Project 4 Report

Jayam Sutariya

CS458

P4-1. Hierarchical Clustering Dendrogram

(a) Randomly generate the following data points:

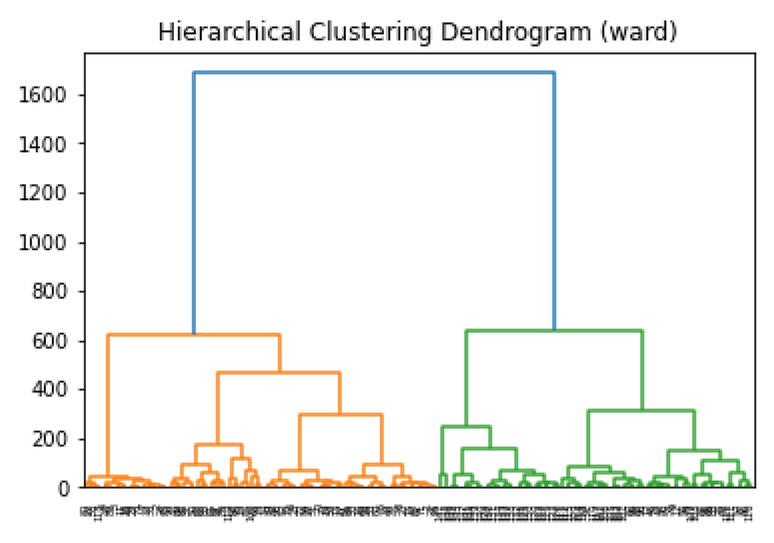
In [14]:

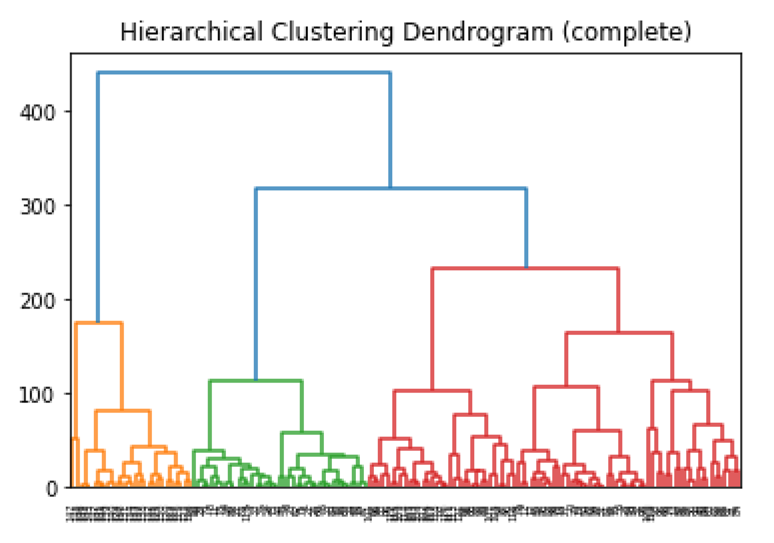
```
# Codes for P4-1(a)
import numpy as np
np.random.seed(0)
X1 = np.random.randn(50,2)+[2,2]
X2 = np.random.randn(50,2)+[6,10]
X3 = np.random.randn(50,2)+[10,2]
X = np.concatenate((X1,X2,X3))
```

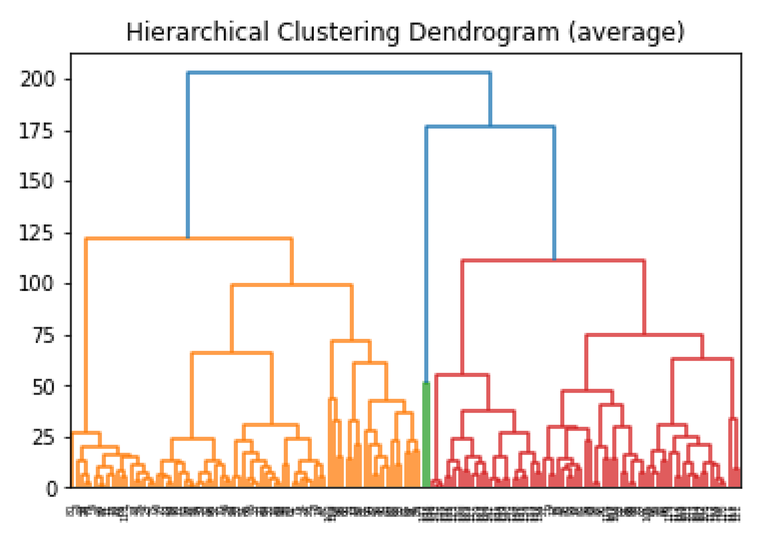
(b) Use sklearn.cluster.AgglomerativeClustering to cluster the points generated in (a). Plot your Dendrogram using different linkage{“ward”, “complete”, “average”, “single”}.

