MLPy Workshop 3

Student 1, Student 2

January 30, 2025

1 Week 3: Clustering

In this workshop, we will work through a set of problems clustering, another cannonical form of unsupervised learning. Clustering is an important tool that is used to discover homogeneous groups of data points within a heterogeneous population. It can be the main goal in some problems, while in others it may be used in EDA to understand the main types of behavior in the data or in feature engineering.

We will start by generating some artificial data, and then we will utilize clustering algorithms described in lectures and explore the impact of feature engineering on the solution. We will then attempt to find clusters in a gene expression dataset.

As usual, the worksheets will be completed in teams of 2-3, using **pair programming**, and we have provided cues to switch roles between driver and navigator. When completing worksheets:

- You will have tasks tagged by (CORE) and (EXTRA).
- Your primary aim is to complete the (CORE) components during the WS session, afterwards you can try to complete the (EXTRA) tasks for your self-learning process.
- Look for the as cue to switch roles between driver and navigator.

Instructions for submitting your workshops can be found at the end of worksheet. As a reminder, you must submit a pdf of your notebook on Learn by 16:00 PM on the Friday of the week the workshop was given.

As you work through the problems it will help to refer to your lecture notes (navigator). The exercises here are designed to reinforce the topics covered this week. Please discuss with the tutors if you get stuck, even early on!

1.1 Outline

1. Problem Definition and Setup: Simulated Example
2. K-means: Simulated Example
3. Hierarchical Clustering: Simulated Example
4. Gene Expression Data
5. Hierarchical Clustering: Gene Expression Data
6. K-means Clustering: Gene Expression Data

2 Problem Definition and Setup: Simulated Example

2.1 Packages

First, lets load in some packages to get us started.

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster import hierarchy
```

2.2 Data: Simulated Example

We will begin with a simple simulated example in which there are truly three clusters. We assume that there are $D = 2$ features and within each cluster, the data points are generated from a spherical normal distribution $N(\mathbf{m}_k, \sigma_k^2 \mathbf{I})$ for clusters $k = 1, 2, 3$, where both the mean $\mathbf{m}_k$ and variance σ_k^2 are different across clusters. Specifically, we assume that:

- Cluster 1: contains $|C_1| = 500$ points with mean vector $\mathbf{m}_1 = \begin{pmatrix} 0 \\ 4 \end{pmatrix}$ with standard deviation $\sigma_1 = 2$.
- Cluster 2: contains $|C_2| = 250$ points with mean vector $\mathbf{m}_2 = \begin{pmatrix} 0 \\ -4 \end{pmatrix}$ with standard deviation $\sigma_2 = 1$.
- Cluster 3: contains $|C_3| = 100$ points with mean vector $\mathbf{m}_3 = \begin{pmatrix} -4 \\ 0 \end{pmatrix}$ with standard deviation $\sigma_3 = 0.5$.

Run the following code to generate the dataset described above.

[2]:
```
# Number of features
D = 2

# Cluster sizes
N_1 = 500
N_2 = 250
N_3 = 100

# Cluster means
m_1 = np.array([0., 4.])
m_2 = np.array([0., -4.])
m_3 = np.array([-4., 0.])

# Cluster standard deviations
sd_1 = 2.
sd_2 = 1.
```

```
sd_3 = 0.5

# Generate the data
rnd = np.random.RandomState(5)
X_1 = rnd.normal(loc = m_1, scale = sd_1, size = (N_1,D))
X_2 = rnd.normal(loc = m_2, scale = sd_2, size = (N_2,D))
X_3 = rnd.normal(loc = m_3, scale = sd_3, size = (N_3,D))
X = np.vstack((X_1, X_2, X_3))

# Save true cluster labels
cl = np.hstack((np.repeat(1,N_1),np.repeat(2,N_2),np.repeat(3,N_3)))
```

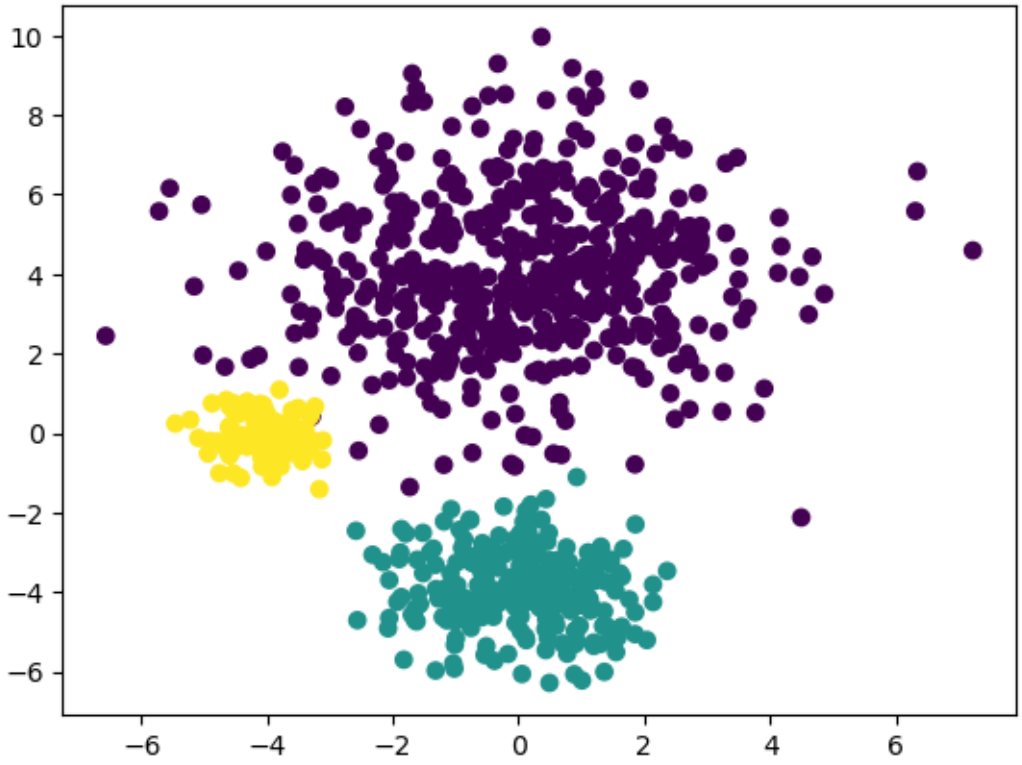
[3]:
```
# Check that the size is correct
print(X.shape)
```

```
(850, 2)
```

2.2.1 Exercise 1 (CORE)

Visualise the data and color by the true cluster labels.

[4]:
```
# Plot the data and color by cluster membership
plt.scatter(X[:,0], X[:,1], c=cl)
plt.show()
```



3 K-means Clustering: Simulated Example

To perform K-means clustering, we will use `KMeans()` in `sklearn.cluster`. Documentation is available here, and for an overview of clustering methods available in `sklearn`, see link. There are different inputs we can specify when calling `KMeans()` such as:

- `n_clusters`: the number of clusters.
- `init`: which specifies the intialization of the centroids, e.g. can be set to `k-means++` for K-means++ initialization or `random` for random initialization.
- `n_init`: which specifies the number of times the algorithm is run with different random initializations
- `random_state`: this can bet set to a fixed number to make results reproducible.

We can then use the `.fit()` method of `KMeans` to run the K-means algorithm on our data.

After fitting, some of the relevant attributes of interest include:

- `labels_`: cluster assignments of the data points.
- `cluster_centers_`: mean corresponding to each cluster, stored in a matrix of size: number of clusters K times number features D.
- `inertia_`: the total within-cluster variation.

3.0.1 Exercise 2 (CORE)

Let's start by exploring how the clustering changes across the K-means iterations. To do, set:

- number of clusters to 3
- initialization to random
- number of times the algorithm is run to 1
- fix the random seed to a number of your choice (e.g. 0)

a) Now, fit the K-means algorithms with different values of the maximum number of iterations fixed to 1,2,3, and the default value of 300.

b) Plot the clustering solution for the four different cases and comment on how it changes.

c) How many iterations are needed for the convergence?

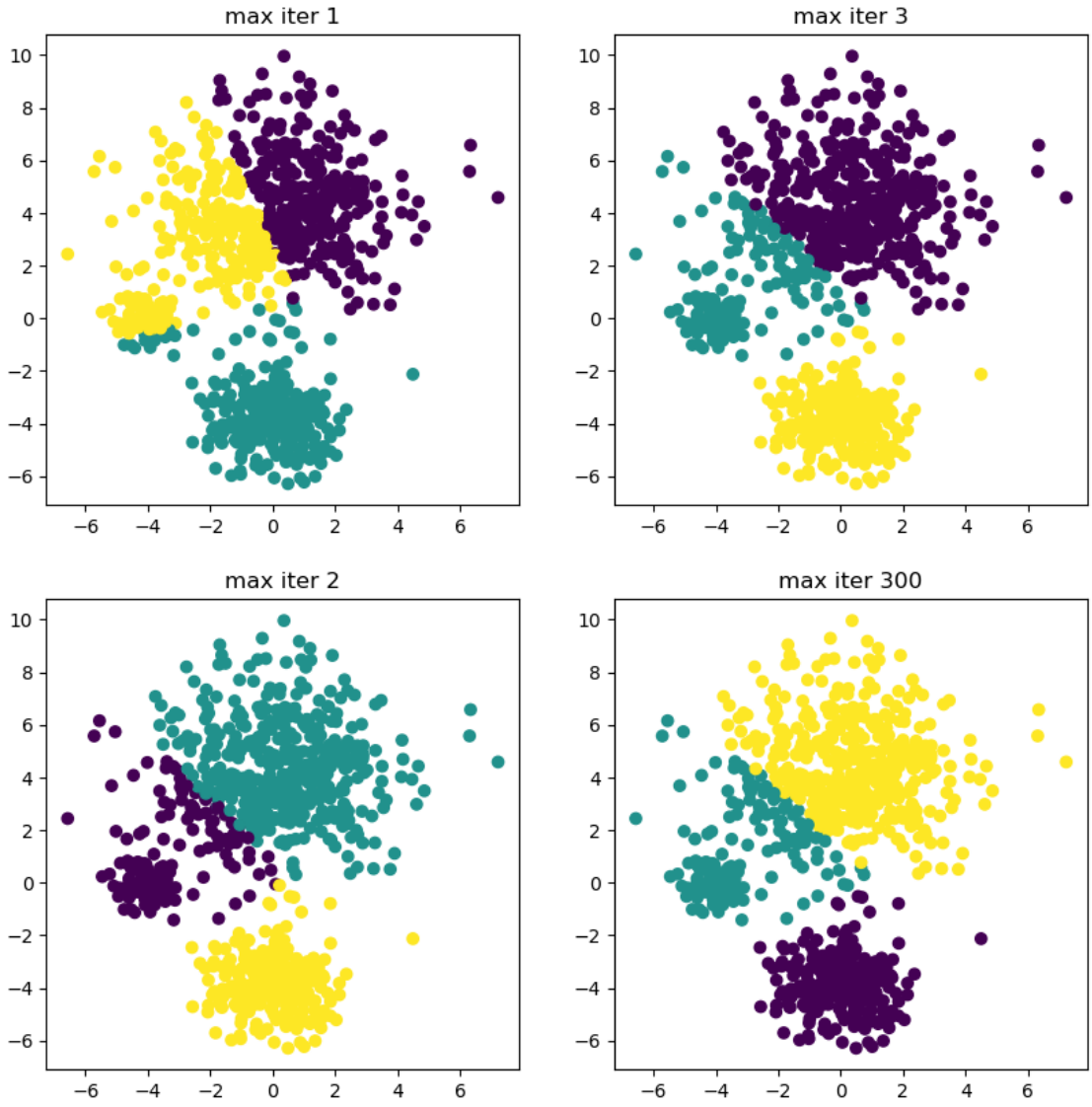
Hint

- To find the number of iterations, check the attributes of `KMeans`

[5]:
```
# Part a
num_clusters = 3

np.random.seed(0)

max_one = KMeans(n_clusters=num_clusters, max_iter=1).fit(X)
max_two = KMeans(n_clusters=num_clusters, max_iter=2).fit(X)
max_three = KMeans(n_clusters=num_clusters, max_iter=3).fit(X)
max_iter = KMeans(n_clusters=num_clusters).fit(X)
```

[6]:
```
# Part b

