ML0101EN-Clus-Hierarchical-Cars-py-v1

September 19, 2021

1 Hierarchical Clustering

Estimated time needed: **25** minutes

1.1 Objectives

After completing this lab you will be able to:

- Use scikit-learn to Hierarchical clustering
- Create dendograms to visualize the clustering

Table of contents

```
<ol>
    <li><a href="https://#hierarchical_agglomerative">Hierarchical Clustering - Agglomerative</
        <ol>
            <li><a href="https://#generating_data">Generating Random Data</a></li>
            <li><a href="https://#agglomerative_clustering">Agglomerative Clustering</a></li>
            <li><a href="https://#dendrogram">Dendrogram Associated for the Agglomerative Hiera
        </ol>
    <li><a href="https://#clustering_vehicle_dataset">Clustering on the Vehicle Dataset</a></li
        <ol>
            <li><a href="https://#data_cleaning">Data Cleaning</a></li>
            <li><a href="https://#clustering_using_scipy">Clustering Using Scipy</a></li>
            <li><a href="https://#clustering_using_skl">Clustering using scikit-learn</a></li>
        </ol>
</ol>
```

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!

[1]:
```python
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

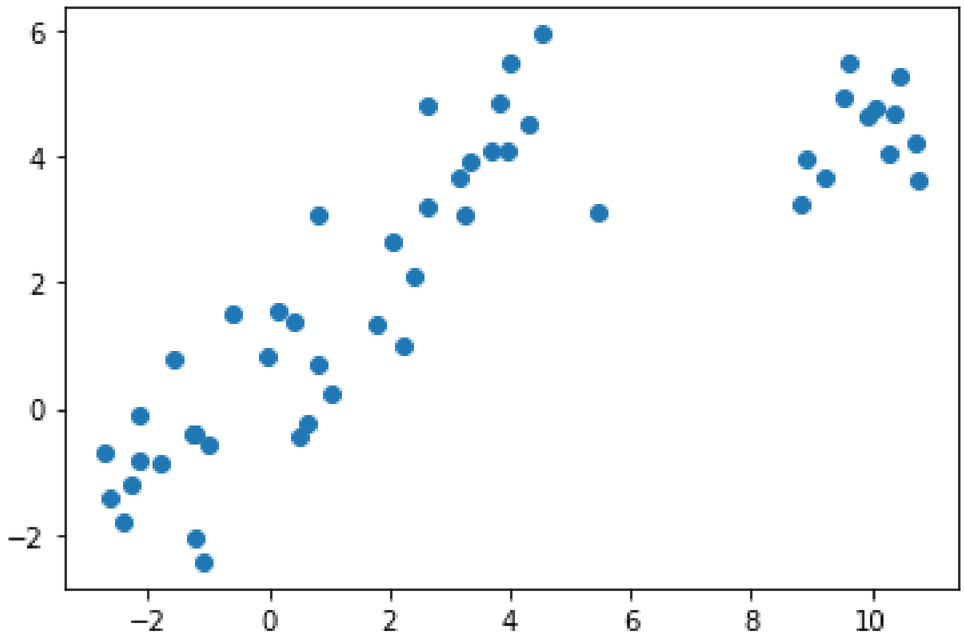
Save the result to X1 and y1.

[2]:
```
X1, y1 = make_blobs(n_samples=50, centers=[[4,4], [-2, -1], [1, 1], [10,4]],␣
↪cluster_std=0.9)
```

Plot the scatter plot of the randomly generated data.

[3]:
```
plt.scatter(X1[:, 0], X1[:, 1], marker='o')
```

[3]:
```
<matplotlib.collections.PathCollection at 0x7fac62896320>
```



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

Save the result to a variable called agglom .

[4]:
```
agglom = AgglomerativeClustering(n_clusters = 4, linkage = 'average')
```

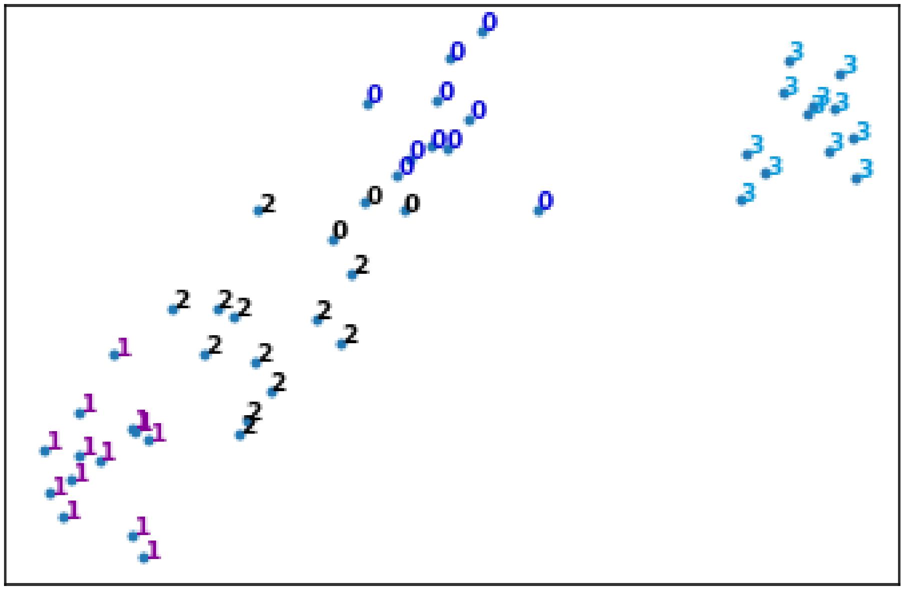
Fit the model with X2 and y2 from the generated data above.

[5]:
```
agglom.fit(X1,y1)
```

[5]:
```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
                        connectivity=None, linkage='average', memory=None,
                        n_clusters=4, pooling_func='deprecated')
```

Run the following code to show the clustering! Remember to read the code and comments to gain more understanding on how the plotting works.

[6]:
```
# 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, X1 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)

[7]:
```
dist_matrix = distance_matrix(X1,X1)
print(dist_matrix)
```

```
[[0.         0.15331976 0.61848595 … 0.28829684 0.91798526 0.05961192]
 [0.15331976 0.         0.71713021 … 0.3738127  1.02343872 0.20933051]
 [0.61848595 0.71713021 0.         … 0.34335527 0.30741724 0.56860821]
 …
 [0.28829684 0.3738127  0.34335527 … 0.         0.65010363 0.24927623]
 [0.91798526 1.02343872 0.30741724 … 0.65010363 0.         0.86499082]
 [0.05961192 0.20933051 0.56860821 … 0.24927623 0.86499082 0.        ]]
```

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 .