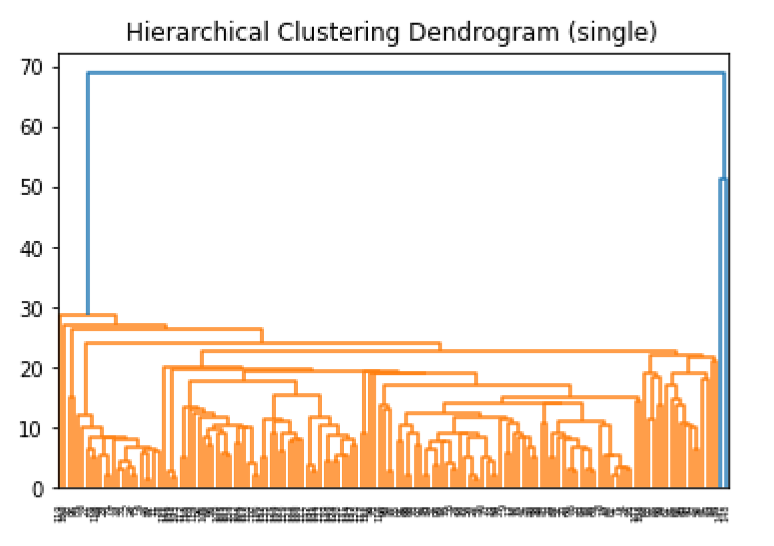
In [29]:

```
# Codes for P4-1(b)
from matplotlib import pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage

def plot_dendrogram(model, method, **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)

    link = linkage(linkage_matrix, method)
    # Plot the corresponding dendrogram
    dendrogram(link, **kwargs)

model = AgglomerativeClustering(distance_threshold=0, n_clusters=None)
model = model.fit(X)

type = ["ward", "complete", "average", "single"]
for t in type:
    title = "Hierarchical Clustering Dendrogram (" + t + ")"
    plt.title(title)
    plot_dendrogram(model, t)
    #plt.xlabel("Number of points in node (or index of point if no parenthesis).")
    plt.show()
```









P4-2. Clustering structured dataset

(a) Generate a swiss roll dataset:

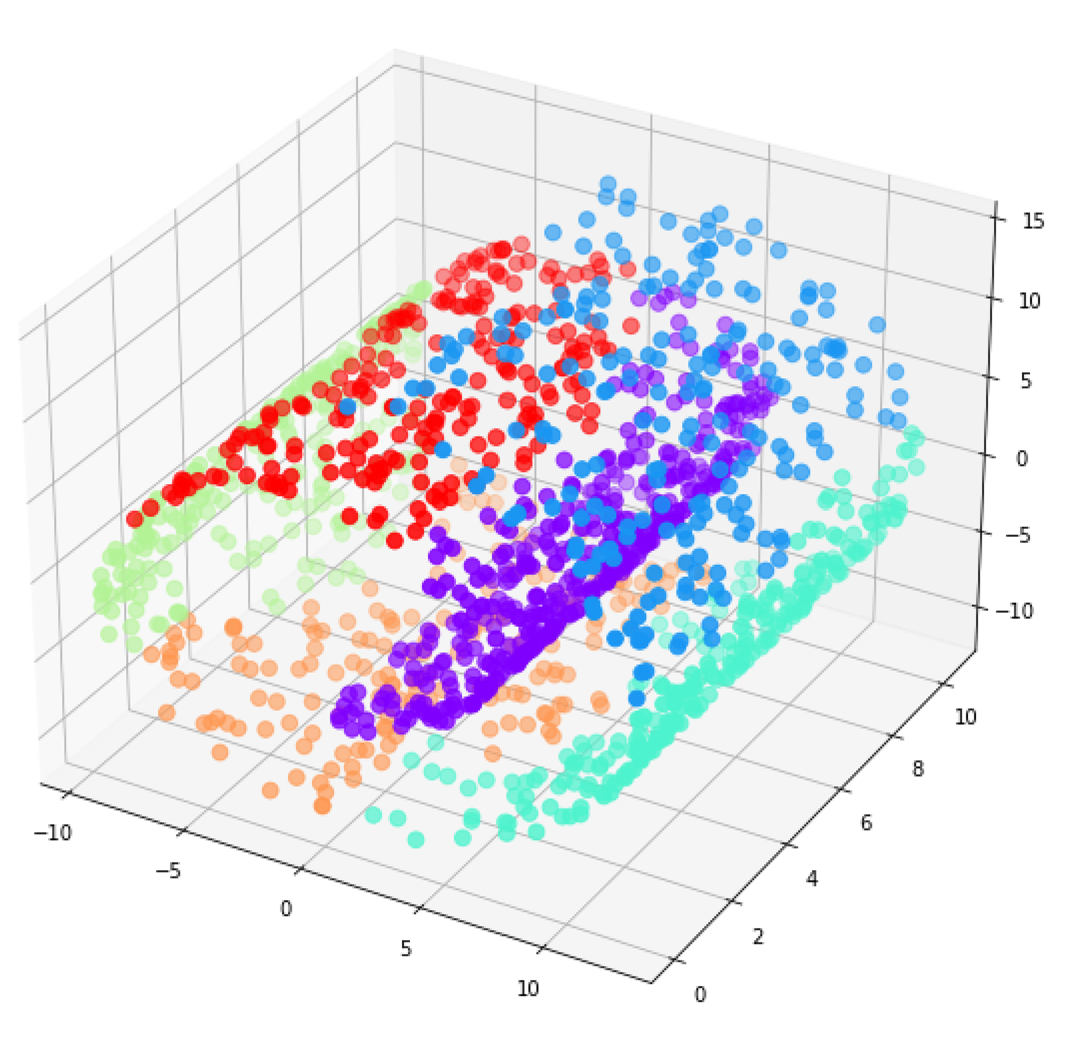
In [70]:

```
# Codes for P4-2(a)
from sklearn.datasets import make_swiss_roll
# Generate data (swiss roll dataset)
n_samples = 1500
noise = 0.05
X, _ = make_swiss_roll(n_samples, noise=noise)
# Make it thinner
X[:, 1] *= .5
```