# four scatter plots in one figure

fig, axs = plt.subplots(2, 2, figsize=(10, 10))
axs[0, 0].scatter(X[:,0], X[:,1], c=max_one.labels_)
axs[0, 0].set_title('max iter 1')
axs[1, 0].scatter(X[:,0], X[:,1], c=max_two.labels_)
axs[1, 0].set_title('max iter 2')
axs[0, 1].scatter(X[:,0], X[:,1], c=max_three.labels_)
axs[0, 1].set_title('max iter 3')
axs[1, 1].scatter(X[:,0], X[:,1], c=max_iter.labels_)
axs[1, 1].set_title('max iter 300')
```

[6]:
```
Text(0.5, 1.0, 'max iter 300')
```



As the iterations progress from 1 to 2 to 3. We observe that the 3rd iteration has reached the max iteration of 300.

In the first iteration, the algorithm captures much more overlaping data when it shouldn't, by the 3rd iteration this has been largely corrected.

[7]:
```
# Part c
max_iter.n_iter_
```

[7]: 5

The algorithm needed 5 iterations to converge.

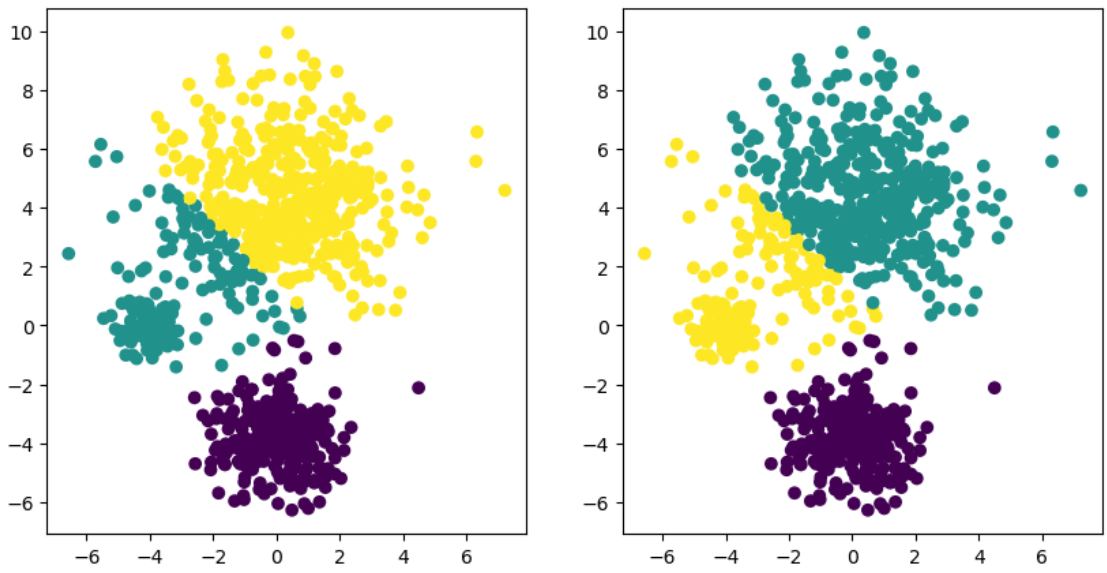
3.0.2 Exercise 3 (CORE)

Next, compare the random intialization with K-means++ (in this case fix the number of different initializations to 10). Plot both clustering solutions. Which requires fewer iterations? and which provides a lower within-cluster variation?

[8]:
```
plus_plus = KMeans(n_clusters=num_clusters, init='k-means++', n_init=10).fit(X)
```

[9]:
```
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].scatter(X[:,0], X[:,1], c=max_iter.labels_)
axs[1].scatter(X[:,0], X[:,1], c=plus_plus.labels_)
```

[9]: <matplotlib.collections.PathCollection at 0x15dfebb30>



[10]:
```
print(plus_plus.n_iter_, plus_plus.inertia_)
print(max_iter.n_iter_, max_iter.inertia_)
```

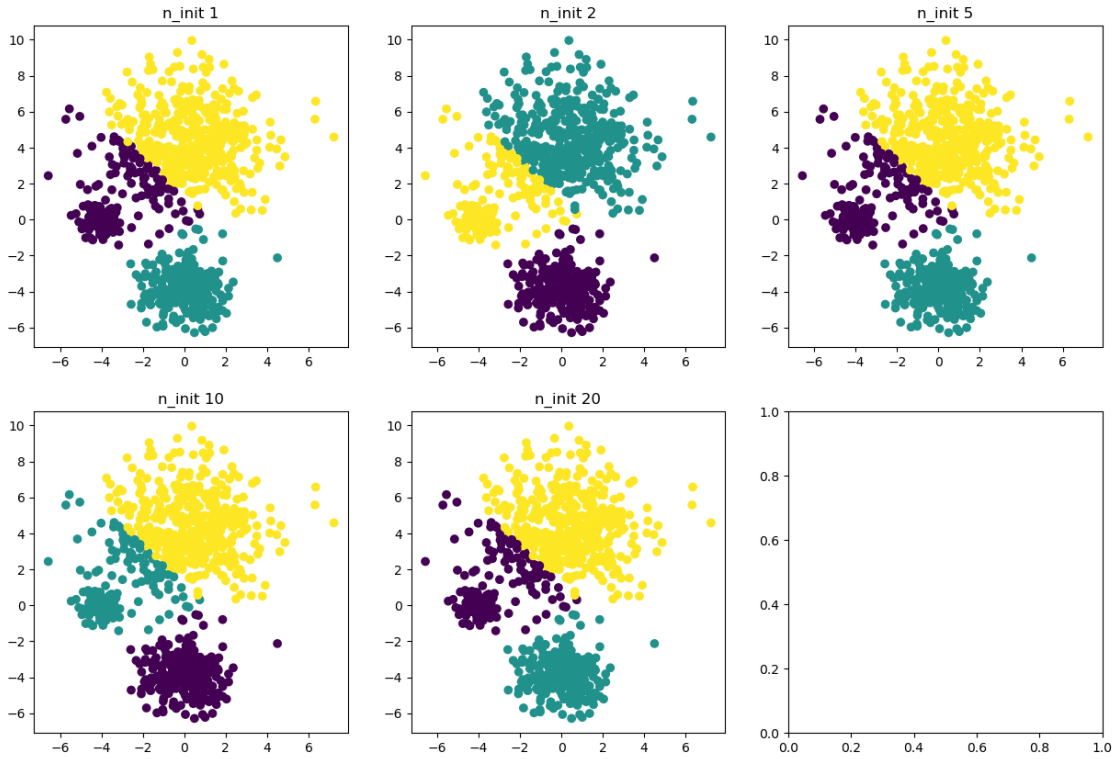
```
4 3833.282646884114
5 3833.367971404591
```

The K-Means++ converges in 4 iterations and has a total within cluster variation which is 0.1 less than the max_iter with random initialisation.

3.0.3 Exercise 4 (CORE)

Find the clustering solution using a different number of initializations equal to 1, 2, 5, 10, and 20. Visualize the results and print the within-cluster variation. Based on the results, how many intializations are needed? Try changing the random state; how does that change your conclusions?

[11]:
```
one_init   = KMeans(n_clusters=num_clusters, init='random', n_init=1,␣
 ↪random_state=110).fit(X)
two_init   = KMeans(n_clusters=num_clusters, init='random', n_init=2).fit(X)
five_init  = KMeans(n_clusters=num_clusters, init='random', n_init=5).fit(X)
ten_init   = KMeans(n_clusters=num_clusters, init='random', n_init=10).fit(X)
twenty_init  = KMeans(n_clusters=num_clusters, init='random', n_init=20).fit(X)

fig, axs = plt.subplots(2, 3, figsize=(15, 10))
axs[0, 0].scatter(X[:,0], X[:,1], c=one_init.labels_)
axs[0, 0].set_title('n_init 1')
axs[0, 1].scatter(X[:,0], X[:,1], c=two_init.labels_)
axs[0, 1].set_title('n_init 2')
axs[0, 2].scatter(X[:,0], X[:,1], c=five_init.labels_)
axs[0, 2].set_title('n_init 5')
axs[1, 0].scatter(X[:,0], X[:,1], c=ten_init.labels_)
axs[1, 0].set_title('n_init 10')
axs[1, 1].scatter(X[:,0], X[:,1], c=twenty_init.labels_)
axs[1, 1].set_title('n_init 20')
plt.show()
```



[12]:
```
print(one_init.inertia_)
print(two_init.inertia_)
print(five_init.inertia_)
print(ten_init.inertia_)
print(twenty_init.inertia_)
```

```
3833.367971404591
3833.3076162106936
3833.282646884114
3833.307616210693
3833.307616210693
```

With 5 initialisations, we observe a 0.06 reduction in within cluster variation.

Now, is a good point to switch driver and navigator

3.0.4 Exercise 5 (CORE)

Since we simulated the data, we know the true number of clusters. However, in practice this number is rarely known. Find the K-means solution with different choices of K and plot the within-cluster variation as a function of K. Comment on the results.

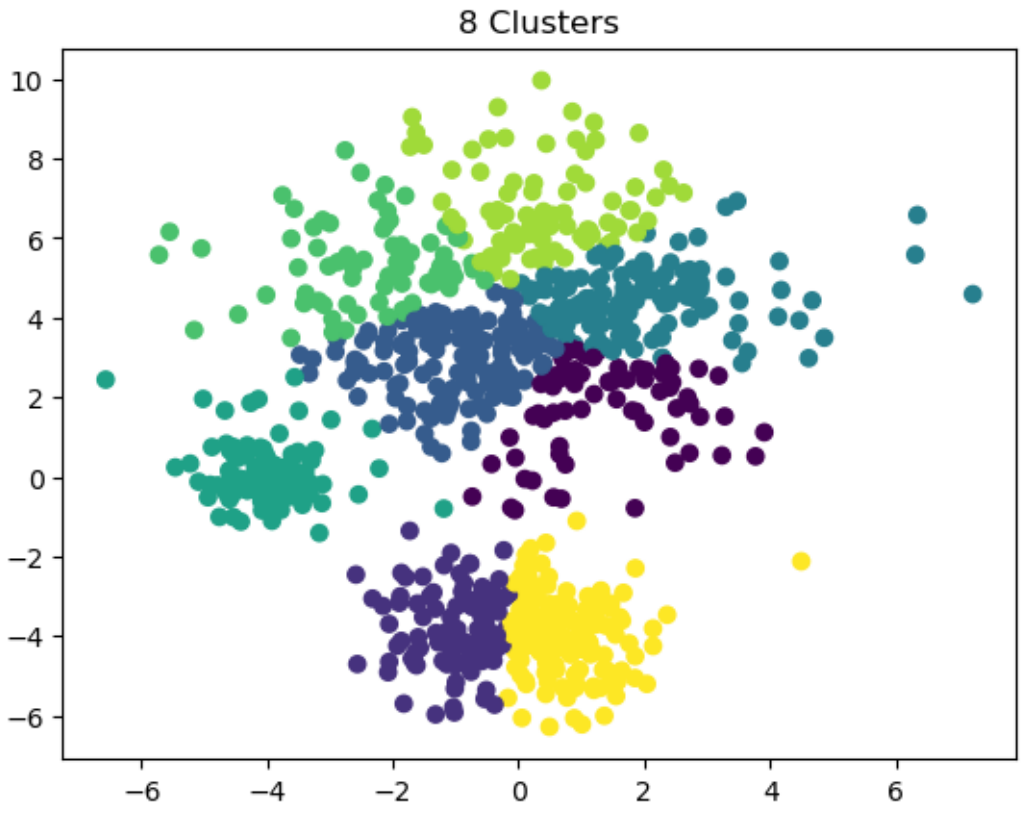
[13]:
```
for i in range(4, 10):
    num_clusters = i
```

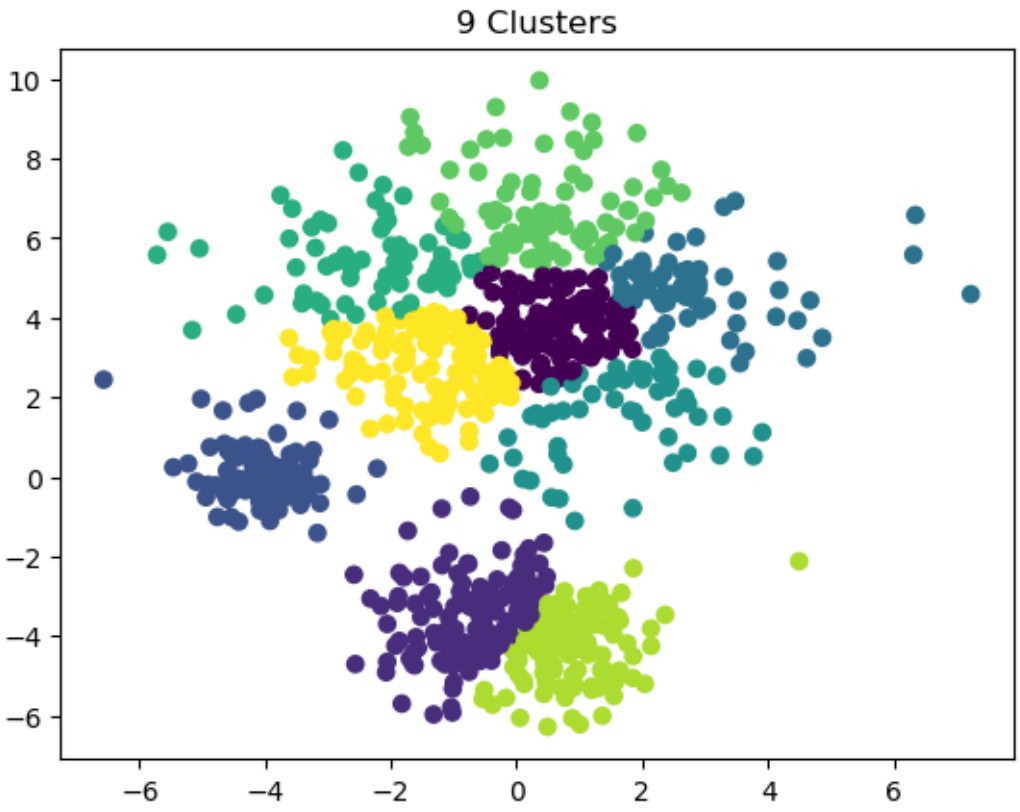
```
    ten_K = KMeans(n_clusters=num_clusters, n_init=10).fit(X)
    plt.scatter(X[:, 0], X[:, 1], c=ten_K.labels_)
    plt.title(f'{num_clusters} Clusters')
    plt.show()
```


5 Clusters
10
8
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6


6 Clusters
10
8
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6


7 Clusters
10
8
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6


8 Clusters
10
8
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6



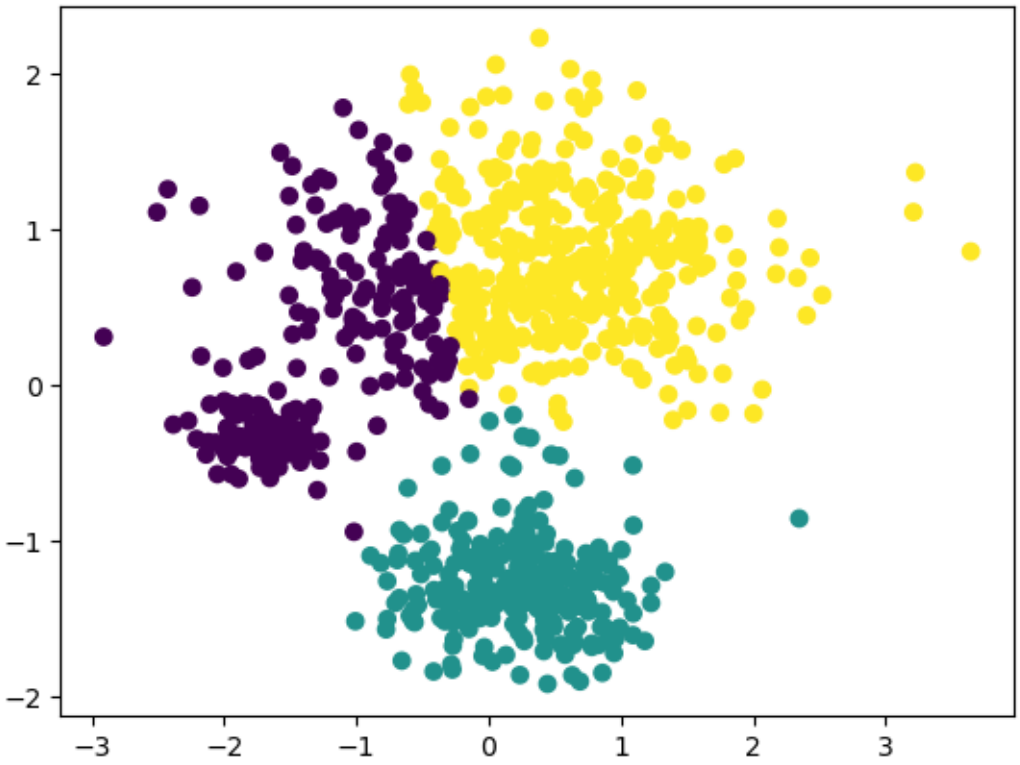
3.0.5 Exercise 6 (CORE)

Now standardize the data and re-run the K-means algorithm. Qualitatively, how has standardising the data impacted performance? Can you argue why you observe what you see?

[14]:
```
from sklearn.preprocessing import StandardScaler
```

[15]:
```
X_scaled = X.copy()

