# K-mean_clustering

April 23, 2022

## 1 Data Exploration & K-means

[ ]:
```
import numpy as np
import pandas as pd
import sklearn
import seaborn as sns
import matplotlib.pyplot as plt
import random

random.seed(10)
```

[ ]:
```
df = pd.read_csv('processed_response.zip')
```

[ ]:
```
df.head()
```

[ ]:
```
   Polls ID  Assessment reports Student ID  \
0   12522.0                           41.0
1   12522.0                          335.0
2   12522.0                          318.0
3   12522.0                          315.0
4   12522.0                          297.0

                             Poll Responses Response  \
0  The strengths of Plato's approach is his const…
1  In the breakout we discussed if outside the ca…
2  Back to cmmon confusion time: the section 'und…
3  Most difficult weakness is that his position w…
4  I'm still trying to understand the significanc…

  Assessment reports Hashtag  Assessment reports Score  time_stamp  \
0        #objectivemorality                       2.0           1
1        #objectivemorality                       3.0           1
2        #objectivemorality                       2.0           1
3        #objectivemorality                       2.0           1
4        #objectivemorality                       2.0           1

                                  tokenized_responses  \
```

```
0  ['The', 'strengths', 'of', "Plato's", 'approac…
1  ['In', 'the', 'breakout', 'we', 'discussed', '…
2  ['Back', 'to', 'cmmon', 'confusion', 'time', '…
3  ['Most', 'difficult', 'weakness', 'is', 'that'…
4  ["I'm", 'still', 'trying', 'to', 'understand',…

                                   stemmed_responses  \
0  ['the', 'strength', 'of', 'plato', 'approach',…
1  ['in', 'the', 'breakout', 'we', 'discuss', 'if…
2  ['back', 'to', 'cmmon', 'confus', 'time', 'the…
3  ['most', 'difficult', 'weak', 'is', 'that', 'h…
4  ["i'm", 'still', 'tri', 'to', 'understand', 't…

                                     clean_responses  \
0  ['strength', 'plato', 'approach', 'construct',…
1  ['breakout', 'discuss', 'outsid', 'cave', 'mig…
2  ['back', 'cmmon', 'confus', 'time', 'section',…
3  ['difficult', 'weak', 'posit', 'understand', '…
4  ["i'm", 'still', 'tri', 'understand', 'signifi…

                                              string     LOs/ HCs College  \
0  strength plato approach construct whole framew…  objmorality      AH
1  breakout discuss outsid cave might bigger cave…  objmorality      AH
2  back cmmon confus time section understand inte…  objmorality      AH
3  difficult weak posit understand testabl like i…  objmorality      AH
4  i'm still tri understand signific cave analog …  objmorality      AH

  Course
0  AH111
1  AH111
2  AH111
3  AH111
4  AH111
```

[ ]:
```
df['string'] = df['string'].values.astype('U')
```

## 2 Transform dataframe into tfidf

Key points - Tf-idf computes weights of the words (how relevant they are in the document) - The higher the TF-IDF score, the rare or unique the term is, and vice versa - TF and IDF are calculated in different ways (either by ranking frequency values, or by dividing the words frequency overall by the number of words given in the dictionary)

Formula

- Term Frequency (TF):
  - Calculate the term frequency
  - tf(t,d) = count of t in d / number of words in d

- Inverse Document Frequency(IDF)
  - Weigh down the frequent terms may appear a lot of times but have little importance
  - idf(t) = log(N/(df + 1))
- Calculation: TF is multiplied by IDF

[ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vector = TfidfVectorizer()
tfidf_matrix = tfidf_vector.fit_transform(df['string'])
tfidf_matrix
```

[ ]:
```
<181941x79400 sparse matrix of type '<class 'numpy.float64'>'
        with 6018555 stored elements in Compressed Sparse Row format>
```

[ ]:
```
#Explore td-idf for one doc: print words & tf-idf scores
feature_names = tfidf_vector.get_feature_names()
doc = 3 #random doc
feature_index = tfidf_matrix[doc,:].nonzero()[1] #Return the indices of the␣
 ↪elements that are non-zero.
tfidf_scores = zip(feature_index, [tfidf_matrix[doc, x] for x in feature_index])
for w, s in [(feature_names[i], s) for (i, s) in tfidf_scores]:
  print(w, s)
```

```
get 0.1602432886945829
way 0.14154666427634024
ani 0.18754112365767017
whether 0.1909802791293325
decid 0.22724769256321592
lot 0.1967549100540339
quit 0.2530798913029075
struggl 0.2884601285317333
peopl 0.13840463662592684
incomprehens 0.451983870307689
like 0.13508086186616905
testabl 0.27084504953544314
posit 0.1880356167579713
difficult 0.23321560468563032
understand 0.15017855307918143
cave 0.3964584008260598
weak 0.24596725240200004
```

```
/Users/swimmingcircle/Library/Python/3.9/lib/python/site-
packages/sklearn/utils/deprecation.py:87: FutureWarning: Function
get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will
be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

