

Problem 1

1a

Both resulting matrices are identical. This makes sense because we are only applying transformations to the data in a different order. Both IDF and word count normalization depend on the original data, therefore, the order does not matter which order you apply. See Appendix A for python code used to normalize the data.

1b

Article "tmnt raph" is the closest to "tmnt mike", reference Table 1 for distance calculations.

Table 1: Euclidean distance between given document and the document named "tmnt mike"

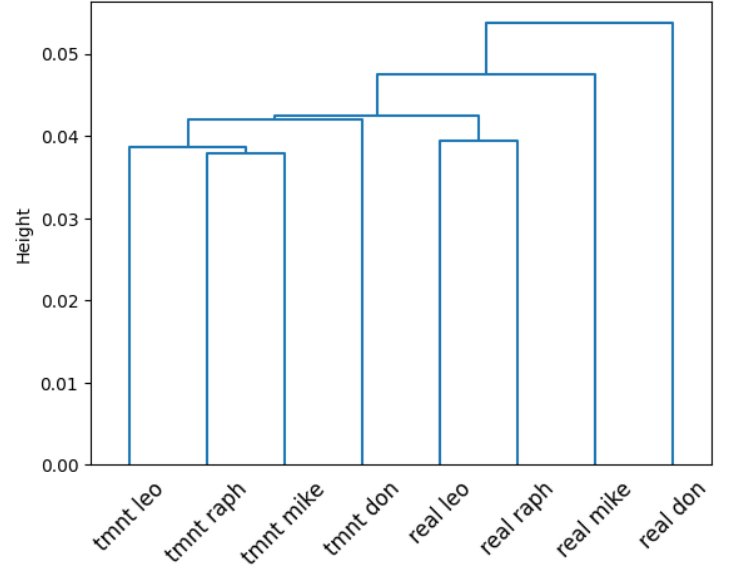
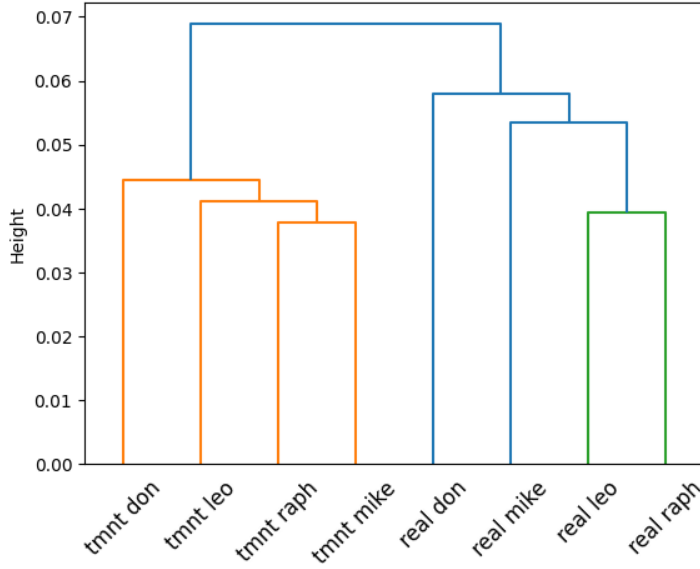
Article	Euclidean distance to "tmnt mike"
tmnt leo	0.0412
tmnt raph	0.0378
tmnt don	0.0420
real leo	0.0489
real raph	0.0449
real mike	0.0475
real don	0.0638

1c

The distance matrix is found in Table 2. When running hierarchical agglomerative clustering, complete linkage provides the best clusters when $K = 2$ because it will cluster the Teenage Mutant Ninja Turtles articles together and the artist articles together, reference Figures 1a and 1b for the dendrograms.

Table 2: Distance matrix for documents

	tmnt leo	tmnt raph	tmnt mike	tmnt don	real leo	real raph	real mike	real don
tmnt leo	0.0000	0.0386	0.0412	0.0444	0.0425	0.0496	0.0648	0.0671
tmnt raph	0.0386	0.0000	0.0378	0.0429	0.0550	0.0462	0.0668	0.0688
tmnt mike	0.0412	0.0378	0.0000	0.0420	0.0489	0.0449	0.0475	0.0638
tmnt don	0.0444	0.0429	0.0420	0.0000	0.0537	0.0504	0.0619	0.0537
real leo	0.0425	0.0550	0.0489	0.0537	0.0000	0.0394	0.0534	0.0553
real raph	0.0496	0.0462	0.0449	0.0504	0.0394	0.0000	0.0504	0.0544
real mike	0.0648	0.0668	0.0475	0.0619	0.0534	0.0504	0.0000	0.0580
real don	0.0671	0.0688	0.0638	0.0537	0.0553	0.0544	0.0580	0.0000



(a) Hierarchical Agglomerative Clustering with complete linkage (b) Hierarchical Agglomerative Clustering with complete linkage

1d

The top 20 most common words and their word count are found in Table 3. They make up 24.3425% of the word count. It takes **254** words to account for at least 50% of the words in the documents.

Table 3: Top 20 most common words

Word	the	and	his	was	leonardo	that	for	with	micangelo	raphael
Count	2664	1127	636	453	342	297	282	279	277	212
Word	from	this	which	turtles	donatello	him	series	were	who	one
Count	209	189	129	127	117	116	115	111	110	103

1e

Our collection of 8 Wikipedia articles appears to follow Zipf's law. When looking at Figure 2, you can see the data is generally linear with a slope = 1. This implies that the number of times words appear in our article generally follows Zipf's law.

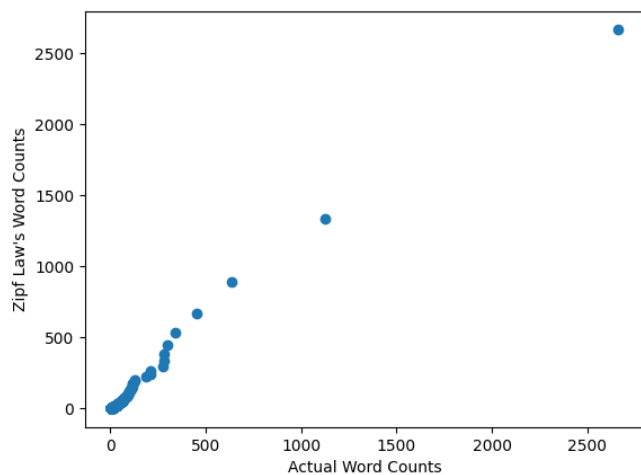


Figure 2: Word counts in our articles vs Zipf's Law theoretical word counts

Problem 2 a

$$\frac{1}{2} \sum_{k=1}^K \frac{1}{n_k} \sum_{(i)=k} \sum_{(j)=k} \|X_i - X_j\|_2^2$$

add and subtract \bar{X}_k

$$\sum_{(i)=k} \sum_{(j)=k} \|(X_i - \bar{X}_k) - (X_j - \bar{X}_k)\|_2^2$$

by inner product

$$\sum_{(i)=k} \sum_{(j)=k} \|X_i - \bar{X}_k\|_2^2 + \|X_j - \bar{X}_k\|_2^2 - 2 \langle X_i - \bar{X}_k, X_j - \bar{X}_k \rangle$$

by summation rules

$$\sum_{(i)=k} \sum_{(j)=k} \|X_i - \bar{X}_k\|_2^2 + \sum_{(i)=k} \sum_{(j)=k} \|X_j - \bar{X}_k\|_2^2 - 2 \langle X_i - \bar{X}_k, X_j - \bar{X}_k \rangle$$

$$\frac{1}{2} \sum_{k=1}^K \frac{1}{n_k} \left[n_k \sum_{(i)=k} \|X_i - \bar{X}_k\|_2^2 + n_k \sum_{(j)=k} \|X_j - \bar{X}_k\|_2^2 - 0 \right]$$

$$\sum_{(i)=k} X_i - \bar{X} \leftarrow \text{because}$$

$$= \sum_{(i)=k} X_i - n_k \bar{X}$$

$$= n_k \bar{X} - n_k \bar{X} = 0$$

continued next page

Problem 2 a continued.

$$\frac{1}{2} \sum_{k=1}^K \frac{1}{n_k} \left[n_k \sum_{C(i)=k} \|x_i - \bar{x}_k\|_2^2 + n_k \sum_{C(j)=k} \|x_j - \bar{x}_k\|_2^2 \right]$$

$$= \frac{1}{2} \sum_{k=1}^K \frac{1}{n_k} \left[\cancel{2n_k} \sum_{C(i)=k} \|x_i - \bar{x}_k\|_2^2 \right]$$

$$= \sum_{k=1}^K \sum_{C(i)=k} \|x_i - \bar{x}_k\|_2^2$$

Problem 2b

$$\sum_{i=1}^m \|z_i - c\|_2^2$$

$$= \sum_{i=1}^m \left(\sqrt{(z_1 - c)^2 + (z_2 - c)^2 + \dots + (z_m - c)^2} \right)^2$$

$$= \sum_{i=1}^m (z_1 - c)^2 + (z_2 - c)^2 + \dots + (z_m - c)^2$$

$$\frac{\partial}{\partial c} = \sum_{i=1}^m 2(z_1 - c) + 2(z_2 - c) + \dots + 2(z_m - c)$$

$$\text{set } \frac{\partial}{\partial c} = 0 = \sum_{i=1}^m 2(z_1 + z_2 + \dots + z_m) + \sum_{i=1}^m 2mc$$

$$2m \sum_{i=1}^m z_i - 2m^2 c = 0$$

$$2m \sum_{i=1}^m z_i = 2m^2 c$$

$$c = \frac{1}{m} \sum_{i=1}^m z_i$$

Problem 2c

m_k = current centroid of cluster k

$c(i)$ = cluster assignment of i

$$c(i) = \operatorname{argmin}_{1 \leq k \leq K} \|X_i - m_k\|^2$$

$$m_k^{t+1} = \frac{1}{n_k} \sum_{c(i)=k} X_i$$

which minimizes

$$\sum_{k=1}^K \sum_{c(i)=k} \|X_i - \bar{X}_k\|^2$$

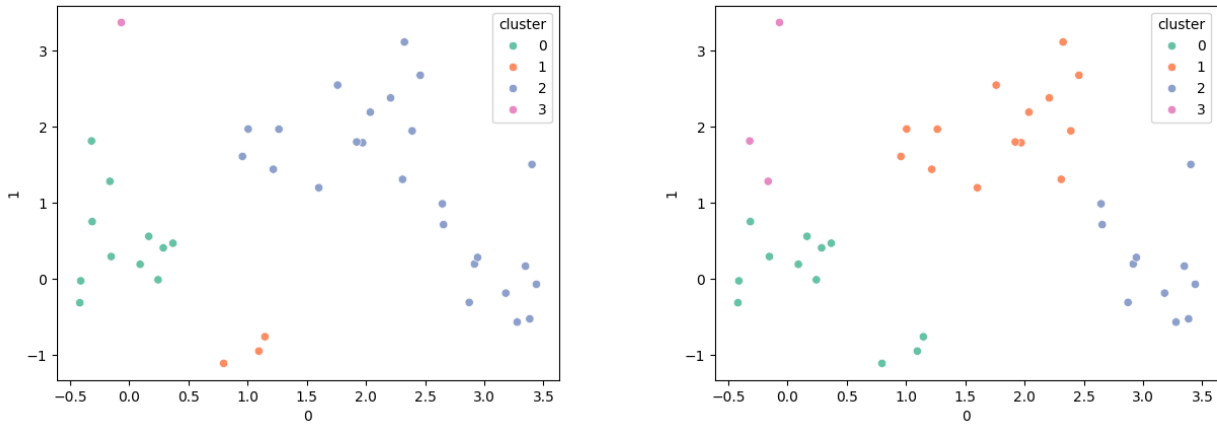
$$\sum_{k=1}^K \sum_{c(i)=k} \|X_i - m_k^t\|^2 \geq \sum_{k=1}^K \sum_{c(i)=k} \|X_i - m_k^{t+1}\|^2$$