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X=X_scaled)

k_scaled = KMeans(n_clusters=3).fit(X_scaled)

plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=k_scaled.labels_)
```

[15]:
```
<matplotlib.collections.PathCollection at 0x15e5b24e0>
```



When compared to previous clusters, the purple cluster has more datapoints. This may be because the standardisation has decreased the dissimilarity leading to more cluster overlap.

4 Hierarchical Clustering: Simulated Example

To perform hierarchical clustering, we will use the `linkage()` function from `scipy.cluster.hierarchy`. The inputs to specify include

- the data.
- `metric`: specifies the dissimarlity between data points. Defaults to the Euclidean distance.
- `method`: specifies the type of linkage, e.g. complete, single, or average.

Then, we can use `dendrogram()` from `scipy.cluster.hierarchy` to plot the dendrogram.

Note that you can also use `AgglomerativeClustering` from `sklearn.cluster`, which similarly has options for `metric` to specify the distance and `linkage` to specify the type of linkage. However, `sklearn` does not have its own functions for plotting the dendogram and use must use the tools from `scipy.cluster.hierarchy`.

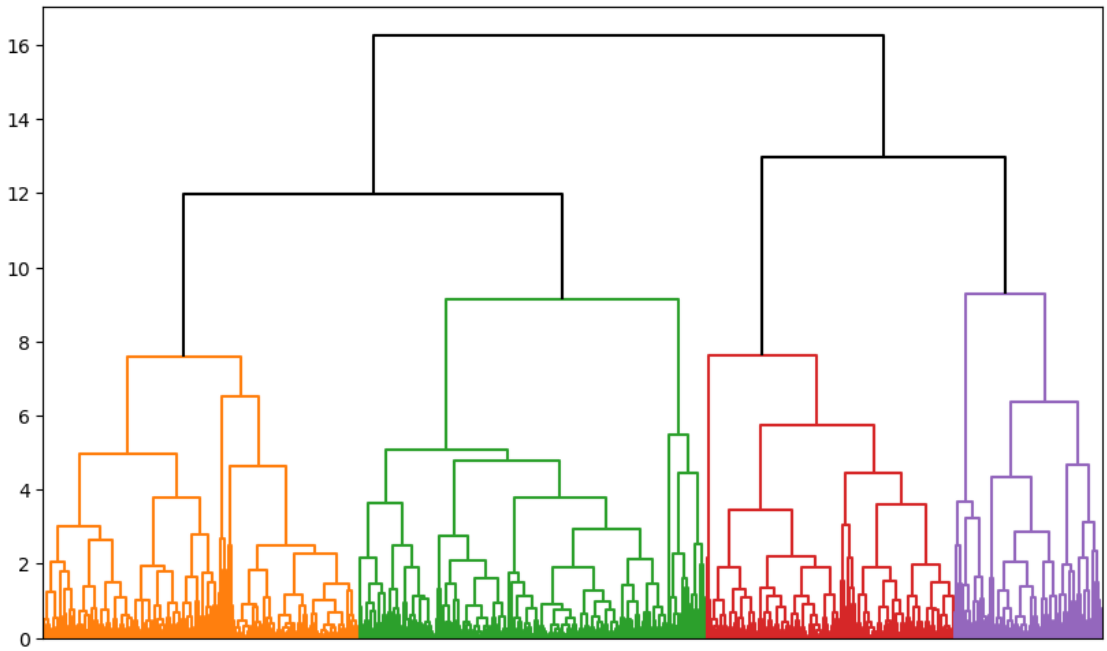
4.0.1 Exercise 7 (CORE)

a) Use hierarchical clustering with complete linkage and the Euclidean distance to cluster the simulated data. Name the object `hc_comp`.

[16]:
```
# The name of the returned object from hierarchy.linkage should be hc_comp for part b
hc_comp = hierarchy.linkage(X, method='complete')
```

b) Plot the dendogram by running the code below. Try changing the 'color_threshold' to a number (e.g. 11) to color the branches of the tree below the threshold with different colors, thus, identifying the clusters if the tree were to be cut at that threshold.

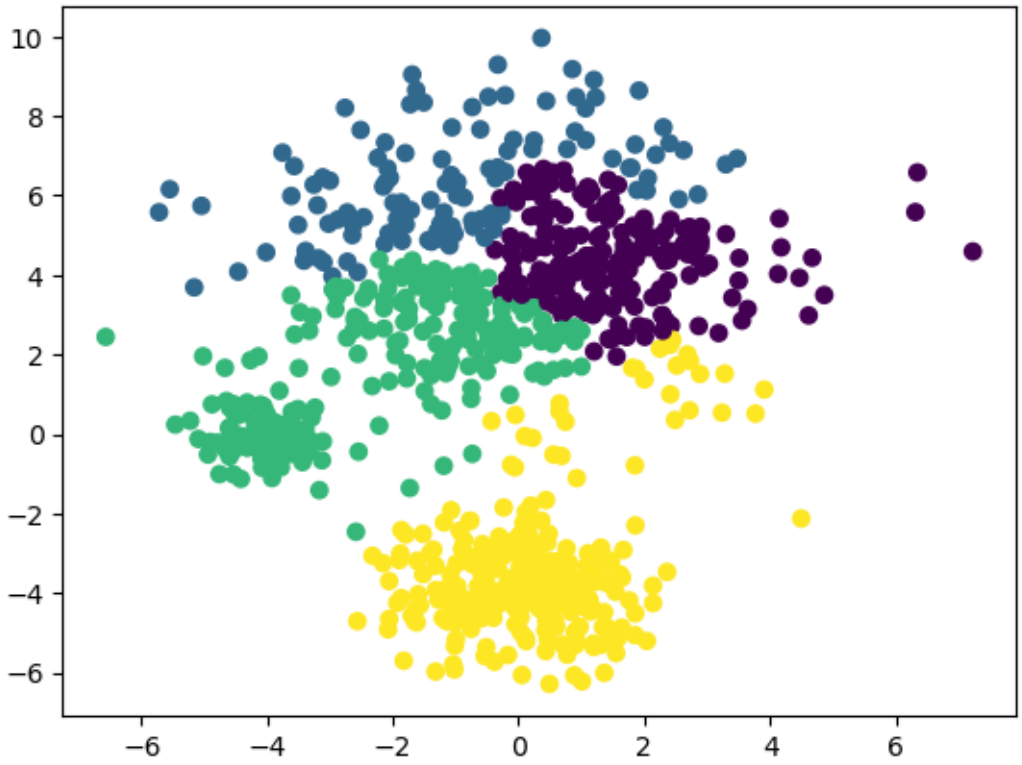
[17]:
```
# Plot the dendrogram
cargs = {'color_threshold': 11,'above_threshold_color':'black'}
fig, ax = plt.subplots(1, 1, figsize=(10,6))
hierarchy.dendrogram(hc_comp, ax=ax, **cargs, no_labels=True)
plt.show()
```



c) Now, use the function `cut_tree()` from `scipy.cluster.hierarchy` to determine the cluster labels associated with a given cut of the dendrogram. You can either specify the number of clusters via `n_clusters` or the height/threshold at which to cut via `height`. Plot the data colored by cluster membership.

[18]:
```
# Cut the tree at a specified number of clusters
c_labels = hierarchy.cut_tree(hc_comp, n_clusters=4)