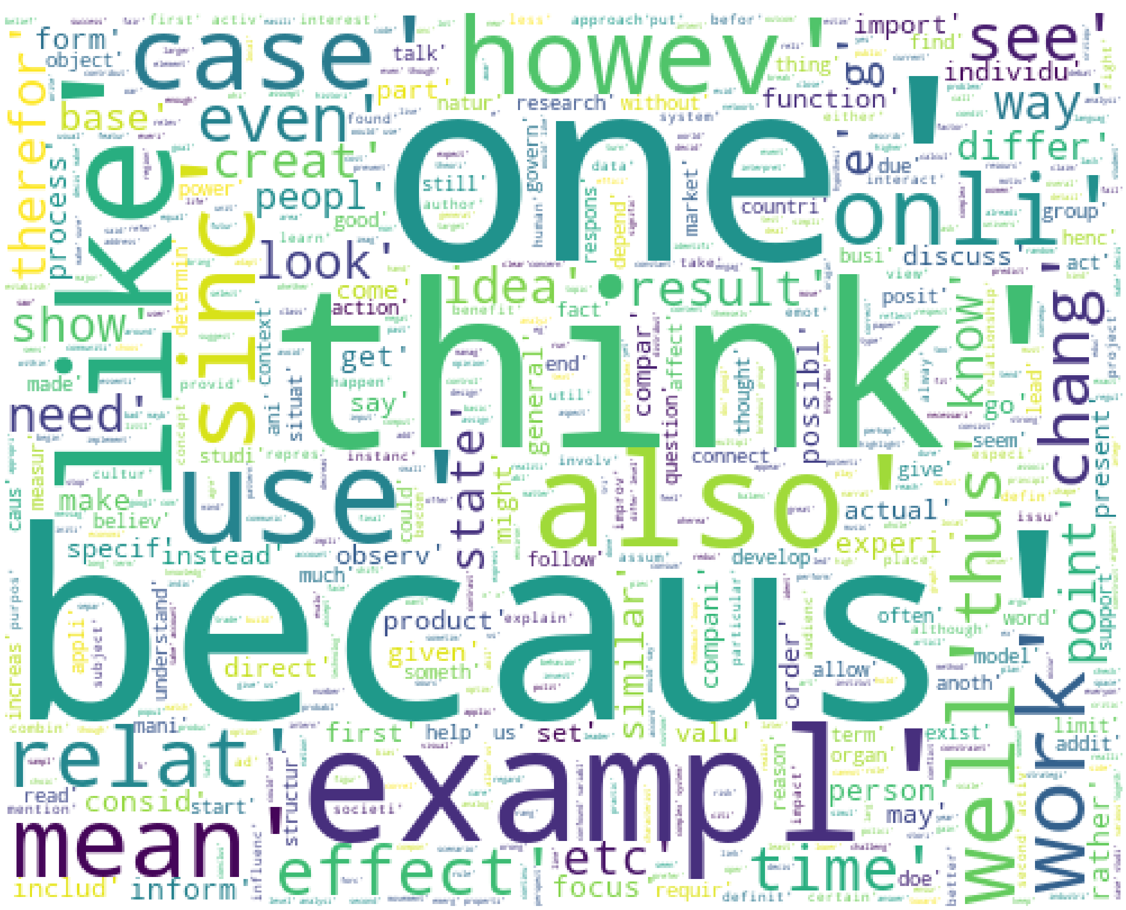
We explore words and their tf-idf scores for a random poll response in our dataset.

## 3 Visualized by wordcloud

### 3.1 Visualize bags of words

[ ]:
```
# Import the wordcloud library
from wordcloud import WordCloud

# Join the different processed titles together.
long_string = ','.join(df['clean_responses'])

plt.figure(figsize=(40, 30))
# Create a WordCloud object
wordcloud = WordCloud(width = 500, height = 400, background_color="white",␣
↪max_words=5000)
# Generate a word cloud
wordcloud.generate(long_string)
# Visualize the word cloud
wordcloud.to_image()
```

[ ]:




<Figure size 2880x2160 with 0 Axes>

### 3.2 Visualize tf-idf

We visualize the word cloud for tf-idf matrix as well, since it takes a long time to run, we slice the matrix that remains around 20% (15000/total # of cols) of the information.

[ ]:
```
tfidf_matrix
```

[ ]:
```
<181941x79400 sparse matrix of type '<class 'numpy.float64'>'
        with 6018555 stored elements in Compressed Sparse Row format>
```

