```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Basics of hierarchical clustering

- Creating a distance matrix using linkage
  - `method`: how to calculate the proximity of clusters
  - `metric`: distance metric
  - `optimal_ordering`: order data points
- Type of Methods
  - single: based on two closest objects
  - complete: based on two farthest objects
  - average: based on the arithmetic mean of all objects
  - centroids: based on the geometric mean of all objects
  - median: based on the median of all objects
  - ward: based on the sum of squares

```
comic_con = pd.read_csv('./dataset/comic_con.csv', index_col=0)
comic_con.head()
```

	x_coordinate	y_coordinate
0	17	4
1	20	6
2	35	0
3	14	0
4	37	4

```
from scipy.cluster.vq import whiten

comic_con['x_scaled'] = whiten(comic_con['x_coordinate'])
comic_con['y_scaled'] = whiten(comic_con['y_coordinate'])
```

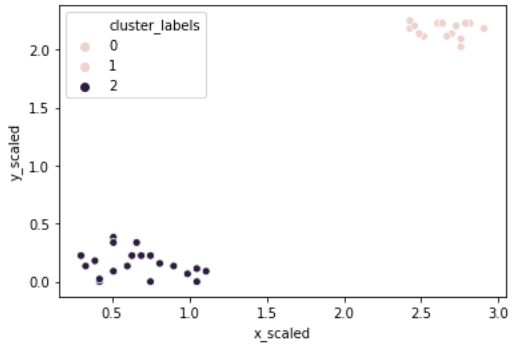
Hierarchical clustering: single method

Let us use the same footfall dataset and check if any changes are seen if we use a different method for clustering.

```
# Use the linkage()
distance_matrix = linkage(comic_con[['x_scaled', 'y_scaled']], method='single', metric='euclidean')

# Assign cluster labels
comic_con['cluster_labels'] = fcluster(distance_matrix, 2, criterion='maxclust')

# Plot clusters
sns.scatterplot(x='x_scaled', y='y_scaled', hue='cluster_labels', data=comic_con);
```



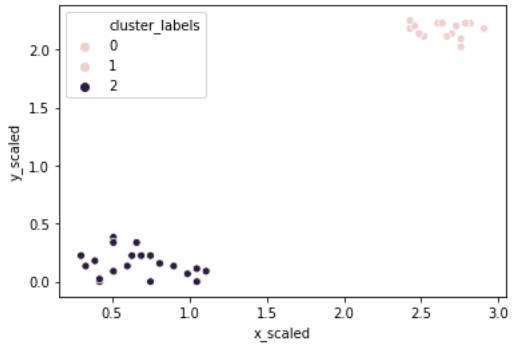
Hierarchical clustering: complete method

For the third and final time, let us use the same footfall dataset and check if any changes are seen if we use a different method for clustering.

```
# Use the linkage()
distance_matrix = linkage(comic_con[['x_scaled', 'y_scaled']], method='complete', metric='euclidean')

# Assign cluster labels
comic_con['cluster_labels'] = fcluster(distance_matrix, 2, criterion='maxclust')

# Plot clusters
sns.scatterplot(x='x_scaled', y='y_scaled', hue='cluster_labels', data=comic_con);
```



Visualize clusters

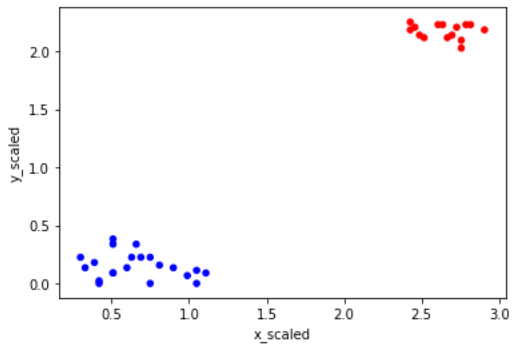
- Why visualize clusters?
  - Try to make sense of the clusters formed
  - An additional step in validation of clusters
  - Spot trends in data

Visualize clusters with matplotlib

We have discussed that visualizations are necessary to assess the clusters that are formed and spot trends in your data. Let us now focus on visualizing the footfall dataset from Comic-Con using the matplotlib module.

```
# Define a colors dictionary for clusters
colors = {1:'red', 2:'blue'}

# Plot the scatter plot
comic_con.plot.scatter(x='x_scaled', y='y_scaled', c=comic_con['cluster_labels'].apply(lambda x: colors[x]));
```

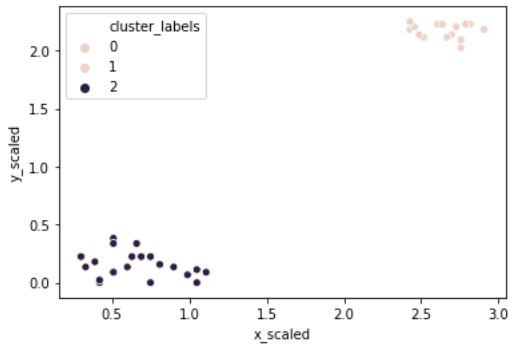


Visualize clusters with seaborn

Let us now visualize the footfall dataset from Comic Con using the seaborn module. Visualizing clusters using seaborn is easier with the inbuild `hue` function for cluster labels.

```
# Plot a scatter plot using seaborn
sns.scatterplot(x='x_scaled', y='y_scaled', hue='cluster_labels', data=comic_con)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x27d3a82d1c8>
```



How many clusters?

- Introduction to dendrograms
  - Strategy till now - decide clusters on visual inspection
  - Dendrograms help in showing progressions as clusters are merged
  - A dendrogram is a branching diagram that demonstrates how each cluster is composed by branching out into its child nodes

Create a dendrogram

Dendrograms are branching diagrams that show the merging of clusters as we move through the distance matrix. Let us use the Comic Con footfall data to create a dendrogram.

```
from scipy.cluster.hierarchy import dendrogram

# Create a dendrogram
dn = dendrogram(distance_matrix)
```