plt.scatter(X[:,0], X[:,1], c = c_labels)
```

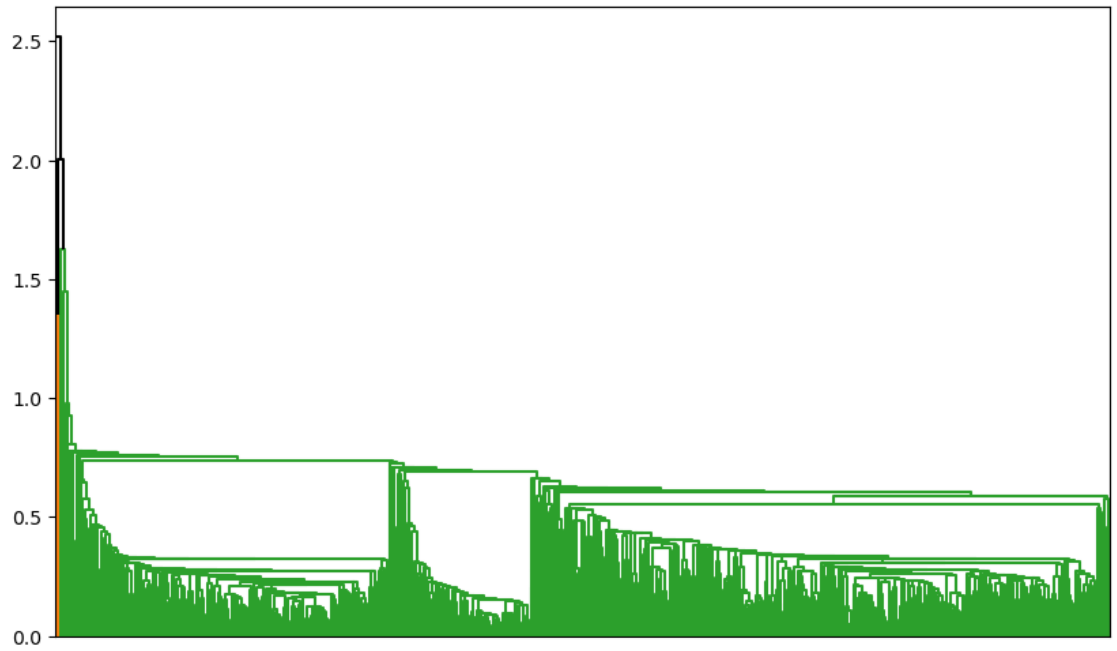
[18]:
```
<matplotlib.collections.PathCollection at 0x15e679430>
```



4.0.2 Exercise 8 (CORE)

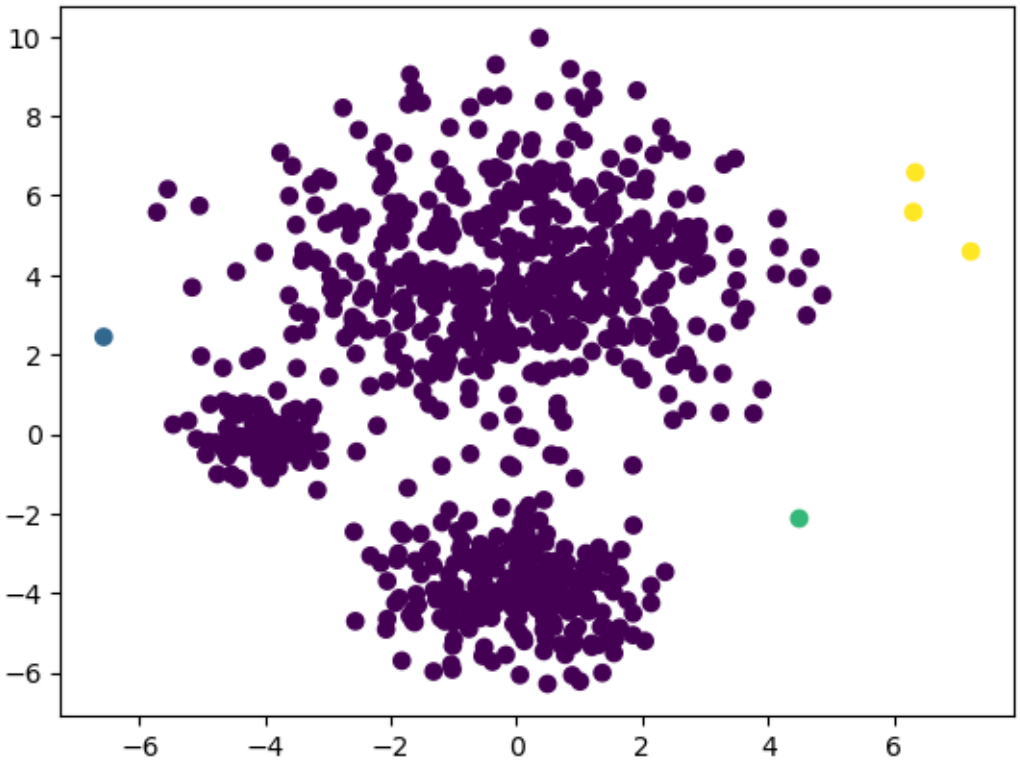
Now try changing the linkage to single and average. Does this affect on the results?

[19]:
```
# Change to single linkage, plot the dendrogram, and visualize the clustering␣
↪solution by cutting the tree
hc_single = hierarchy.linkage(X, method='single')
cargs = {'color_threshold': 2,'above_threshold_color':'black'}
fig, ax = plt.subplots(1, 1, figsize=(10,6))
hierarchy.dendrogram(hc_single, ax=ax, **cargs, no_labels=True)
plt.show()
```

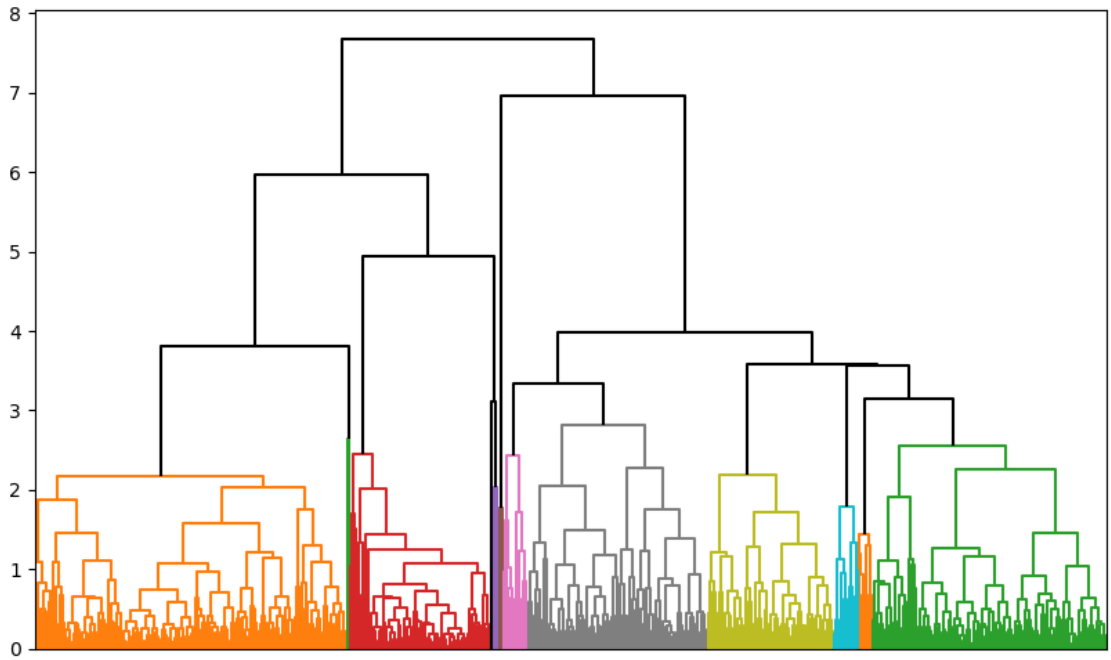


[20]:
```
c_labels = hierarchy.cut_tree(hc_single, n_clusters=4)
plt.scatter(X[:,0], X[:,1], c = c_labels)
```

[20]: <matplotlib.collections.PathCollection at 0x15e61f230>

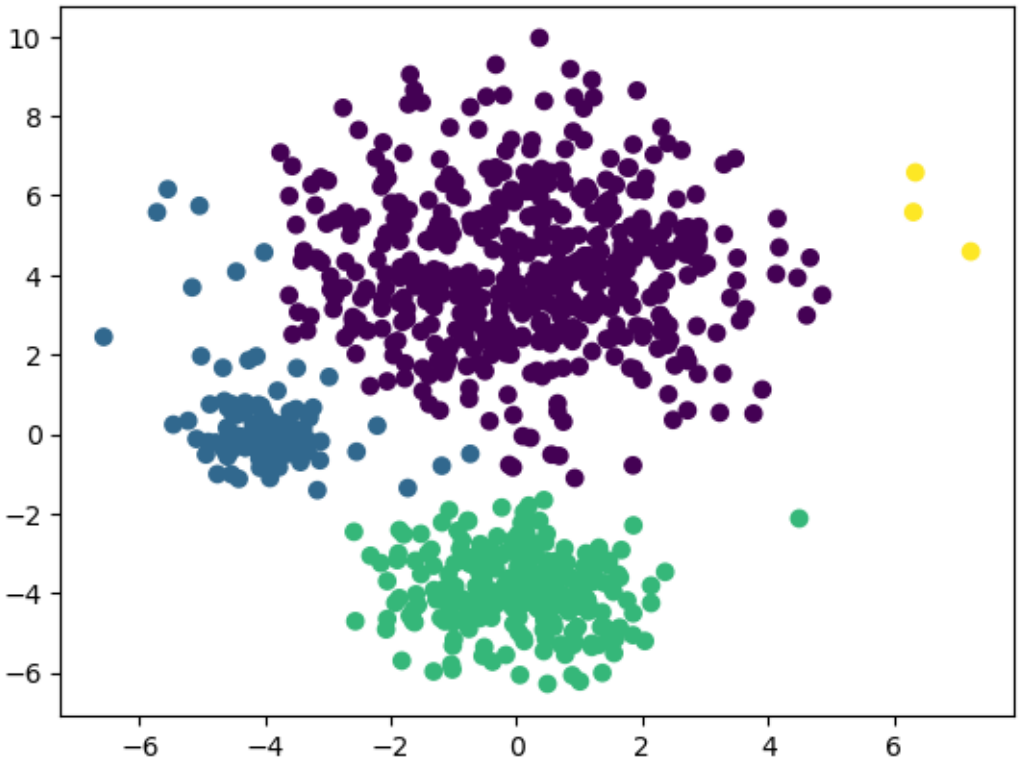


[21]:
```
# Change to average linkage, plot the dendrogram, and visualize the clustering␣
↪solution by cutting the tree
hc_avg = hierarchy.linkage(X, method='average')

cargs = {'color_threshold': 3,'above_threshold_color':'black'}
fig, ax = plt.subplots(1, 1, figsize=(10,6))
hierarchy.dendrogram(hc_avg, ax=ax, **cargs, no_labels=True)
plt.show()
```



[22]:
```
c_labels = hierarchy.cut_tree(hc_avg, n_clusters=4)
plt.scatter(X[:,0], X[:,1], c = c_labels)
```

[22]: <matplotlib.collections.PathCollection at 0x15e52b3e0>



Average seems to do the best job capturing the true nature of clustering. When comparing to a human output I would cluster similar to the average clustering.

Now, is a good point to switch driver and navigator

5 Gene Expression Data

Now, we will consider a more complex real dataset with a larger feature space.

The dataset is the **NCI cancer microarray dataset** discussed in both *Introduction to Statistical Learning* and *Elements of Statistical Learning*. The dataset consists of $D = 6830$ gene expression measurements for each of $N = 64$ cancer cell lines. The aim is to determine whether there are groups among the cell lines with similar gene expression patterns. This is an example of a high-dimensional dataset with D much larger than N, which makes visualization difficult. The $N = 64$ cancer cell lines have been obtained from samples of cancerous tisses, corresponding to 14 different types of cancer. However, our focus remains unsupervised learning and we will use the cancer labels only to plot.

We first need to read in the dataset.

[23]:
```
#Fetch the data and cancer labels
url_data = 'https://web.stanford.edu/~hastie/ElemStatLearn/datasets/nci.data.
  ↪csv'
```

```
url_labels = 'https://web.stanford.edu/~hastie/ElemStatLearn/datasets/nci.label.
 ↪txt'

X = pd.read_csv(url_data)
y = pd.read_csv(url_labels, header=None)

# clean data and follow convention in the notes that features are columns:
X = X.drop(labels='Unnamed: 0', axis=1).T
```

[24]:
```
X.shape
```

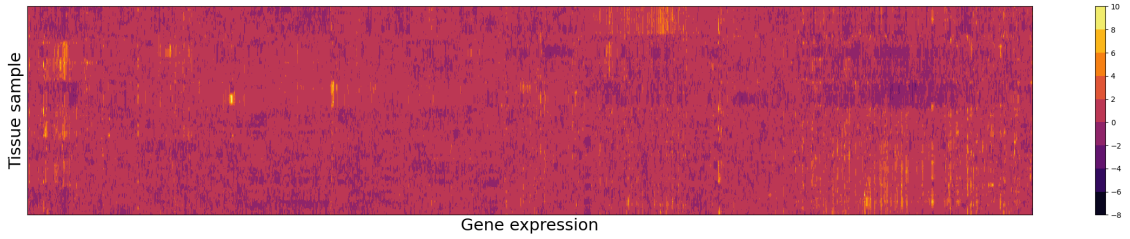
[24]: (64, 6830)

[25]:
```
y.shape
```

[25]: (64, 1)

Let's visualise the data with a contour plot.

[26]:
```
# Contour plot of the gene expression data
fig = plt.figure(figsize=(30,5))
ax = fig.add_subplot(111)
contours = ax.contourf(X, cmap='inferno')#, vmax=4, vmin=-4)
cbar = plt.colorbar(contours)
ax.set_xticks([])
ax.set_yticks([])
ax.set_xlabel("Gene expression", fontsize=22)
ax.set_ylabel("Tissue sample", fontsize=22)
plt.show()
```



We now convert our pandas dataframe into a numpy array and create integer labels for cancer type (for plotting purposes)

If you visualise the labels, you will notice there are lots of inconsistencies with white space etc. Run the following code to clean the labels.

[27]:
```
# Print the unique labels and counts
y.value_counts()
```

[27]:
```
0
RENAL                    4
OVARIAN                  4
MELANOMA                 3
NSCLC                    3
COLON                    3
BREAST                   2
NSCLC                    2
MELANOMA                 2
NSCLC                    2
LEUKEMIA                 2
PROSTATE                 2
COLON                    2
CNS                      2
CNS                      2
BREAST                   2
RENAL                    2
MELANOMA                 2
RENAL                    1
RENAL                    1
OVARIAN                  1
OVARIAN                  1
NSCLC                    1
NSCLC                    1
RENAL                    1
MCF7D-repro              1
MELANOMA                 1
BREAST                   1
MCF7A-repro              1
LEUKEMIA                 1
LEUKEMIA                 1
LEUKEMIA                 1
LEUKEMIA                 1
K562B-repro              1
K562A-repro              1
COLON                    1
COLON                    1
CNS                      1
BREAST                   1
BREAST                   1
UNKNOWN                  1
Name: count, dtype: int64
```

[28]:
```
my_lab = y.to_numpy().flatten()
```

[29]:
```
for j in range(my_lab.size):
    my_lab[j] = my_lab[j].strip()
```

[30]:
```
types = list(np.unique(my_lab))
types
```

[30]:
```
['BREAST',
 'CNS',
 'COLON',
 'K562A-repro',
 'K562B-repro',
 'LEUKEMIA',
 'MCF7A-repro',
 'MCF7D-repro',
 'MELANOMA',
 'NSCLC',
 'OVARIAN',
 'PROSTATE',
 'RENAL',
 'UNKNOWN']
```

[31]:
```
groups = [types.index(lab) for lab in my_lab]
```

[32]:
```
groups
```

[32]:
```
[1,
 1,
 1,
 12,
 0,
 1,
 1,
 0,
 9,
 9,
 12,
 12,
 12,
 12,
 12,
 12,
 12,
 0,
 9,
 12,
 13,
 10,
 8,
 11,
 10,
```