[8]:
```
Z = hierarchy.linkage(dist_matrix, 'complete')
```

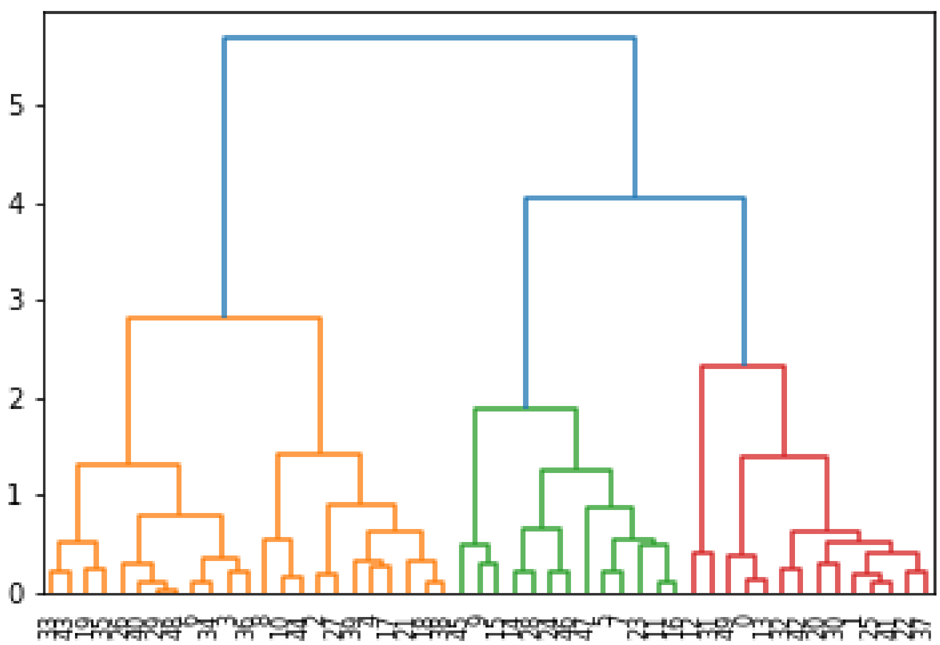
```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:1: ClusterWarning: scipy.cluster: The symmetric
non-negative hollow observation matrix looks suspiciously like an uncondensed
distance matrix
  """Entry point for launching an IPython kernel.
```

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

[9]:
```
dendro = hierarchy.dendrogram(Z)
```

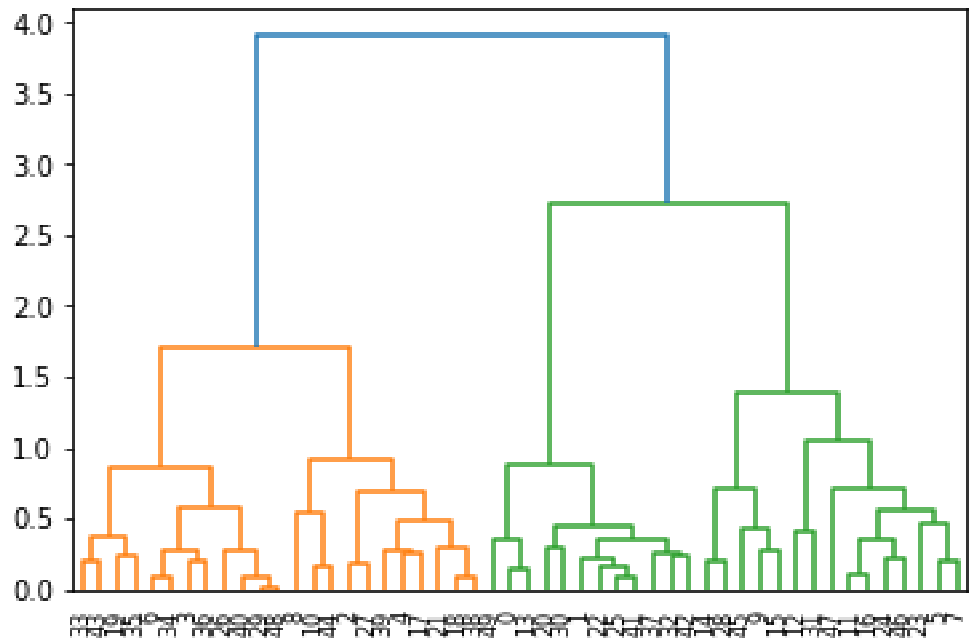


1.2 Practice

We used **complete** linkage for our case, change it to **average** linkage to see how the dendogram changes.

[11]:
```
Z = hierarchy.linkage(dist_matrix, 'average')
dendro = hierarchy.dendrogram(Z)
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:1: ClusterWarning: scipy.cluster: The symmetric
non-negative hollow observation matrix looks suspiciously like an uncondensed
distance matrix
  """Entry point for launching an IPython kernel.
```



Click here for the solution

```
Z = hierarchy.linkage(dist_matrix, 'average')
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.

1.2.1 Download data

To download the data, we will use **`!wget`** to download it from IBM Object Storage.
Did you know? When it comes to Machine Learning, you will likely be working with large datasets. As a business, where can you host your data? IBM is offering a unique opportunity for businesses, with 10 Tb of IBM Cloud Object Storage: Sign up now for free

[12]:
```
!wget -O cars_clus.csv https://cf-courses-data.s3.us.cloud-object-storage.
↪appdomain.cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/
↪Module%204/data/cars_clus.csv
```

