**Hierarchical Clustering - Agglomerative**

We will be looking at a clustering technique, which is **Agglomerative Hierarchical Clustering**. Remember that agglomerative is the bottom up approach.  
  
In this lab, we will be looking at Agglomerative clustering, which is more popular than Divisive clustering.  
  
We will also be using Complete Linkage as the Linkage Criteria.  
***NOTE: You can also try using Average Linkage wherever Complete Linkage would be used to see the difference!***

import numpy as np

import pandas as pd

from scipy import ndimage

from scipy.cluster import hierarchy

from scipy.spatial import distance\_matrix

from matplotlib import pyplot as plt

from sklearn import manifold, datasets

from sklearn.cluster import AgglomerativeClustering

from sklearn.datasets.samples\_generator import make\_blobs

%matplotlib inline

### Generating Random Data

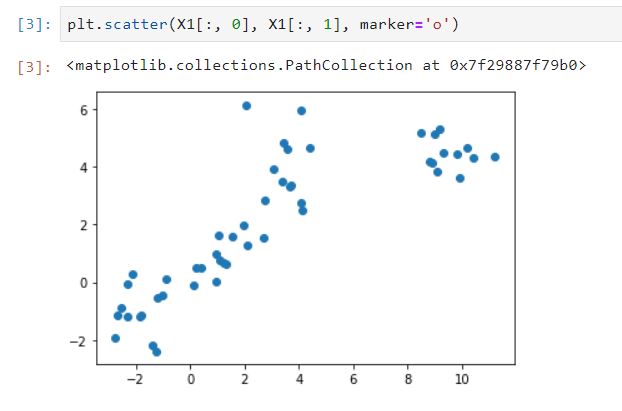
We will be generating a set of data using the **make\_blobs** class.  
  
Input these parameters into make\_blobs:

* **n\_samples**: The total number of points equally divided among clusters.
  + Choose a number from 10-1500
* **centers**: The number of centers to generate, or the fixed center locations.
  + Choose arrays of x,y coordinates for generating the centers. Have 1-10 centers (ex. centers=[[1,1], [2,5]])
* **cluster\_std**: The standard deviation of the clusters. The larger the number, the further apart the clusters
  + Choose a number between 0.5-1.5

X1, y1 = make\_blobs(n\_samples=50, centers=[[4,4], [-2, -1], [1, 1], [10,4]], cluster\_std=0.9)

#plot the random data

plt.scatter(X1[:0],X1[:1],marker=’o’)



### Agglomerative Clustering

We will start by clustering the random data points we just created.

The **Agglomerative Clustering**class will require two inputs:

* **n\_clusters**: The number of clusters to form as well as the number of centroids to generate.
  + Value will be: 4
* **linkage**: 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.
  + Value will be: 'complete'
  + **Note**: It is recommended you try everything with 'average' as well

agglom=AgglomerativeClustering(n\_clusters=4,linkage=’average’)

agglom.fit(X1,y1)

**Plotting**

# Create a figure of size 6 inches by 4 inches.

plt.figure(figsize=(6,4))

# These two lines of code are used to scale the data points down,

# Or else the data points will be scattered very far apart.

# Create a minimum and maximum range of X1.

x\_min, x\_max = np.min(X1, axis=0), np.max(X1, axis=0)

# Get the average distance for X1.

X1 = (X1 - x\_min) / (x\_max - x\_min)

# This loop displays all of the datapoints.

for i in range(X1.shape[0]):

# Replace the data points with their respective cluster value

# (ex. 0) and is color coded with a colormap (plt.cm.spectral)

plt.text(X1[i, 0], X1[i, 1], str(y1[i]),

color=plt.cm.nipy\_spectral(agglom.labels\_[i] / 10.),

fontdict={'weight': 'bold', 'size': 9})

# Remove the x ticks, y ticks, x and y axis

plt.xticks([])

plt.yticks([])

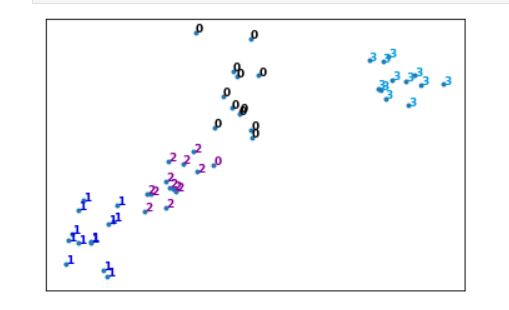
#plt.axis('off')

# Display the plot of the original data before clustering

plt.scatter(X1[:, 0], X1[:, 1], marker='.')