```
 10,
 10,
 10,
 10,
 11,
 9,
 9,
 9,
 5,
 4,
 3,
 5,
 5,
 5,
 5,
 5,
 2,
 2,
 2,
 2,
 2,
 2,
 2,
 6,
 0,
 7,
 0,
 9,
 9,
 9,
 8,
 0,
 0,
 8,
 8,
 8,
 8,
 8,
 8]
```

[33]:
```
# Clean the labels by stripping the white space
y_clean = np.asarray(y).flatten()
for j in range(y_clean.size):
    y_clean[j] = y_clean[j].strip()

cancer_types = list(np.unique(y_clean))
cancer_groups = np.array([cancer_types.index(lab) for lab in y_clean])
```

[34]:
```
print(cancer_types)
```

```
['BREAST', 'CNS', 'COLON', 'K562A-repro', 'K562B-repro', 'LEUKEMIA',
'MCF7A-repro', 'MCF7D-repro', 'MELANOMA', 'NSCLC', 'OVARIAN', 'PROSTATE',
'RENAL', 'UNKNOWN']
```

[35]:
```
X_array = np.asarray(X)
```

5.0.1 Exercise 9 (EXTRA)

Perform a PCA of $\mathbf{X}$ to visualize the data. Plot the first few principal component scores and color by cancer type. Do cell lines within the same cancer types seems to have similar scores? Make a scree plot of the proportion of variance explained. How many components does this suggest?

[167]:
```
from sklearn.decomposition import PCA

PCA(20)
```

6 Hierarchical Clustering: Gene Expression Data

Now, let's perform hierarchical clustering on the gene expression data.

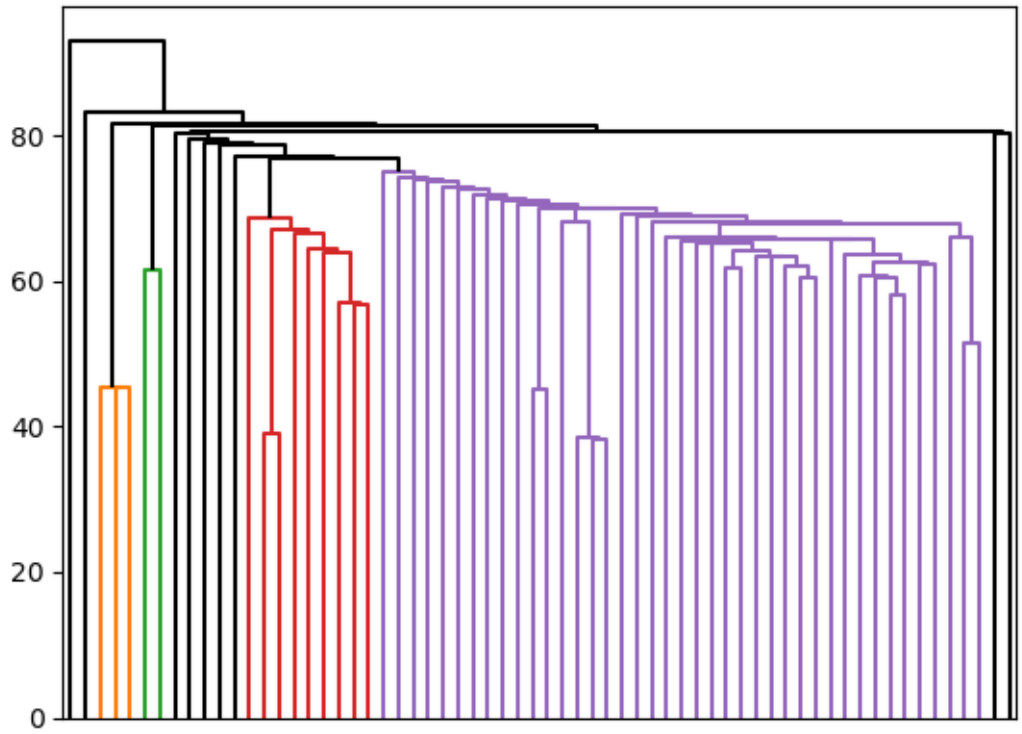
6.0.1 Exercise 10 (CORE)

a) Plot the dendrogram with complete, single, and average linkage. Does the choice of linkage affect the results? Which linkage would you choose?

[73]:
```
# Fit hierarchical clustering with different types of linkage

sing = hierarchy.linkage(X, method='single')
comp = hierarchy.linkage(X, method='complete')
avg  = hierarchy.linkage(X, method='average')
# Plot the dendogram
cargs = {'color_threshold': 75,'above_threshold_color':'black'}
hierarchy.dendrogram(sing, **cargs, no_labels=True)
cut = hierarchy.cut_tree(X, n_clusters=11)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[73], line 9
      7 cargs = {'color_threshold': 75,'above_threshold_color':'black'}
      8 hierarchy.dendrogram(sing, **cargs, no_labels=True)
----> 9 cut = hierarchy.cut_tree(X, n_clusters=11)

File /opt/anaconda3/lib/python3.12/site-packages/scipy/cluster/hierarchy.py:
 ↪1332, in cut_tree(Z, n_clusters, height)
   1286 """
   1287 Given a linkage matrix Z, return the cut tree.
```

```
   1288
   (…)
   1329
   1330 """
   1331 xp = array_namespace(Z)
-> 1332 nobs = num_obs_linkage(Z)
   1333 nodes = _order_cluster_tree(Z)
   1335 if height is not None and n_clusters is not None:

File /opt/anaconda3/lib/python3.12/site-packages/scipy/cluster/hierarchy.py:
 ↪2335, in num_obs_linkage(Z)
   2333 xp = array_namespace(Z)
   2334 Z = _asarray(Z, order='c', xp=xp)
-> 2335 is_valid_linkage(Z, throw=True, name='Z')
   2336 return (Z.shape[0] + 1)

File /opt/anaconda3/lib/python3.12/site-packages/scipy/cluster/hierarchy.py:
 ↪2235, in is_valid_linkage(Z, warning, throw, name)
   2232     raise ValueError('Linkage matrix %smust have shape=2 (i.e. be '
   2233                      'two-dimensional).' % name_str)
   2234 if Z.shape[1] != 4:
-> 2235     raise ValueError('Linkage matrix %smust have 4 columns.' % name_str
   2236 if Z.shape[0] == 0:
   2237     raise ValueError('Linkage must be computed on at least two '
   2238                      'observations.')

ValueError: Linkage matrix 'Z' must have 4 columns.
```



[64]:
```
cargs = {'color_threshold': 95,'above_threshold_color':'black'}
hierarchy.dendrogram(comp, **cargs, no_labels=True)
```

[64]:
```
{'icoord': [[25.0, 25.0, 35.0, 35.0],
  [15.0, 15.0, 30.0, 30.0],
  [5.0, 5.0, 22.5, 22.5],
  [55.0, 55.0, 65.0, 65.0],
  [75.0, 75.0, 85.0, 85.0],
  [60.0, 60.0, 80.0, 80.0],
  [45.0, 45.0, 70.0, 70.0],
  [95.0, 95.0, 105.0, 105.0],
  [57.5, 57.5, 100.0, 100.0],
  [13.75, 13.75, 78.75, 78.75],
  [115.0, 115.0, 125.0, 125.0],
  [155.0, 155.0, 165.0, 165.0],
  [145.0, 145.0, 160.0, 160.0],
  [175.0, 175.0, 185.0, 185.0],
  [152.5, 152.5, 180.0, 180.0],
  [135.0, 135.0, 166.25, 166.25],
  [120.0, 120.0, 150.625, 150.625],
  [46.25, 46.25, 135.3125, 135.3125],
  [205.0, 205.0, 215.0, 215.0],
  [195.0, 195.0, 210.0, 210.0],
```

```
 [225.0, 225.0, 235.0, 235.0],
 [275.0, 275.0, 285.0, 285.0],
 [265.0, 265.0, 280.0, 280.0],
 [255.0, 255.0, 272.5, 272.5],
 [295.0, 295.0, 305.0, 305.0],
 [315.0, 315.0, 325.0, 325.0],
 [300.0, 300.0, 320.0, 320.0],
 [263.75, 263.75, 310.0, 310.0],
 [245.0, 245.0, 286.875, 286.875],
 [345.0, 345.0, 355.0, 355.0],
 [335.0, 335.0, 350.0, 350.0],
 [365.0, 365.0, 375.0, 375.0],
 [385.0, 385.0, 395.0, 395.0],
 [415.0, 415.0, 425.0, 425.0],
 [405.0, 405.0, 420.0, 420.0],
 [390.0, 390.0, 412.5, 412.5],
 [370.0, 370.0, 401.25, 401.25],
 [342.5, 342.5, 385.625, 385.625],
 [435.0, 435.0, 445.0, 445.0],
 [465.0, 465.0, 475.0, 475.0],
 [455.0, 455.0, 470.0, 470.0],
 [505.0, 505.0, 515.0, 515.0],
 [495.0, 495.0, 510.0, 510.0],
 [485.0, 485.0, 502.5, 502.5],
 [462.5, 462.5, 493.75, 493.75],
 [525.0, 525.0, 535.0, 535.0],
 [555.0, 555.0, 565.0, 565.0],
 [545.0, 545.0, 560.0, 560.0],
 [530.0, 530.0, 552.5, 552.5],
 [595.0, 595.0, 605.0, 605.0],
 [625.0, 625.0, 635.0, 635.0],
 [615.0, 615.0, 630.0, 630.0],
 [600.0, 600.0, 622.5, 622.5],
 [585.0, 585.0, 611.25, 611.25],
 [575.0, 575.0, 598.125, 598.125],
 [541.25, 541.25, 586.5625, 586.5625],
 [478.125, 478.125, 563.90625, 563.90625],
 [440.0, 440.0, 521.015625, 521.015625],
 [364.0625, 364.0625, 480.5078125, 480.5078125],
 [265.9375, 265.9375, 422.28515625, 422.28515625],
 [230.0, 230.0, 344.111328125, 344.111328125],
 [202.5, 202.5, 287.0556640625, 287.0556640625],
 [90.78125, 90.78125, 244.77783203125, 244.77783203125]],
'dcoord': [[0.0, 38.23033266509951, 38.23033266509951, 0.0],
 [0.0, 39.99989875885876, 39.99989875885876, 38.23033266509951],
 [0.0, 74.8070242197004, 74.8070242197004, 39.99989875885876],
 [0.0, 60.496506570127174, 60.496506570127174, 0.0],
```

```
[0.0, 63.26414097768565, 63.26414097768565, 0.0],
[60.496506570127174,
 68.99669187921637,
 68.99669187921637,
 63.26414097768565],
[0.0, 75.92470654751754, 75.92470654751754, 68.99669187921637],
[0.0, 80.28907806694906, 80.28907806694906, 0.0],
[75.92470654751754, 83.8398260488138, 83.8398260488138, 80.28907806694906],
[74.8070242197004, 92.7071877871419, 92.7071877871419, 83.8398260488138],
[0.0, 61.55425326193697, 61.55425326193697, 0.0],
[0.0, 45.35338129441764, 45.35338129441764, 0.0],
[0.0, 48.011628188919204, 48.011628188919204, 45.35338129441764],
[0.0, 80.3509997411341, 80.3509997411341, 0.0],
[48.011628188919204, 89.33675089896792, 89.33675089896792, 80.3509997411341],
[0.0, 99.7640549200639, 99.7640549200639, 89.33675089896792],
[61.55425326193697, 99.9901854155011, 99.9901854155011, 99.7640549200639],
[92.7071877871419, 111.51306850739942, 111.51306850739942, 99.9901854155011],
[0.0, 78.9072803641754, 78.9072803641754, 0.0],
[0.0, 97.94830934467525, 97.94830934467525, 78.9072803641754],
[0.0, 80.19940947059743, 80.19940947059743, 0.0],
[0.0, 56.78015373222973, 56.78015373222973, 0.0],
[0.0, 63.674411120061656, 63.674411120061656, 56.78015373222973],
[0.0, 66.23109874088539, 66.23109874088539, 63.674411120061656],
[0.0, 39.10562478531746, 39.10562478531746, 0.0],
[0.0, 68.6060228178101, 68.6060228178101, 0.0],
[39.10562478531746, 71.59949093826448, 71.59949093826448, 68.6060228178101],
[66.23109874088539, 77.32029088122367, 77.32029088122367, 71.59949093826448],
[0.0, 82.1622036028543, 82.1622036028543, 77.32029088122367],
[0.0, 45.15158095717165, 45.15158095717165, 0.0],
[0.0, 86.31140874481736, 86.31140874481736, 45.15158095717165],
[0.0, 80.97092472159423, 80.97092472159423, 0.0],
[0.0, 51.43823072875002, 51.43823072875002, 0.0],
[0.0, 77.06837620497099, 77.06837620497099, 0.0],
[0.0, 80.11985439021791, 80.11985439021791, 77.06837620497099],
[51.43823072875002, 84.32318975839188, 84.32318975839188, 80.11985439021791],
[80.97092472159423, 90.13572613600087, 90.13572613600087, 84.32318975839188],
[86.31140874481736, 92.58579286262689, 92.58579286262689, 90.13572613600087],
[0.0, 73.56083029371364, 73.56083029371364, 0.0],
[0.0, 57.91726379245596, 57.91726379245596, 0.0],
[0.0, 67.07313092778026, 67.07313092778026, 57.91726379245596],
[0.0, 62.17805940041783, 62.17805940041783, 0.0],
[0.0, 62.78163257503842, 62.78163257503842, 62.17805940041783],
[0.0, 72.56828297923231, 72.56828297923231, 62.78163257503842],
[67.07313092778026, 78.42221831846733, 78.42221831846733, 72.56828297923231],
[0.0, 70.93759619159026, 70.93759619159026, 0.0],
[0.0, 61.637503616580325, 61.637503616580325, 0.0],
[0.0, 73.6766982322484, 73.6766982322484, 61.637503616580325],
```

