Hierarchical Clustering - Agglomerative

We will be looking at a clustering technique, which is Agglomerative Hierarchical Clustering. Agglomerative is the bottom up approach.

Agglomerative Clustering is a type of hierarchical clustering algorithm. It is an unsupervised machine learning technique that divides the population into several clusters such that data points in the same cluster are more similar and data points in different clusters are dissimilar.

The Nootebook is divided into 2 parts:

- **Part 1:** Making Agglomerative clustering on random generated dataset
- **Part 2:** Clustering on the Vehicle Dataset

Agglomerative Algorithm

1. Create n clusters, one for each data point

2. Compute the Proximity Matrix (Distance Matrix)

3. Repeat until only a single cluster remains

```
i.  Merge the two closest clusters

ii. Update the proximity matrix
```

In [1]:
```
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 import make_blobs
```

Part 1

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

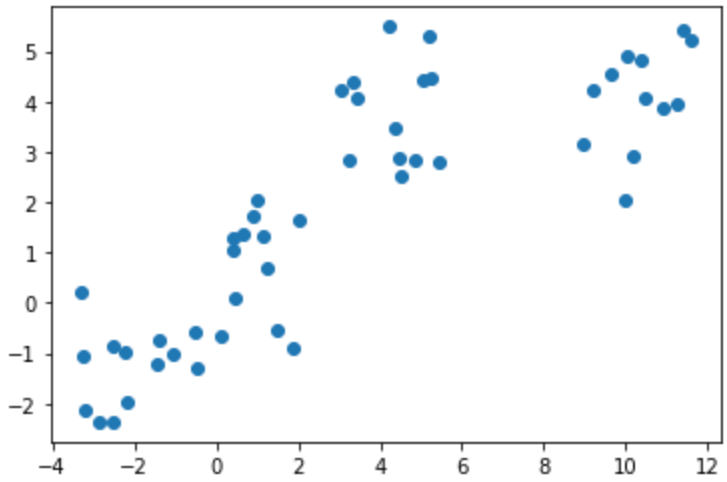
- Choose a number between 0.5-1.5

Save the result to X1 and y1.

In [2]:
```
X1, y1 = make_blobs(n_samples=50, centers=[[4,4], [-2, -1], [1, 1], [10,4]], cluster_std
```

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

Out[3]:
```
<matplotlib.collections.PathCollection at 0x18744b90190>
```



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: determines which distance to use between sets of observation. The algorithm will merge the pairs of cluster that minimize this criterion.

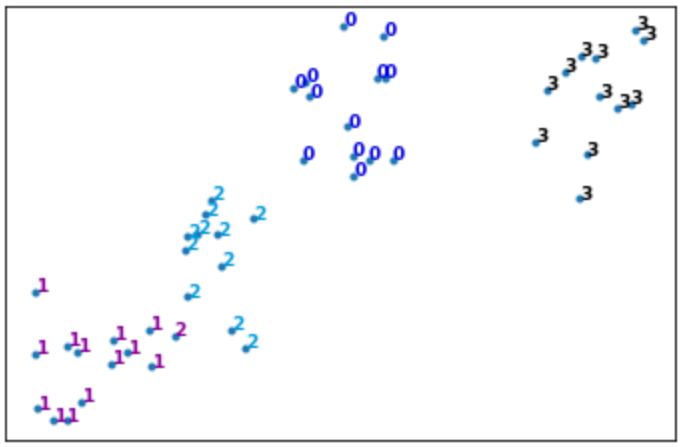
In [4]:
```
agglom = AgglomerativeClustering(n_clusters = 4, linkage = 'average')
```

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

Out[5]:
```
AgglomerativeClustering(linkage='average', n_clusters=4)
```

In [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

A Hierarchical clustering is typically visualized as a dendrogram. 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. 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.