because each iteration you assign X_i to closest m_k

Problem 3

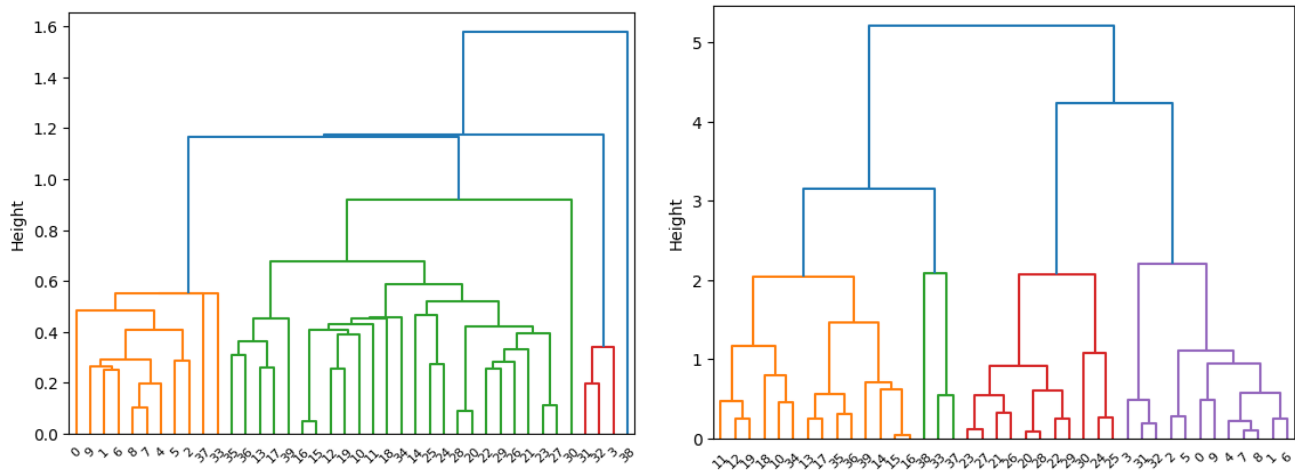
3a,3b

The hierarchical agglomerative clustering using both single linkage and complete linkage are found in Figure 3. Their dendrogram plots are found in Figure 4.



(a) Problem 3 data with K=4 and using single linkage to cluster (b) Problem 3 data with K=4 and using complete linkage to cluster

Figure 3: Hierarchical Agglomerative Cluster plots for problems 3a and 3b

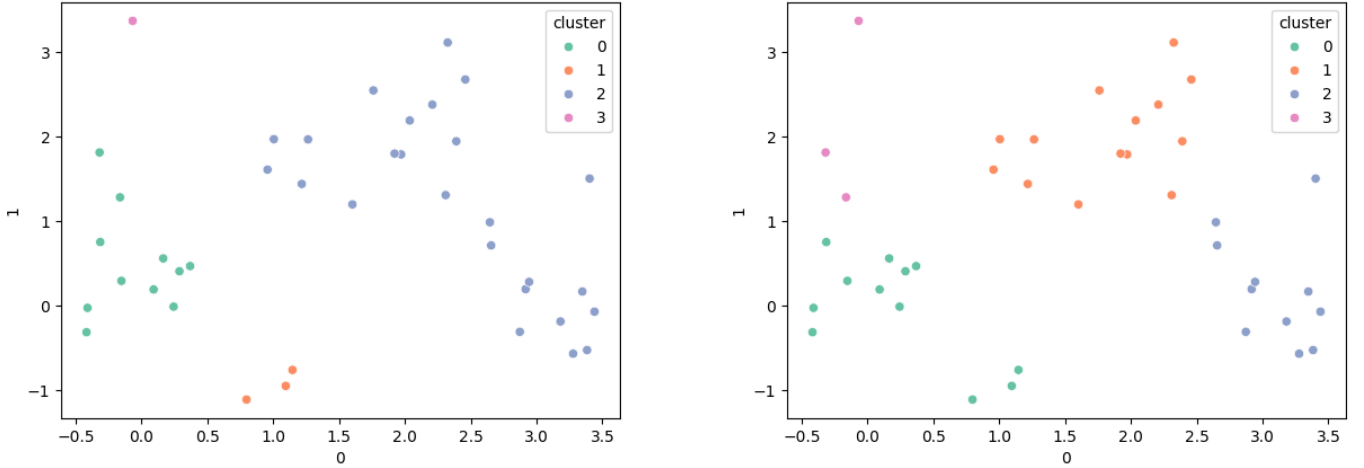


(a) Problem 3 data with K=4 and using single linkage to cluster (b) Problem 3 data with K=4 and using complete linkage to cluster

Figure 4: Hierarchical Agglomerative Cluster dendrogram plots for problems 3a and 3b

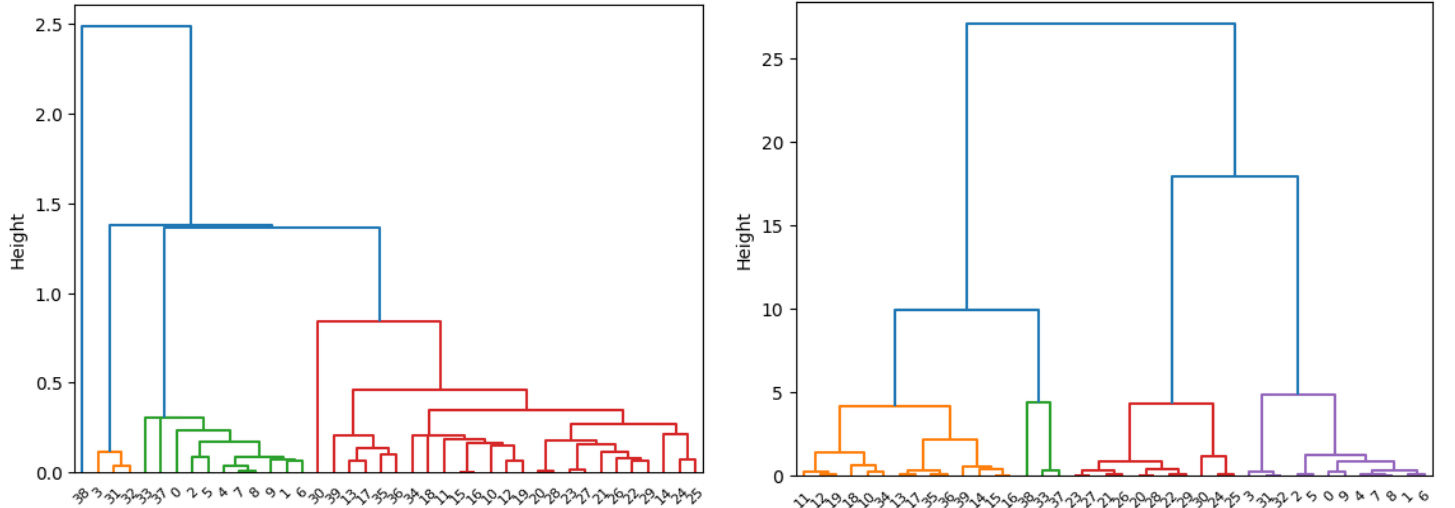
3c

The hierarchical agglomerative clustering using both single linkage and complete linkage with the squared distances, d_{ij}^2 are found in Figure 5. As expected, the dendrograms did not change because $h(\cdot)$ is a monotone increase function. Their dendrogram plots are found in Figure 6.



(a) Problem 3 data with K=4, squared distances, and (b) Problem 3 data with K=4, squared distances, and using single linkage to cluster using complete linkage to cluster

Figure 5: Hierarchical Agglomerative Cluster plots for problem 3c



(a) Problem 3 data with K=4, squared distances, and (b) Problem 3 data with K=4, squared distances, and using single linkage to cluster using complete linkage to cluster

Figure 6: Hierarchical Agglomerative Cluster dendrogram plots for problem 3c

Problem 3d

$$d_{\text{single}}(G, H) = \min_{i \in G, j \in H} d_{ij}$$

$$\text{let } d_{gh} = d_{\text{single}}(G, H)$$

then

$$d_{gh} \leq d_{i_1, j_1} \leq d_{i_1, j_2} \leq \dots \leq d_{i_n, j_n}$$

if $h(\cdot)$ is monotone increasing function
then if $x \leq x'$, $h(x) \leq h(x')$

\therefore

$$h(d_{gh}) \leq h(d_{i_1, j_1}) \leq h(d_{i_1, j_2}) \leq \dots \leq h(d_{i_n, j_n})$$

\therefore

points g, h still closest points

$$d_{\text{complete}}(G, H) = \max_{i \in G, j \in H} d_{ij}$$

$$\text{let } d_{gh} = d_{\text{complete}}(G, H)$$

$$d_{gh} \geq d_{i_1, j_1} \geq d_{i_1, j_2} \geq \dots \geq d_{i_n, j_n}$$

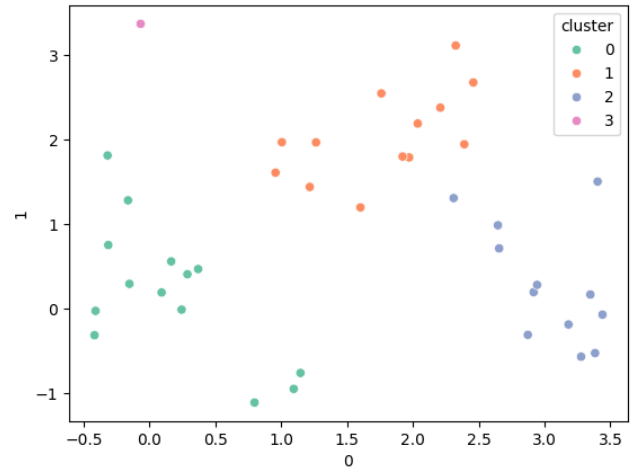
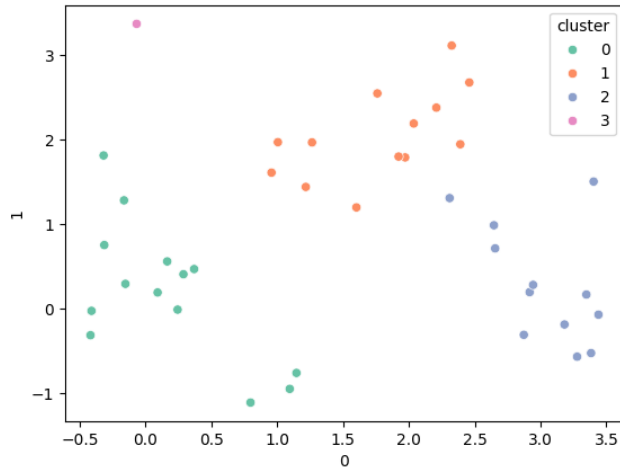
by same monotone increasing function rule