```
 [70.93759619159026, 78.5528923492835, 78.5528923492835, 73.6766982322484],
 [0.0, 67.83789247908632, 67.83789247908632, 0.0],
 [0.0, 65.76413536190013, 65.76413536190013, 0.0],
 [0.0, 70.6227323071043, 70.6227323071043, 65.76413536190013],
 [67.83789247908632, 74.62701246996566, 74.62701246996566, 70.6227323071043],
 [0.0, 76.63179160393543, 76.63179160393543, 74.62701246996566],
 [0.0, 81.8980420004987, 81.8980420004987, 76.63179160393543],
 [78.5528923492835, 84.13402431906778, 84.13402431906778, 81.8980420004987],
 [78.42221831846733, 88.85328368632948, 88.85328368632948, 84.13402431906778],
 [73.56083029371364, 92.65795123720224, 92.65795123720224, 88.85328368632948],
 [92.58579286262689,
  101.82696152499048,
  101.82696152499048,
  92.65795123720224],
 [82.1622036028543,
  105.92309815375836,
  105.92309815375836,
  101.82696152499048],
 [80.19940947059743,
  106.43870527175368,
  106.43870527175368,
  105.92309815375836],
 [97.94830934467525,
  118.25973071690086,
  118.25973071690086,
  106.43870527175368],
 [111.51306850739942,
  138.15044875568614,
  138.15044875568614,
  118.25973071690086]],
'ivl': ['51',
 '48',
 '49',
 '50',
 '42',
 '41',
 '43',
 '44',
 '45',
 '46',
 '47',
 '38',
 '39',
 '40',
 '36',
 '34',
 '35',
```

```
'33',
'37',
'9',
'3',
'4',
'53',
'54',
'55',
'58',
'59',
'60',
'61',
'56',
'57',
'62',
'63',
'19',
'20',
'21',
'17',
'18',
'0',
'1',
'6',
'5',
'7',
'24',
'25',
'10',
'11',
'12',
'16',
'13',
'14',
'15',
'26',
'27',
'52',
'30',
'31',
'32',
'22',
'2',
'8',
'23',
'28',
'29'],
```

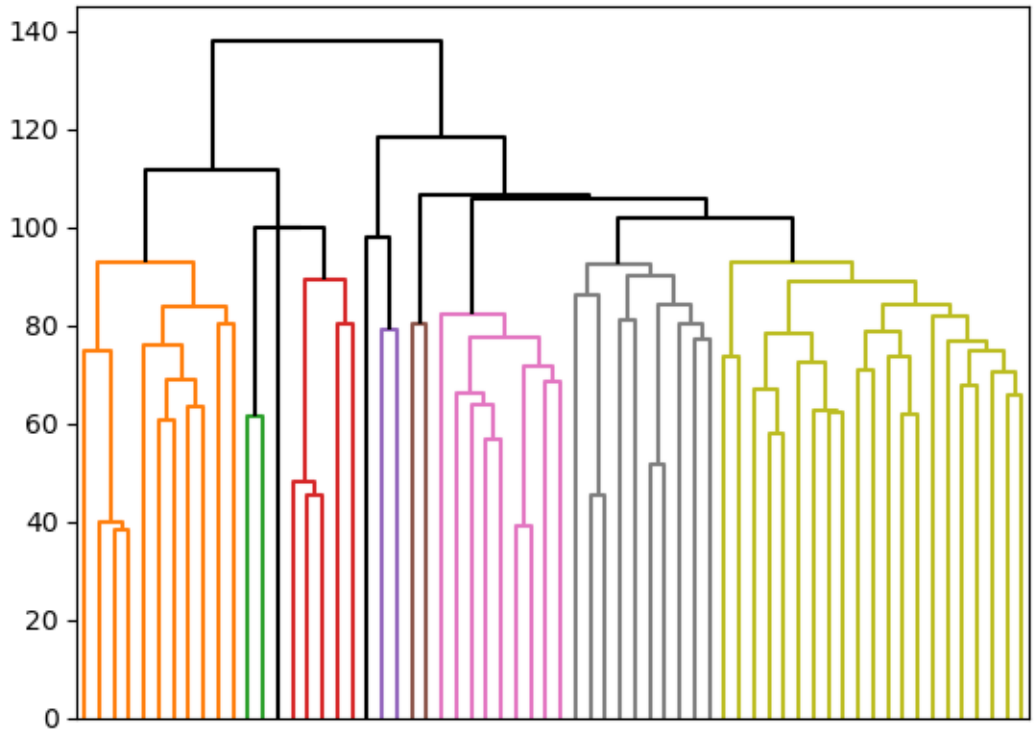
```
'leaves': [51,
 48,
 49,
 50,
 42,
 41,
 43,
 44,
 45,
 46,
 47,
 38,
 39,
 40,
 36,
 34,
 35,
 33,
 37,
 9,
 3,
 4,
 53,
 54,
 55,
 58,
 59,
 60,
 61,
 56,
 57,
 62,
 63,
 19,
 20,
 21,
 17,
 18,
 0,
 1,
 6,
 5,
 7,
 24,
 25,
 10,
 11,
```

```
 12,
 16,
 13,
 14,
 15,
 26,
 27,
 52,
 30,
 31,
 32,
 22,
 2,
 8,
 23,
 28,
 29],
'color_list': ['C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C2',
 'C3',
 'C3',
 'C3',
 'C3',
 'black',
 'black',
 'black',
 'C4',
 'black',
 'C5',
 'C6',
 'C6',
 'C6',
 'C6',
 'C6',
 'C6',
 'C6',
 'C6',
 'C7',
```

```
 'C7',
 'C7',
 'C7',
 'C7',
 'C7',
 'C7',
 'C7',
 'C7',
 'C8',
 'C8',
 'C8',
 'C8',
 'C8',
 'C8',
 'C8',
 'C8',
 'C8',
 'C8',
 'C8',
 'C8',
 'C8',
 'C8',
 'C8',
 'C8',
 'C8',
 'C8',
 'C8',
 'C8',
 'black',
 'black',
 'black',
 'black',
 'black'],
'leaves_color_list': ['C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C1',
 'C2',
 'C2',
 'black',
```

```
'C3',
'C3',
'C3',
'C3',
'C3',
'black',
'C4',
'C4',
'C5',
'C5',
'C6',
'C6',
'C6',
'C6',
'C6',
'C6',
'C6',
'C6',
'C6',
'C6',
'C7',
'C7',
'C7',
'C7',
'C7',
'C7',
'C7',
'C7',
'C7',
'C7',
'C8',
'C8',
'C8',
'C8',
'C8',
'C8',
'C8',
'C8',
'C8',
'C8',
'C8',
'C8',
'C8',
'C8',
'C8',
'C8',
'C8',
'C8',
'C8',
```

```
'C8',
'C8',
'C8']}
```



[63]:
```
cargs = {'color_threshold': 80,'above_threshold_color':'black'}
hierarchy.dendrogram(avg, **cargs, no_labels=True)
```

[63]:
```
{'icoord': [[15.0, 15.0, 25.0, 25.0],
  [45.0, 45.0, 55.0, 55.0],
  [35.0, 35.0, 50.0, 50.0],
  [65.0, 65.0, 75.0, 75.0],
  [42.5, 42.5, 70.0, 70.0],
  [20.0, 20.0, 56.25, 56.25],
  [5.0, 5.0, 38.125, 38.125],
  [95.0, 95.0, 105.0, 105.0],
  [155.0, 155.0, 165.0, 165.0],
  [145.0, 145.0, 160.0, 160.0],
  [135.0, 135.0, 152.5, 152.5],
  [125.0, 125.0, 143.75, 143.75],
  [115.0, 115.0, 134.375, 134.375],
  [100.0, 100.0, 124.6875, 124.6875],
  [85.0, 85.0, 112.34375, 112.34375],
  [185.0, 185.0, 195.0, 195.0],
```

```
[205.0, 205.0, 215.0, 215.0],
[225.0, 225.0, 235.0, 235.0],
[245.0, 245.0, 255.0, 255.0],
[230.0, 230.0, 250.0, 250.0],
[285.0, 285.0, 295.0, 295.0],
[275.0, 275.0, 290.0, 290.0],
[315.0, 315.0, 325.0, 325.0],
[305.0, 305.0, 320.0, 320.0],
[282.5, 282.5, 312.5, 312.5],
[265.0, 265.0, 297.5, 297.5],
[345.0, 345.0, 355.0, 355.0],
[375.0, 375.0, 385.0, 385.0],
[405.0, 405.0, 415.0, 415.0],
[395.0, 395.0, 410.0, 410.0],
[380.0, 380.0, 402.5, 402.5],
[365.0, 365.0, 391.25, 391.25],
[350.0, 350.0, 378.125, 378.125],
[335.0, 335.0, 364.0625, 364.0625],
[281.25, 281.25, 349.53125, 349.53125],
[240.0, 240.0, 315.390625, 315.390625],
[445.0, 445.0, 455.0, 455.0],
[435.0, 435.0, 450.0, 450.0],
[425.0, 425.0, 442.5, 442.5],
[475.0, 475.0, 485.0, 485.0],
[465.0, 465.0, 480.0, 480.0],
[433.75, 433.75, 472.5, 472.5],
[277.6953125, 277.6953125, 453.125, 453.125],
[210.0, 210.0, 365.41015625, 365.41015625],
[190.0, 190.0, 287.705078125, 287.705078125],
[175.0, 175.0, 238.8525390625, 238.8525390625],
[98.671875, 98.671875, 206.92626953125, 206.92626953125],
[495.0, 495.0, 505.0, 505.0],
[535.0, 535.0, 545.0, 545.0],
[525.0, 525.0, 540.0, 540.0],
[515.0, 515.0, 532.5, 532.5],
[585.0, 585.0, 595.0, 595.0],
[605.0, 605.0, 615.0, 615.0],
[590.0, 590.0, 610.0, 610.0],
[575.0, 575.0, 600.0, 600.0],
[565.0, 565.0, 587.5, 587.5],
[555.0, 555.0, 576.25, 576.25],
[523.75, 523.75, 565.625, 565.625],
[500.0, 500.0, 544.6875, 544.6875],
[152.799072265625, 152.799072265625, 522.34375, 522.34375],
[625.0, 625.0, 635.0, 635.0],
[337.5714111328125, 337.5714111328125, 630.0, 630.0],
[21.5625, 21.5625, 483.78570556640625, 483.78570556640625]],
```