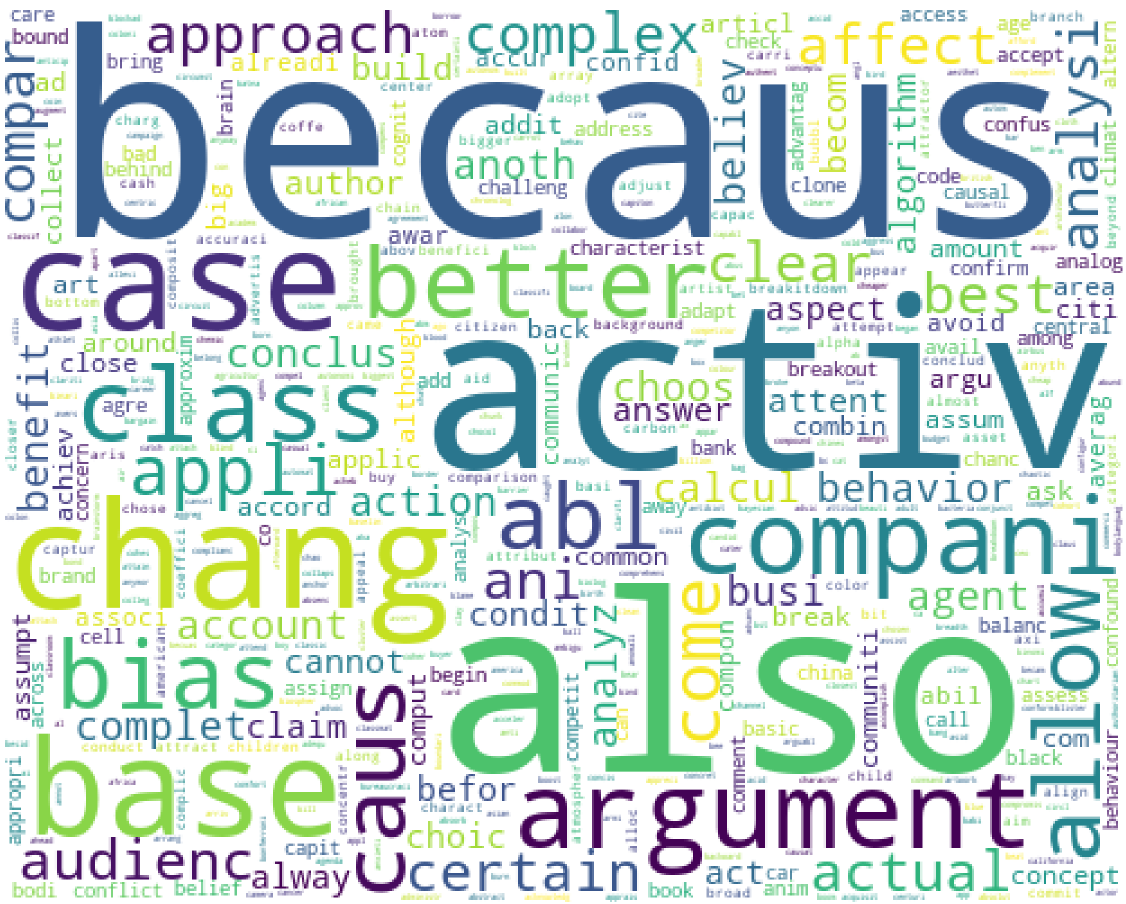
[ ]:
```
response = tfidf_matrix[:, :15000] #slice the matrix
df_tfidf_sklearn = pd.DataFrame(response.toarray(),columns= tfidf_vector.
 ↪get_feature_names()[:15000])
df_tfidf_sklearn.head()
```

[ ]:
```
    __   ___   ____   _____   ______   _______   ________   _________   __________  \
0  0.0   0.0   0.0    0.0     0.0      0.0       0.0        0.0         0.0
1  0.0   0.0   0.0    0.0     0.0      0.0       0.0        0.0         0.0
2  0.0   0.0   0.0    0.0     0.0      0.0       0.0        0.0         0.0
3  0.0   0.0   0.0    0.0     0.0      0.0       0.0        0.0         0.0
4  0.0   0.0   0.0    0.0     0.0      0.0       0.0        0.0         0.0

   ___________  …  conjunst  conjunt  conjur  conjvey  conlad  conland  \
0          0.0  …       0.0      0.0     0.0      0.0     0.0      0.0
1          0.0  …       0.0      0.0     0.0      0.0     0.0      0.0
2          0.0  …       0.0      0.0     0.0      0.0     0.0      0.0
3          0.0  …       0.0      0.0     0.0      0.0     0.0      0.0
4          0.0  …       0.0      0.0     0.0      0.0     0.0      0.0

   conlcud  conlcus  conley  conlfict
0      0.0      0.0     0.0       0.0
1      0.0      0.0     0.0       0.0
2      0.0      0.0     0.0       0.0
3      0.0      0.0     0.0       0.0
4      0.0      0.0     0.0       0.0

[5 rows x 15000 columns]
```

[ ]:
```
#some up all frequency
tf_idf_counter = df_tfidf_sklearn.T.sum(axis=1)
```

[ ]:
```
wordcloud = WordCloud(width = 500, height = 400, background_color="white",␣
 ↪max_words=5000)
wordcloud.generate_from_frequencies(tf_idf_counter)
wordcloud.to_image()
```

[ ]:



From the word clouds, we layout the common words for bags of words and - Bags of words: think, one, because, also, however, like, example, use, work, only, need, know, mean, relate, use, effect. - Tf-idf: because, also, change, base, case, active, companies, arguement, affect, class, apply,complex, allow, better, bias.

We obseverve that though the most common words for both bags of words and tf-idf only provide us words the appear freuqnently, tf-idf gives us more useful information.

## 3.3 Cluster by K means

### 3.3.1 5 clusters

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

num_clusters = 5

km = KMeans(n_clusters=num_clusters)

%time km.fit(tfidf_matrix)
```

```
clusters = km.labels_.tolist()

df['cluster'] = np.array(clusters)
terms = tfidf_vector.get_feature_names_out()
```