# Display the plot

plt.show()

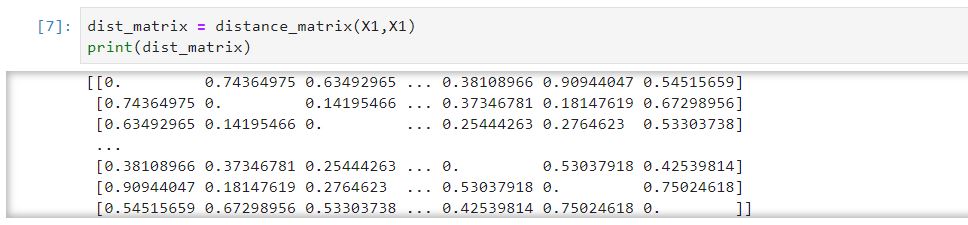


### Dendrogram Associated for the Agglomerative Hierarchical Clustering

Remember that a **distance matrix** contains the **distance from each point to every other point of a dataset**.  
Use the function **distance\_matrix,**which requires **two inputs**. Use the Feature Matrix, **X2**as both inputs and save the distance matrix to a variable called **dist\_matrix**  
  
Remember that the distance values are symmetric, with a diagonal of 0's. This is one way of making sure your matrix is correct.  
(print out dist\_matrix to make sure it's correct)

dist\_matrix=distance\_matrix(X1,X1)

print(dist\_matrix)



Using the **linkage**class from hierarchy, pass in the parameters:

* The distance matrix
* 'complete' for complete linkage

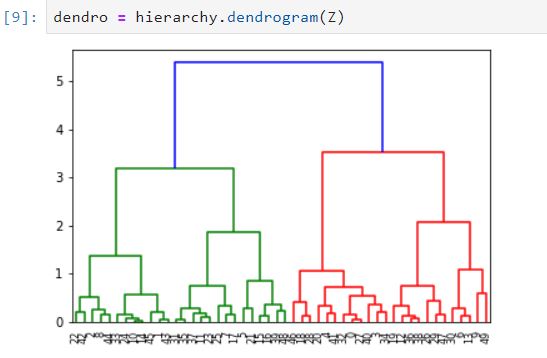
Save the result to a variable called **Z**

Z=hierarchy.linkage(dist\_matrix,’complete’)

A Hierarchical clustering is typically visualized as a dendrogram as shown in the following cell. Each merge is represented by a horizontal line. The y-coordinate of the horizontal line is the similarity of the two clusters that were merged, where cities are viewed as singleton clusters. By moving up from the bottom layer to the top node, a dendrogram allows us to reconstruct the history of merges that resulted in the depicted clustering.

Next, we will save the dendrogram to a variable called **dendro**. In doing this, the dendrogram will also be displayed. Using the **dendrogram**class from hierarchy, pass in the parameter:

* Z

dendro=hierarchy.dendrogram(Z)

# Clustering on Vehicle dataset

Imagine that an automobile manufacturer has developed prototypes for a new vehicle. Before introducing the new model into its range, the manufacturer wants to determine which existing vehicles on the market are most like the prototypes--that is, how vehicles can be grouped, which group is the most similar with the model, and therefore which models they will be competing against.

Our objective here, is to use clustering methods, to find the most distinctive clusters of vehicles. It will summarize the existing vehicles and help manufacturers to make decision about the supply of new models.

### Download data

To download the data, we will use **!wget** to download it from IBM Object Storage.

!wget -O cars\_clus.csv <https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/cars_clus.csv>

## Read data

lets read dataset to see what features the manufacturer has collected about the existing models.

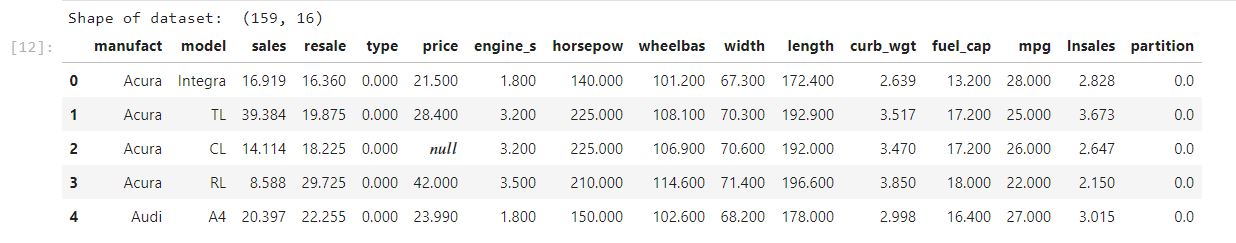
filename = 'cars\_clus.csv'