$$h(d_{gh}) \geq h(d_{i_1, j_1}) \geq \dots \geq h(d_{i_n, j_n})$$

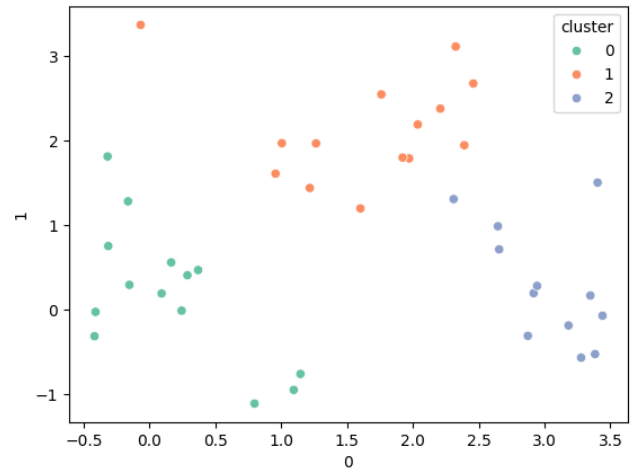
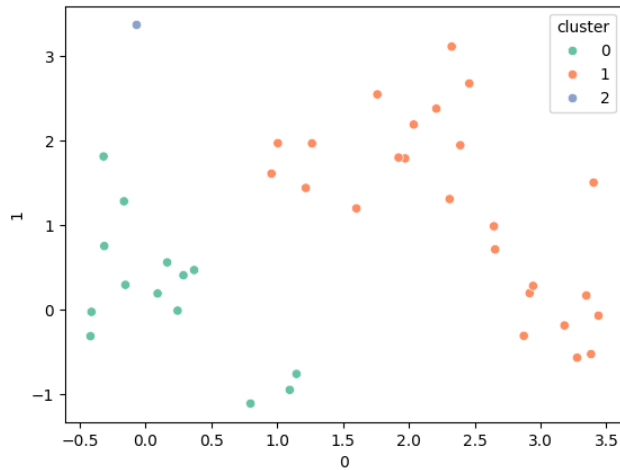
$\therefore d_{gh}$ is still the furthest points

3e

The hierarchical agglomerative clustering using average linkage with $K = 3, 4$ and both d_{ij} and d_{ij}^2 was conducted. When using $K = 4$, the clusters did not change between using d_{ij} and d_{ij}^2 , reference Figures 7a and 7b; however, it did change when using $K = 3$, reference Figures 7c and 7d. The dendrogram plots are found in Figure 8.

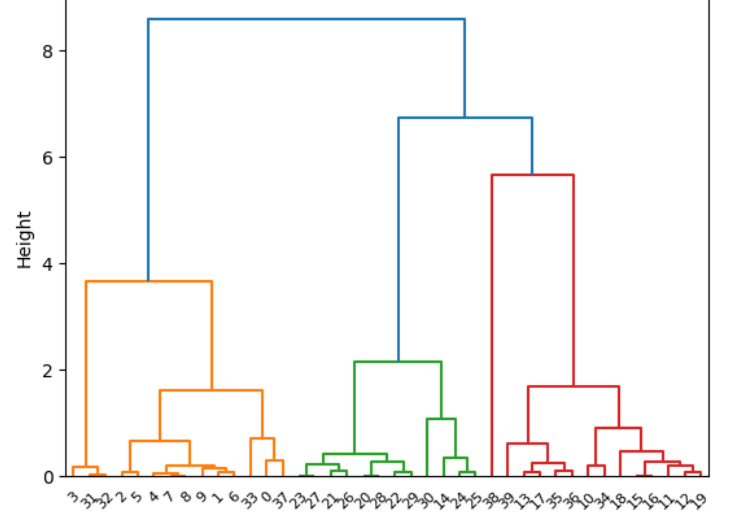
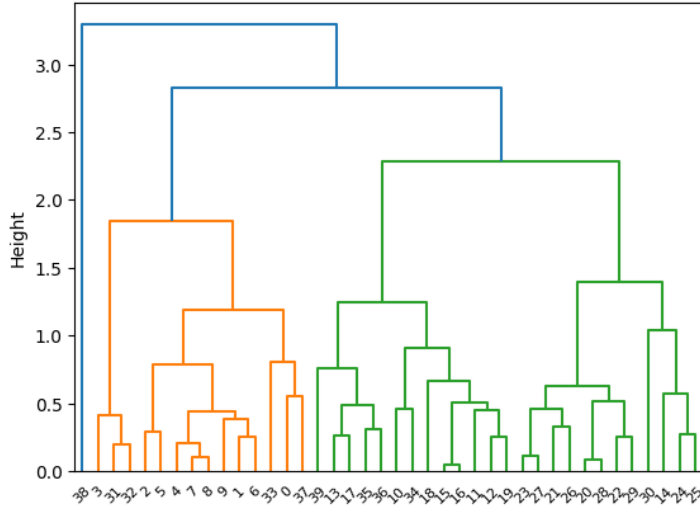


(a) Problem 3 data with $K=4$, non-squared squared distances, and using average linkage to cluster (b) Problem 3 data with $K=4$, squared distances, and using average linkage to cluster



(c) Problem 3 data with $K=3$, non-squared distances, and using average linkage to cluster (d) Problem 3 data with $K=3$, squared distances, and using average linkage to cluster

Figure 7: Hierarchical Agglomerative Cluster plots for problem 3e



(a) Problem 3 data with $K=4$, squared distances, and using single linkage to cluster (b) Problem 3 data with $K=4$, squared distances, and using complete linkage to cluster

Figure 8: Hierarchical Agglomerative Cluster dendrogram plots for problem 3c

Appendix A

October 4, 2023

```
[ ]: import pyreadr
import pandas as pd
import numpy as np
from scipy.spatial.distance import pdist, squareform
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import (dendrogram, cut_tree, linkage)

[ ]: def plot_dendrogram(model,filename, **kwargs):
    # Create linkage matrix and then plot the dendrogram

    # create the counts of samples under each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1 # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack(
        [model.children_, model.distances_, counts]
    ).astype(float)

    # Plot the corresponding dendrogram
    dendrogram(linkage_matrix, **kwargs)
    plt.ylabel('Height')
    plt.xticks(rotation=45)
    plt.savefig('plots/{}.png'.format(filename),bbox_inches='tight')

def compute_linkage(hclust):
    """
```

Create linkage matrix used to plot a dendrogram

Follows [sklearn example](https://scikit-learn.org/stable/auto_examples/cluster/plot_agglomerative_dendrogram.html)

Parameters

hclust : `sklearn.cluster.AgglomerativeClustering`
Fitted hierarchical clustering object.

Returns

linkage_matrix : np.ndarray
Array to be passed to `dendrogram` from `scipy.cluster.hierarchy`.

"""

```
counts = np.zeros(hclust.children_.shape[0])
n_samples = len(hclust.labels_)
for i, merge in enumerate(hclust.children_):
    current_count = 0
    for child_idx in merge:
        if child_idx < n_samples:
            current_count += 1 # leaf node
        else:
            current_count += counts[child_idx - n_samples]
    counts[i] = current_count

linkage_matrix = np.column_stack([hclust.children_, hclust.distances_,
                                  counts]).astype(float)

return linkage_matrix
```

1 Problem 1

```
[ ]: problem_1_data = pyreadr.read_r('data/hw1prob1.Rdata')
```

```
[ ]: for value in problem_1_data.values():
      problem_1_df = pd.DataFrame(value)
```

```
[ ]: problem_1_df.columns
```

```
[ ]: Index(['abandoned', 'abbeville', 'abbey', 'abbracciava', 'abilities',
          'ability', 'able', 'abode', 'about', 'above',
          ...
```

```

    ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
    ' ', ' ', ' ', ' ', ' '],
    dtype='object', length=6820)

```

```
[ ]: problem_1_df
```

```
[ ]:
      abandoned  abbeville  abbey  abbracciava  abilities  ability  able  \
tmnt leo        1.0        0.0    0.0          0.0        2.0      0.0  0.0
tmnt raph        0.0        0.0    1.0          0.0        1.0      0.0  0.0
tmnt mike        0.0        0.0    0.0          0.0        2.0      1.0  1.0
tmnt don         0.0        0.0    0.0          0.0        1.0      0.0  0.0
real leo         1.0        0.0    0.0          0.0        0.0      4.0  1.0
real raph        0.0        1.0    0.0          0.0        0.0      0.0  1.0
real mike        0.0        0.0    0.0          1.0        0.0      0.0  1.0
real don         0.0        0.0    0.0          0.0        0.0      0.0  0.0

```

```

      abode  about  above  ...
tmnt leo    0.0    5.0    1.0  ...  0.0    0.0    0.0    0.0    0.0  0.0
tmnt raph    0.0    3.0    0.0  ...  0.0    0.0    0.0    0.0    0.0  0.0
tmnt mike    1.0    4.0    0.0  ...  0.0    0.0    0.0    0.0    0.0  0.0
tmnt don     0.0    3.0    0.0  ...  0.0    0.0    0.0    0.0    0.0  0.0
real leo     0.0   15.0    2.0  ...  1.0    1.0    1.0    1.0    1.0  1.0
real raph    0.0   12.0    1.0  ...  1.0    0.0    0.0    1.0    0.0  1.0
real mike    0.0    3.0    1.0  ...  1.0    1.0    1.0    1.0    1.0  1.0
real don     0.0    4.0    0.0  ...  0.0    0.0    0.0    1.0    0.0  1.0

```

```

tmnt leo      0.0          0.0  1.0  0.0
tmnt raph      0.0          0.0  0.0  0.0
tmnt mike      0.0          0.0  1.0  0.0
tmnt don       0.0          0.0  0.0  0.0
real leo       1.0          1.0  1.0  1.0
real raph      1.0          0.0  1.0  1.0
real mike      1.0          1.0  1.0  1.0
real don       0.0          0.0  1.0  1.0

```

[8 rows x 6820 columns]

1.1 Problem 1 a

```
[ ]: word_counts = problem_1_df.sum(axis=1)
```

```
[ ]: word_counts
```

```
[ ]: tmnt leo      3114.0
      tmnt raph    1976.0
      tmnt mike    3330.0

```



```

tmnt don      2143.0
real leo      8962.0
real raph     6524.0
real mike     4618.0
real don      1766.0
dtype: float64

```

1.1.1 Determine the number of documents that contain each word

This will allow me to keep track of number of documents that contain the given word, by making it 1 if it contains the word, 0 otherwise

```
[ ]: documents_count_df = problem_1_df.applymap(lambda x: 1 if x != 0 and pd.
      ↪notna(x) else 0)
```

Sum each column to get n_w for each word

```
[ ]: n_w = documents_count_df.sum()
      n_w.head()
```

```
[ ]: abandoned      2
      abbeville      1
      abbey          1
      abbracciava    1
      abilities      4
      dtype: int64

```

```
[ ]: dtm1 = pd.DataFrame(columns=problem_1_df.columns)
      for k,v in word_counts.items():
          dtm1.loc[k] = problem_1_df.loc[k]/v
```

```
[ ]: D = 8
```

```
[ ]: dtm1 = dtm1 * np.log(D/dtm1.columns.map(n_w.to_dict()))
```

1.1.2 dtm2 - first scale by IDF and then normalize by word count

```
[ ]: dtm2 = pd.DataFrame(columns=problem_1_df.columns)
```

```
[ ]: dtm2 = problem_1_df * np.log(D/dtm2.columns.map(n_w.to_dict()))
```

```
[ ]: for k,v in word_counts.items():
      dtm2.loc[k] = dtm2.loc[k]/v
```