```
CPU times: user 4min 35s, sys: 10.1 s, total: 4min 45s
Wall time: 42.1 s
```

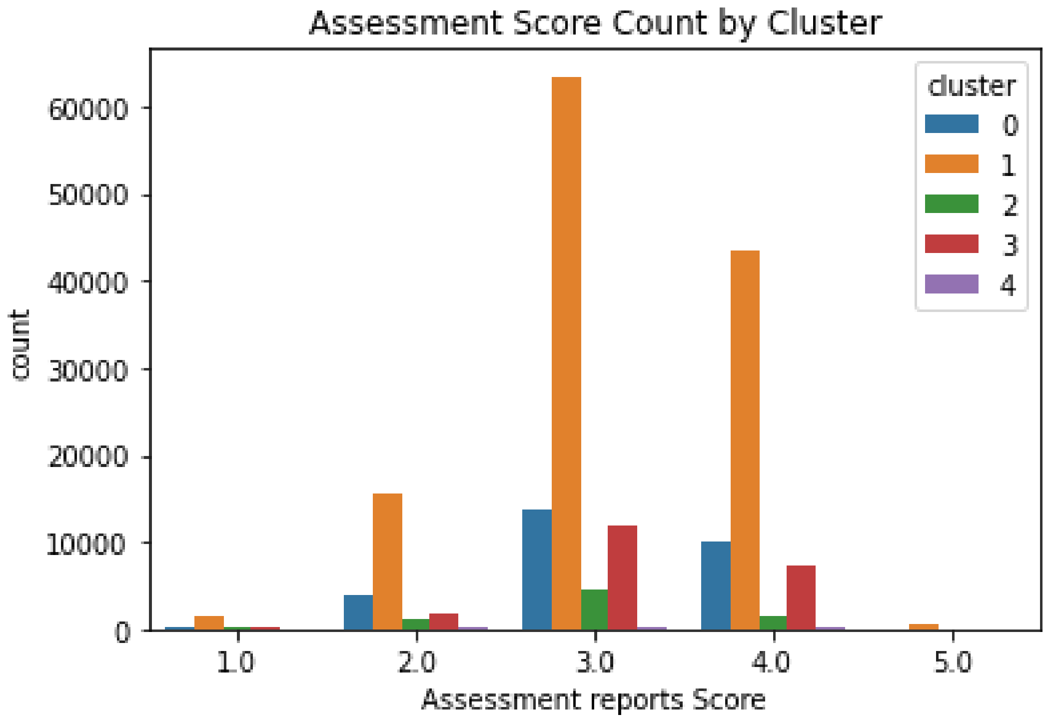
[ ]:
```
print("Top terms per cluster:")
print()
#sort cluster centers by proximity to centroid
order_centroids = km.cluster_centers_.argsort()[:, ::-1] #sort in decending␣
↪order

for i in range(num_clusters):
    print("Cluster %d words:" % i, end='')

    for ind in order_centroids[i, :15]: # with 15 words per cluster
        print(' %s' % terms[ind],end=',')
    print() #add whitespace

print()
print()
```

```
Top terms per cluster:

Cluster 0 words: data, variabl, model, use, would, sampl, valu, distribut,
probabl, test, hypothesi, observ, mean, studi, differ,
Cluster 1 words: would, use, becaus, one, peopl, differ, think, make, exampl,
also, like, could, system, way, argument,
Cluster 2 words: problem, solut, solv, constraint, water, use, rightproblem,
differ, identifi, state, goal, subproblem, breakitdown, step, one,
Cluster 3 words: compani, market, product, custom, risk, busi, com, countri,
https, invest, cost, economi, price, googl, doc,
Cluster 4 words: poll, complet, student, present,
faazillexzvnfbceqmeprnivsrzaadmewqcnu, fabian, fabianokafor, fabiola, fabl,
fabric, fabul, fac, faabi, facad, facbook,
```

[ ]:
```
sns.countplot('Assessment reports Score', hue = 'cluster', data = df)
plt.title('Assessment Score Count by Cluster')
```

```
/Users/swimmingcircle/Library/Python/3.9/lib/python/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable
as a keyword arg: x. From version 0.12, the only valid positional argument will
be `data`, and passing other arguments without an explicit keyword will result
in an error or misinterpretation.
  warnings.warn(
```

[ ]: Text(0.5, 1.0, 'Assessment Score Count by Cluster')




### 3.3.2 10 clusters

[ ]:
```
km_10 = KMeans(n_clusters=10)

%time km_10.fit(tfidf_matrix)

clusters10 = km_10.labels_.tolist()

df['cluster10'] = np.array(clusters10)
terms10 = tfidf_vector.get_feature_names_out()
```

```
CPU times: user 6min 34s, sys: 15.1 s, total: 6min 49s
Wall time: 1min 4s
```

[ ]:
```
print("Top terms per cluster:")
print()
#sort cluster centers by proximity to centroid
order_centroids = km_10.cluster_centers_.argsort()[:, ::-1]