```
--2021-09-19 01:46:36--  https://cf-courses-data.s3.us.cloud-object-
storage.appdomain.cloud/IBMDeveloperSkillsNetwork-ML0101EN-
SkillsNetwork/labs/Module%204/data/cars_clus.csv
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-
courses-data.s3.us.cloud-object-storage.appdomain.cloud)… 169.63.118.104
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-
courses-data.s3.us.cloud-object-storage.appdomain.cloud)|169.63.118.104|:443…
connected.
HTTP request sent, awaiting response… 200 OK
Length: 17774 (17K) [text/csv]
Saving to: 'cars_clus.csv'

cars_clus.csv       100%[===================>]  17.36K  --.-KB/s    in 0s

2021-09-19 01:46:36 (228 MB/s) - 'cars_clus.csv' saved [17774/17774]
```

1.3 Read data

Let's read dataset to see what features the manufacturer has collected about the existing models.

[13]:
```python
filename = 'cars_clus.csv'

#Read csv
pdf = pd.read_csv(filename)
print ("Shape of dataset: ", pdf.shape)

pdf.head(5)
```

```
Shape of dataset:  (159, 16)
```

[13]:
```
  manufact    model   sales  resale   type   price engine_s horsepow wheelbas  \
0    Acura  Integra  16.919  16.360  0.000  21.500    1.800  140.000  101.200
1    Acura       TL  39.384  19.875  0.000  28.400    3.200  225.000  108.100
2    Acura       CL  14.114  18.225  0.000  $null$    3.200  225.000  106.900
3    Acura       RL   8.588  29.725  0.000  42.000    3.500  210.000  114.600
4     Audi       A4  20.397  22.255  0.000  23.990    1.800  150.000  102.600

    width   length curb_wgt fuel_cap     mpg lnsales  partition
0  67.300  172.400    2.639   13.200  28.000   2.828        0.0
1  70.300  192.900    3.517   17.200  25.000   3.673        0.0
2  70.600  192.000    3.470   17.200  26.000   2.647        0.0
3  71.400  196.600    3.850   18.000  22.000   2.150        0.0
4  68.200  178.000    2.998   16.400  27.000   3.015        0.0
```

The feature sets include price in thousands (price), engine size (engine_s), horsepower (horsepow), wheelbase (wheelbas), width (width), length (length), curb weight (curb_wgt), fuel capacity (fuel_cap) and fuel efficiency (mpg).

Data Cleaning

Let's clean the dataset by dropping the rows that have null value:

[14]:
```
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_numeric, errors='coerce')
pdf = pdf.dropna()
pdf = pdf.reset_index(drop=True)
print ("Shape of dataset after cleaning: ", pdf.size)
pdf.head(5)
```

```
Shape of dataset before cleaning:  2544
Shape of dataset after cleaning:  1872
```

[14]:
```
  manufact    model   sales  resale  type  price  engine_s  horsepow  \
0    Acura  Integra  16.919  16.360   0.0  21.50       1.8     140.0
1    Acura       TL  39.384  19.875   0.0  28.40       3.2     225.0
2    Acura       RL   8.588  29.725   0.0  42.00       3.5     210.0
3     Audi       A4  20.397  22.255   0.0  23.99       1.8     150.0
4     Audi       A6  18.780  23.555   0.0  33.95       2.8     200.0

   wheelbas  width  length  curb_wgt  fuel_cap   mpg  lnsales  partition
0     101.2   67.3   172.4     2.639      13.2  28.0    2.828        0.0
1     108.1   70.3   192.9     3.517      17.2  25.0    3.673        0.0
2     114.6   71.4   196.6     3.850      18.0  22.0    2.150        0.0
3     102.6   68.2   178.0     2.998      16.4  27.0    3.015        0.0
4     108.7   76.1   192.0     3.561      18.5  22.0    2.933        0.0
```

1.3.1 Feature selection

Let's select our feature set:

[15]:
```
featureset = pdf[['engine_s',  'horsepow', 'wheelbas', 'width', 'length',␣
 ↪'curb_wgt', 'fuel_cap', 'mpg']]
```

1.3.2 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.

[16]:
```python
from sklearn.preprocessing import MinMaxScaler
x = featureset.values #returns a numpy array
min_max_scaler = MinMaxScaler()
feature_mtx = min_max_scaler.fit_transform(x)
feature_mtx [0:5]
```

[16]:
```
array([[0.11428571, 0.21518987, 0.18655098, 0.28143713, 0.30625832,
        0.2310559 , 0.13364055, 0.43333333],
       [0.31428571, 0.43037975, 0.3362256 , 0.46107784, 0.5792277 ,
        0.50372671, 0.31797235, 0.33333333],
       [0.35714286, 0.39240506, 0.47722343, 0.52694611, 0.62849534,
        0.60714286, 0.35483871, 0.23333333],
       [0.11428571, 0.24050633, 0.21691974, 0.33532934, 0.38082557,
        0.34254658, 0.28110599, 0.4       ],
       [0.25714286, 0.36708861, 0.34924078, 0.80838323, 0.56724368,
        0.5173913 , 0.37788018, 0.23333333]])
```

Clustering using Scipy

In this part we use Scipy package to cluster the dataset.

First, we calculate the distance matrix.

[17]:
```python
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])
D
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:3: DeprecationWarning: scipy.zeros is deprecated
and will be removed in SciPy 2.0.0, use numpy.zeros instead
  This is separate from the ipykernel package so we can avoid doing imports
until
```

[17]:
```
array([[0.        , 0.57777143, 0.75455727, …, 0.28530295, 0.24917241,
        0.18879995],
       [0.57777143, 0.        , 0.22798938, …, 0.36087756, 0.66346677,
        0.62201282],
       [0.75455727, 0.22798938, 0.        , …, 0.51727787, 0.81786095,
        0.77930119],
       …,
       [0.28530295, 0.36087756, 0.51727787, …, 0.        , 0.41797928,
        0.35720492],
       [0.24917241, 0.66346677, 0.81786095, …, 0.41797928, 0.        ,
```