1.1.3 dtm1 vs dtm2

They are identical. This makes sense because you are only applying transformations in a different order. Both IDF and word count normalization depend on the original data and it does not matter which order you apply.

```
[ ]: dtm1
```

```
[ ]:      abandoned  abbeville    abbey  abbracciava  abilities  ability  \  
tmnt leo    0.000445    0.000000    0.000000    0.00000    0.000445    0.000000  
tmnt raph   0.000000    0.000000    0.001052    0.00000    0.000351    0.000000  
tmnt mike   0.000000    0.000000    0.000000    0.00000    0.000416    0.000416  
tmnt don    0.000000    0.000000    0.000000    0.00000    0.000323    0.000000  
real leo    0.000155    0.000000    0.000000    0.00000    0.000000    0.000619  
real raph   0.000000    0.000319    0.000000    0.00000    0.000000    0.000000  
real mike   0.000000    0.000000    0.000000    0.00045    0.000000    0.000000  
real don    0.000000    0.000000    0.000000    0.00000    0.000000    0.000000
```

```
      able    abode  about    above  ...  \  
tmnt leo    0.000000    0.000000    0.0  0.000223  ...  0.000000    0.000000  
tmnt raph   0.000000    0.000000    0.0  0.000000  ...  0.000000    0.000000  
tmnt mike   0.000208    0.000624    0.0  0.000000  ...  0.000000    0.000000  
tmnt don    0.000000    0.000000    0.0  0.000000  ...  0.000000    0.000000  
real leo    0.000077    0.000000    0.0  0.000155  ...  0.000109    0.000155  
real raph   0.000106    0.000000    0.0  0.000106  ...  0.000150    0.000000  
real mike   0.000150    0.000000    0.0  0.000150  ...  0.000212    0.000300  
real don    0.000000    0.000000    0.0  0.000000  ...  0.000000    0.000000
```

```
      \  
tmnt leo    0.000000    0.000000    0.000000    0.000000    0.000000    0.000000  
tmnt raph   0.000000    0.000000    0.000000    0.000000    0.000000    0.000000  
tmnt mike   0.000000    0.000000    0.000000    0.000000    0.000000    0.000000  
tmnt don    0.000000    0.000000    0.000000    0.000000    0.000000    0.000000  
real leo    0.000155    0.000077    0.000155    0.000077    0.000109    0.000155  
real raph   0.000000    0.000106    0.000000    0.000106    0.000150    0.000000  
real mike   0.000300    0.000150    0.000300    0.000150    0.000212    0.000300  
real don    0.000000    0.000392    0.000000    0.000392    0.000000    0.000000
```

```
tmnt leo    0.000092    0.000000  
tmnt raph   0.000000    0.000000  
tmnt mike   0.000086    0.000000  
tmnt don    0.000000    0.000000  
real leo    0.000032    0.000077  
real raph   0.000044    0.000106  
real mike   0.000062    0.000150  
real don    0.000163    0.000392
```

```
[8 rows x 6820 columns]
```

```
[ ]: dtm2
```

```
[ ]:      abandoned  abbeville      abbey  abbracciava  abilities  ability  \
tmnt leo    0.000445   0.000000   0.000000      0.00000   0.000445  0.000000
tmnt raph   0.000000   0.000000   0.001052      0.00000   0.000351  0.000000
tmnt mike   0.000000   0.000000   0.000000      0.00000   0.000416  0.000416
tmnt don    0.000000   0.000000   0.000000      0.00000   0.000323  0.000000
real leo    0.000155   0.000000   0.000000      0.00000   0.000000  0.000619
real raph   0.000000   0.000319   0.000000      0.00000   0.000000  0.000000
real mike   0.000000   0.000000   0.000000      0.00045   0.000000  0.000000
real don    0.000000   0.000000   0.000000      0.00000   0.000000  0.000000
```

```
      able      abode  about      above  ...      \
tmnt leo  0.000000  0.000000    0.0  0.000223  ...  0.000000  0.000000
tmnt raph 0.000000  0.000000    0.0  0.000000  ...  0.000000  0.000000
tmnt mike 0.000208  0.000624    0.0  0.000000  ...  0.000000  0.000000
tmnt don  0.000000  0.000000    0.0  0.000000  ...  0.000000  0.000000
real leo  0.000077  0.000000    0.0  0.000155  ...  0.000109  0.000155
real raph 0.000106  0.000000    0.0  0.000106  ...  0.000150  0.000000
real mike 0.000150  0.000000    0.0  0.000150  ...  0.000212  0.000300
real don  0.000000  0.000000    0.0  0.000000  ...  0.000000  0.000000
```

```
      \
tmnt leo  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
tmnt raph 0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
tmnt mike 0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
tmnt don  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
real leo  0.000155  0.000077  0.000155  0.000077  0.000109  0.000155
real raph 0.000000  0.000106  0.000000  0.000106  0.000150  0.000000
real mike 0.000300  0.000150  0.000300  0.000150  0.000212  0.000300
real don  0.000000  0.000392  0.000000  0.000392  0.000000  0.000000
```

```
tmnt leo  0.000092  0.000000
tmnt raph 0.000000  0.000000
tmnt mike 0.000086  0.000000
tmnt don  0.000000  0.000000
real leo  0.000032  0.000077
real raph 0.000044  0.000106
real mike 0.000062  0.000150
real don  0.000163  0.000392
```

[8 rows x 6820 columns]

1.2 Problem 1 b

```
[ ]: dtm3 = pd.DataFrame(columns=problem_1_df.columns)
      for k,v in word_counts.items():
          dtm3.loc[k] = problem_1_df.loc[k]/v
```

```
[ ]: tmnt_mike = np.array(dtm3.loc['tmnt mike'])
```

Iterate through the data frame and calculate the euclidean norm for each vector compared to tmnt_mike

The closest document is tmnt raph

```
[ ]: for row in dtm3.iterrows():
      euclidean_distance = np.linalg.norm(tmnt_mike - np.array(row[1]))
      print('{}: {}'.format(row[0], euclidean_distance))
```

```
tmnt leo: 0.04118467877954197
tmnt raph: 0.03782949128933655
tmnt mike: 0.0
tmnt don: 0.042006124349499935
real leo: 0.04887327167743683
real raph: 0.04487836641687292
real mike: 0.047500179860007326
real don: 0.06376091296934941
```

1.3 Problem 1c

Complete linkage does a better job when K=2 because it creates two groups, one of turtles and one of the artists.

```
[ ]: euclidean_distances = pdist(dtm3, metric='euclidean')
      distance_matrix = squareform(euclidean_distances)
```

```
[ ]: distance_df = pd.DataFrame(distance_matrix, index=dtm3.index, columns=dtm3.
      ↪index)
      distance_df
```

```
[ ]:
      tmnt leo  tmnt raph  tmnt mike  tmnt don  real leo  real raph  \
tmnt leo  0.000000  0.038599  0.041185  0.044445  0.042532  0.049604
tmnt raph  0.038599  0.000000  0.037829  0.042870  0.054986  0.046176
tmnt mike  0.041185  0.037829  0.000000  0.042006  0.048873  0.044878
tmnt don   0.044445  0.042870  0.042006  0.000000  0.053655  0.050400
real leo   0.042532  0.054986  0.048873  0.053655  0.000000  0.039363
real raph  0.049604  0.046176  0.044878  0.050400  0.039363  0.000000
real mike  0.064754  0.066762  0.047500  0.061904  0.053363  0.050380
real don   0.067113  0.068849  0.063761  0.053722  0.055306  0.054376

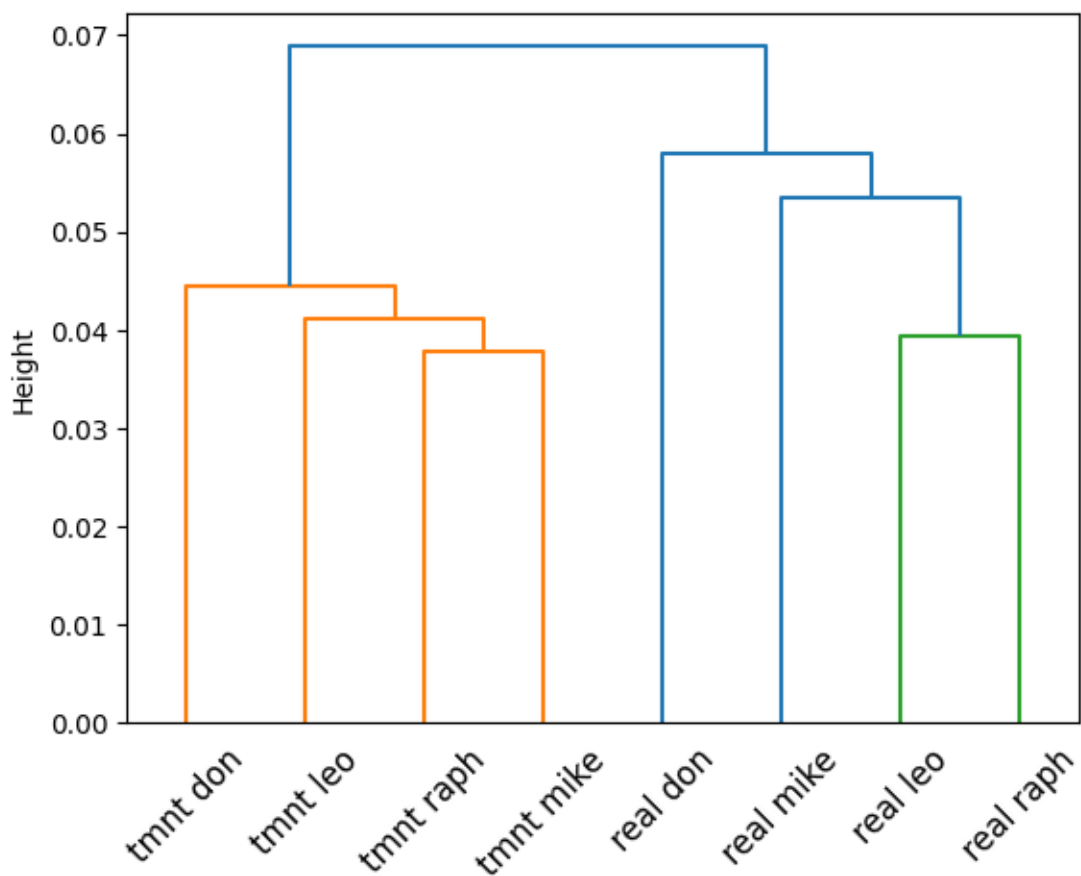
      real mike  real don
tmnt leo      0.064754  0.067113
```

tmnt raph	0.066762	0.068849
tmnt mike	0.047500	0.063761
tmnt don	0.061904	0.053722
real leo	0.053363	0.055306
real raph	0.050380	0.054376
real mike	0.000000	0.057994
real don	0.057994	0.000000