for i in range(10):
    print("Cluster %d words:" % i, end='')
```

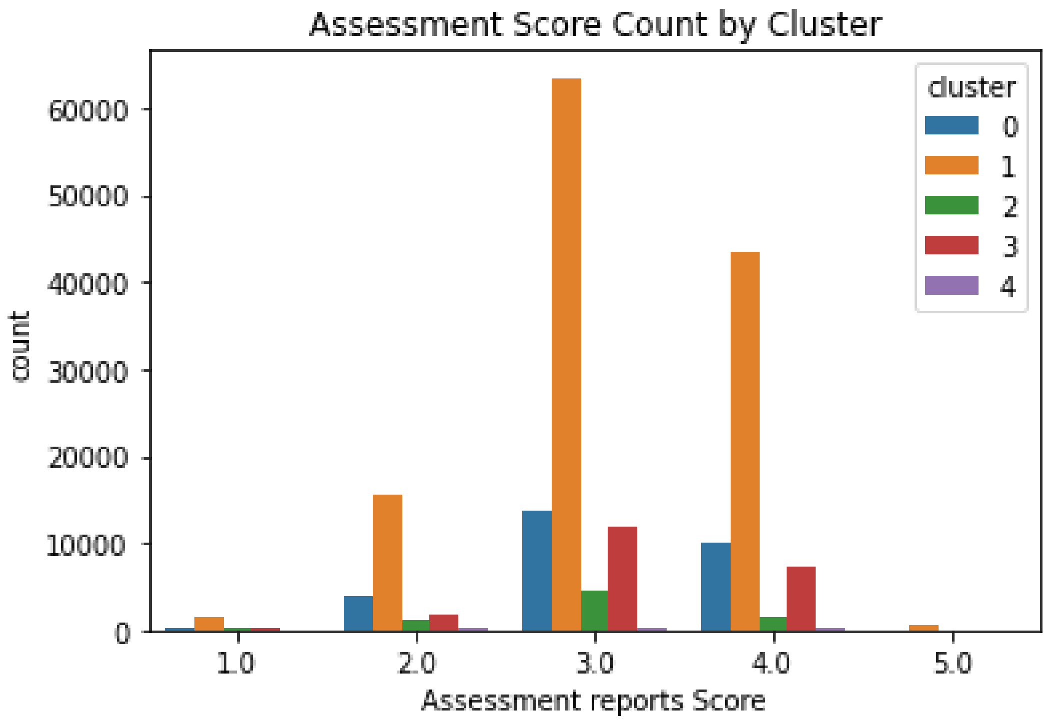
```
    for ind in order_centroids[i, :15]: #replace 6 with n words per cluster
        print(' %s' % terms10[ind],end=',')
    print() #add whitespace

print()
print()
```

```
Top terms per cluster:

Cluster 0 words: doc, https, com, googl, edit, document, usp, share, kgi, edu,
minerva, spreadsheet, drive, colab, gid,
Cluster 1 words: would, time, use, function, number, valu, becaus, one, chang,
probabl, vector, get, node, first, tree,
Cluster 2 words: system, level, emerg, agent, interact, individu, properti,
complex, network, differ, predict, behavior, analysi, social, understand,
Cluster 3 words: variabl, studi, hypothesi, test, treatment, control, observ,
group, experi, effect, would, confound, differ, use, research,
Cluster 4 words: compani, market, product, custom, risk, busi, countri, invest,
cost, economi, price, would, growth, financi, increas,
Cluster 5 words: poll, complet, student, present,
faazillexzvnfbceqmeprnivsrzaadmewqcnu, fabian, fabianokafor, fabiola, fabl,
fabric, fabul, fac, faabi, facad, facbook,
Cluster 6 words: peopl, think, use, one, would, becaus, differ, make, way, like,
also, exampl, understand, could, person,
Cluster 7 words: data, model, distribut, sampl, use, probabl, mean, would, valu,
predict, line, normal, graph, differ, variabl,
Cluster 8 words: argument, thesi, sentenc, evid, induct, logic, premis, deduct,
true, conclus, statement, use, truth, claim, clone,
Cluster 9 words: problem, solut, solv, constraint, use, water, rightproblem,
identifi, differ, state, goal, breakitdown, subproblem, step, one,
```

[ ]:
```
# Calculate the score distribution per cluster
sns.countplot('Assessment reports Score', hue = 'cluster', data = df)
plt.title('Assessment Score Count by Cluster')
```

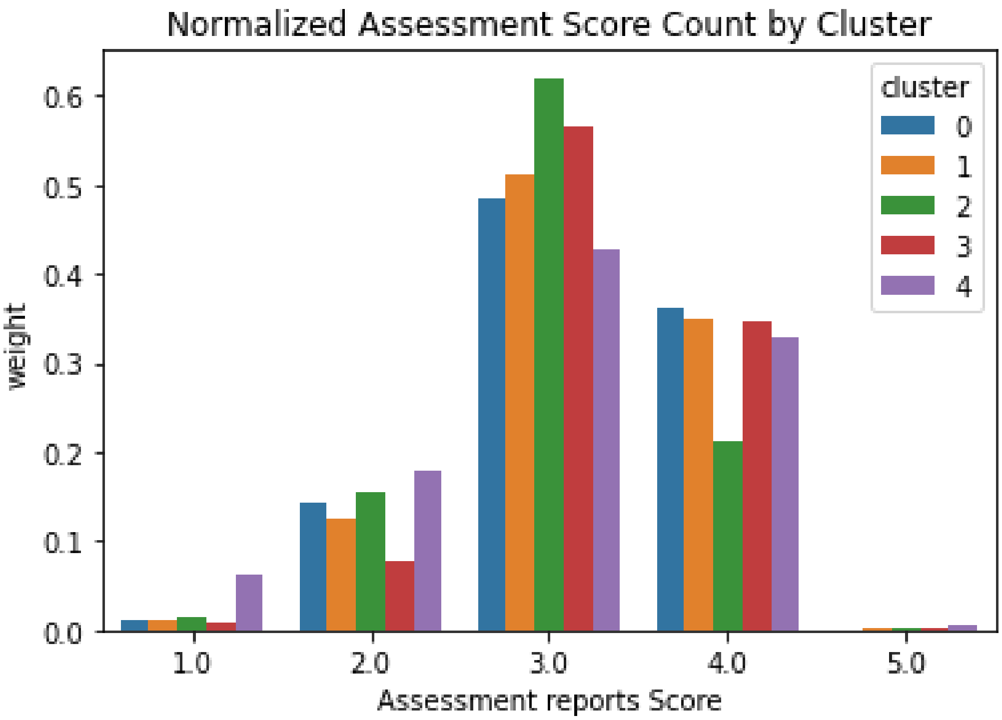
```
/Users/swimmingcircle/Library/Python/3.9/lib/python/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable
as a keyword arg: x. From version 0.12, the only valid positional argument will
be `data`, and passing other arguments without an explicit keyword will result
in an error or misinterpretation.
  warnings.warn(
```

[ ]:
```
Text(0.5, 1.0, 'Assessment Score Count by Cluster')
```

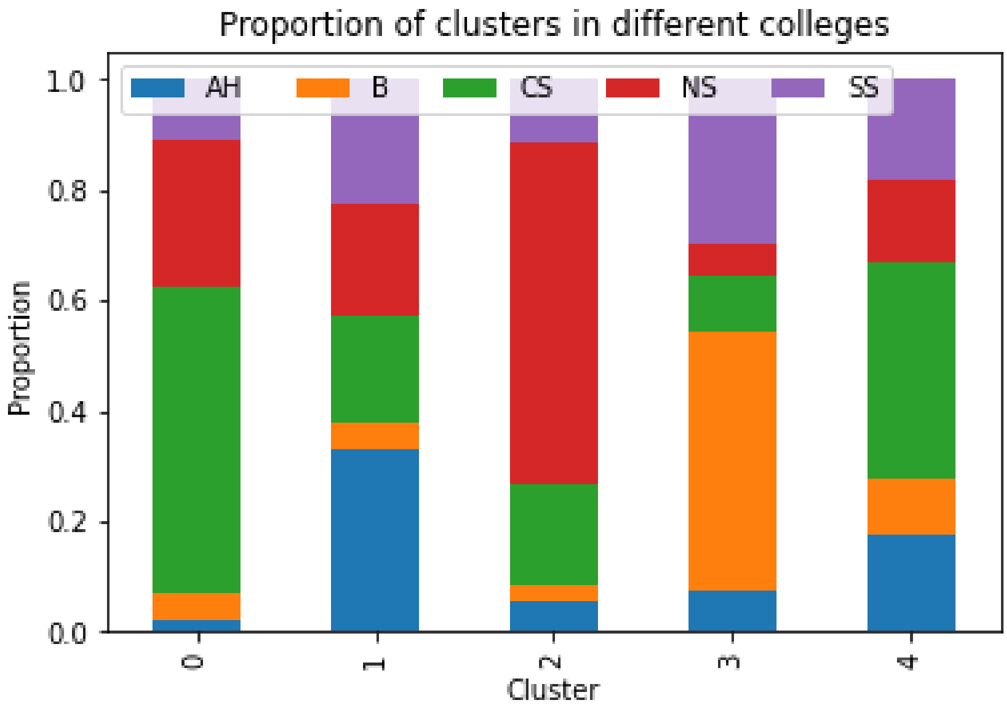



[ ]:
```
# Normalize the score distribution per cluster
norm_df = df["Assessment reports Score"].groupby(df["cluster"]).value_counts().
 ↪rename('count').reset_index()
norm_df = norm_df.assign(weight=norm_df['count']/norm_df.
 ↪groupby('cluster')['count'].transform('sum'))

