Clustering

The clustering technique can be very handy when it comes to unlabeled data. Since most of the data in the real-world is unlabeled and annotating the data has higher costs, clustering techniques can be used to label unlabeled data.

Hierarchical Clustering

https://stackabuse.com/hierarchical-clustering-with-python-and-scikit-learn/

Two types of hierarchical clustering:

Agglomerative

Divisive

In the former, data points are clustered using a bottom-up approach starting with individual data points, while in the latter top-down approach is followed where all the data points are treated as one big cluster and the clustering process involves dividing the one big cluster into several small clusters.

Consider collection of data points represented by a numpy array

```
import numpy as np

X = np.array([[5,3],
    [10,15],
    [15,12],
    [24,10],
    [30,30],
    [85,70],
    [71,80],
    [60,78],
    [70,55],
    [80,91],])
```

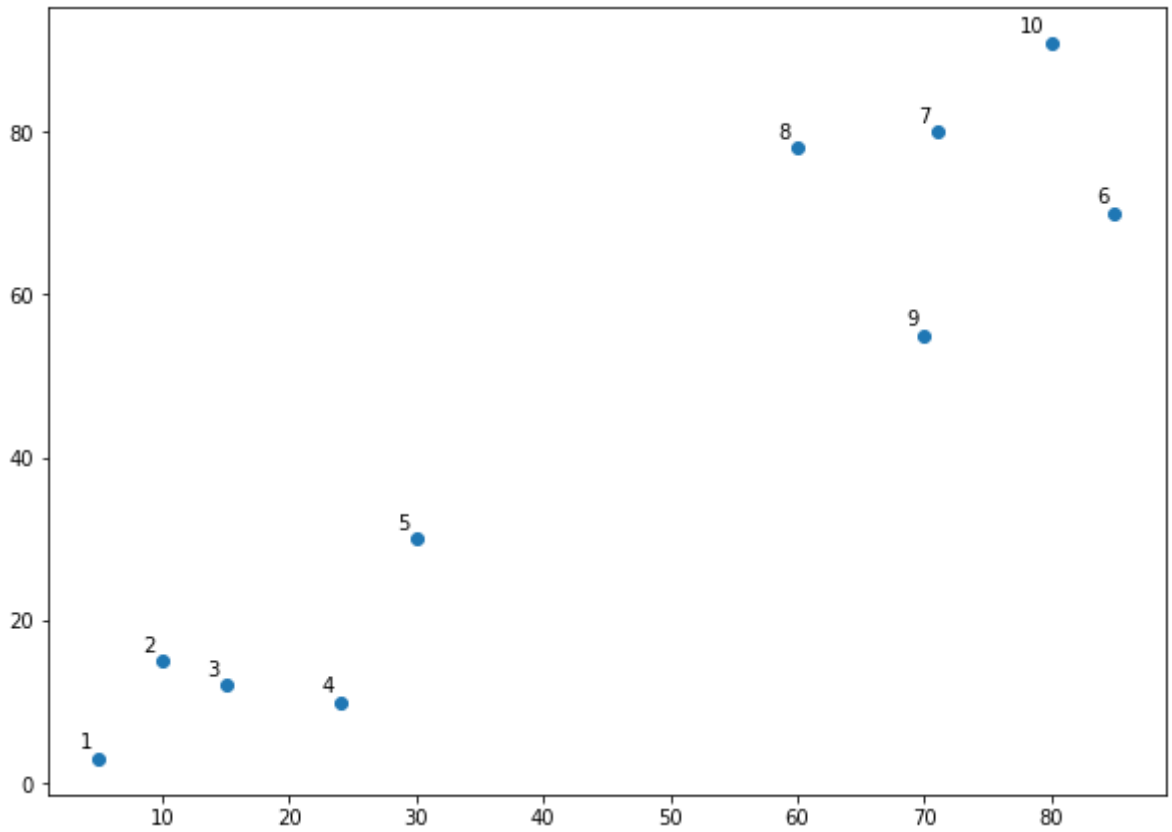
** Plot the above data points.**

```
import matplotlib.pyplot as plt

labels = range(1, 11)
plt.figure(figsize=(10, 7))
```

```
plt.subplots_adjust(bottom=0.1)
plt.scatter(X[:,0],X[:,1], label='True Position')

for label, x, y in zip(labels, X[:, 0], X[:, 1]):
    plt.annotate(
        label,
        xy=(x, y), xytext=(-3, 3),
        textcoords='offset points', ha='right', va='bottom')
plt.show()
```



Use of dendrograms in hierarchical clustering, let's draw the **dendrograms** for our data points.