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

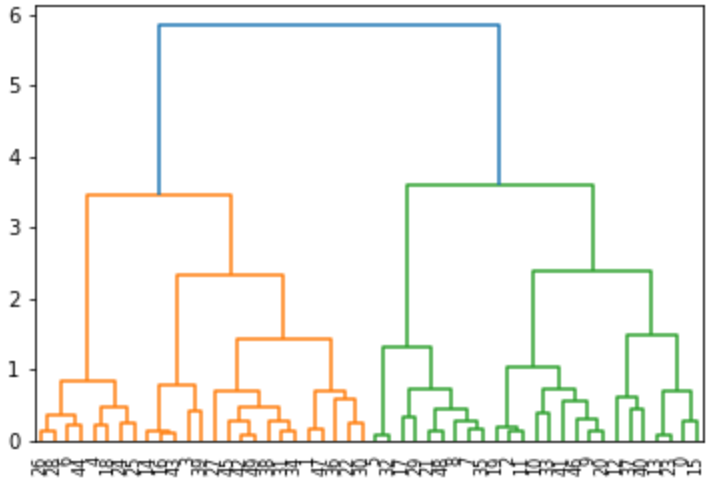
In [7]:
```
dist_matrix = distance_matrix(X1,X1)
print(dist_matrix)
```

```
[[0.         0.83428488 0.1785977  ... 0.83167431 0.39660198 0.9323158 ]
 [0.83428488 0.         0.65647882 ... 0.04292983 0.99120681 0.12932693]
 [0.1785977  0.65647882 0.         ... 0.65514176 0.47784482 0.75373444]
 ...
 [0.83167431 0.04292983 0.65514176 ... 0.         0.9717485  0.16396088]
 [0.39660198 0.99120681 0.47784482 ... 0.9717485  0.         1.11331966]
 [0.9323158  0.12932693 0.75373444 ... 0.16396088 1.11331966 0.        ]]
```

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

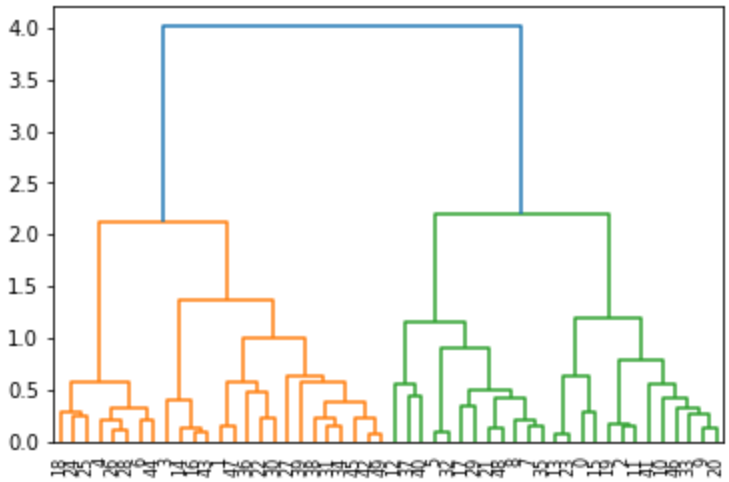
```
C:\Users\HP\AppData\Local\Temp\ipykernel_20152\850864696.py:1: ClusterWarning: scipy.clu
```

```
ster: The symmetric non-negative hollow observation matrix looks suspiciously like an un
condensed distance matrix
  Z = hierarchy.linkage(dist_matrix, 'complete')
```



In [9]:
```
Z = hierarchy.linkage(dist_matrix, 'average')
dendro = hierarchy.dendrogram(Z)
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_20152\3726426701.py:1: ClusterWarning: scipy.cl
uster: The symmetric non-negative hollow observation matrix looks suspiciously like an u
ncondensed distance matrix
  Z = hierarchy.linkage(dist_matrix, 'average')
```



In [10]:
```
import os
os.chdir(r"C:\Users\HP\Downloads\cars_clus")

pdf = pd.read_csv('cars_clus.csv')
print ("Shape of dataset: ", pdf.shape)

pdf.head(5)
```

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

Out[10]:

	manufact	model	sales	resale	type	price	engine_s	horsepow	wheelbas	width	length	curb_wgt	fu
0	Acura	Integra	16.919	16.360	0.000	21.500	1.800	140.000	101.200	67.300	172.400	2.639	
1	Acura	TL	39.384	19.875	0.000	28.400	3.200	225.000	108.100	70.300	192.900	3.517	
2	Acura	CL	14.114	18.225	0.000	*null*	3.200	225.000	106.900	70.600	192.000	3.470	
3	Acura	RL	8.588	29.725	0.000	42.000	3.500	210.000	114.600	71.400	196.600	3.850	
4	Audi	A4	20.397	22.255	0.000	23.990	1.800	150.000	102.600	68.200	178.000	2.998	

In [11]:
```
# Data Cleaning
```

```
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
```