(b) Use sklearn.cluster.AgglomerativeClustering to cluster the points generated in (a), where you set the parameters as n_clusters=6, connectivity=connectivity, linkage='ward'.

In [66]:

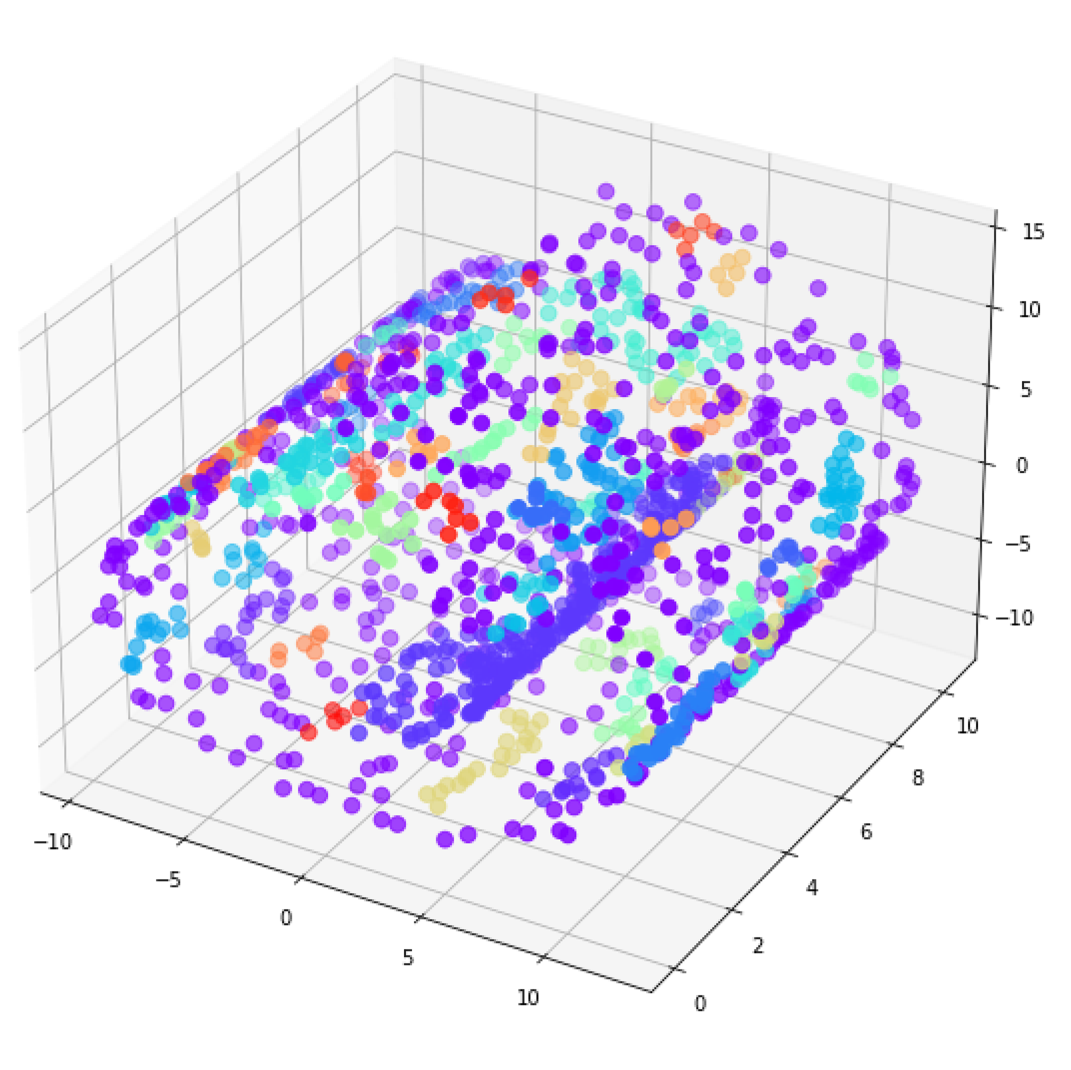
```
# Codes for P4-2(b)
from matplotlib import pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from sklearn.neighbors import kneighbors_graph

connectivity = kneighbors_graph(X, n_neighbors=10, include_self=False)

model = AgglomerativeClustering(n_clusters=6, connectivity=connectivity, linkage='ward')
model = model.fit(X)

fig = plt.figure()
fig.set_size_inches(10, 10)
ax = fig.add_subplot(projection='3d')
x = X[:, 0]
y = X[:, 1]
z = X[:, 2]

ax.scatter(x,y,z, c=model.labels_, s=60, cmap="rainbow")
plt.show()
```