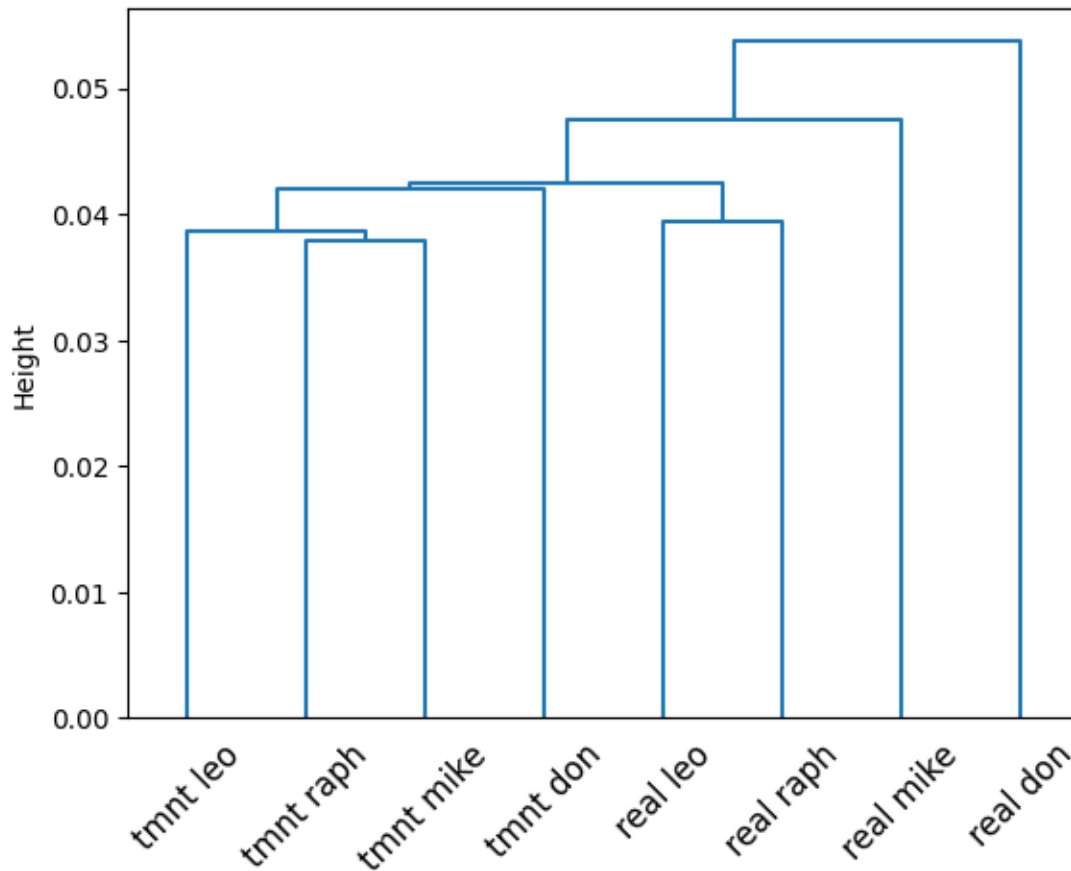
```
[ ]: HClust = AgglomerativeClustering
hc_comp = HClust(distance_threshold=0,n_clusters=None,linkage='complete')
hc_comp.fit(dtm3)
```

```
[ ]: AgglomerativeClustering(distance_threshold=0, linkage='complete',
                             n_clusters=None)
```

```
[ ]: plot_dendrogram(hc_comp, filename = 'tmnt_complete',labels = ["tmnt leo","tmnt_
raph","tmnt mike","tmnt don","real leo","real raph","real mike","real don"])
```



```
[ ]: hc_sing = HClust(distance_threshold=0,n_clusters=None,linkage='single')
hc_sing.fit(dtm3)
plot_dendrogram(hc_sing, filename = "tmnt_single",labels = ["tmnt leo","tmnt_
↪raph","tmnt mike","tmnt don","real leo","real raph","real mike","real don"])
```



1.4 Problem 1d

```
[ ]: word_counts = problem_1_df.sum()
```

```
[ ]: first_20 = pd.DataFrame(word_counts.sort_values(ascending=False),_
↪columns=['count']).head(n=20)
first_20
```

```
[ ]:
count
the      2664.0
and      1127.0
his       636.0
was       453.0
leonardo  342.0
```


that	297.0
for	282.0
with	279.0
michelangelo	277.0
raphael	212.0
from	209.0
this	189.0
which	129.0
turtles	127.0
donatello	117.0
him	116.0
series	115.0
were	111.0
who	110.0
one	103.0

```
[ ]: first_20.sum()/word_counts.sum()
```

```
[ ]: count    0.243425
      dtype: float64
```

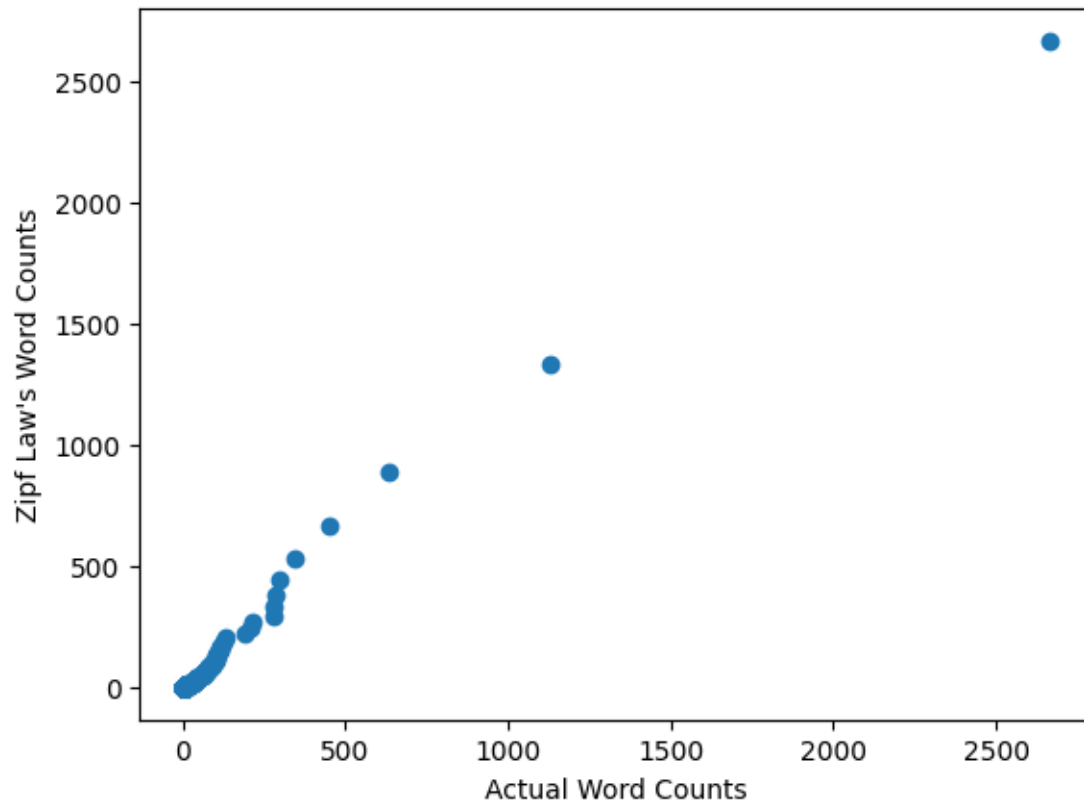
```
[ ]: running_sum = 0
      words = 0
      for count in word_counts.sort_values(ascending=False):
          words+=1
          running_sum+=count
          if running_sum/word_counts.sum()>=0.5:
              print(words)
              print(running_sum)
              break
```

```
254
16231.0
```

1.5 Problem 1e

```
[ ]: num_of_words = 6820
      largest_count = 2664.0
      zipf_law_values = np.array([largest_count/(i+1) for i in range(num_of_words)])
      actual_values = word_counts.sort_values(ascending=False).values
```

```
[ ]: plt.scatter(actual_values,zipf_law_values)
      plt.xlabel('Actual Word Counts')
      plt.ylabel('Zipf Law\'s Word Counts')
      plt.savefig('plots/zipf_law.png')
```



2 Problem 3

```
[ ]: problem_3_data = pyreadr.read_r('data/hw1prob3.Rdata')
```

```
[ ]: for k,v in problem_3_data.items():
    if k == 'x':
        x = pd.DataFrame(v)
    if k == 'd':
        distance = pd.DataFrame(v)
```

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

```
[ ]:
      0      1
0 -0.313227  0.755891
1  0.091822  0.194922
2 -0.417814 -0.310620
3  0.797640 -1.107350
4  0.164754  0.562465
```

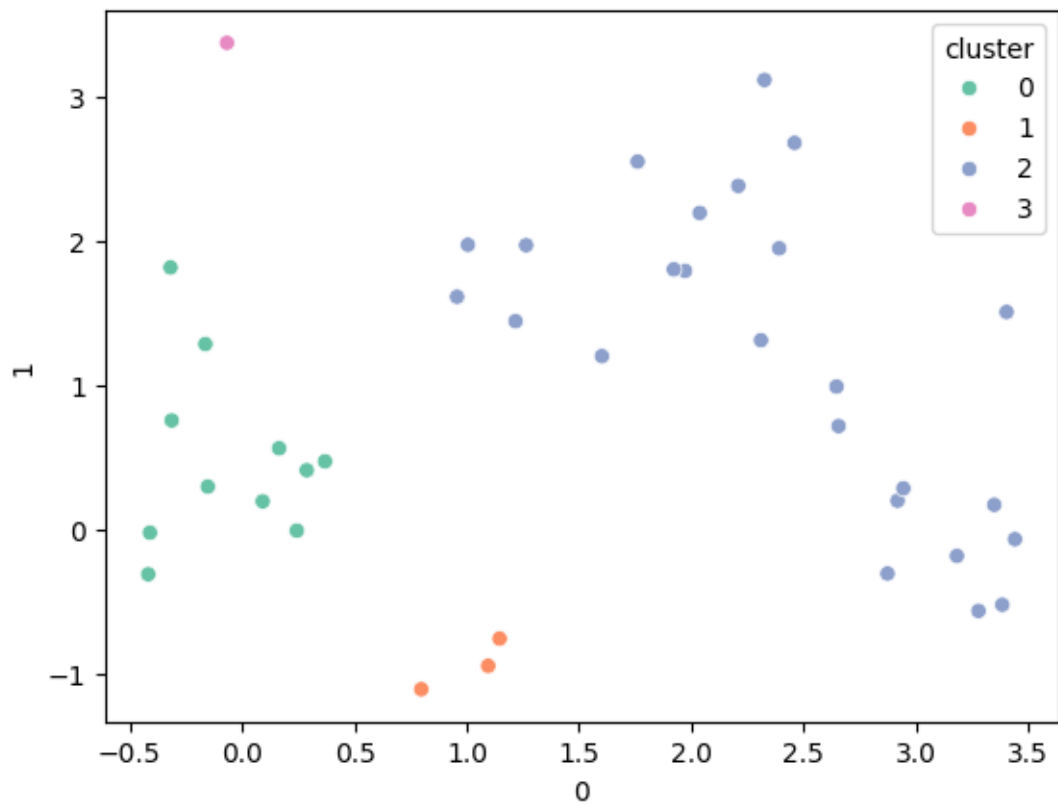
```
[ ]: distances = pd.read_csv('data/distances.csv')
distances = distances.drop(columns='Unnamed: 0', axis=1)
```

```
[ ]: def h_cluster(data: pd.DataFrame,
                  linkage: str,
                  num_clusters: int,
                  metric: str = 'euclidean',
                  distances: pd.DataFrame = None,
                  figure_name: str = 'plot') -> None:
    df = data.copy()
    hclust = HClust(distance_threshold=0, n_clusters=None, metric=metric,
    ↪linkage=linkage)
    if isinstance(distances, pd.DataFrame):
        hclust.fit(distances)
    else:
        hclust.fit(df)
    linkage_matrix = compute_linkage(hclust)
    data_labels = cut_tree(linkage_matrix, num_clusters)
    df['cluster'] = data_labels
    sns.scatterplot(x=df[0], y=df[1], hue=df['cluster'], palette = sns.
    ↪color_palette("Set2")[0:num_clusters])
    plt.savefig('plots/{}.png'.format(figure_name))

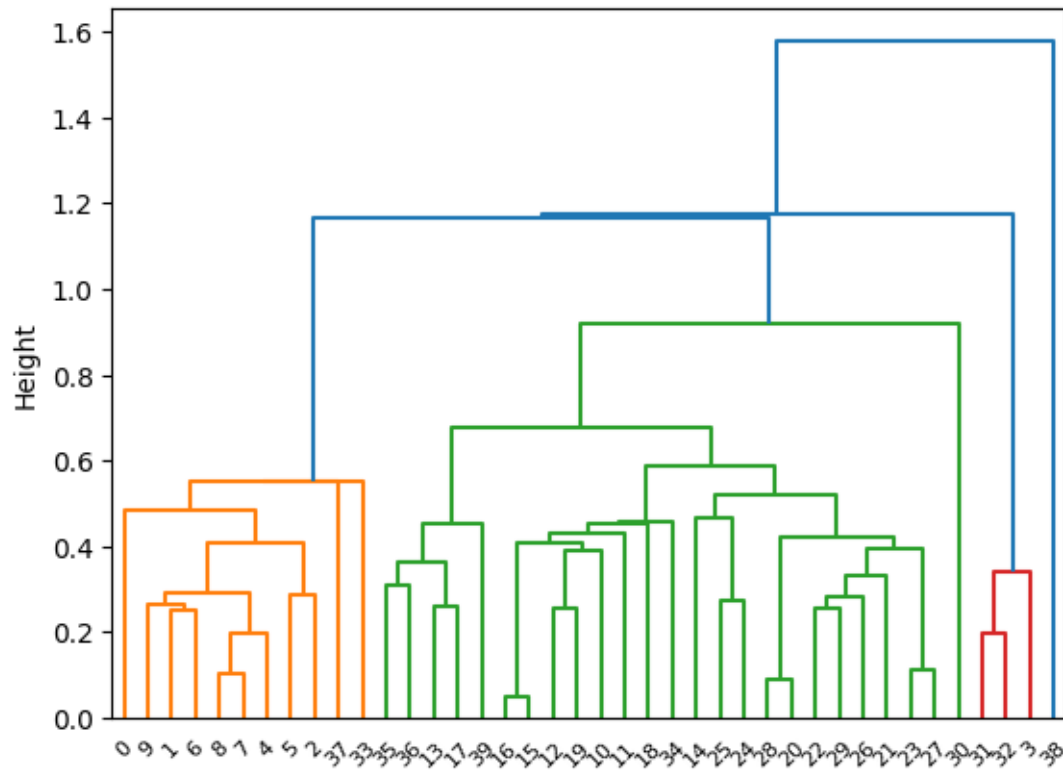
    return None
```