Out[11]:

	manufact	model	sales	resale	type	price	engine_s	horsepow	wheelbas	width	length	curb_wgt	fuel_
0	Acura	Integra	16.919	16.360	0.0	21.50	1.8	140.0	101.2	67.3	172.4	2.639	1
1	Acura	TL	39.384	19.875	0.0	28.40	3.2	225.0	108.1	70.3	192.9	3.517	1
2	Acura	RL	8.588	29.725	0.0	42.00	3.5	210.0	114.6	71.4	196.6	3.850	1
3	Audi	A4	20.397	22.255	0.0	23.99	1.8	150.0	102.6	68.2	178.0	2.998	1
4	Audi	A6	18.780	23.555	0.0	33.95	2.8	200.0	108.7	76.1	192.0	3.561	1

In [12]:
```
# Feature Selection
featureset = pdf[['engine_s',  'horsepow', 'wheelbas', 'width', 'length', 'curb_wgt', 'f
```

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.

In [13]:
```
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]
```

Out[13]:
```
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]])
```

In [14]:
```
# Clustering using Scipy
# 1- Calculate 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])
D
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_20152\3287657191.py:5: DeprecationWarning: scip
y.zeros is deprecated and will be removed in SciPy 2.0.0, use numpy.zeros instead
  D = scipy.zeros([leng,leng])
```

Out[14]:
```
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

In [15]:
```
import pylab
import scipy.cluster.hierarchy
Z = hierarchy.linkage(D, 'complete')
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_20152\227076933.py:3: ClusterWarning: scipy.clu
ster: The symmetric non-negative hollow observation matrix looks suspiciously like an un
condensed distance matrix
  Z = hierarchy.linkage(D, 'complete')
```

In [16]:
```
from scipy.cluster.hierarchy import fcluster
k = 5
clusters = fcluster(Z, k, criterion='maxclust')
clusters
```

Out[16]:
```
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)
```

Clustering using scikit-learn

In [17]:
```
from sklearn.metrics.pairwise import euclidean_distances
dist_matrix = euclidean_distances(feature_mtx,feature_mtx)
print(dist_matrix)
```

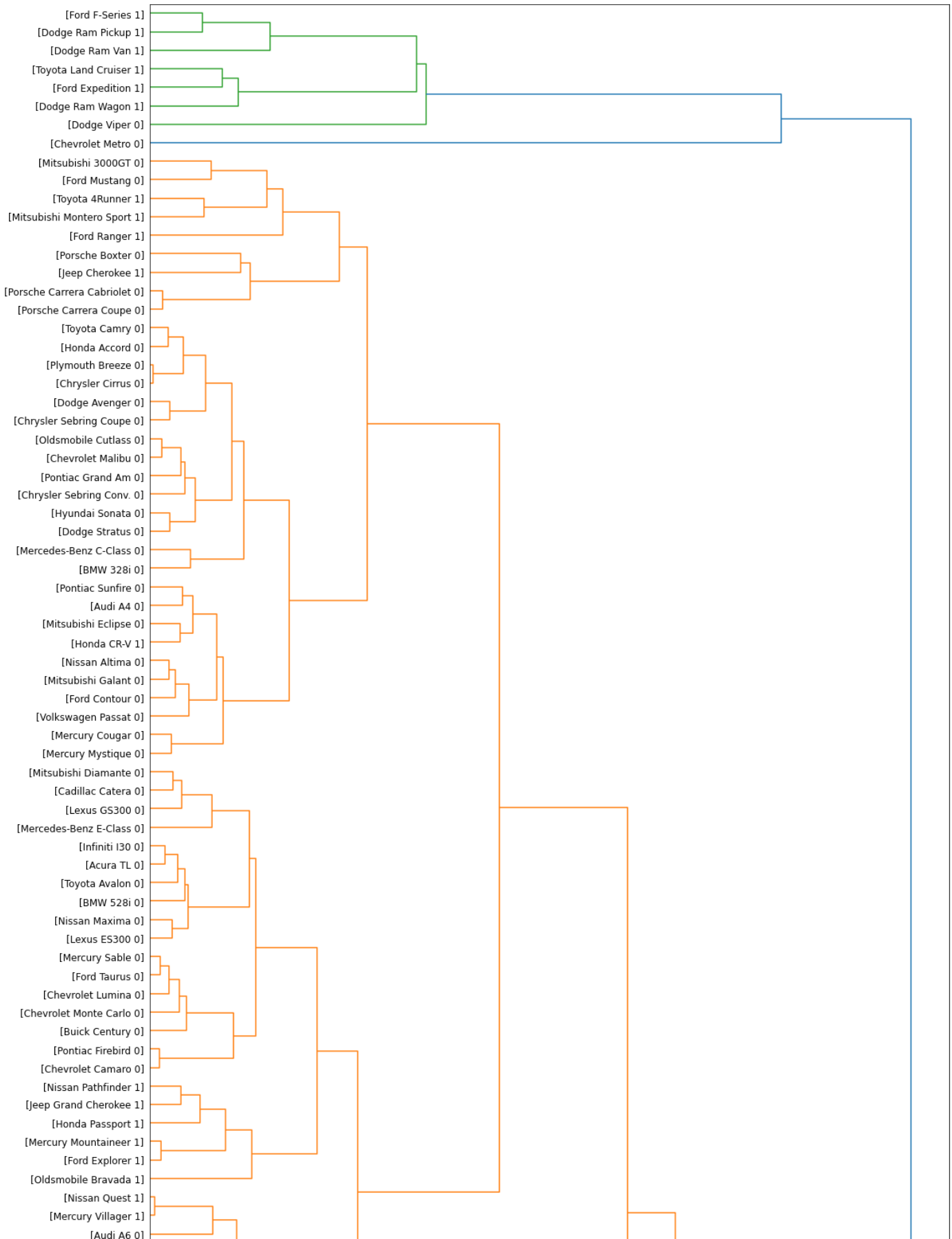
```
[[0.         0.57777143 0.75455727 ... 0.28530295 0.24917241 0.18879995]
 [0.57777143 0.         0.22798938 ... 0.36087756 0.66346677 0.62201282]
 [0.75455727 0.22798938 0.         ... 0.51727787 0.81786095 0.77930119]
 ...
 [0.28530295 0.36087756 0.51727787 ... 0.         0.41797928 0.35720492]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