(c) Use sklearn.cluster.DBSCAN to cluster the points generated in (a). Plot the clustered data in a 3D figure and use different colors different clusters in your figure.

In [97]:

```
# Codes for P4-2(c)
from matplotlib import pyplot as plt
from sklearn.cluster import DBSCAN

model = DBSCAN(eps=.8, min_samples=5)
model = model.fit(X)

labels = model.labels_
n_clusters = len(set(model.labels_)) - (1 if -1 in labels else 0)
#print(n_clusters)

fig = plt.figure()
fig.set_size_inches(10, 10)
ax = fig.add_subplot(projection='3d')
x = X[:, 0]
y = X[:, 1]
z = X[:, 2]

ax.scatter(x,y,z, c=model.labels_, s=60, cmap="rainbow")
plt.show()
```



After several adjustments of the parameters of the DBSCAN clustering algorithm, the desired results were not achieved. Either the clusters were all one class or there were too many classes. Perhaps this is because of some connectivity issue. Or perhaps we must flatten the image to 2D using PCA and then perform DBSCAN. Nonetheless, it seems very difficult to cluster the make_swiss_rolls dataset using DBSCAN optimally.

On the other hand, the AgglomerativeClustering algorithm with ward linkage method and connectivity seems to optimally cluster the dataset and performs much better.

P4-3. Clustering the handwritten digits data

(a) Use the hand-written digits dataset embedded in scikit-learn and use the following methods to cluster the data.

In [135…

```
# Codes for P4-3(a)
from sklearn import datasets, metrics
from sklearn.cluster import DBSCAN
from sklearn.cluster import KMeans

digits = datasets.load_digits()

n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))

kmeans = KMeans(n_clusters=10)
kmeans = kmeans.fit(data)

db = DBSCAN(eps=.12, min_samples=1)
db = db.fit(data)

accuracy = metrics.accuracy_score(digits.target, kmeans.labels_)
print("KMeans accuracy:", accuracy)

accuracy2 = metrics.accuracy_score(digits.target, db.labels_)
print("DBSCAN accuracy:", accuracy2)
```

```
KMeans accuracy: 0.14913745130773512
DBSCAN accuracy: 0.005564830272676683
```

(b) Evaluate these methods based on the labels of the data and discuss which method gives you the best results in terms of accuracy.

Unless there are no ideal parameters that improve the accuracy of these algorithms, I was not able to find the optimal parameters. In general, however, the KMeans performed slightly better than the DBSCAN.