2.1 Problem 3a

```
[ ]: h_cluster(data = x, linkage='single', num_clusters=4,
    ↪figure_name='single_linkage')
```

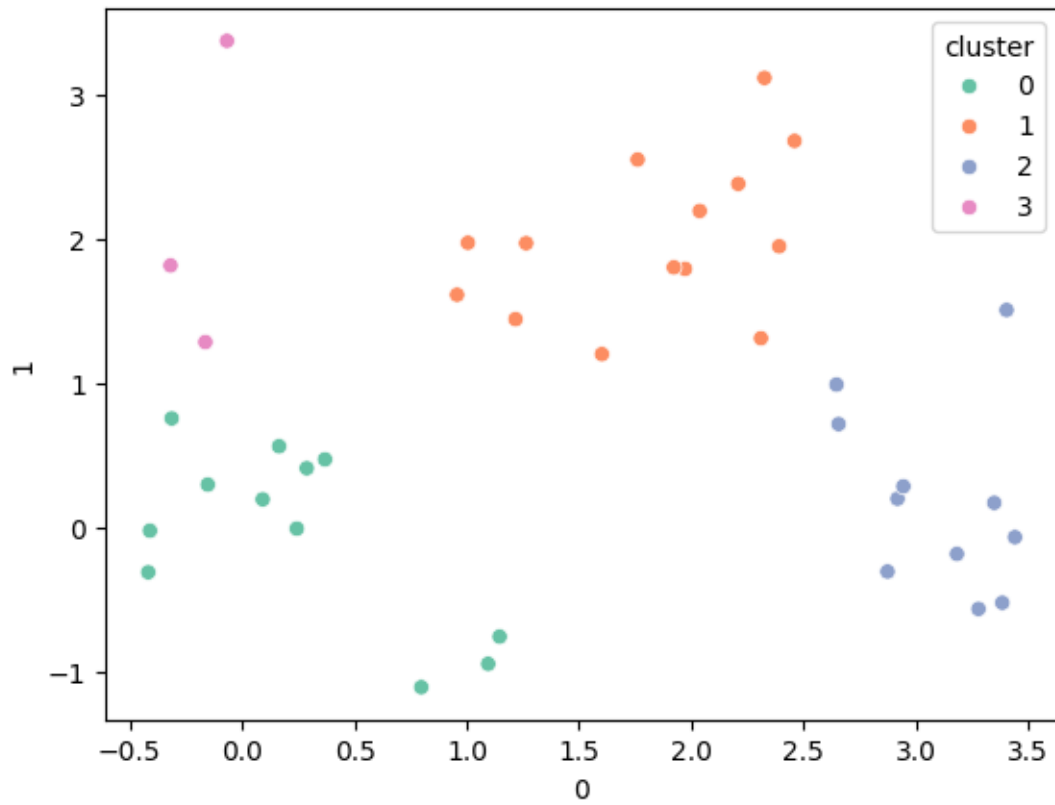


```
[ ]: hc = HClust(distance_threshold=0,n_clusters=None ,linkage='single')
      hc.fit(x)
      plot_dendrogram(hc, filename="3a_single_dendrogram")
```

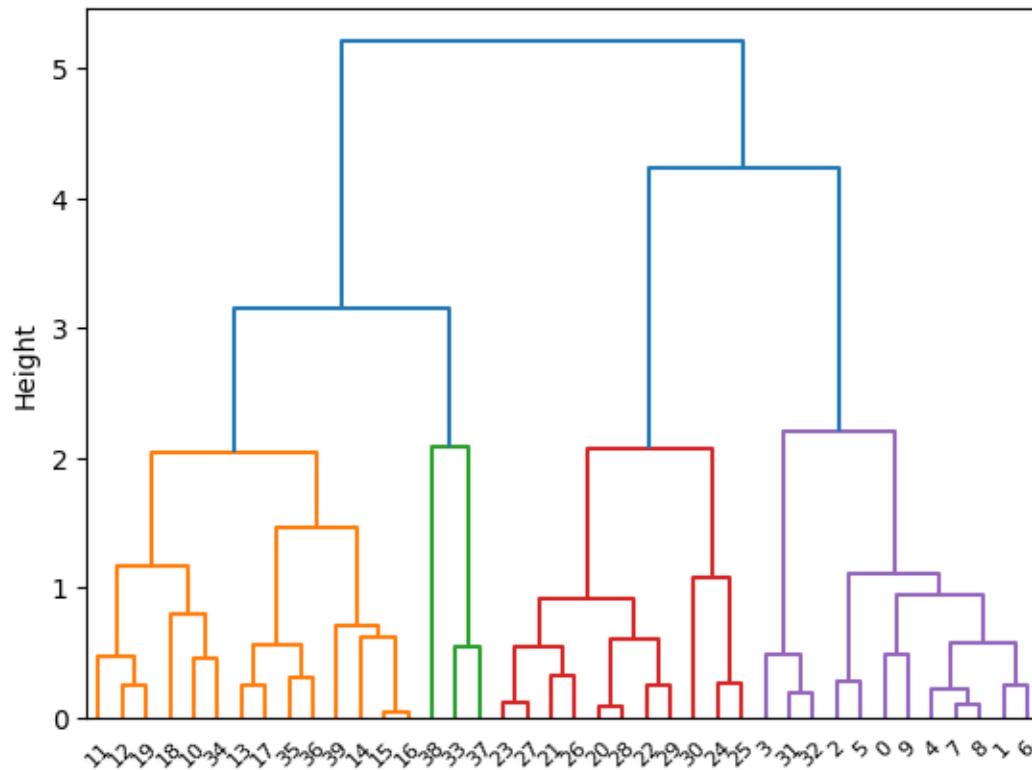


2.2 Problem 3b

```
[ ]: h_cluster(data = x, linkage='complete', num_clusters=4,
               figure_name='complete_linkage')
```

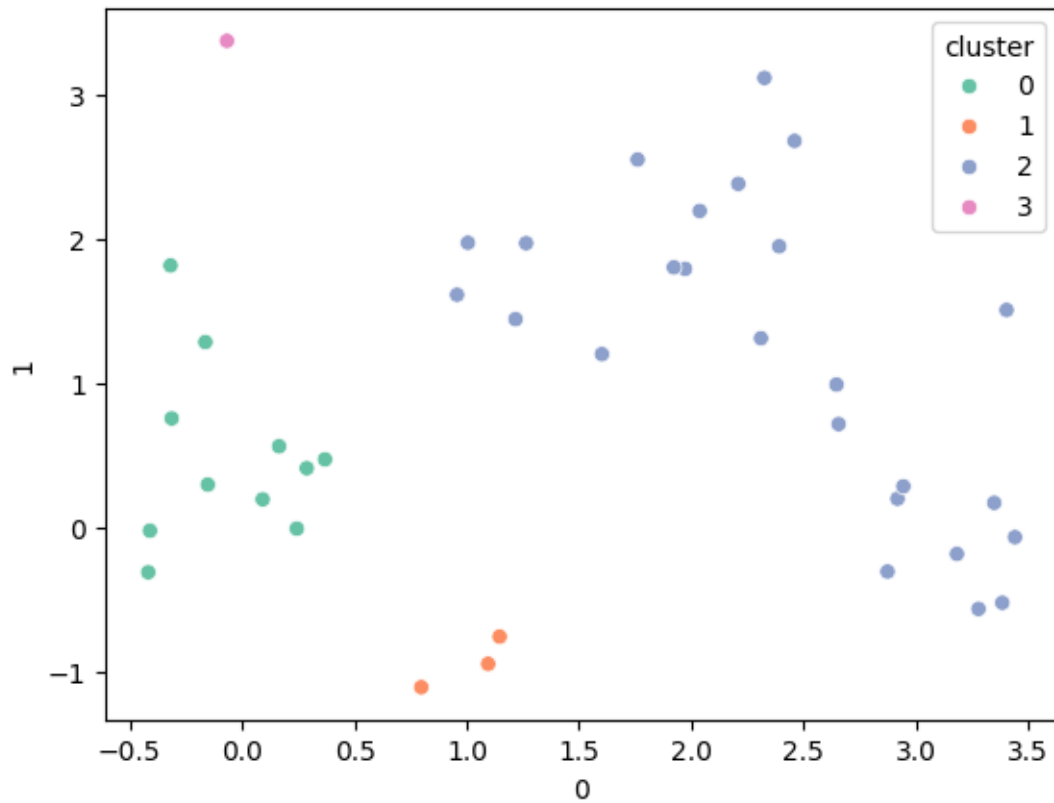


```
[ ]: hc = HClust(distance_threshold=0,n_clusters=None ,linkage='complete')
      hc.fit(x)
      plot_dendrogram(hc, filename="3a_complete_dendrogram", color_threshold = 2.5)
```