#Read csv

pdf = pd.read\_csv(filename)

print ("Shape of dataset: ", pdf.shape)

pdf.head(5)



print ("Shape of dataset before cleaning: ", pdf.size)

pdf[[ 'sales', 'resale', 'type', 'price', 'engine\_s',

'horsepow', 'wheelbas', 'width', 'length', 'curb\_wgt', 'fuel\_cap',

'mpg', 'lnsales']] =pdf[[ 'sales', 'resale', 'type', 'price', 'engine\_s',

'horsepow', 'wheelbas', 'width', 'length', 'curb\_wgt', 'fuel\_cap',

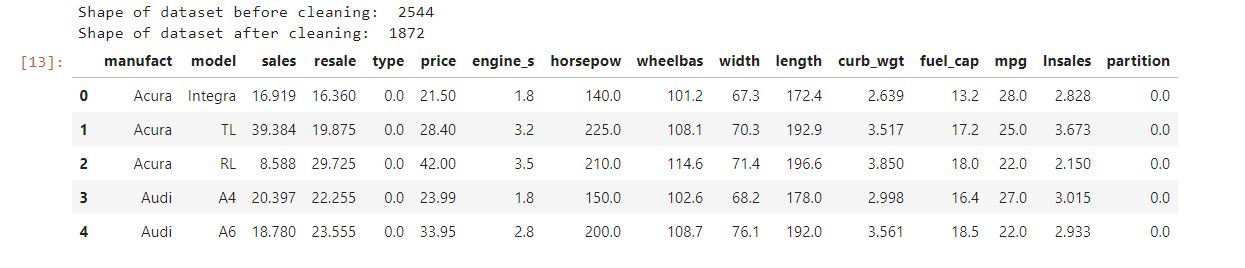
'mpg', 'lnsales']] .apply(pd.to\_numberic,error=’coerce’)

pdf=pdf.dropna()

pdf=pdf.reset\_index(drop=True)

print ("Shape of dataset after cleaning: ", pdf.size)

pdf.head(5)



### Feature selection

Lets select our feature set:

featureset = pdf[['engine\_s', 'horsepow', 'wheelbas', 'width', 'length', 'curb\_wgt', 'fuel\_cap', 'mpg']]

### Normalization

Now we can normalize the feature set. **MinMaxScaler** transforms features by scaling each feature to a given range. It is by default (0, 1). That is, this estimator scales and translates each feature individually such that it is between zero and one.

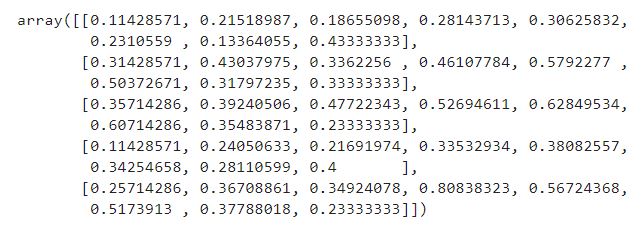
from sklearn.preprocessing import MinMaxScaler

x=featureset.values

minmaxscl=MinMaxScaler()

feature\_mxt= minmaxscl.fit\_transform(X)

feature\_mxt[0:5]



## Clustering using Scipy

In this part we use Scipy package to cluster the dataset: First, we calculate the distance matrix.

import scipy

leng = feature\_mtx.shape[0]

D = scipy.zeros([leng,leng])

for i in range(leng):

for j in range(leng):

D[i,j] = scipy.spatial.distance.euclidean(feature\_mtx[i], feature\_mtx[j])

In agglomerative clustering, at each iteration, the algorithm must update the distance matrix to reflect the distance of the newly formed cluster with the remaining clusters in the forest. The following methods are supported in Scipy for calculating the distance between the newly formed cluster and each: - single - complete - average - weighted - centroid

We use **complete** for our case, but feel free to change it to see how the results change.