```
 0.15212198],
[0.18879995, 0.62201282, 0.77930119, …, 0.35720492, 0.15212198,
 0.        ]])
```

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.

[18]:
```
import pylab
import scipy.cluster.hierarchy
Z = hierarchy.linkage(D, 'complete')
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:3: ClusterWarning: scipy.cluster: The symmetric
non-negative hollow observation matrix looks suspiciously like an uncondensed
distance matrix
  This is separate from the ipykernel package so we can avoid doing imports
until
```

Essentially, 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:

[19]:
```
from scipy.cluster.hierarchy import fcluster
max_d = 3
clusters = fcluster(Z, max_d, criterion='distance')
clusters
```

[19]:
```
array([ 1,  5,  5,  6,  5,  4,  6,  5,  5,  5,  5,  5,  4,  4,  5,  1,  6,
        5,  5,  5,  4,  2, 11,  6,  6,  5,  6,  5,  1,  6,  6, 10,  9,  8,
        9,  3,  5,  1,  7,  6,  5,  3,  5,  3,  8,  7,  9,  2,  6,  6,  5,
        4,  2,  1,  6,  5,  2,  7,  5,  5,  5,  4,  4,  3,  2,  6,  6,  5,
        7,  4,  7,  6,  6,  5,  3,  5,  5,  6,  5,  4,  4,  1,  6,  5,  5,
        5,  6,  4,  5,  4,  1,  6,  5,  6,  6,  5,  5,  5,  7,  7,  7,  2,
        2,  1,  2,  6,  5,  1,  1,  1,  7,  8,  1,  1,  6,  1,  1],
      dtype=int32)
```

Also, you can determine the number of clusters directly:

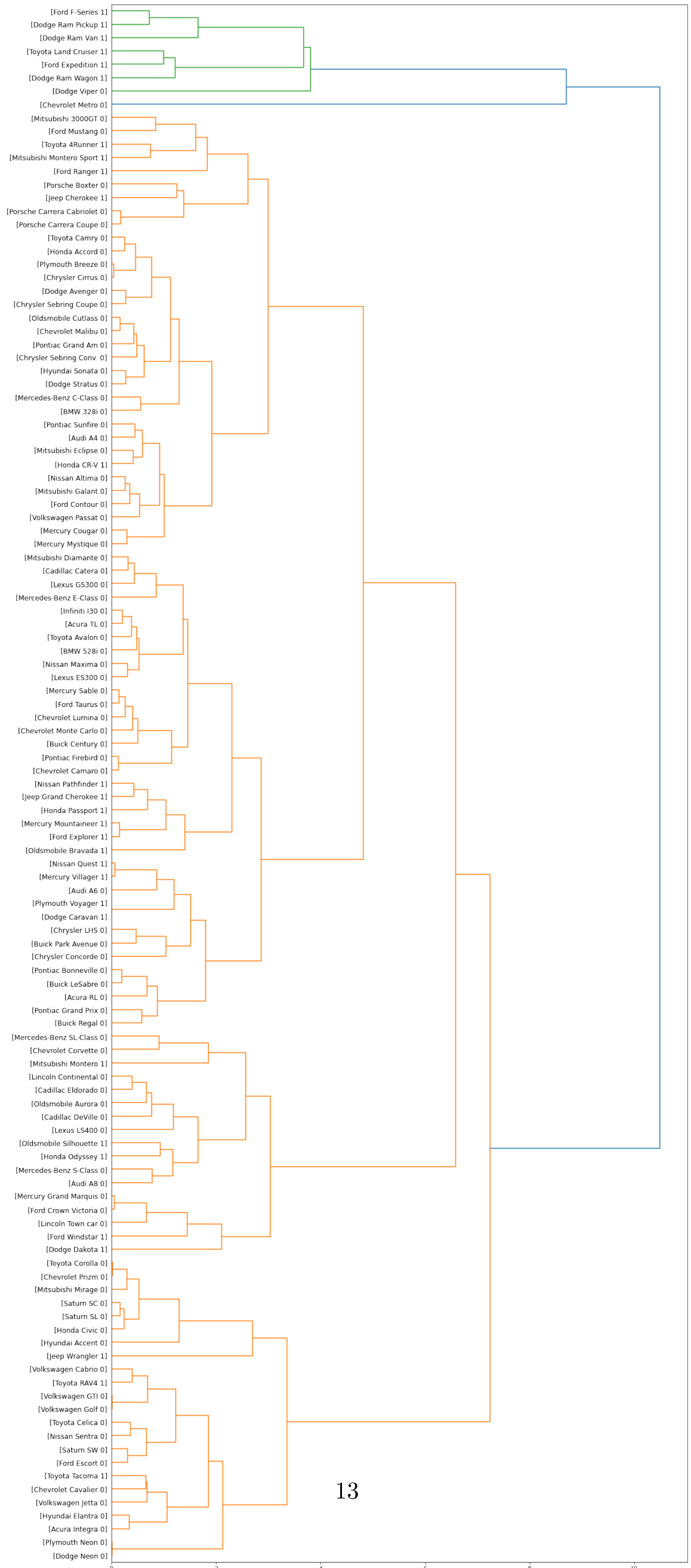
[20]:
```
from scipy.cluster.hierarchy import fcluster
k = 5
clusters = fcluster(Z, k, criterion='maxclust')
clusters
```

[20]:
```
array([1, 3, 3, 3, 3, 2, 3, 3, 3, 3, 3, 3, 2, 2, 3, 1, 3, 3, 3, 3, 2, 1,
       5, 3, 3, 3, 3, 3, 1, 3, 3, 4, 4, 4, 4, 2, 3, 1, 3, 3, 3, 2, 3, 2,
```