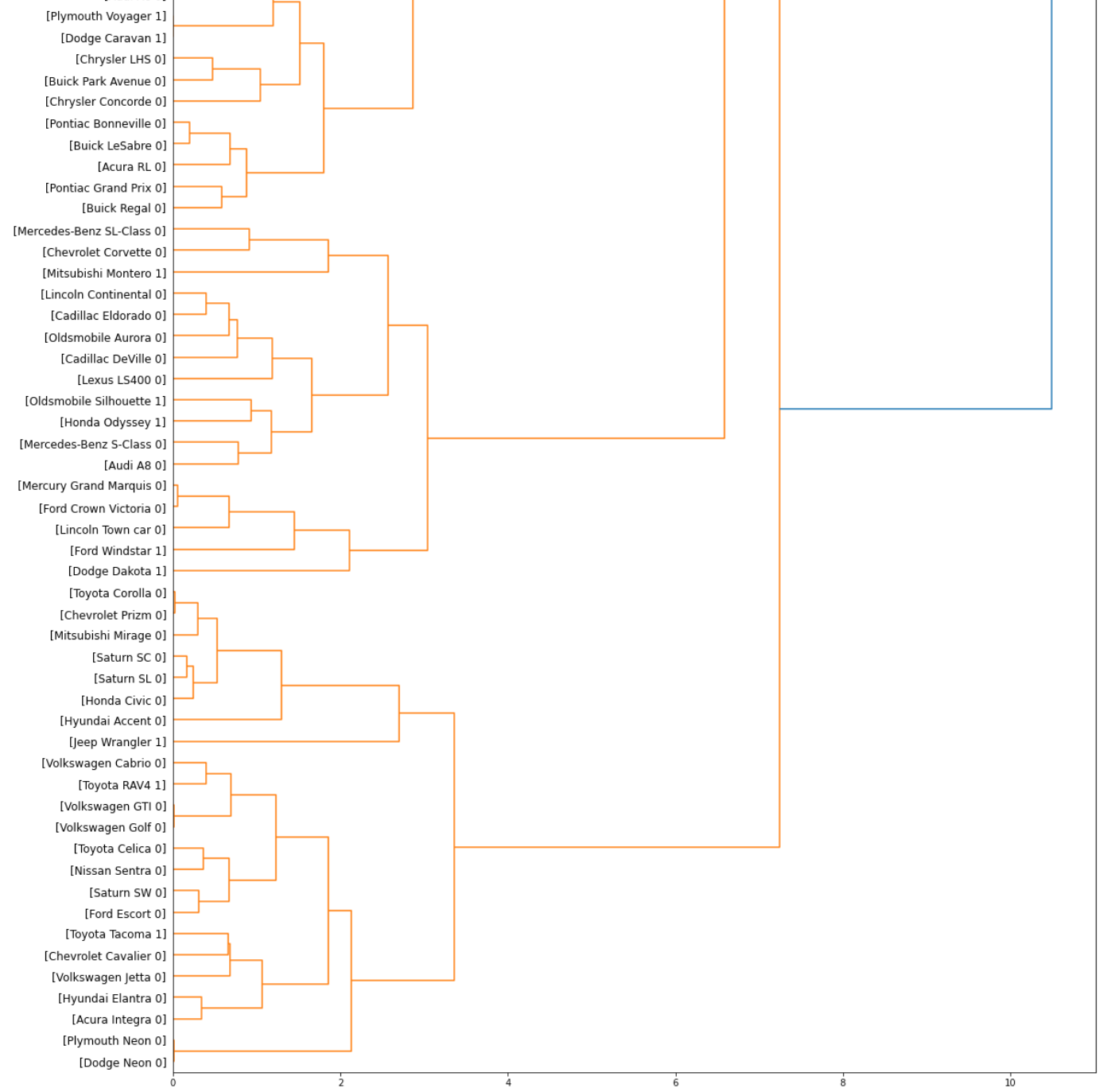
In [18]:
```
Z_using_dist_matrix = hierarchy.linkage(dist_matrix, 'complete')
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_20152\1633147189.py:1: ClusterWarning: scipy.cl
```

```
uster: The symmetric non-negative hollow observation matrix looks suspiciously like an u
ncondensed distance matrix
  Z_using_dist_matrix = hierarchy.linkage(dist_matrix, 'complete')
```