```
'dcoord': [[0.0, 61.55425326193697, 61.55425326193697, 0.0],
 [0.0, 45.35338129441764, 45.35338129441764, 0.0],
 [0.0, 46.72729023673859, 46.72729023673859, 45.35338129441764],
 [0.0, 80.3509997411341, 80.3509997411341, 0.0],
 [46.72729023673859, 85.41874237671023, 85.41874237671023, 80.3509997411341],
 [61.55425326193697, 94.05713411884713, 94.05713411884713, 85.41874237671023],
 [0.0, 97.62270297875179, 97.62270297875179, 94.05713411884713],
 [0.0, 39.10562478531746, 39.10562478531746, 0.0],
 [0.0, 56.78015373222973, 56.78015373222973, 0.0],
 [0.0, 60.32136099283471, 60.32136099283471, 56.78015373222973],
 [0.0, 65.03493727460739, 65.03493727460739, 60.32136099283471],
 [0.0, 66.08965144723348, 66.08965144723348, 65.03493727460739],
 [0.0, 69.21690713768909, 69.21690713768909, 66.08965144723348],
 [39.10562478531746, 72.07973687484922, 72.07973687484922, 69.21690713768909],
 [0.0, 77.27159134710296, 77.27159134710296, 72.07973687484922],
 [0.0, 80.97092472159423, 80.97092472159423, 0.0],
 [0.0, 73.56083029371364, 73.56083029371364, 0.0],
 [0.0, 45.15158095717165, 45.15158095717165, 0.0],
 [0.0, 71.27390115422887, 71.27390115422887, 0.0],
 [45.15158095717165, 74.91282982789339, 74.91282982789339, 71.27390115422887],
 [0.0, 62.17805940041783, 62.17805940041783, 0.0],
 [0.0, 62.60555272962195, 62.60555272962195, 62.17805940041783],
 [0.0, 57.91726379245596, 57.91726379245596, 0.0],
 [0.0, 66.1778876229592, 66.1778876229592, 57.91726379245596],
 [62.60555272962195, 67.7373213448314, 67.7373213448314, 66.1778876229592],
 [0.0, 70.41020026319184, 70.41020026319184, 67.7373213448314],
 [0.0, 70.93759619159026, 70.93759619159026, 0.0],
 [0.0, 61.637503616580325, 61.637503616580325, 0.0],
 [0.0, 65.76413536190013, 65.76413536190013, 0.0],
 [0.0, 69.12748707863923, 69.12748707863923, 65.76413536190013],
 [61.637503616580325,
  70.34299206950027,
  70.34299206950027,
  69.12748707863923],
 [0.0, 72.51594249535299, 72.51594249535299, 70.34299206950027],
 [70.93759619159026, 73.62302893444844, 73.62302893444844, 72.51594249535299],
 [0.0, 76.67513536627712, 76.67513536627712, 73.62302893444844],
 [70.41020026319184, 77.36448359110311, 77.36448359110311, 76.67513536627712],
 [74.91282982789339, 79.24936058528212, 79.24936058528212, 77.36448359110311],
 [0.0, 51.43823072875002, 51.43823072875002, 0.0],
 [0.0, 67.46269863944634, 67.46269863944634, 51.43823072875002],
 [0.0, 77.78729021009927, 77.78729021009927, 67.46269863944634],
 [0.0, 77.06837620497099, 77.06837620497099, 0.0],
 [0.0, 79.03982311967985, 79.03982311967985, 77.06837620497099],
 [77.78729021009927, 80.72823407505152, 80.72823407505152, 79.03982311967985],
 [79.24936058528212, 83.28803585536917, 83.28803585536917, 80.72823407505152],
 [73.56083029371364, 85.66805452760087, 85.66805452760087, 83.28803585536917],
```

```
 [80.97092472159423, 87.05121898221603, 87.05121898221603, 85.66805452760087],
 [0.0, 90.31513376525673, 90.31513376525673, 87.05121898221603],
 [77.27159134710296, 90.80837059102737, 90.80837059102737, 90.31513376525673],
 [0.0, 80.19940947059743, 80.19940947059743, 0.0],
 [0.0, 38.23033266509951, 38.23033266509951, 0.0],
 [0.0, 39.2979702102202, 39.2979702102202, 38.23033266509951],
 [0.0, 72.2098824201147, 72.2098824201147, 39.2979702102202],
 [0.0, 60.496506570127174, 60.496506570127174, 0.0],
 [0.0, 63.26414097768565, 63.26414097768565, 0.0],
 [60.496506570127174,
  65.74219903775725,
  65.74219903775725,
  63.26414097768565],
 [0.0, 71.99332788246787, 71.99332788246787, 65.74219903775725],
 [0.0, 74.98528106513152, 74.98528106513152, 71.99332788246787],
 [0.0, 78.03069045653744, 78.03069045653744, 74.98528106513152],
 [72.2098824201147, 84.81251811853673, 84.81251811853673, 78.03069045653744],
 [80.19940947059743, 92.06102581215346, 92.06102581215346, 84.81251811853673],
 [90.80837059102737, 92.99565249473405, 92.99565249473405, 92.06102581215346],
 [0.0, 96.0569297563875, 96.0569297563875, 0.0],
 [92.99565249473405, 98.41984521601518, 98.41984521601518, 96.0569297563875],
 [97.62270297875179,
  103.15960016309877,
  103.15960016309877,
  98.41984521601518]],
'ivl': ['40',
 '38',
 '39',
 '36',
 '34',
 '35',
 '33',
 '37',
 '55',
 '56',
 '57',
 '62',
 '58',
 '63',
 '59',
 '60',
 '61',
 '19',
 '17',
 '18',
 '24',
 '25',
```

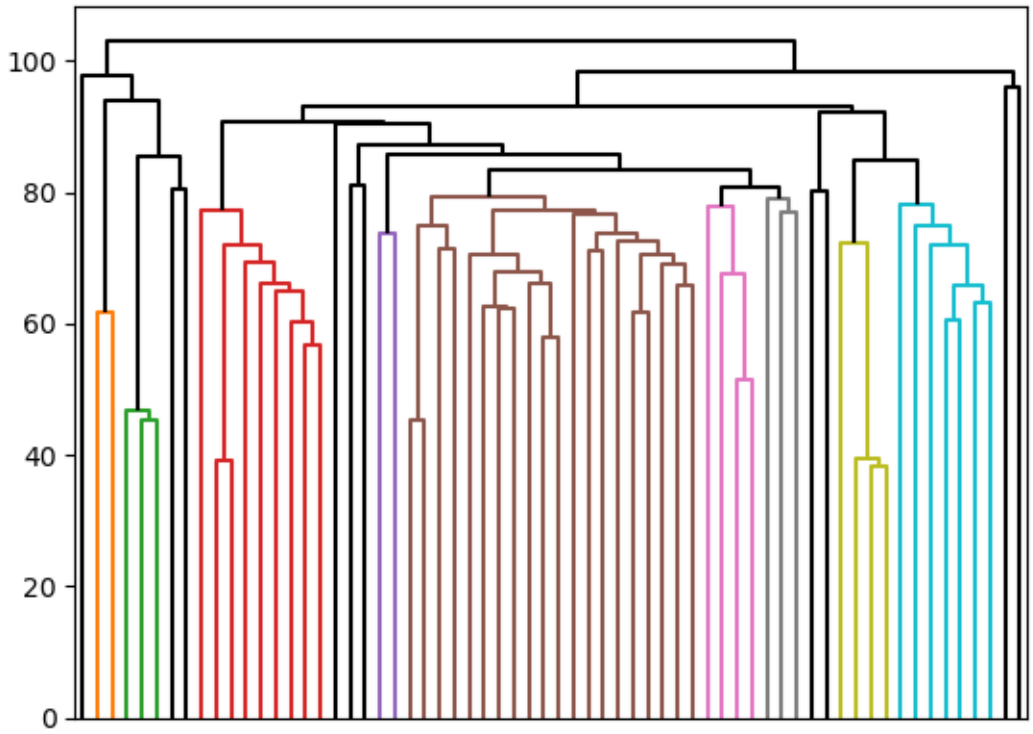
```
 '20',
 '21',
 '8',
 '22',
 '16',
 '13',
 '14',
 '15',
 '10',
 '11',
 '12',
 '32',
 '26',
 '27',
 '52',
 '30',
 '31',
 '23',
 '28',
 '29',
 '3',
 '2',
 '0',
 '1',
 '6',
 '5',
 '7',
 '53',
 '54',
 '51',
 '48',
 '49',
 '50',
 '46',
 '47',
 '42',
 '41',
 '43',
 '44',
 '45',
 '4',
 '9'],
'leaves': [40,
 38,
 39,
 36,
 34,
```

```
35,
33,
37,
55,
56,
57,
62,
58,
63,
59,
60,
61,
19,
17,
18,
24,
25,
20,
21,
8,
22,
16,
13,
14,
15,
10,
11,
12,
32,
26,
27,
52,
30,
31,
23,
28,
29,
3,
2,
0,
1,
6,
5,
7,
53,
54,
51,
```

```
 48,
 49,
 50,
 46,
 47,
 42,
 41,
 43,
 44,
 45,
 4,
 9],
'color_list': ['C1',
 'C2',
 'C2',
 'black',
 'black',
 'black',
 'black',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'black',
 'C4',
 'C5',
 'C5',
 'C5',
 'C5',
 'C5',
 'C5',
 'C5',
 'C5',
 'C5',
 'C5',
 'C5',
 'C5',
 'C5',
 'C5',
 'C5',
 'C5',
 'C5',
 'C5',
 'C5',
 'C5',
 'C5',
```

```
 'C5',
 'C6',
 'C6',
 'C6',
 'C7',
 'C7',
 'black',
 'black',
 'black',
 'black',
 'black',
 'black',
 'black',
 'C8',
 'C8',
 'C8',
 'C9',
 'C9',
 'C9',
 'C9',
 'C9',
 'C9',
 'black',
 'black',
 'black',
 'black',
 'black',
 'black'],
'leaves_color_list': ['black',
 'C1',
 'C1',
 'C2',
 'C2',
 'C2',
 'black',
 'black',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'C3',
 'black',
 'black',
```

```
'black',
'C4',
'C4',
'C5',
'C5',
'C5',
'C5',
'C5',
'C5',
'C5',
'C5',
'C5',
'C5',
'C5',
'C5',
'C5',
'C5',
'C5',
'C5',
'C5',
'C5',
'C5',
'C5',
'C6',
'C6',
'C6',
'C6',
'C7',
'C7',
'C7',
'black',
'black',
'C8',
'C8',
'C8',
'C8',
'C9',
'C9',
'C9',
'C9',
'C9',
'C9',
'C9',
'black',
'black']}
```



Yes the choice of linkage does impact the clustering. Within single linkage we see long chains forming. This may be more relevant when dealing with geneology since genes are often related and grouping genes through this long chain may capture some internal connection between the different gene strands.

b) Select a linkage and a number of clusters (by examining the dendrogram and jumps in the heights of the clusters merged). Plot the dendogram and color the branches to identify the clusters. Use the option `labels = np.asarray(y_clean), leaf_font_size=10` in `hierarchy.dendrogram` to add the cancer types as labels for each data point. Do you observe any patterns between the clusters and cancer types? You may also want to use `pd.crosstab` to compute a cross-tabulation to compare the clusters and cancer types.

[ ]:
```
hierarchy.dendrogram(sing, labels=np.array(y_clean), leaf_font_size=10)
```

[ ]:
```
{'icoord': [[35.0, 35.0, 45.0, 45.0],
  [25.0, 25.0, 40.0, 40.0],
  [55.0, 55.0, 65.0, 65.0],
  [135.0, 135.0, 145.0, 145.0],
  [195.0, 195.0, 205.0, 205.0],
  [185.0, 185.0, 200.0, 200.0],
  [175.0, 175.0, 192.5, 192.5],
  [165.0, 165.0, 183.75, 183.75],
  [155.0, 155.0, 174.375, 174.375],
  [140.0, 140.0, 164.6875, 164.6875],
```

```
[125.0, 125.0, 152.34375, 152.34375],
[315.0, 315.0, 325.0, 325.0],
[355.0, 355.0, 365.0, 365.0],
[345.0, 345.0, 360.0, 360.0],
[335.0, 335.0, 352.5, 352.5],
[445.0, 445.0, 455.0, 455.0],
[495.0, 495.0, 505.0, 505.0],
[485.0, 485.0, 500.0, 500.0],
[475.0, 475.0, 492.5, 492.5],
[465.0, 465.0, 483.75, 483.75],
[450.0, 450.0, 474.375, 474.375],
[435.0, 435.0, 462.1875, 462.1875],
[425.0, 425.0, 448.59375, 448.59375],
[415.0, 415.0, 436.796875, 436.796875],
[555.0, 555.0, 565.0, 565.0],
[545.0, 545.0, 560.0, 560.0],
[535.0, 535.0, 552.5, 552.5],
[575.0, 575.0, 585.0, 585.0],
[543.75, 543.75, 580.0, 580.0],
[525.0, 525.0, 561.875, 561.875],
[515.0, 515.0, 543.4375, 543.4375],
[425.8984375, 425.8984375, 529.21875, 529.21875],
[405.0, 405.0, 477.55859375, 477.55859375],
[605.0, 605.0, 615.0, 615.0],
[595.0, 595.0, 610.0, 610.0],
[441.279296875, 441.279296875, 602.5, 602.5],
[395.0, 395.0, 521.8896484375, 521.8896484375],
[385.0, 385.0, 458.44482421875, 458.44482421875],
[375.0, 375.0, 421.722412109375, 421.722412109375],
[343.75, 343.75, 398.3612060546875, 398.3612060546875],
[320.0, 320.0, 371.05560302734375, 371.05560302734375],
[305.0, 305.0, 345.5278015136719, 345.5278015136719],
[295.0, 295.0, 325.26390075683594, 325.26390075683594],
[285.0, 285.0, 310.13195037841797, 310.13195037841797],
[275.0, 275.0, 297.565975189209, 297.565975189209],
[265.0, 265.0, 286.2829875946045, 286.2829875946045],
[255.0, 255.0, 275.64149379730225, 275.64149379730225],
[245.0, 245.0, 265.3207468986511, 265.3207468986511],
[235.0, 235.0, 255.16037344932556, 255.16037344932556],
[225.0, 225.0, 245.08018672466278, 245.08018672466278],
[215.0, 215.0, 235.0400933623314, 235.0400933623314],
[138.671875, 138.671875, 225.0200466811657, 225.0200466811657],
[115.0, 115.0, 181.84596084058285, 181.84596084058285],
[105.0, 105.0, 148.42298042029142, 148.42298042029142],
[95.0, 95.0, 126.71149021014571, 126.71149021014571],
[85.0, 85.0, 110.85574510507286, 110.85574510507286],
[75.0, 75.0, 97.92787255253643, 97.92787255253643],
```