sns.barplot(x="Assessment reports Score", y="weight", hue = 'cluster',␣
 ↪data=norm_df)
plt.title('Normalized Assessment Score Count by Cluster')
```

[ ]:
```
Text(0.5, 1.0, 'Normalized Assessment Score Count by Cluster')
```



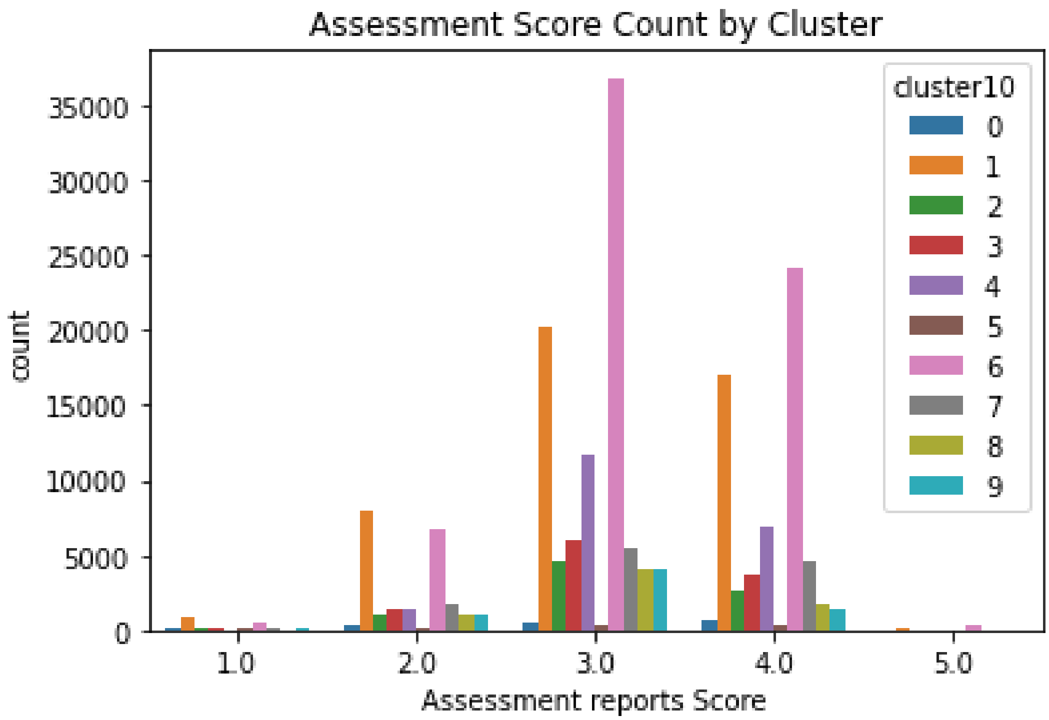

[ ]:
```
# College distribution per cluster
cross_tab_prop = pd.crosstab(index=df['cluster'],
                             columns=df['College'],
                             normalize="index")
cross_tab_prop