```
       4, 3, 4, 1, 3, 3, 3, 2, 1, 1, 3, 3, 1, 3, 3, 3, 3, 2, 2, 2, 1, 3,
       3, 3, 3, 2, 3, 3, 3, 3, 2, 3, 3, 3, 3, 2, 2, 1, 3, 3, 3, 3, 3, 2,
       3, 2, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 1, 1, 3, 3, 1, 1, 1,
       3, 4, 1, 1, 3, 1, 1], dtype=int32)
```

Now, plot the dendrogram:

[21]:
```
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')
```


[Ford F-Series 1]
[Dodge Ram Pickup 1]
[Dodge Ram Van 1]
[Toyota Land Cruiser 1]
[Ford Expedition 1]
[Dodge Ram Wagon 1]
[Dodge Viper 0]
[Chevrolet Metro 0]
[Mitsubishi 3000GT 0]
[Ford Mustang 0]
[Toyota 4Runner 1]
[Mitsubishi Montero Sport 1]
[Ford Ranger 1]
[Porsche Boxter 0]
[Jeep Cherokee 1]
[Porsche Carrera Cabriolet 0]
[Porsche Carrera Coupe 0]
[Toyota Camry 0]
[Honda Accord 0]
[Plymouth Breeze 0]
[Chrysler Cirrus 0]
[Dodge Avenger 0]
[Chrysler Sebring Coupe 0]
[Oldsmobile Cutlass 0]
[Chevrolet Malibu 0]
[Pontiac Grand Am 0]
[Chrysler Sebring Conv. 0]
[Hyundai Sonata 0]
[Dodge Stratus 0]
[Mercedes-Benz C-Class 0]
[BMW 328i 0]
[Pontiac Sunfire 0]
[Audi A4 0]
[Mitsubishi Eclipse 0]
[Honda CR-V 1]
[Nissan Altima 0]
[Mitsubishi Galant 0]
[Ford Contour 0]
[Volkswagen Passat 0]
[Mercury Cougar 0]
[Mercury Mystique 0]
[Mitsubishi Diamante 0]
[Cadillac Catera 0]
[Lexus GS300 0]
[Mercedes-Benz E-Class 0]
[Infiniti I30 0]
[Acura TL 0]
[Toyota Avalon 0]
[BMW 528i 0]
[Nissan Maxima 0]
[Lexus ES300 0]
[Mercury Sable 0]
[Ford Taurus 0]
[Chevrolet Lumina 0]
[Chevrolet Monte Carlo 0]
[Buick Century 0]
[Pontiac Firebird 0]
[Chevrolet Camaro 0]
[Nissan Pathfinder 1]
[Jeep Grand Cherokee 1]
[Honda Passport 1]
[Mercury Mountaineer 1]
[Ford Explorer 1]
[Oldsmobile Bravada 1]
[Nissan Quest 1]
[Mercury Villager 1]
[Audi A6 0]
[Plymouth Voyager 1]
[Dodge Caravan 1]
[Chrysler LHS 0]
[Buick Park Avenue 0]
[Chrysler Concorde 0]
[Pontiac Bonneville 0]
[Buick LeSabre 0]
[Acura RL 0]
[Pontiac Grand Prix 0]
[Buick Regal 0]
[Mercedes-Benz SL-Class 0]
[Chevrolet Corvette 0]
[Mitsubishi Montero 1]
[Lincoln Continental 0]
[Cadillac Eldorado 0]
[Oldsmobile Aurora 0]
[Cadillac DeVille 0]
[Lexus LS400 0]
[Oldsmobile Silhouette 1]
[Honda Odyssey 1]
[Mercedes-Benz S-Class 0]
[Audi A8 0]
[Mercury Grand Marquis 0]
[Ford Crown Victoria 0]
[Lincoln Town car 0]
[Ford Windstar 1]
[Dodge Dakota 1]
[Toyota Corolla 0]
[Chevrolet Prizm 0]
[Mitsubishi Mirage 0]
[Saturn SC 0]
[Saturn SL 0]
[Honda Civic 0]
[Hyundai Accent 0]
[Jeep Wrangler 1]
[Volkswagen Cabrio 0]
[Toyota RAV4 1]
[Volkswagen GTI 0]
[Volkswagen Golf 0]
[Toyota Celica 0]
[Nissan Sentra 0]
[Saturn SW 0]
[Ford Escort 0]
[Toyota Tacoma 1]
[Chevrolet Cavalier 0]
[Volkswagen Jetta 0]
[Hyundai Elantra 0]
[Acura Integra 0]
[Plymouth Neon 0]
[Dodge Neon 0]
0
2
4
6
8
10

Clustering using scikit-learn

Let's redo it again, but this time using the scikit-learn package:

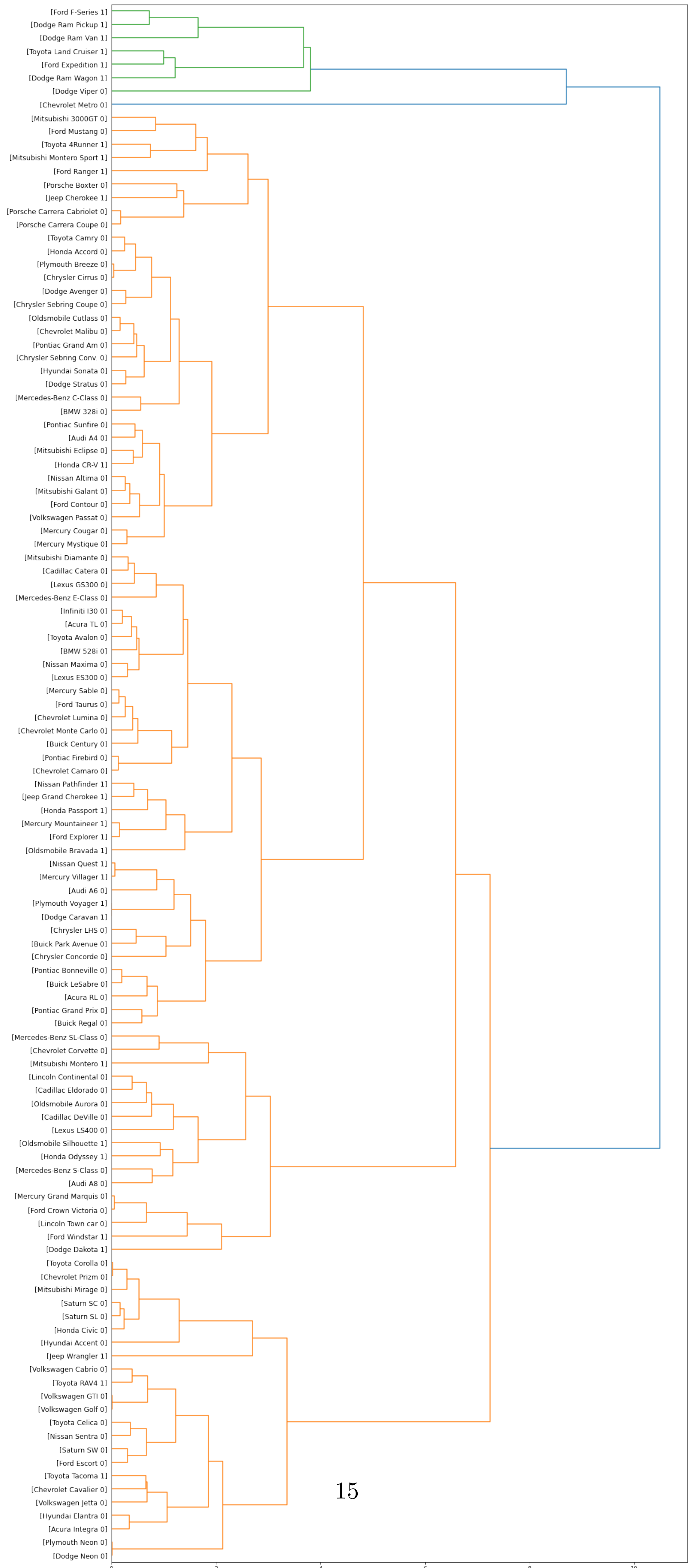
[22]:
```
from sklearn.metrics.pairwise import euclidean_distances
dist_matrix = euclidean_distances(feature_mtx,feature_mtx)
print(dist_matrix)
```