import pylab

import scipy.cluster.hierarchy

Z = hierarchy.linkage(D, 'complete')

ssentially, Hierarchical clustering does not require a pre-specified number of clusters. However, in some applications we want a partition of disjoint clusters just as in flat clustering. So you can use a cutting line:

from scipy.cluster.hierarchy import fcluster

max\_d=3

clusters=fcluster(Z,max\_d,criterion=’distance’)

clusters

Also, you can determine the number of clusters directly:

from scipy.cluster.hierarchy import fcluster

k = 5

clusters = fcluster(Z, k, criterion='maxclust')

clusters

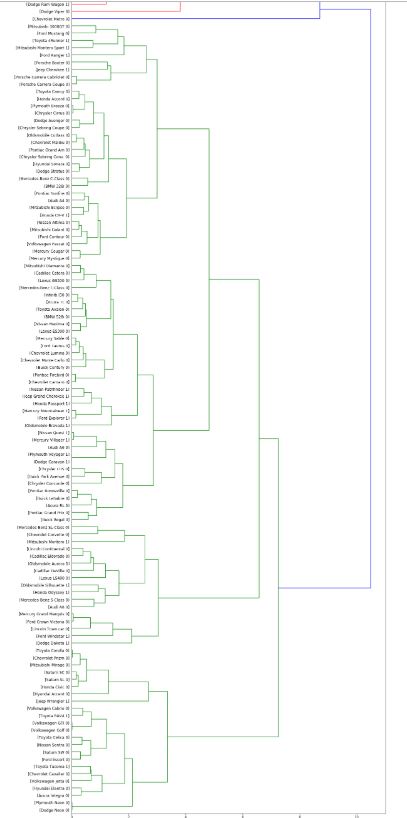
Now we will plot the dendogram

fig = pylab.figure(figsize=(18,50))

def llf(id):

return '[%s %s %s]' % (pdf['manufact'][id], pdf['model'][id], int(float(pdf['type'][id])) )

dendro = hierarchy.dendrogram(Z, leaf\_label\_func=llf, leaf\_rotation=0, leaf\_font\_size =12, orientation = 'right')



## Clustering using scikit-learn

Lets redo it again, but this time using scikit-learn package:

dist\_matrix=distance\_matrix(feature\_mtx,,feature\_mtx)

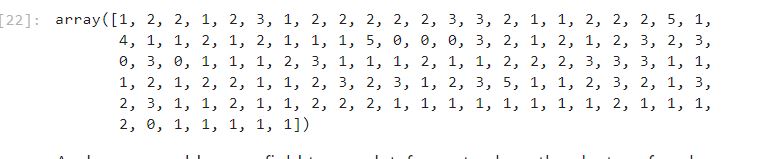
Now, we can use the 'AgglomerativeClustering' function from scikit-learn library to cluster the dataset. The AgglomerativeClustering performs a hierarchical clustering using a bottom up approach. The linkage criteria determines the metric used for the merge strategy:

* Ward minimizes the sum of squared differences within all clusters. It is a variance-minimizing approach and in this sense is similar to the k-means objective function but tackled with an agglomerative hierarchical approach.
* Maximum or complete linkage minimizes the maximum distance between observations of pairs of clusters.
* Average linkage minimizes the average of the distances between all observations of pairs of clusters.

agglom = AgglomerativeClustering(n\_clusters = 6, linkage = 'complete')

agglom.fit(feature\_mtx)

agglom.labels\_



pdf['cluster\_'] = agglom.labels\_

pdf.head()

import matplotlib.cm as cm

n\_clusters = max(agglom.labels\_)+1

colors = cm.rainbow(np.linspace(0, 1, n\_clusters))

cluster\_labels = list(range(0, n\_clusters))

# Create a figure of size 6 inches by 4 inches.

plt.figure(figsize=(16,14))

for color, label in zip(colors, cluster\_labels):

subset = pdf[pdf.cluster\_ == label]

for i in subset.index:

plt.text(subset.horsepow[i], subset.mpg[i],str(subset['model'][i]), rotation=25)

plt.scatter(subset.horsepow, subset.mpg, s= subset.price\*10, c=color, label='cluster'+str(label),alpha=0.5)