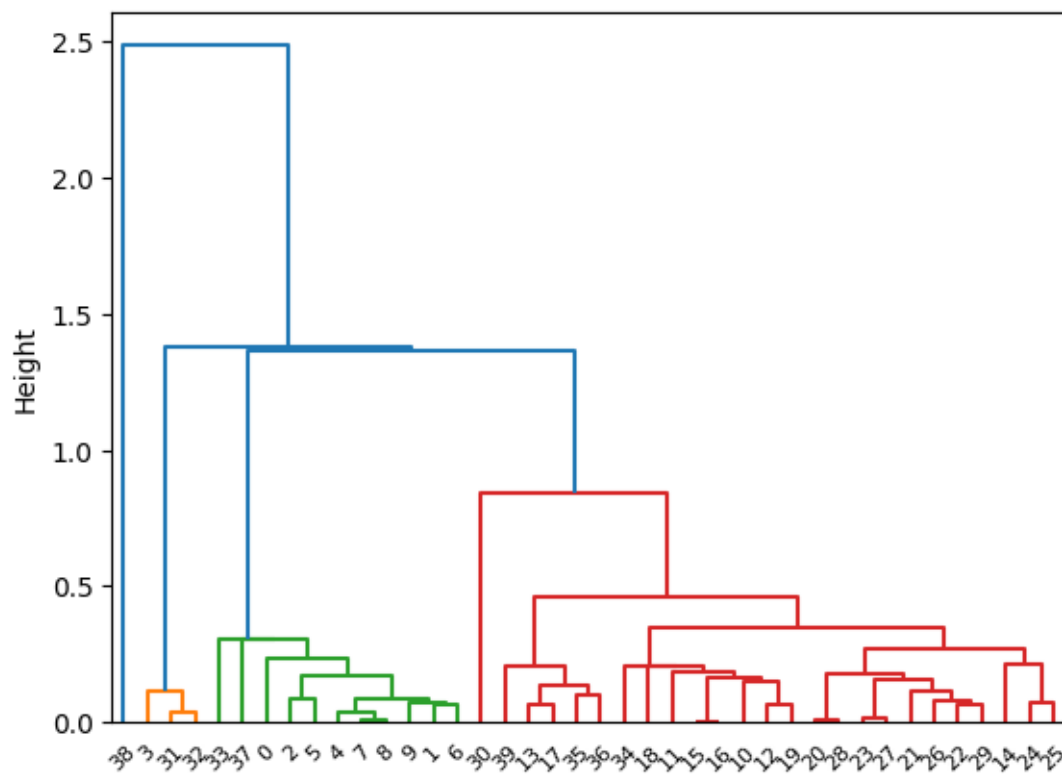



2.3 Problem 3c

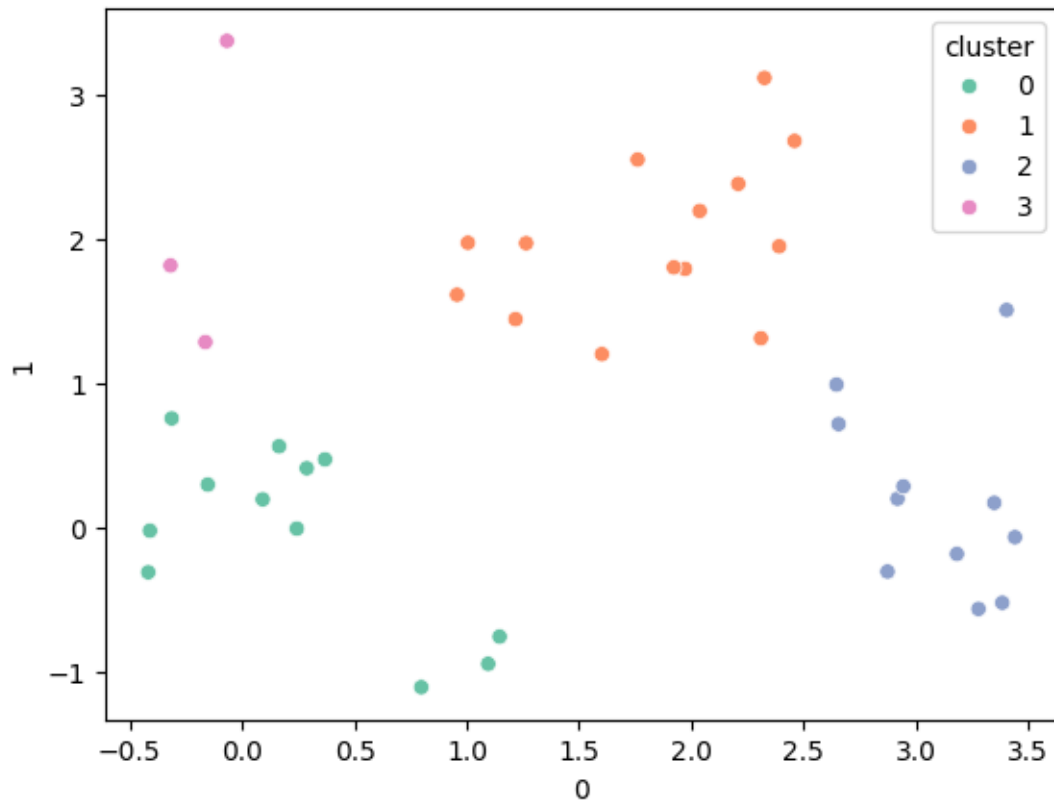
```
[ ]: h_cluster(data = x, metric='precomputed', distances=distances**2,
               linkage='single', num_clusters=4, figure_name='single_squared')
```



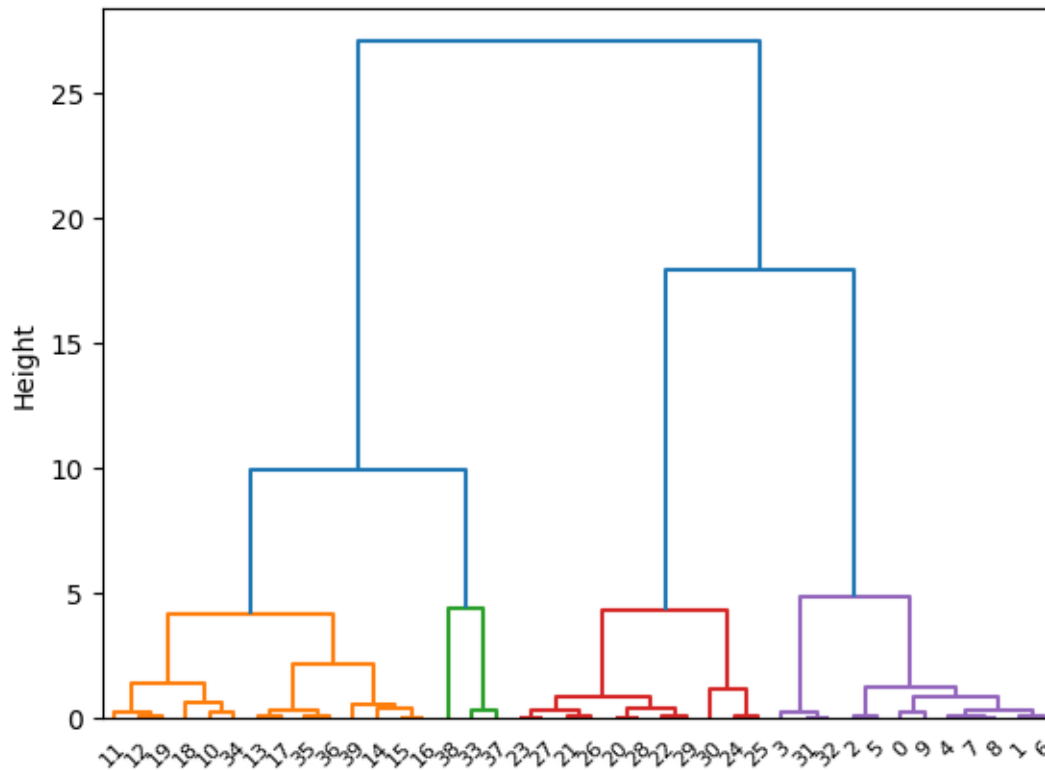
```
[ ]: hc = HClust(distance_threshold=0,metric='precomputed',n_clusters=None,
↳,linkage='single')
hc.fit(distances**2)
plot_dendrogram(hc, filename="3c_single_squared_dendrogram", color_threshold=1)
```



```
[ ]: h_cluster(data = x, metric='precomputed', distances=distances**2,
↳ linkage='complete', num_clusters=4, figure_name='complete_squared')
```

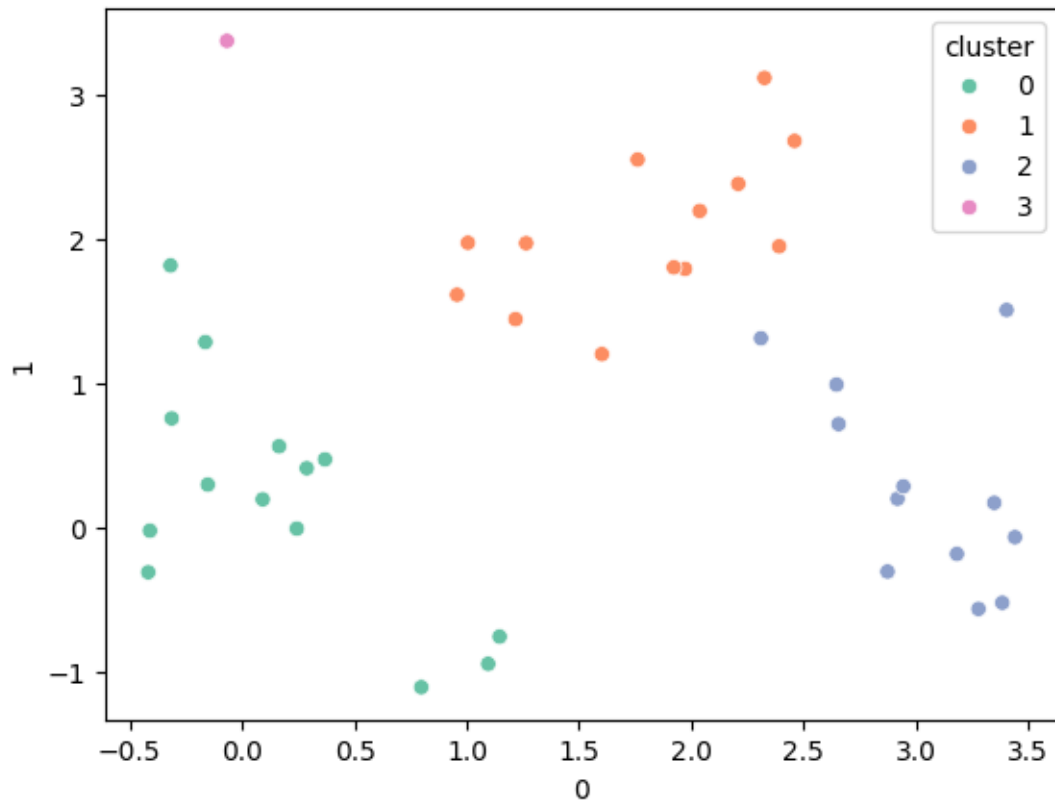


```
[ ]: hc = HClust(distance_threshold=0,metric='precomputed',n_clusters=None,
↳,linkage='complete')
hc.fit(distances**2)
plot_dendrogram(hc, filename="3c_complete_squared_dendrogram", color_threshold,
↳= 7.5)
```