```
[[0.         0.57777143 0.75455727 … 0.28530295 0.24917241 0.18879995]
 [0.57777143 0.         0.22798938 … 0.36087756 0.66346677 0.62201282]
 [0.75455727 0.22798938 0.         … 0.51727787 0.81786095 0.77930119]
 …
 [0.28530295 0.36087756 0.51727787 … 0.         0.41797928 0.35720492]
 [0.24917241 0.66346677 0.81786095 … 0.41797928 0.         0.15212198]
 [0.18879995 0.62201282 0.77930119 … 0.35720492 0.15212198 0.        ]]
```

[23]:
```
Z_using_dist_matrix = hierarchy.linkage(dist_matrix, 'complete')
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:1: ClusterWarning: scipy.cluster: The symmetric
non-negative hollow observation matrix looks suspiciously like an uncondensed
distance matrix
  """Entry point for launching an IPython kernel.
```

[24]:
```
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_using_dist_matrix,  leaf_label_func=llf,␣
 ↪leaf_rotation=0, leaf_font_size =12, orientation = 'right')
```


[Ford F-Series 1]
[Dodge Ram Pickup 1]
[Dodge Ram Van 1]
[Toyota Land Cruiser 1]
[Ford Expedition 1]
[Dodge Ram Wagon 1]
[Dodge Viper 0]
[Chevrolet Metro 0]
[Mitsubishi 3000GT 0]
[Ford Mustang 0]
[Toyota 4Runner 1]
[Mitsubishi Montero Sport 1]
[Ford Ranger 1]
[Porsche Boxter 0]
[Jeep Cherokee 1]
[Porsche Carrera Cabriolet 0]
[Porsche Carrera Coupe 0]
[Toyota Camry 0]
[Honda Accord 0]
[Plymouth Breeze 0]
[Chrysler Cirrus 0]
[Dodge Avenger 0]
[Chrysler Sebring Coupe 0]
[Oldsmobile Cutlass 0]
[Chevrolet Malibu 0]
[Pontiac Grand Am 0]
[Chrysler Sebring Conv. 0]
[Hyundai Sonata 0]
[Dodge Stratus 0]
[Mercedes-Benz C-Class 0]
[BMW 328i 0]
[Pontiac Sunfire 0]
[Audi A4 0]
[Mitsubishi Eclipse 0]
[Honda CR-V 1]
[Nissan Altima 0]
[Mitsubishi Galant 0]
[Ford Contour 0]
[Volkswagen Passat 0]
[Mercury Cougar 0]
[Mercury Mystique 0]
[Mitsubishi Diamante 0]
[Cadillac Catera 0]
[Lexus GS300 0]
[Mercedes-Benz E-Class 0]
[Infiniti I30 0]
[Acura TL 0]
[Toyota Avalon 0]
[BMW 528i 0]
[Nissan Maxima 0]
[Lexus ES300 0]
[Mercury Sable 0]
[Ford Taurus 0]
[Chevrolet Lumina 0]
[Chevrolet Monte Carlo 0]
[Buick Century 0]
[Pontiac Firebird 0]
[Chevrolet Camaro 0]
[Nissan Pathfinder 1]
[Jeep Grand Cherokee 1]
[Honda Passport 1]
[Mercury Mountaineer 1]
[Ford Explorer 1]
[Oldsmobile Bravada 1]
[Nissan Quest 1]
[Mercury Villager 1]
[Audi A6 0]
[Plymouth Voyager 1]
[Dodge Caravan 1]
[Chrysler LHS 0]
[Buick Park Avenue 0]
[Chrysler Concorde 0]
[Pontiac Bonneville 0]
[Buick LeSabre 0]
[Acura RL 0]
[Pontiac Grand Prix 0]
[Buick Regal 0]
[Mercedes-Benz SL-Class 0]
[Chevrolet Corvette 0]
[Mitsubishi Montero 1]
[Lincoln Continental 0]
[Cadillac Eldorado 0]
[Oldsmobile Aurora 0]
[Cadillac DeVille 0]
[Lexus LS400 0]
[Oldsmobile Silhouette 1]
[Honda Odyssey 1]
[Mercedes-Benz S-Class 0]
[Audi A8 0]
[Mercury Grand Marquis 0]
[Ford Crown Victoria 0]
[Lincoln Town car 0]
[Ford Windstar 1]
[Dodge Dakota 1]
[Toyota Corolla 0]
[Chevrolet Prizm 0]
[Mitsubishi Mirage 0]
[Saturn SC 0]
[Saturn SL 0]
[Honda Civic 0]
[Hyundai Accent 0]
[Jeep Wrangler 1]
[Volkswagen Cabrio 0]
[Toyota RAV4 1]
[Volkswagen GTI 0]
[Volkswagen Golf 0]
[Toyota Celica 0]
[Nissan Sentra 0]
[Saturn SW 0]
[Ford Escort 0]
[Toyota Tacoma 1]
[Chevrolet Cavalier 0]
[Volkswagen Jetta 0]
[Hyundai Elantra 0]
[Acura Integra 0]
[Plymouth Neon 0]
[Dodge Neon 0]
0
2
4
6
8
10

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.

[25]:
```
agglom = AgglomerativeClustering(n_clusters = 6, linkage = 'complete')
agglom.fit(dist_matrix)