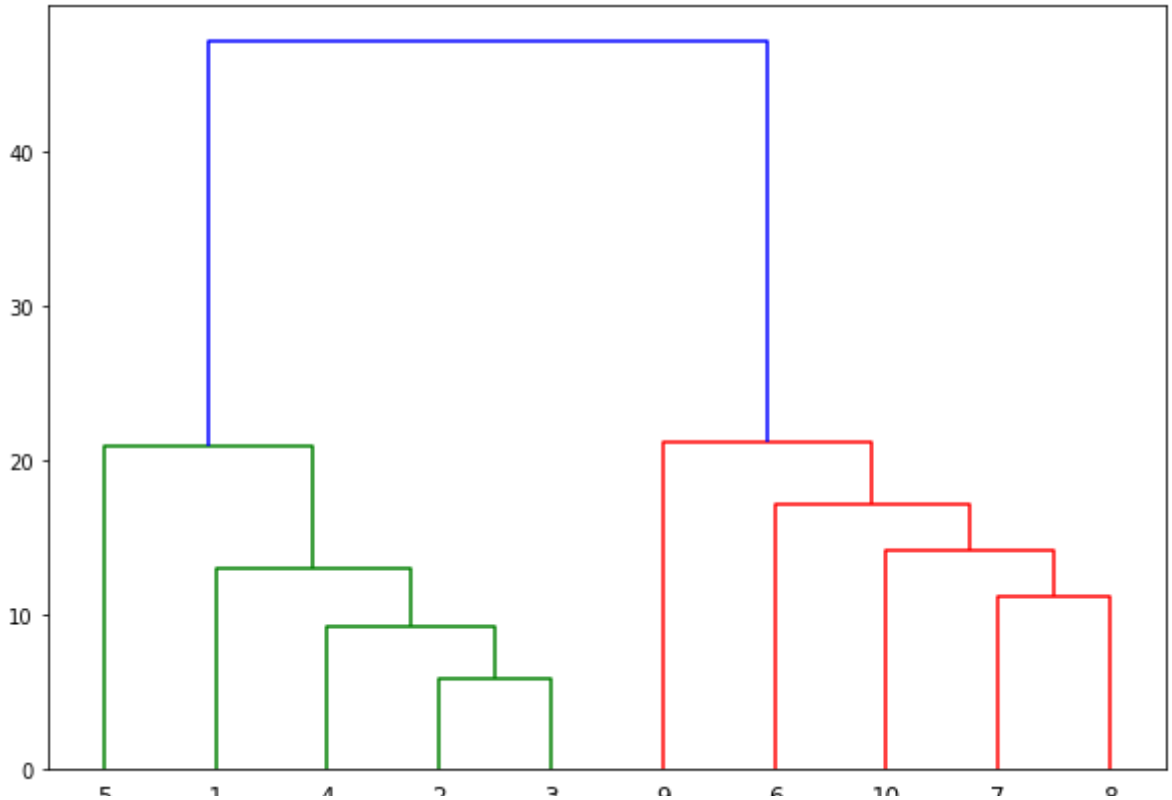
We will use the **scipy library** for that purpose.

```
from scipy.cluster.hierarchy import dendrogram, linkage
from matplotlib import pyplot as plt

linked = linkage(X, 'single')

labelList = range(1, 11)

plt.figure(figsize=(10, 7))
dendrogram(linked,
            orientation='top',
            labels=labelList,
            distance_sort='descending',
            show_leaf_counts=True)
plt.show()
```



Using the AgglomerativeClustering from sklearn

Agglomerative Clustering

Recursively merges the pair of clusters that minimally increases a given linkage distance.

Read more in the User Guide.