In [19]:
```
fig = pylab.figure(figsize=(18,50))
def llf(id):
    return '[%s %s %s]' % (pdf['manufact'][id], pdf['model'][id], int(float(pdf['type'][

dendro = hierarchy.dendrogram(Z_using_dist_matrix,  leaf_label_func=llf, leaf_rotation=0
```





In [20]:
```
agglom = AgglomerativeClustering(n_clusters = 6, linkage = 'complete')
agglom.fit(dist_matrix)

agglom.labels_
```

```
C:\Users\HP\anaconda3\lib\site-packages\sklearn\cluster\_agglomerative.py:542: ClusterWa
rning: scipy.cluster: The symmetric non-negative hollow observation matrix looks suspici
ously like an uncondensed distance matrix
  out = hierarchy.linkage(X, method=linkage, metric=affinity)
```

Out[20]:
```
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], dtype=int64)
```

In [21]:
```
pdf['cluster_'] = agglom.labels_
pdf.head()
```

Out[21]:

	manufact	model	sales	resale	type	price	engine_s	horsepow	wheelbas	width	length	curb_wgt	fuel_

0	Acura	Integra	16.919	16.360	0.0	21.50	1.8	140.0	101.2	67.3	172.4	2.639
1	Acura	TL	39.384	19.875	0.0	28.40	3.2	225.0	108.1	70.3	192.9	3.517
2	Acura	RL	8.588	29.725	0.0	42.00	3.5	210.0	114.6	71.4	196.6	3.850
3	Audi	A4	20.397	22.255	0.0	23.99	1.8	150.0	102.6	68.2	178.0	2.998
4	Audi	A6	18.780	23.555	0.0	33.95	2.8	200.0	108.7	76.1	192.0	3.561