agglom.labels_
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/sklearn/cluster/hierarchical.py:471: ClusterWarning: scipy.cluster: The
symmetric non-negative hollow observation matrix looks suspiciously like an
uncondensed distance matrix
  out = hierarchy.linkage(X, method=linkage, metric=affinity)
```

[25]:
```
array([1, 2, 2, 3, 2, 4, 3, 2, 2, 2, 2, 2, 4, 4, 2, 1, 3, 2, 2, 2, 4, 1,
       5, 3, 3, 2, 3, 2, 1, 3, 3, 0, 0, 0, 0, 4, 2, 1, 3, 3, 2, 4, 2, 4,
       0, 3, 0, 1, 3, 3, 2, 4, 1, 1, 3, 2, 1, 3, 2, 2, 2, 4, 4, 4, 1, 3,
       3, 2, 3, 4, 3, 3, 3, 2, 4, 2, 2, 3, 2, 4, 4, 1, 3, 2, 2, 2, 3, 4,
       2, 4, 1, 3, 2, 3, 3, 2, 2, 2, 3, 3, 3, 1, 1, 1, 1, 3, 2, 1, 1, 1,
       3, 0, 1, 1, 3, 1, 1])
```

We can add a new field to our dataframe to show the cluster of each row:

[26]:
```
pdf['cluster_'] = agglom.labels_
pdf.head()
```

[26]:
```
  manufact    model   sales  resale  type  price  engine_s  horsepow  \
0    Acura  Integra  16.919  16.360   0.0  21.50       1.8     140.0
1    Acura       TL  39.384  19.875   0.0  28.40       3.2     225.0
2    Acura       RL   8.588  29.725   0.0  42.00       3.5     210.0
3     Audi       A4  20.397  22.255   0.0  23.99       1.8     150.0
4     Audi       A6  18.780  23.555   0.0  33.95       2.8     200.0

   wheelbas  width  length  curb_wgt  fuel_cap   mpg  lnsales  partition  \
0     101.2   67.3   172.4     2.639      13.2  28.0    2.828        0.0
1     108.1   70.3   192.9     3.517      17.2  25.0    3.673        0.0
2     114.6   71.4   196.6     3.850      18.0  22.0    2.150        0.0
```

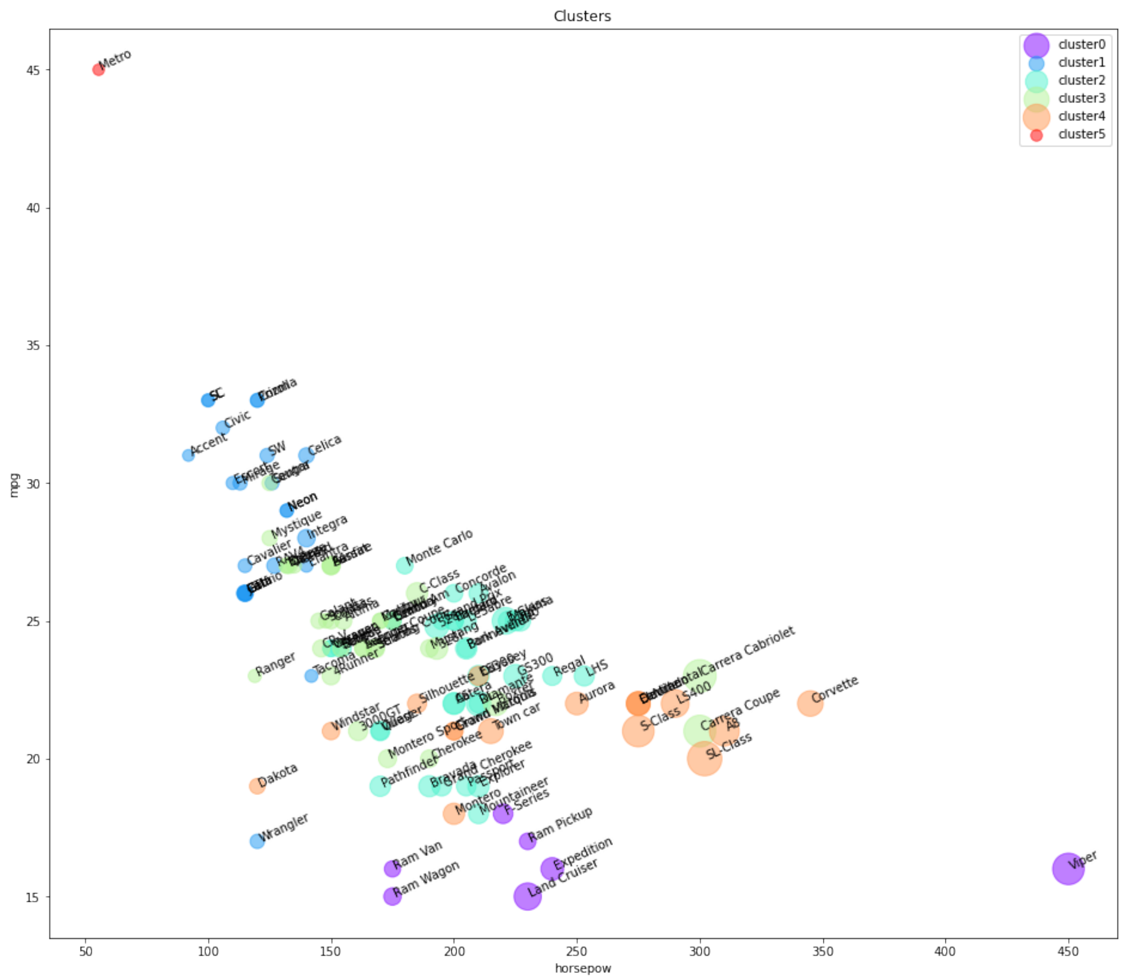
```
3     102.6   68.2   178.0     2.998      16.4  27.0    3.015         0.0
4     108.7   76.1   192.0     3.561      18.5  22.0    2.933         0.0