Parameters n_clustersint or None, default=2 The number of clusters to find. It must be None if distance_threshold is not None.

affinitystr or callable, default='euclidean' Metric used to compute the linkage. Can be "euclidean", "l1", "l2", "manhattan", "cosine", or "precomputed". If linkage is "ward", only "euclidean" is accepted. If "precomputed", a distance matrix (instead of a similarity matrix) is needed as input for the fit method.

memorystr or object with the joblib.Memory interface, default=None Used to cache the output of the computation of the tree. By default, no caching is done. If a string is given, it is the path to the caching directory.

connectivityarray-like or callable, default=None Connectivity matrix. Defines for each sample the neighboring samples following a given structure of the data. This can be a connectivity matrix itself or a callable that transforms the data into a connectivity matrix, such as derived from kneighbors_graph. Default is None, i.e, the hierarchical clustering algorithm is unstructured.

compute_full_tree'auto' or bool, default='auto' Stop early the construction of the tree at n_clusters. This is useful to decrease computation time if the number of clusters is not small compared to the number of samples. This option is useful only when specifying a connectivity matrix. Note also that when varying the number of clusters and using caching, it may be

advantageous to compute the full tree. It must be True if distance_threshold is not None. By default compute_full_tree is “auto”, which is equivalent to True when distance_threshold is not None or that n_clusters is inferior to the maximum between 100 or 0.02 * n_samples. Otherwise, “auto” is equivalent to False.

*linkage{“ward”, “complete”, “average”, “single”}, *default=”ward” Which linkage criterion to use. The linkage criterion determines which distance to use between sets of observation. The algorithm will merge the pairs of cluster that minimize this criterion.

ward minimizes the variance of the clusters being merged.

average uses the average of the distances of each observation of the two sets.

complete or maximum linkage uses the maximum distances between all observations of the two sets.

single uses the minimum of the distances between all observations of the two sets.

New in version 0.20: Added the ‘single’ option

distance_thresholdfloat, default=None The linkage distance threshold above which, clusters will not be merged. If not None, n_clusters must be None and compute_full_tree must be True.

New in version 0.21.

Attributes n_clusters_int The number of clusters found by the algorithm. If distance_threshold=None, it will be equal to the given n_clusters.

labels_ndarray of shape (n_samples) cluster labels for each point

n_leaves_int Number of leaves in the hierarchical tree.

n_connected_components_int The estimated number of connected components in the graph.

New in version 0.21: n_connected_components_ was added to replace n_components_.

children_array-like of shape (n_samples-1, 2) The children of each non-leaf node. Values less than n_samples correspond to leaves of the tree which are the original samples. A node i greater than or equal to n_samples is a non-leaf node and has children children_[i - n_samples]. Alternatively at the i-th iteration, children[i][0] and children[i][1] are merged to form node n_samples + iward minimizes the variance of the clusters being merged.

average uses the average of the distances of each observation of the two sets.

complete or maximum linkage uses the maximum distances between all observations of the two sets.

single uses the minimum of the distances between all observations of the two sets.

```
1 from sklearn.cluster import AgglomerativeClustering
2
3 cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage='ward')
4 cluster.fit_predict(X)
```

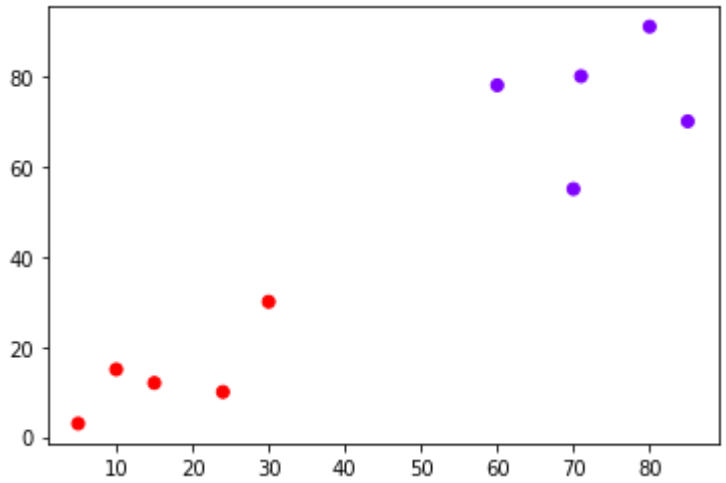
```
array([1, 1, 1, 1, 1, 0, 0, 0, 0, 0])
```

```
print(cluster.labels_)
```

```
[1 1 1 1 1 0 0 0 0 0]
```

```
plt.scatter(X[:,0],X[:,1], c=cluster.labels_, cmap='rainbow')
```

```
<matplotlib.collections.PathCollection at 0x7f2a1e2cdd68>
```



Example 2- Using shopping.csv

Importing Libraries

```
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
import numpy as np
```