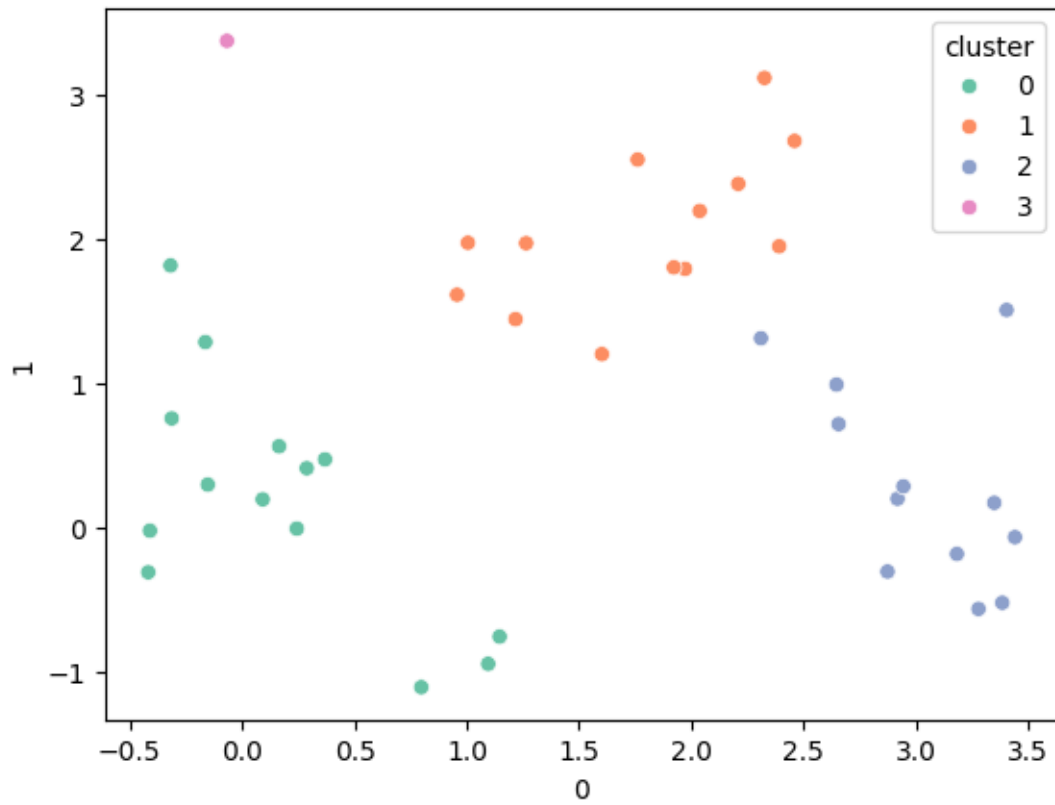


2.4 Problem 3e

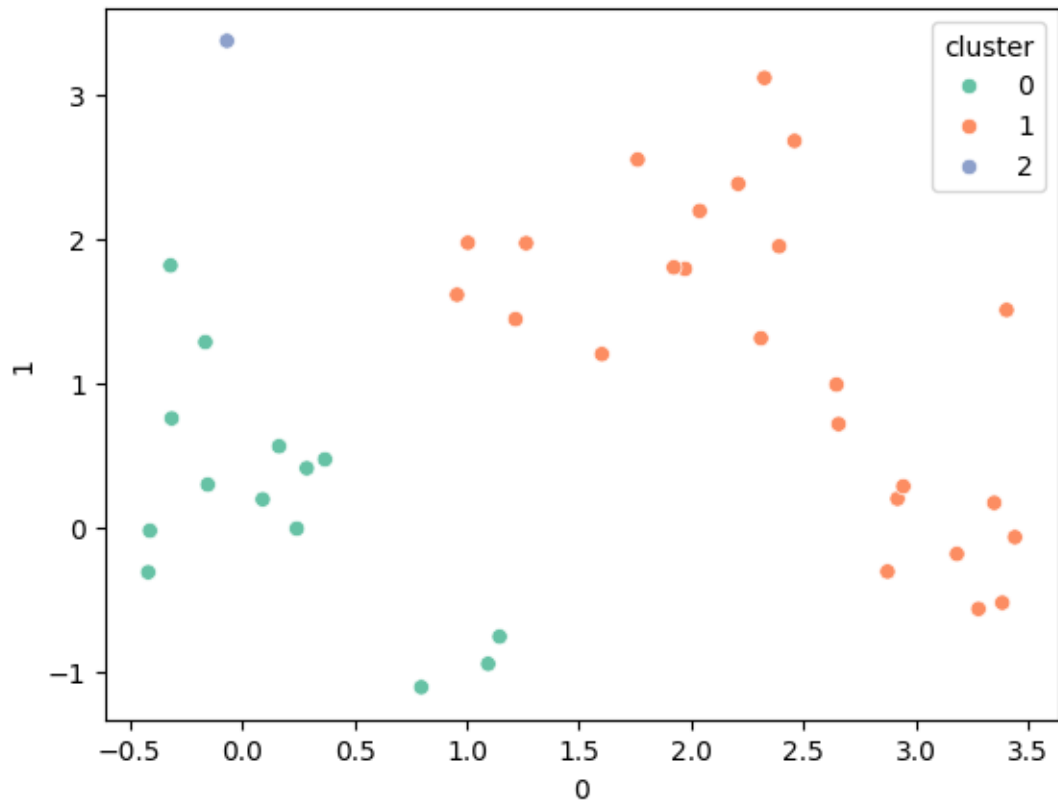
```
[ ]: h_cluster(data = x, linkage='average', num_clusters=4,
  ↳figure_name='average_linkage_4')
```



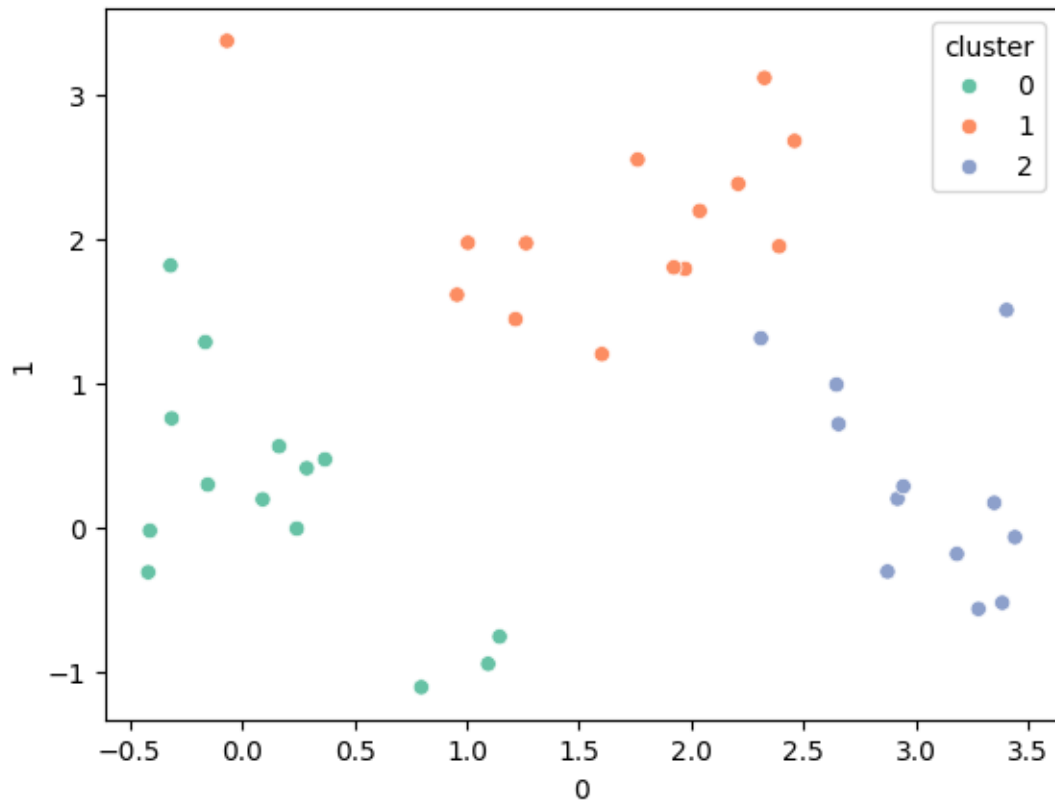
```
[ ]: h_cluster(data = x, metric='precomputed', distances=distances**2,
↳ linkage='average', num_clusters=4, figure_name='average_squared_4')
```

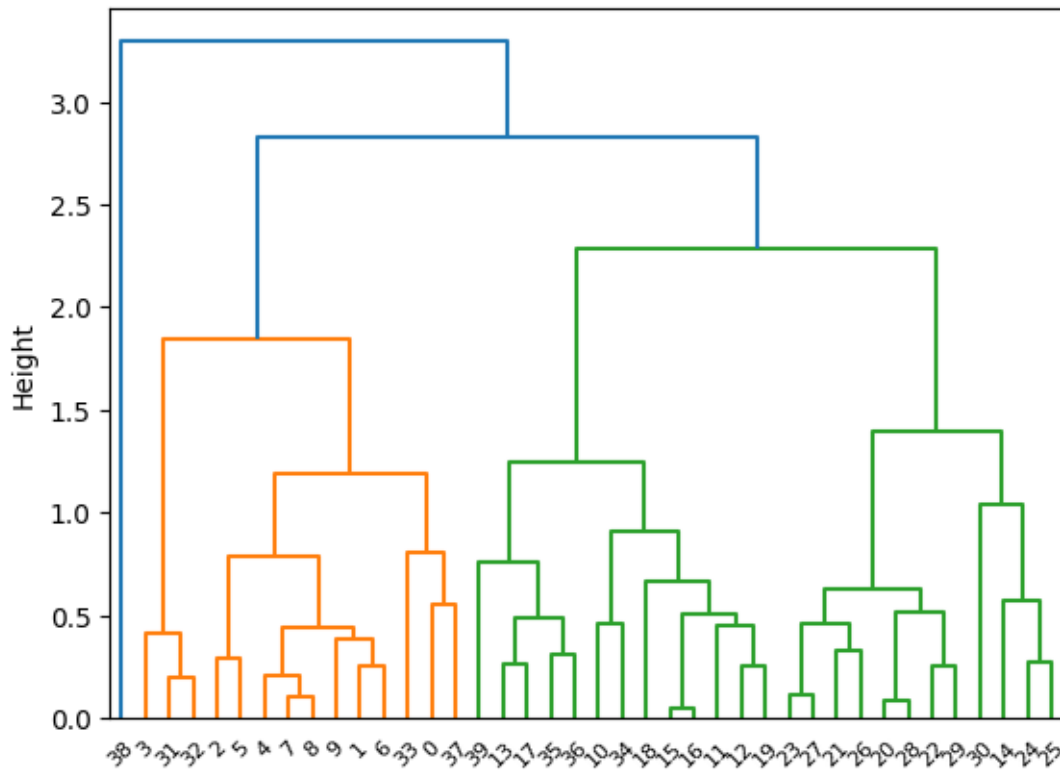
```
[ ]: h_cluster(data = x, linkage='average', num_clusters=3,
  ↳figure_name='average_linkage_3')
```



```
[ ]: h_cluster(data = x, metric='precomputed', distances=distances**2,
↳linkage='average', num_clusters=3, figure_name='average_squared_3')
```



```
[ ]: hc = HClust(distance_threshold=0,metric='precomputed',n_clusters=None,
↳,linkage='average')
hc.fit(distances)
plot_dendrogram(hc, filename="3e_average_dendrogram")
```



```
[ ]: hc = HClust(distance_threshold=0,metric='precomputed',n_clusters=None,
↪,linkage='average')
hc.fit(distances**2)
plot_dendrogram(hc, filename="3e_average_squared_dendrogram")
```

