.0	0.0	0.0	
q8	NaN	NaN	NaN	
q9	NaN	NaN	NaN	
q10	NaN	NaN	NaN	
q11	NaN	NaN	NaN	
q12	NaN	NaN	NaN	
q13	NaN	NaN	NaN	
q14	7	10	9	
q15	510.0	132.0	1792.0	
q16a	0	0	0	
q16b	0.0	0.0	0.0	
q16c	0.0	0.0	0.0	
q16d	3.0	1.0	3.0	
q16e	0.013725	0.045455	0.027344	
q16f	0.0	0.0	0.0	
q16g	0	1	1	
q16h	0	1	1	
q16i	0	0	0	
q16j	0.0	0.0	0.0	
q16k	0	0	0	
q16l	0	0	0	
q16m	3.0	0.0	12.0	
q16n	0.0	0.0	1.0	
q16o	0.0	1.0	1.0	
q16p	0.0	0.0	1.0	

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):

  ks = [2, 3, 4, 5, 6, 7, 8]
  scores = []
  x = df[['q4', 'q5', 'q6']].dropna()
  for k in ks:
    km = KMeans(n_clusters = k, random_state = 42)
    km.fit(x)
    score = silhouette_score(x, km.labels_)
    scores.append(score)
  return ks[scores.index(max(scores))]

  raise NotImplementedError()

print(q1(df))
```

```
2
```

What is the best choice of k?

```
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: The missing values from q7-q10 mainly stem from the result of taking the logarithms of q3-q7, which when taking the log of 0, will result in a `NaN` value. So, to properly clean this data, we will find the subset of data specified for this question (q7-q10), and then replace the `NaN` values with 0's

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

  ks = [2, 3, 4, 5, 6, 7, 8]
  scores = []
  x = df[['q7', 'q8', 'q9', 'q10']].dropna()
  for k in ks:
    km = KMeans(n_clusters = k, random_state = 42)
    km.fit(x)
    score = silhouette_score(x, km.labels_)
    scores.append(score)
  return ks[scores.index(max(scores))]
  raise NotImplementedError()

print(q2(df))
```

```
2
```

What is the best choice of k?

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):

  ks = [2, 3, 4, 5, 6, 7, 8]
  scores = []
  x = df[['q11', 'q12', 'q13']].dropna()
  for k in ks:
```

```
        km = KMeans(n_clusters = k, random_state = 42)
        km.fit(x)
        score = silhouette_score(x, km.labels_)
        scores.append(score)
    return ks[scores.index(max(scores))]
    raise NotImplementedError()

print(q3(df))
```

```
8
```

What is the best choice of k?

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, ...}`).

```
#Make sure you return the answer value in this function.
#The return value should be an dictionary. Eg : {0:1000,1:500,2:1460}.
from collections import Counter, defaultdict
def q4(df):
    x = df[["q11", "q12", "q13"]].dropna()
    km = KMeans(n_clusters=8, random_state = 42)
    km.fit(x)
    return Counter(km.labels_)

    # YOUR CODE HERE
    raise NotImplementedError()


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

```
Counter({2: 9962, 4: 4483, 7: 4251, 5: 3434, 1: 3064, 0: 2055, 6: 1632, 3: 1228})
```

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):
  x = df[["q11", "q12", "q13"]].dropna()
  km = KMeans(n_clusters=8, random_state = 42)
  km.fit(x)
  return km.cluster_centers_[6]
  # YOUR CODE HERE
  raise NotImplementedError()
```

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

```
[ 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):
  x = df[["q11", "q12", "q13"]].dropna()
  km = KMeans(n_clusters=8, random_state = 42)
  km.fit(x)
  print(km.cluster_centers_)
  return 24.31256856
  raise NotImplementedError()
```

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

```
[[ 3.22372749 52.6463163  44.13010706]
 [26.65194971  3.35620509 69.99171457]
 [ 0.30151074  0.41439068 99.28407749]
 [98.15302932  0.96006515  0.88692182]
 [14.77627648 24.31256856 60.91198216]
 [47.56644166  3.80018912 48.6333343 ]
 [ 1.13148897 98.30148897  0.56707721]
 [33.32621234 32.8740678  33.79618409]]
24.31256856
```

Question 7 (1 point)

Cluster the dataset using $k = 5$ 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.

Drop/replace missing values for q7-q15 as you have done in previous questions. For q14-q15, feel free to drop rows that have `NaN` values.

```
#Make sure you return the answer value in this function.
#The return value should be a float.
def q7(df):
  x = df[['q7', 'q8', 'q9', 'q10', 'q11', 'q12', 'q13', 'q14', 'q15']].dropna()
  km = KMeans(n_clusters = 5, random_state = 42)
  km.fit(x)
  score = silhouette_score(x, km.labels_)
  return score
  # YOUR CODE HERE
  raise NotImplementedError()
```

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

```
0.5481158706623568
```

Question 8 (1 point)

Cluster the dataset using $k = 5$ and using features q7-q15 (refer to the column descriptions if needed). Drop/replace missing values as you have done before.