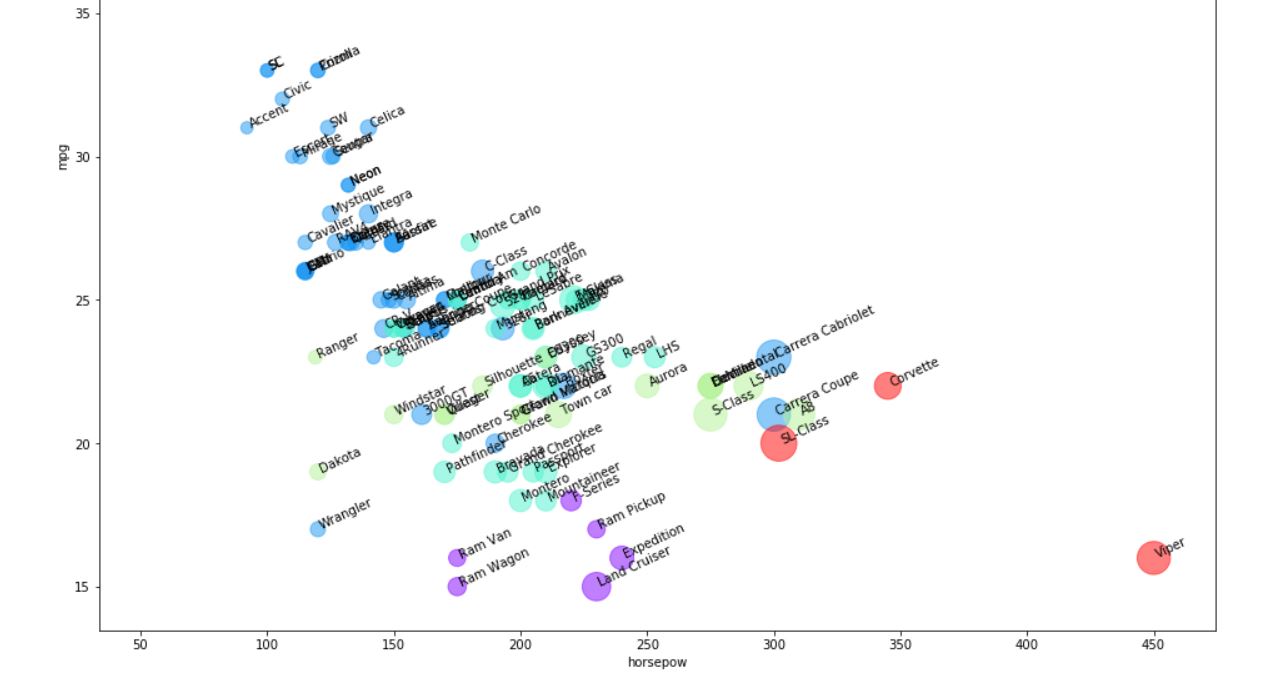
# plt.scatter(subset.horsepow, subset.mpg)

plt.legend()

plt.title('Clusters')

plt.xlabel('horsepow')

plt.ylabel('mpg')



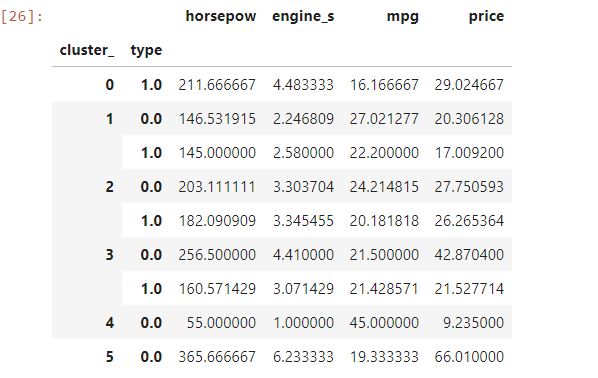
As you can see, we are seeing the distribution of each cluster using the scatter plot, but it is not very clear where is the centroid of each cluster. Moreover, there are 2 types of vehicles in our dataset, "truck" (value of 1 in the type column) and "car" (value of 1 in the type column). So, we use them to distinguish the classes, and summarize the cluster. First we count the number of cases in each group:

pdf.groupby(['cluster\_','type'])['cluster\_'].count()

Now we can look at the characteristics of each cluster:

agg\_cars = pdf.groupby(['cluster\_','type'])['horsepow','engine\_s','mpg','price'].mean()

agg\_cars



It is obvious that we have 3 main clusters with the majority of vehicles in those.

**Cars**:

* Cluster 1: with almost high mpg, and low in horsepower.
* Cluster 2: with good mpg and horsepower, but higher price than average.
* Cluster 3: with low mpg, high horsepower, highest price.

**Trucks**:

* Cluster 1: with almost highest mpg among trucks, and lowest in horsepower and price.
* Cluster 2: with almost low mpg and medium horsepower, but higher price than average.
* Cluster 3: with good mpg and horsepower, low price.

Please notice that we did not use **type** , and **price** of cars in the clustering process, but Hierarchical clustering could forge the clusters and discriminate them with quite high accuracy.

plt.figure(figsize=(16,10))

for color, label in zip(colors, cluster\_labels):

subset = agg\_cars.loc[(label,),]

for i in subset.index:

plt.text(subset.loc[i][0]+5, subset.loc[i][2], 'type='+str(int(i)) + ', price='+str(int(subset.loc[i][3]))+'k')

plt.scatter(subset.horsepow, subset.mpg, s=subset.price\*20, c=color, label='cluster'+str(label))

plt.legend()

plt.title('Clusters')

plt.xlabel('horsepow')

plt.ylabel('mpg')