cross_tab_prop.plot(kind='bar',
                    stacked=True)

plt.legend(loc="upper left", ncol = 5)
plt.xlabel("Cluster")
plt.ylabel("Proportion")
plt.title("Proportion of clusters in different colleges")
plt.show()
```




[ ]:
```
sns.countplot('Assessment reports Score', hue = 'cluster10', data = df)
plt.title('Assessment Score Count by Cluster')
```

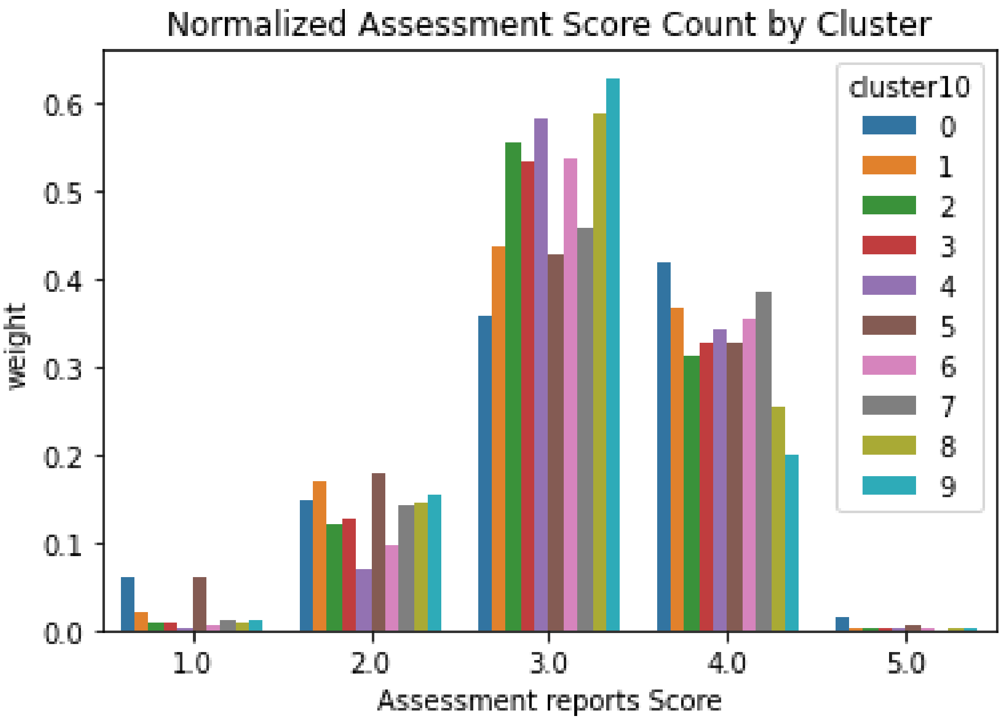
```
/Users/swimmingcircle/Library/Python/3.9/lib/python/site-
packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable
as a keyword arg: x. From version 0.12, the only valid positional argument will
be `data`, and passing other arguments without an explicit keyword will result
in an error or misinterpretation.
  warnings.warn(
```

[ ]:
```
Text(0.5, 1.0, 'Assessment Score Count by Cluster')
```

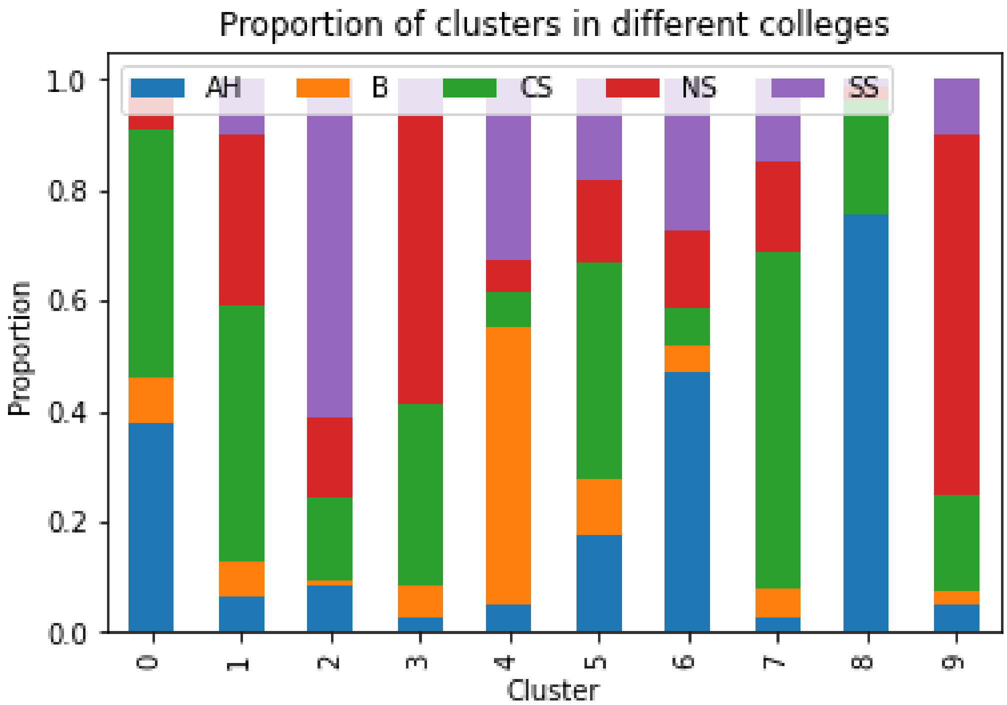



[ ]:
```
norm_df10 = df["Assessment reports Score"].groupby(df["cluster10"]).
    ↪value_counts().rename('count').reset_index()
norm_df10 = norm_df10.assign(weight=norm_df10['count']/norm_df10.
    ↪groupby('cluster10')['count'].transform('sum'))