```
 [625.0, 625.0, 635.0, 635.0],
 [86.46393627626821, 86.46393627626821, 630.0, 630.0],
 [60.0, 60.0, 358.2319681381341, 358.2319681381341],
 [32.5, 32.5, 209.11598406906705, 209.11598406906705],
 [15.0, 15.0, 120.80799203453353, 120.80799203453353],
 [5.0, 5.0, 67.90399601726676, 67.90399601726676]],
'dcoord': [[0.0, 45.35338129441764, 45.35338129441764, 0.0],
 [0.0, 45.442952284557975, 45.442952284557975, 45.35338129441764],
 [0.0, 61.55425326193697, 61.55425326193697, 0.0],
 [0.0, 39.10562478531746, 39.10562478531746, 0.0],
 [0.0, 56.78015373222973, 56.78015373222973, 0.0],
 [0.0, 56.96831086560776, 56.96831086560776, 56.78015373222973],
 [0.0, 63.74587788862488, 63.74587788862488, 56.96831086560776],
 [0.0, 64.3726220563973, 64.3726220563973, 63.74587788862488],
 [0.0, 66.4870807234396, 66.4870807234396, 64.3726220563973],
 [39.10562478531746, 66.90877972818537, 66.90877972818537, 66.4870807234396],
 [0.0, 68.58009711229795, 68.58009711229795, 66.90877972818537],
 [0.0, 45.15158095717165, 45.15158095717165, 0.0],
 [0.0, 38.23033266509951, 38.23033266509951, 0.0],
 [0.0, 38.59604166158164, 38.59604166158164, 38.23033266509951],
 [0.0, 68.20511741691348, 68.20511741691348, 38.59604166158164],
 [0.0, 61.637503616580325, 61.637503616580325, 0.0],
 [0.0, 60.496506570127174, 60.496506570127174, 0.0],
 [0.0, 61.92928070280477, 61.92928070280477, 60.496506570127174],
 [0.0, 63.26414097768565, 63.26414097768565, 61.92928070280477],
 [0.0, 63.4286314852814, 63.4286314852814, 63.26414097768565],
 [61.637503616580325, 64.19222732437795, 64.19222732437795, 63.4286314852814],
 [0.0, 65.12892102156222, 65.12892102156222, 64.19222732437795],
 [0.0, 65.29849226098817, 65.29849226098817, 65.12892102156222],
 [0.0, 65.47270790078538, 65.47270790078538, 65.29849226098817],
 [0.0, 57.91726379245596, 57.91726379245596, 0.0],
 [0.0, 60.350522708371386, 60.350522708371386, 57.91726379245596],
 [0.0, 60.80441099581368, 60.80441099581368, 60.350522708371386],
 [0.0, 62.17805940041783, 62.17805940041783, 0.0],
 [60.80441099581368, 62.42947288420547, 62.42947288420547, 62.17805940041783],
 [0.0, 63.57607597056812, 63.57607597056812, 62.42947288420547],
 [0.0, 65.70067054382524, 65.70067054382524, 63.57607597056812],
 [65.47270790078538, 65.7710667950194, 65.7710667950194, 65.70067054382524],
 [0.0, 65.91204531855446, 65.91204531855446, 65.7710667950194],
 [0.0, 51.43823072875002, 51.43823072875002, 0.0],
 [0.0, 65.93814644245552, 65.93814644245552, 51.43823072875002],
 [65.91204531855446, 67.83789247908632, 67.83789247908632, 65.93814644245552],
 [0.0, 68.16587538955464, 68.16587538955464, 67.83789247908632],
 [0.0, 68.81308416013594, 68.81308416013594, 68.16587538955464],
 [0.0, 69.02726291794762, 69.02726291794762, 68.81308416013594],
 [68.20511741691348, 69.98455840670991, 69.98455840670991, 69.02726291794762],
 [45.15158095717165, 69.98966545127753, 69.98966545127753, 69.98455840670991],
```

```
 [0.0, 70.51728495164731, 70.51728495164731, 69.98966545127753],
 [0.0, 70.93759619159026, 70.93759619159026, 70.51728495164731],
 [0.0, 71.27390115422887, 71.27390115422887, 70.93759619159026],
 [0.0, 71.74470569620742, 71.74470569620742, 71.27390115422887],
 [0.0, 72.55172510839922, 72.55172510839922, 71.74470569620742],
 [0.0, 72.85034831888515, 72.85034831888515, 72.55172510839922],
 [0.0, 73.56083029371364, 73.56083029371364, 72.85034831888515],
 [0.0, 74.00465691447805, 74.00465691447805, 73.56083029371364],
 [0.0, 74.17664690374679, 74.17664690374679, 74.00465691447805],
 [0.0, 74.90674331908522, 74.90674331908522, 74.17664690374679],
 [68.58009711229795, 76.67936684081786, 76.67936684081786, 74.90674331908522],
 [0.0, 77.13961534533145, 77.13961534533145, 76.67936684081786],
 [0.0, 78.72362916253624, 78.72362916253624, 77.13961534533145],
 [0.0, 78.9072803641754, 78.9072803641754, 78.72362916253624],
 [0.0, 79.50676497995694, 79.50676497995694, 78.9072803641754],
 [0.0, 80.19940947059743, 80.19940947059743, 79.50676497995694],
 [0.0, 80.3509997411341, 80.3509997411341, 0.0],
 [80.19940947059743, 80.49364395273476, 80.49364395273476, 80.3509997411341],
 [61.55425326193697, 81.30137854951302, 81.30137854951302, 80.49364395273476],
 [45.442952284557975,
  81.66618694677368,
  81.66618694677368,
  81.30137854951302],
 [0.0, 83.2325224404959, 83.2325224404959, 81.66618694677368],
 [0.0, 93.06565171073733, 93.06565171073733, 83.2325224404959]],
'ivl': ['LEUKEMIA',
 'NSCLC',
 'LEUKEMIA',
 'K562B-repro',
 'K562A-repro',
 'LEUKEMIA',
 'LEUKEMIA',
 'NSCLC',
 'BREAST',
 'BREAST',
 'RENAL',
 'NSCLC',
 'MELANOMA',
 'BREAST',
 'BREAST',
 'MELANOMA',
 'MELANOMA',
 'MELANOMA',
 'MELANOMA',
 'MELANOMA',
 'MELANOMA',
 'BREAST',
```

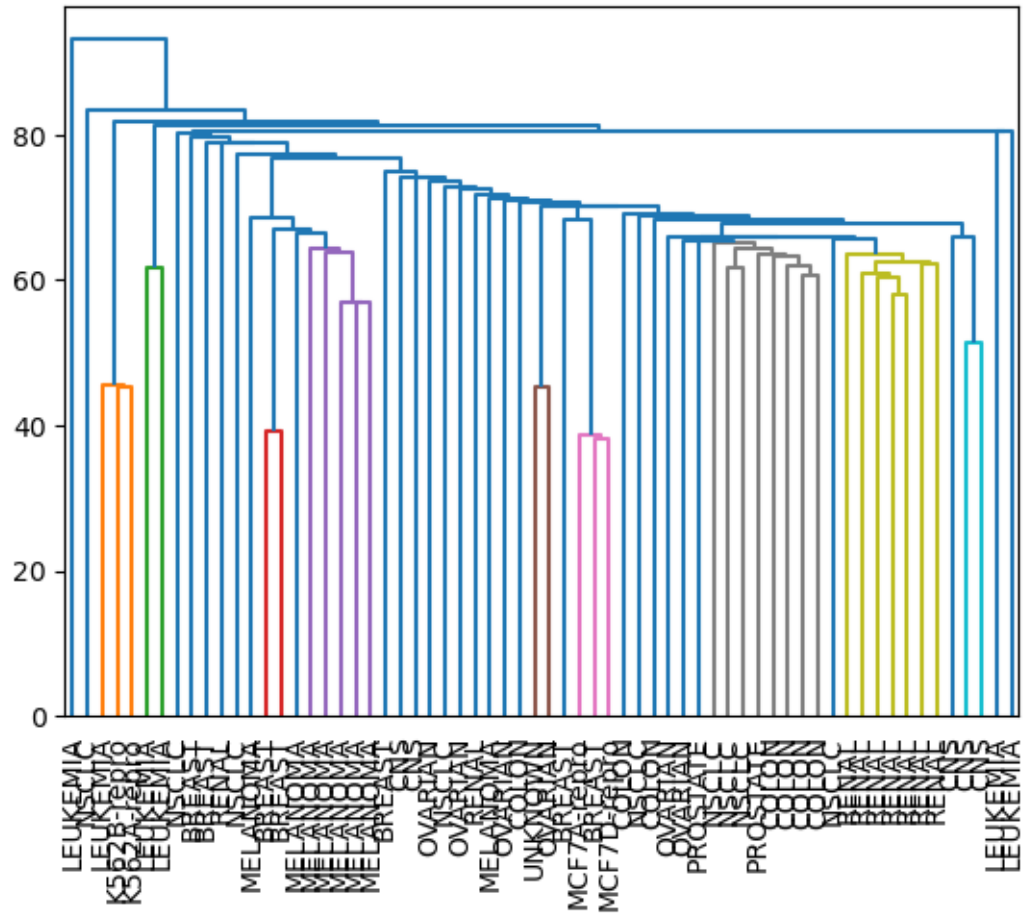
```
 'CNS',
 'CNS',
 'OVARIAN',
 'NSCLC',
 'OVARIAN',
 'RENAL',
 'MELANOMA',
 'OVARIAN',
 'COLON',
 'UNKNOWN',
 'OVARIAN',
 'BREAST',
 'MCF7A-repro',
 'BREAST',
 'MCF7D-repro',
 'COLON',
 'NSCLC',
 'COLON',
 'OVARIAN',
 'OVARIAN',
 'PROSTATE',
 'NSCLC',
 'NSCLC',
 'NSCLC',
 'PROSTATE',
 'COLON',
 'COLON',
 'COLON',
 'COLON',
 'NSCLC',
 'RENAL',
 'RENAL',
 'RENAL',
 'RENAL',
 'RENAL',
 'RENAL',
 'RENAL',
 'RENAL',
 'CNS',
 'CNS',
 'CNS',
 'LEUKEMIA',
 'LEUKEMIA'],
'leaves': [40,
 9,
 36,
 34,
 35,
```

```
38,
39,
54,
17,
4,
19,
18,
55,
56,
57,
62,
58,
63,
59,
60,
61,
7,
6,
5,
24,
53,
25,
3,
22,
27,
46,
20,
21,
51,
48,
49,
50,
47,
52,
42,
26,
28,
23,
32,
30,
31,
29,
44,
45,
41,
43,
8,
```

```
 10,
 16,
 13,
 11,
 12,
 14,
 15,
 2,
 0,
 1,
 33,
 37],
'color_list': ['C1',
 'C1',
 'C2',
 'C3',
 'C4',
 'C4',
 'C4',
 'C4',
 'C0',
 'C0',
 'C0',
 'C5',
 'C6',
 'C6',
 'C0',
 'C7',
 'C7',
 'C7',
 'C7',
 'C7',
 'C7',
 'C7',
 'C0',
 'C0',
 'C8',
 'C8',
 'C8',
 'C8',
 'C8',
 'C8',
 'C0',
 'C0',
 'C0',
 'C9',
 'C0',
```

```
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0'],
'leaves_color_list': ['C0',
 'C0',
 'C1',
 'C1',
 'C1',
 'C2',
 'C2',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C0',
 'C3',
 'C3',
 'C0',
 'C4',
 'C4',
 'C4',
```

```
'C4',
'C4',
'C0',
'C0',
'C0',
'C0',
'C0',
'C0',
'C0',
'C0',
'C0',
'C0',
'C5',
'C5',
'C0',
'C6',
'C6',
'C6',
'C0',
'C0',
'C0',
'C0',
'C0',
'C0',
'C0',
'C7',
'C7',
'C7',
'C7',
'C7',
'C7',
'C7',
'C7',
'C0',
'C8',
'C8',
'C8',
'C8',
'C8',
'C8',
'C8',
'C8',
'C0',
'C9',
'C9',
'C0',
'C0']}
```



Now, is a good point to switch driver and navigator

7 K-means Clustering: Gene Expression Data

Now, let's perform k-means clustering on the gene expression data.

7.0.1 Exercise 11 (CORE)

Perform K-means clustering with the same number of clusters that you selected for hierarchical clustering. Are the results similar?

[72]:
```
x_means = KMeans(n_clusters=10).fit_transform(X)
```

7.0.2 Exercise 12 (EXTRA)

Plot the two clustering solutions along with a plot of the data colored by the cancer types in the space spanned by the first two principal components.

[ ]:
```

```

8 Competing the Worksheet

At this point you have hopefully been able to complete all the CORE exercises and attempted the EXTRA ones. Now is a good time to check the reproducibility of this document by restarting the notebook's kernel and rerunning all cells in order.

Before generating the PDF, please go to Edit -> Edit Notebook Metadata and change 'Student 1' and 'Student 2' in the **name** attribute to include your name.

Once that is done and you are happy with everything, you can then run the following cell to generate your PDF.

[ ]:
```
!jupyter nbconvert --to pdf mlp_week03.ipynb
```