   cluster_
0         1
1         2
2         2
3         3
4         2
```

[27]:
```
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')
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all points.
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all points.
```

[27]: Text(0, 0.5, '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 0 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:

[28]:
```
pdf.groupby(['cluster_','type'])['cluster_'].count()
```

[28]:
```
cluster_  type
0         0.0      1
          1.0      6
1         0.0     20
          1.0      3
2         0.0     26
          1.0     10
3         0.0     28
          1.0      5
4         0.0     12
          1.0      5
5         0.0      1
Name: cluster_, dtype: int64
```

Now we can look at the characteristics of each cluster:

[29]:
```
agg_cars = pdf.
 ↪groupby(['cluster_','type'])['horsepow','engine_s','mpg','price'].mean()
agg_cars
```

```
/home/jupyterlab/conda/envs/python/lib/python3.6/site-
packages/ipykernel_launcher.py:1: FutureWarning: Indexing with multiple keys
(implicitly converted to a tuple of keys) will be deprecated, use a list
instead.
  """Entry point for launching an IPython kernel.
```

[29]:
```
                 horsepow  engine_s        mpg      price
cluster_ type
0        0.0   450.000000  8.000000  16.000000  69.725000
         1.0   211.666667  4.483333  16.166667  29.024667
1        0.0   118.500000  1.890000  29.550000  14.226100
         1.0   129.666667  2.300000  22.333333  14.292000
2        0.0   203.615385  3.284615  24.223077  27.988692
         1.0   182.000000  3.420000  20.300000  26.120600
3        0.0   168.107143  2.557143  25.107143  24.693786
         1.0   155.600000  2.840000  22.000000  19.807000
4        0.0   267.666667  4.566667  21.416667  46.417417
         1.0   173.000000  3.180000  20.600000  24.308400
5        0.0    55.000000  1.000000  45.000000   9.235000
```

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:

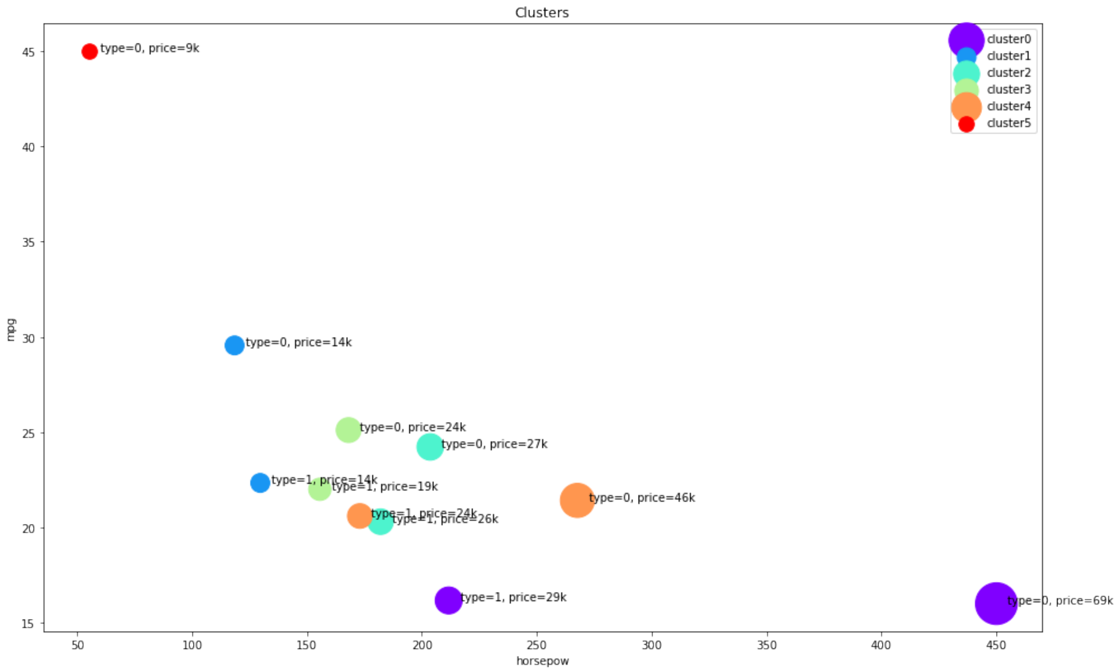
- Cluster 1: with almost highest mpg among trucks, and lowest in horsepower and price.
- Cluster 2: with almost low mpg and medium horsepower, but higher price than average.
- Cluster 3: with good mpg and horsepower, low price.

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 a high accuracy.

[30]:
```
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')
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2-D array with
a single row if you intend to specify the same RGB or RGBA value for all points.
```

[30]: Text(0, 0.5, 'mpg')



Want to learn more?

IBM SPSS Modeler is a comprehensive analytics platform that has many machine learning algorithms. It has been designed to bring predictive intelligence to decisions made by individuals, by groups, by systems – by your enterprise as a whole. A free trial is available through this course, available here: SPSS Modeler

Also, you can use Watson Studio to run these notebooks faster with bigger datasets. Watson Studio is IBM's leading cloud solution for data scientists, built by data scientists. With Jupyter notebooks, RStudio, Apache Spark and popular libraries pre-packaged in the cloud, Watson Studio enables data scientists to collaborate on their projects without having to install anything. Join the fast-growing community of Watson Studio users today with a free account at Watson Studio

1.3.3 Thank you for completing this lab!

1.4 Author

Saeed Aghabozorgi

1.4.1 Other Contributors

Joseph Santarcangelo

1.5 Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-01-11	2.2	Lakshmi	Changed distance matrix in agglomerative clustering
2020-11-03	2.1	Lakshmi	Updated URL
2020-08-27	2.0	Lavanya	Moved lab to course repo in GitLab

##

[ ]:

```

```