Importing data using pandas

```
customer_data = pd.read_csv('shopping.csv')
```

Examining the data

```
customer_data.describe()
```

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000

```
1 customer_data.head()
```

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

```
1 data = customer_data.iloc[:, 3:5].values
2 data
```

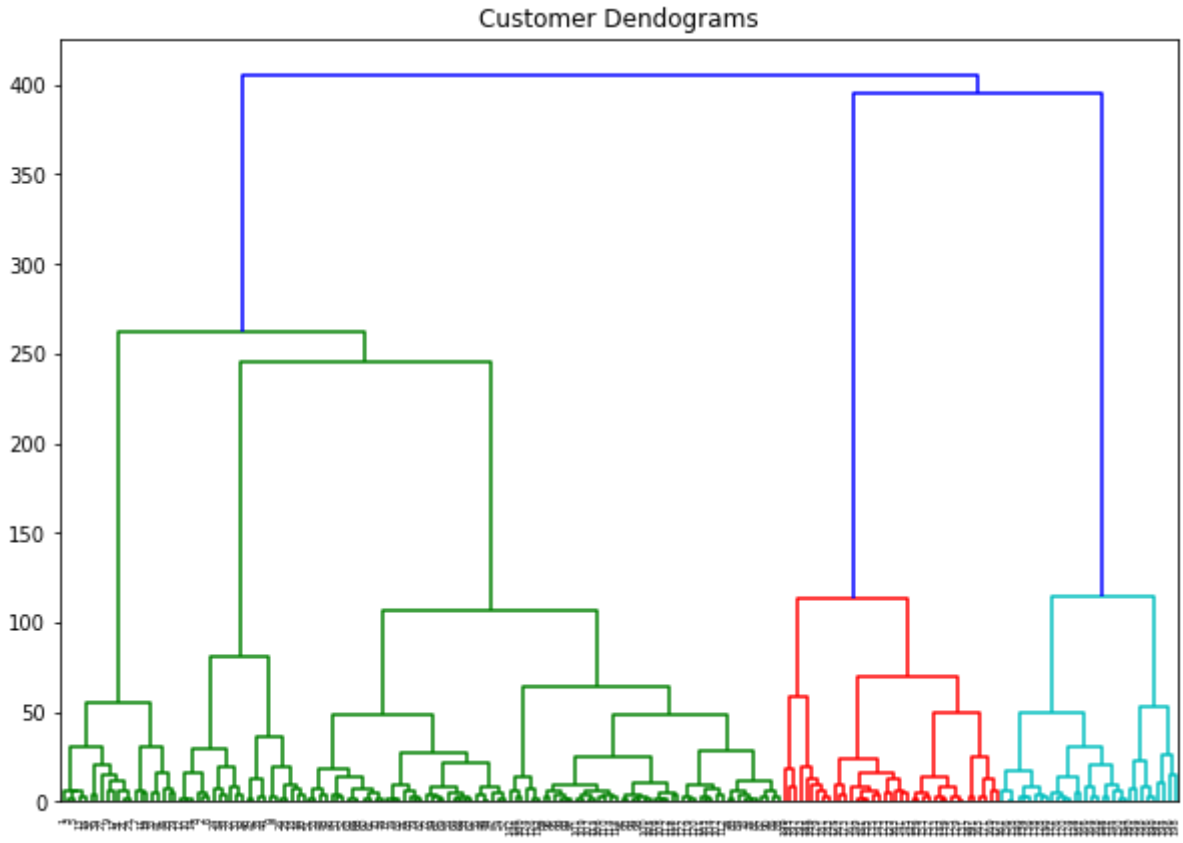
```
array([[ 15,  39],
       [ 15,  81],
       [ 16,   6],
       [ 16,  77],
       [ 17,  40],
       [ 17,  76],
       [ 18,   6],
       [ 18,  94],
       [ 19,   3],
       [ 19,  72],
       [ 19,  14],
       [ 19,  99],
       [ 20,  15],
       [ 20,  77],
       [ 20,  13],
       [ 20,  79],
       [ 21,  35],
       [ 21,  66],
       [ 23,  29],
       [ 23,  98],
       [ 24,  35],
       [ 24,  73],
       [ 25,   5],
       [ 25,  73],
       [ 28,  14],
       [ 28,  82],
       [ 28,  32],
       [ 28,  61],
       [ 29,  31],
       [ 29,  87],
       [ 30,   4],
       [ 30,  73],
       [ 33,   4],
       [ 33,  92],
       [ 33,  14],
       [ 33,  81],
       [ 34,  17],
       [ 34,  73],
       [ 37,  26],
       [ 37,  75],
       [ 38,  35],
       [ 38,  92],
       [ 39,  36],
       [ 39,  61],
       [ 39,  28],
       [ 39,  65],
       [ 40,  55],
       [ 40,  47],
       [ 40,  42],
       [ 40,  42],
       [ 42,  52],
       [ 42,  60],
       [ 43,  54],
       [ 43,  60],
       [ 43,  45],
       [ 43,  41],
       [ 44,  50],
       [ 44,  46],
       [ 46,  51],
       [ 46,  46],
       [ 46,  56],
```

```
       [ 46,  55],
       [ 47,  52],
       [ 47,  59],
       [ 48,  51],
       [ 48,  59],
       [ 48,  50],
       [ 48,  48],
       [ 48,  59],
       [ 48,  47],
       [ 49,  55],
       [ 49,  42],
       [ 50,  49],
       [ 50,  56],
       [ 54,  47],
       [ 54,  54],
       [ 54,  53],
       [ 54,  48],
       [ 54,  52],
       [ 54,  42],
       [ 54,  51],
       [ 54,  55],
       [ 54,  41],
       [ 54,  44],
       [ 54,  57],
       [ 54,  46],
       [ 57,  58],
       [ 57,  55],
       [ 58,  60],