What is the average q3 value in each of the clusters?

```
#Make sure you return the answer value in this function.
#The return value should be an Array. Eg : [10, 30, 54].
def q8(df):
  x = df[['q7', 'q8', 'q9', 'q10', 'q11', 'q12', 'q13', 'q14', 'q15']].dropna()
  km = KMeans(n_clusters = 5, random_state = 42)
  km.fit(x)
  x["id"] = km.labels_
  df2 = df[["q3", "q15"]]
  y = pd.merge(x, df2, how="left", on="q15")
  means = y.groupby("id").mean()
  return np.array(means["q3"])
  # YOUR CODE HERE
  raise NotImplementedError()
```

```
#This is a graded cell, do not edit
```

```
print(np.round_(q8(df), decimals=1, out=None))
```

```
[2.2 2.9 3.6 2.2 3.6]
```

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.

```
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.

```
print(df.dropna().shape)
print(df.drop(columns=["q8", "q9"]).dropna().shape)
```

```
(7177, 43)
(19582, 41)
```

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`).

```
nona = pd.get_dummies(df, columns = ["q8", "q9"]).dropna()
nona.head()
```

	user_id	q3	q4	q5	q6	q7	q10	q11	q12	q13
47453	Gd_IGX3BmRYbPD84ovLEoA	8	2	1	8	2.08	2.08	18.18	9.09	72.73
53000	Ihx1EQHDTIoXM35Cc08r2Q	2	1	1	2	0.69	0.69	25.00	25.00	50.00
64580	N22hkNXzJdz_v_KocOy6vA	1	0	0	1	0.00	0.00	0.00	0.00	100.00
84662	UZ2TflixHLqkCL9G6ykCNw	5	0	0	4	1.61	1.39	0.00	0.00	100.00
50079	HcL7R7ingTW8nenpD3X2cg	8	8	5	13	2.08	2.56	30.77	19.23	50.00

5 rows × 284 columns

Now, normalize the remaining values.

```
df2 = pd.DataFrame(normalize(nona.drop(columns=["user_id", "q16s", "q16t"])))
df2.head()
```

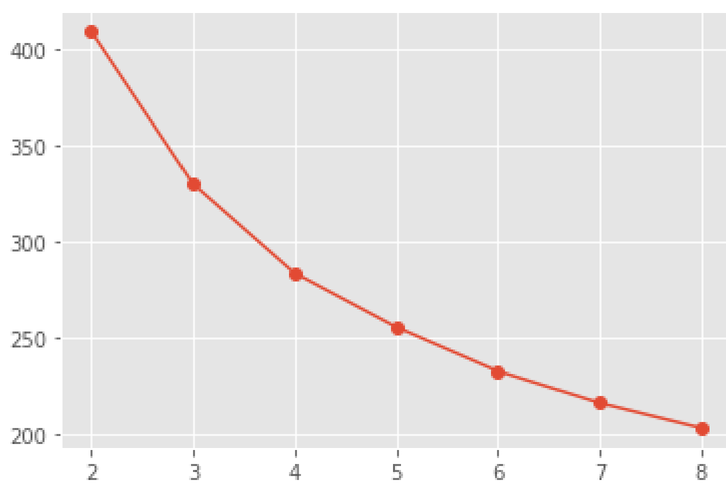
	0	1	2	3	4	5	6	7	
0	0.011846	0.002962	0.001481	0.011846	0.003080	0.003080	0.026920	0.013460	0.1
1	0.003713	0.001856	0.001856	0.003713	0.001281	0.001281	0.046410	0.046410	0.(
2	0.000492	0.000000	0.000000	0.000492	0.000000	0.000000	0.000000	0.000000	0.(
3	0.003793	0.000000	0.000000	0.003034	0.001221	0.001054	0.000000	0.000000	0.(
4	0.007589	0.007589	0.004743	0.012333	0.001973	0.002429	0.029191	0.018243	0.(

5 rows × 281 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
ks = [2, 3, 4, 5, 6, 7, 8]
errors = []
x = df2
for k in ks:
  km = KMeans(n_clusters = k, random_state = 42)
  km.fit(x)
  error = km.inertia_
  errors.append(error)
plt.plot(ks, errors, marker = "o")
```

```
[<matplotlib.lines.Line2D at 0x7f8e42c510d0>]
```



Based on the SSE and elbow method, the best k to use for this dataset is k=4.

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: YOUR QUESTION HERE

```
print("From the best cluster in question 3, are the percentage of types of votes equal
def q10(df):
  x = df[["q11", "q12", "q13"]].dropna()
  km = KMeans(n_clusters=8, random_state = 42)
  km.fit(x)
  return (km.cluster_centers_)
print(q10(df))
```

```
From the best cluster in question 3, are the percentage of types of votes equally
[[ 3.22372749 52.6463163  44.13010706]
 [26.65194971  3.35620509 69.99171457]
 [ 0.30151074  0.41439068 99.28407749]
 [98.15302932  0.96006515  0.88692182]
 [14.77627648 24.31256856 60.91198216]
 [47.56644166  3.80018912 48.6333343 ]
 [ 1.13148897 98.30148897  0.56707721]
 [33.32621234 32.8740678  33.79618409]]
```

Written Answer

The question is asking that in the best cluster from problem 3, are the percentages of different types of votes per review equally distributed? Since there are 3 types of votes, having centers close to 33.3% would mean the votes are equally distributed in the cluster. Looking at the centers, it is clear that most are not equally distributed.

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
```

```
#This is a graded cell, do not edit
print(bonus_df.head())
```