sns.barplot(x="Assessment reports Score", y="weight", hue = 'cluster10',␣
    ↪data=norm_df10)
plt.title('Normalized Assessment Score Count by Cluster')
```

[ ]:
```
Text(0.5, 1.0, 'Normalized Assessment Score Count by Cluster')
```




[ ]:
```
cross_tab_prop = pd.crosstab(index=df['cluster10'],
                             columns=df['College'],
                             normalize="index")
cross_tab_prop


cross_tab_prop.plot(kind='bar',
                    stacked=True)

plt.legend(loc="upper left", ncol = 5)
plt.xlabel("Cluster")
plt.ylabel("Proportion")
plt.title("Proportion of clusters in different colleges")
plt.show()
```




# 4 Visualize K-means result

We use `TruncatedSVD` to decompose `tfidf_matrix`, because the matrix is too fat to plot it directly and `TruncatedSVD` works well with sparse data. We then attempt to plot the clusters with 2 components.

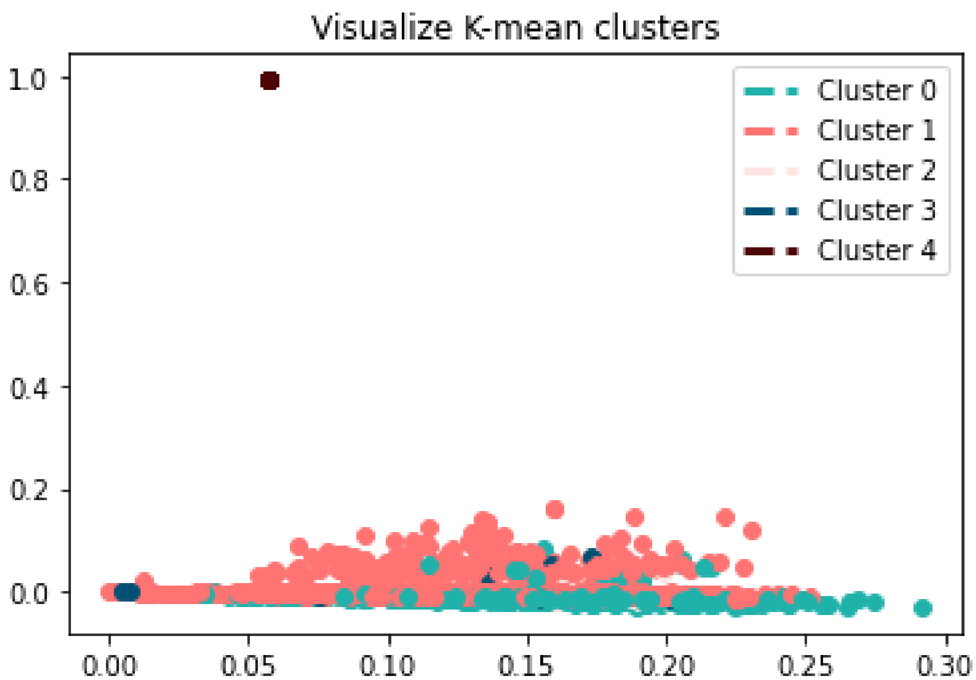
[ ]:
```
from sklearn.decomposition import TruncatedSVD

labels_color_map = {
    0: '#20b2aa', 1: '#ff7373', 2: '#ffe4e1', 3: '#005073', 4: '#4d0404'}

svd = TruncatedSVD(n_components=2)
reduced_matrix = svd.fit_transform(tfidf_matrix)
print(reduced_matrix.shape)
```

```
(181941, 2)
```

[ ]:
```
#slice the matrix even more
rr_matrix = reduced_matrix[:3000,:]
```

[ ]:
```
from matplotlib.lines import Line2D

fig, ax = plt.subplots()
```

```
lines = [Line2D([0], [0], color=c, linewidth=3, linestyle='--') for c in␣
↪labels_color_map.values()]
for index, instance in enumerate(rr_matrix):
    pca_comp_1, pca_comp_2 = rr_matrix[index]
    color = labels_color_map[clusters[index]]
    ax.scatter(pca_comp_1, pca_comp_2, c=color)
plt.title('Visualize K-mean clusters')
plt.legend(lines, ['Cluster 0', 'Cluster 1','Cluster 2','Cluster 3','Cluster␣
↪4'])
plt.show()
```




However, since the reduced matrix after `TruncatedSVD` is still too big to plot, we slice it into 3000 data points. Therefore, some clusters are not inlcuded. From the visualization, it seems that the data isn't accurately clustered. However, it can because of over reduced and slicing of it, or 2 components aren't the right numbers of dimensions to visualize it.