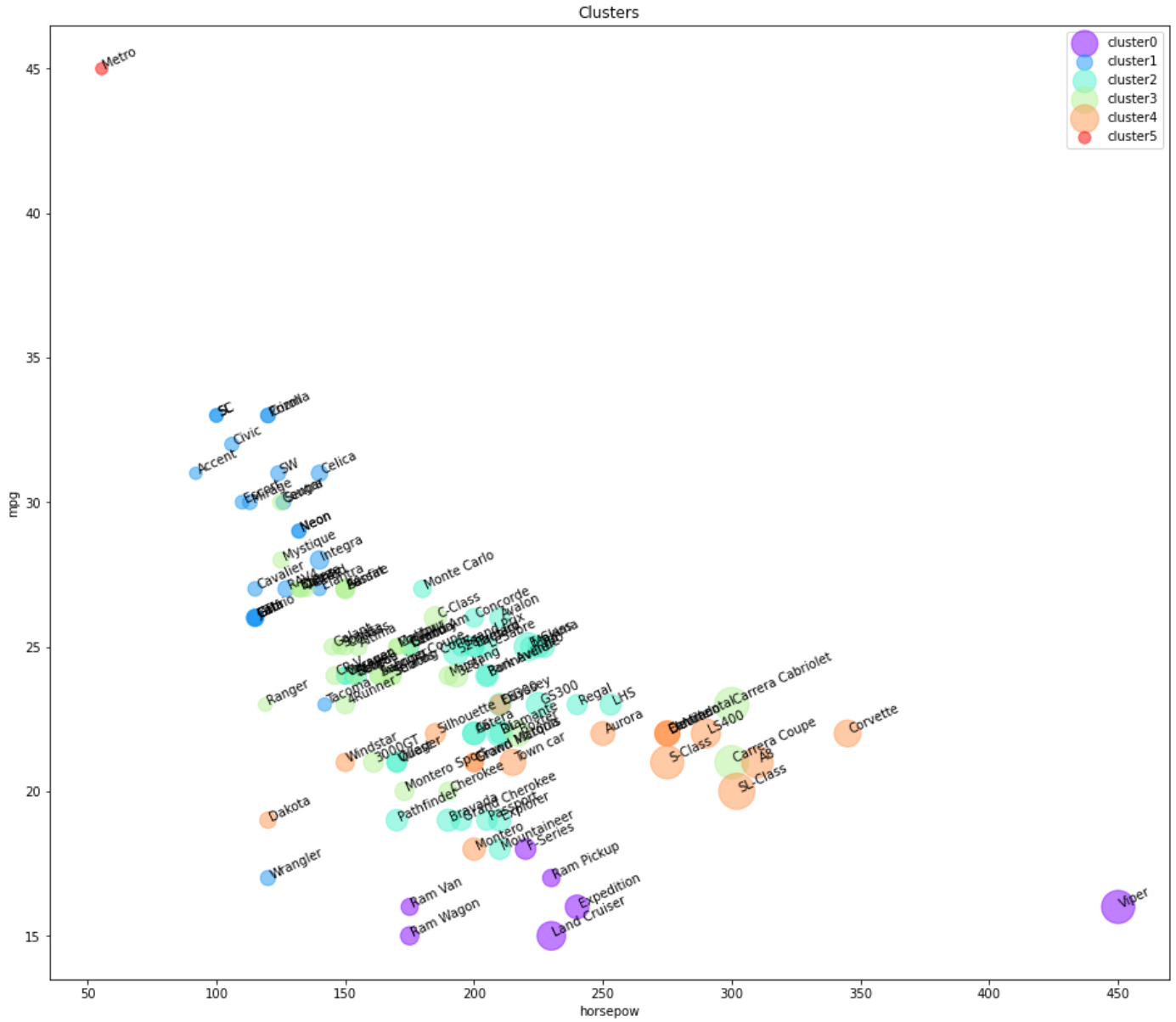
In [22]:

```
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
    plt.scatter(subset.horsepow, subset.mpg, s= subset.price*10, c=color, label='cluster
#    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 a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o 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 a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o 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 a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o 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 a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o 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 a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o 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 a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o specify the same RGB or RGBA value for all points.
```

Out[22]:

```
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:

In [23]:
```
pdf.groupby(['cluster_','type'])['cluster_'].count()
```

Out[23]:
```
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
```

In [24]:
```
agg_cars = pdf.groupby(['cluster_','type'])['horsepow','engine_s','mpg','price'].mean()
agg_cars
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_20152\3307995906.py:1: FutureWarning: Indexing
```