       [ 58,  46],
       [ 59,  55],
       [ 59,  41],
       [ 60,  49],
       [ 60,  40],
       [ 60,  42],
       [ 60,  52],
       [ 60,  47],
       [ 60,  50],
       [ 61,  42],
       [ 61,  49],
       [ 62,  41],
       [ 62,  48],
       [ 62,  59],
       [ 62,  55],
       [ 62,  56],
       [ 62,  42],
       [ 63,  50],
       [ 63,  46],
       [ 63,  43],
       [ 63,  48],
       [ 63,  52],
       [ 63,  54],
       [ 64,  42],
       [ 64,  46],
       [ 65,  48],
       [ 65,  50],
       [ 65,  43],
       [ 65,  59],
       [ 67,  43],
       [ 67,  57],
       [ 67,  56],
       [ 67,  40],
       [ 69,  58],
```

```
[ 69,  91],
[ 70,  29],
[ 70,  77],
[ 71,  35],
[ 71,  95],
[ 71,  11],
[ 71,  75],
[ 71,   9],
[ 71,  75],
[ 72,  34],
[ 72,  71],
[ 73,   5],
[ 73,  88],
[ 73,   7],
[ 73,  73],
[ 74,  10],
[ 74,  72],
[ 75,   5],
[ 75,  93],
[ 76,  40],
[ 76,  87],
[ 77,  12],
[ 77,  97],
[ 77,  36],
[ 77,  74],
[ 78,  22],
[ 78,  90],
[ 78,  17],
[ 78,  88],
[ 78,  20],
[ 78,  76],
[ 78,  16],
[ 78,  89],
[ 78,   1],
[ 78,  78],
[ 78,   1],
[ 78,  73],
[ 79,  35],
[ 79,  83],
[ 81,   5],
[ 81,  93],
[ 85,  26],
[ 85,  75],
[ 86,  20],
[ 86,  95],
[ 87,  27],
[ 87,  63],
[ 87,  13],
[ 87,  75],
[ 87,  10],
[ 87,  92],
[ 88,  13],
[ 88,  86],
[ 88,  15],
[ 88,  69],
[ 93,  14],
[ 93,  90],
[ 97,  32],
[ 97,  86],
```

```
1 import scipy.cluster.hierarchy as shc
```

```
2
3 plt.figure(figsize=(10, 7))
4 plt.title("Customer Dendograms")
5 dend = shc.dendrogram(shc.linkage(data, method='ward'))
```




```
1 from sklearn.cluster import AgglomerativeClustering
2
3 cluster = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
4 cluster.fit_predict(data)
```

```
array([4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3,
       4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 1,
       4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 0, 2, 0, 2,
       1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2])
```

```
1 plt.figure(figsize=(10, 7))
2 plt.scatter(data[:,0], data[:,1], c=cluster.labels_, cmap='rainbow')
```