```
with multiple keys (implicitly converted to a tuple of keys) will be deprecated, use a l
ist instead.
  agg_cars = pdf.groupby(['cluster_','type'])['horsepow','engine_s','mpg','price'].mean
()
```

Out[24]:

cluster_	type	horsepow	engine_s	mpg	price
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

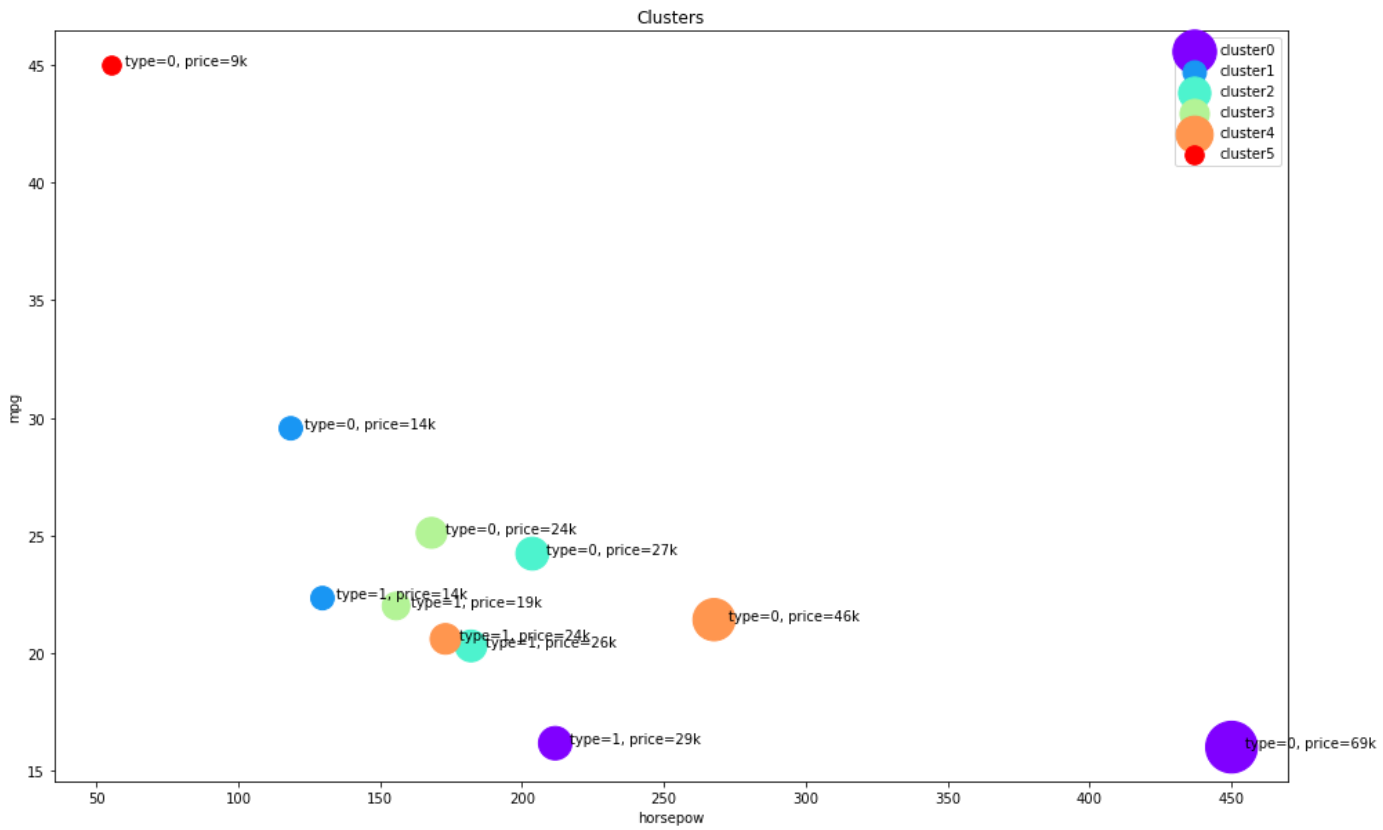
In [25]:

```
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='+
    plt.scatter(subset.horsepow, subset.mpg, s=subset.price*20, c=color, label='cluster'
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 a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o 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 a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o 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 a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o 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 a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o 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 a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o 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 a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o specify the same RGB or RGBA value for all points.
```

Out[25]:

```
Text(0, 0.5, 'mpg')
```


Clusters
type=0, price=9k
type=0, price=14k
type=0, price=24k
type=0, price=27k
type=1, price=14k
type=1, price=19k
type=1, price=24k
type=1, price=26k
type=0, price=46k
type=1, price=29k
type=0, price=69k
cluster0
cluster1
cluster2
cluster3
cluster4
cluster5
mpg
horsepow
45
40
35
30
25
20
15
50
100
150
200
250
300
350
400
450

In [ ]: