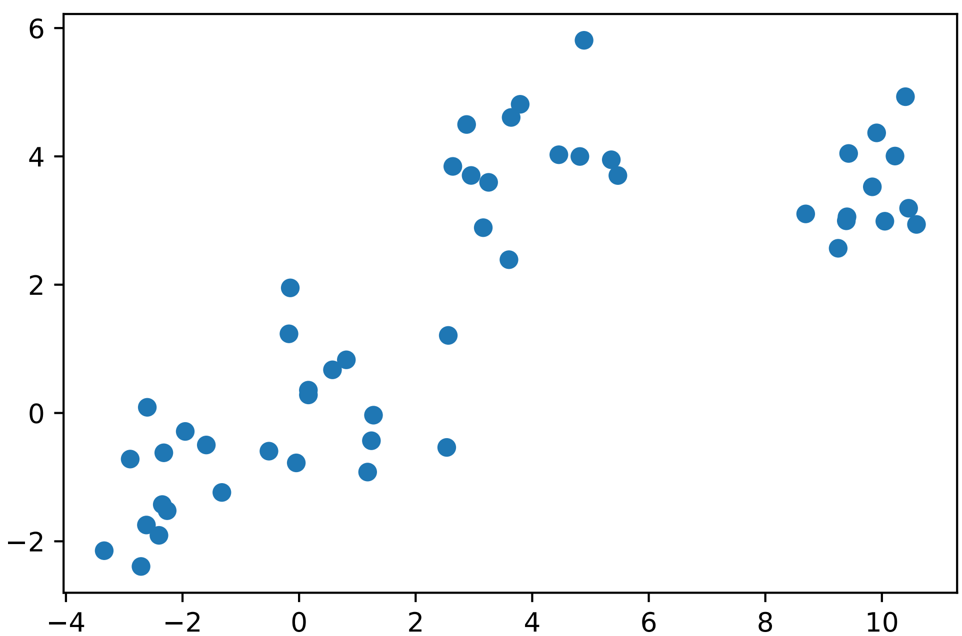
Demonstration of the model

In [3]:
```
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
```

```
C:\Users\Faraz Khoubsirat\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:143:
FutureWarning: The sklearn.datasets.samples_generator module is  deprecated in version
0.22 and will be removed in version 0.24. The corresponding classes / functions should i
nstead be imported from sklearn.datasets. Anything that cannot be imported from sklearn.
datasets is now part of the private API.
  warnings.warn(message, FutureWarning)
```

In [10]:
```
# sample datset
X1, y1 = make_blobs(n_samples=50, centers=[[4,4], [-2, -1], [1, 1], [10,4]], cluster_st
plt.scatter(X1[:, 0], X1[:, 1], marker='o')
```

Out[10]: `<matplotlib.collections.PathCollection at 0x25ceda293a0>`



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'

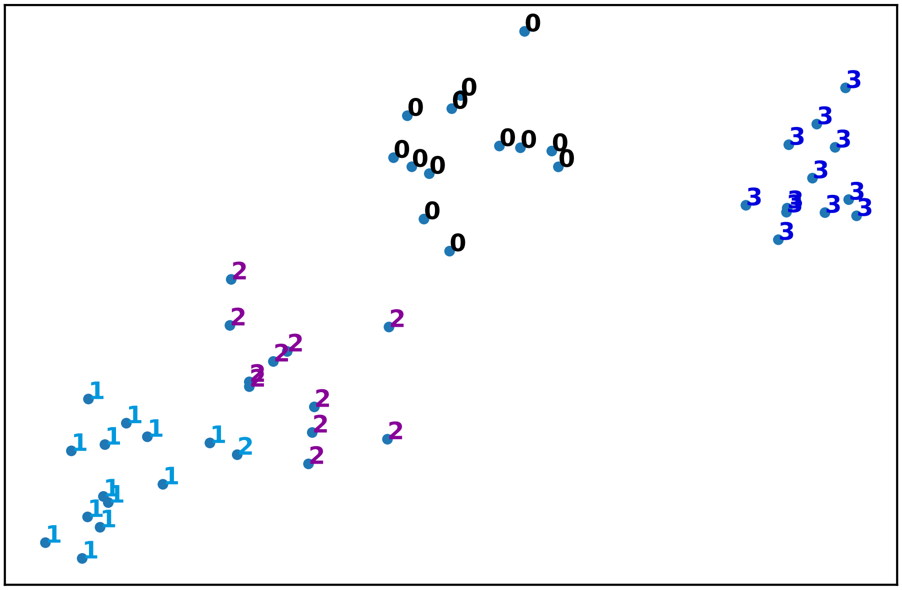
- **Note**: It is recommended you try everything with 'average' as well

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

In [15]:
```
agglom = AgglomerativeClustering(n_clusters = 4, linkage = 'average')
agglom.fit(X1,y1)
```

Out[15]:
```
AgglomerativeClustering(linkage='average', n_clusters=4)
```

In [16]:
```
# Plotting
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

Using the function **distance_matrix,** which requires **two inputs**. 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)

In [19]:
```
dist_matrix = distance_matrix(X1,X1)
print(dist_matrix)
```

```
[[0.         0.54505247 0.76722602 ... 0.40569215 0.65531324 0.56688677]
 [0.54505247 0.         0.44243693 ... 0.9364442  0.32403696 0.06527833]
 [0.76722602 0.44243693 0.         ... 1.16467248 0.12714398 0.37743613]
 ...
 [0.40569215 0.9364442  1.16467248 ... 0.         1.05762271 0.96575343]
 [0.65531324 0.32403696 0.12714398 ... 1.05762271 0.         0.26050846]
 [0.56688677 0.06527833 0.37743613 ... 0.96575343 0.26050846 0.        ]]
```

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

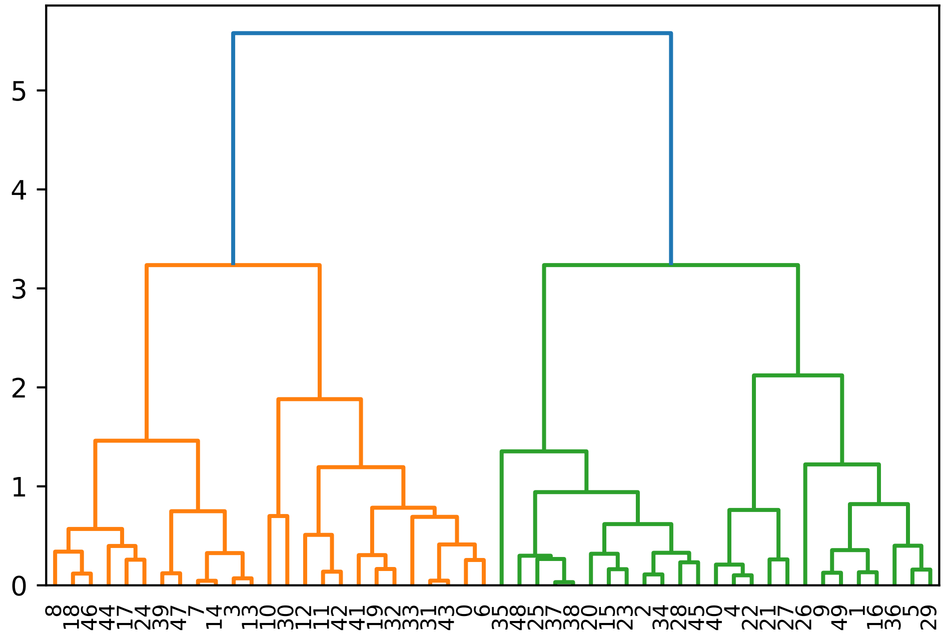
- The distance matrix
- 'complete' for complete linkage

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

In [20]:
```
Z = hierarchy.linkage(dist_matrix, 'complete')
```

```
<ipython-input-20-3814b774a052>:1: ClusterWarning: scipy.cluster: The symmetric non-nega
tive hollow observation matrix looks suspiciously like an uncondensed distance matrix
  Z = hierarchy.linkage(dist_matrix, 'complete')
```

In [21]:
```
dendro = hierarchy.dendrogram(Z)
```



Objective:

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.

Data Proccesing

In [1]:
```
# Get data
!wget -O cars_clus.csv https://cf-courses-data.s3.us.cloud-object-storage.appdomain.clo
```

In [6]:
```
# read data
df = pd.read_csv('cars_clus.csv')
df.head()
```

Out[6]:

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

In [7]:
```
# Data Cleaning
df[[ 'sales', 'resale', 'type', 'price', 'engine_s',
       'horsepow', 'wheelbas', 'width', 'length', 'curb_wgt', 'fuel_cap',
       'mpg', 'lnsales']] = df[['sales', 'resale', 'type', 'price', 'engine_s',
       'horsepow', 'wheelbas', 'width', 'length', 'curb_wgt', 'fuel_cap',
       'mpg', 'lnsales']].apply(pd.to_numeric, errors='coerce')
```

```
df = df.dropna()
df = df.reset_index(drop=True)
df.head(5)
```

Out[7]:

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

In [8]:

```
# Feature Selection
features = df[['engine_s',  'horsepow', 'wheelbas', 'width', 'length', 'curb_wgt', 'fue
features.head()
```

Out[8]:

	engine_s	horsepow	wheelbas	width	length	curb_wgt	fuel_cap	mpg
0	1.8	140.0	101.2	67.3	172.4	2.639	13.2	28.0
1	3.2	225.0	108.1	70.3	192.9	3.517	17.2	25.0
2	3.5	210.0	114.6	71.4	196.6	3.850	18.0	22.0
3	1.8	150.0	102.6	68.2	178.0	2.998	16.4	27.0
4	2.8	200.0	108.7	76.1	192.0	3.561	18.5	22.0

In [9]:

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

Out[9]:

```
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]])
```

Clustring using scipy

In [10]:

```
# Generating a distance matrix
import scipy
leng = feature_mtx.shape[0]
D = np.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
```

Out[10]:
```
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 [11]:
```python
# Agglomerative clustering
import pylab
import scipy.cluster.hierarchy
Z = hierarchy.linkage(D, 'complete')
```

```
<ipython-input-11-91d18692f7d5>:4: ClusterWarning: scipy.cluster: The symmetric non-nega
tive hollow observation matrix looks suspiciously like an uncondensed distance matrix
  Z = hierarchy.linkage(D, 'complete')
```

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:

In [12]:
```python
from scipy.cluster.hierarchy import fcluster
max_d = 3
clusters = fcluster(Z, max_d, criterion='distance')
clusters
```

Out[12]:
```
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:

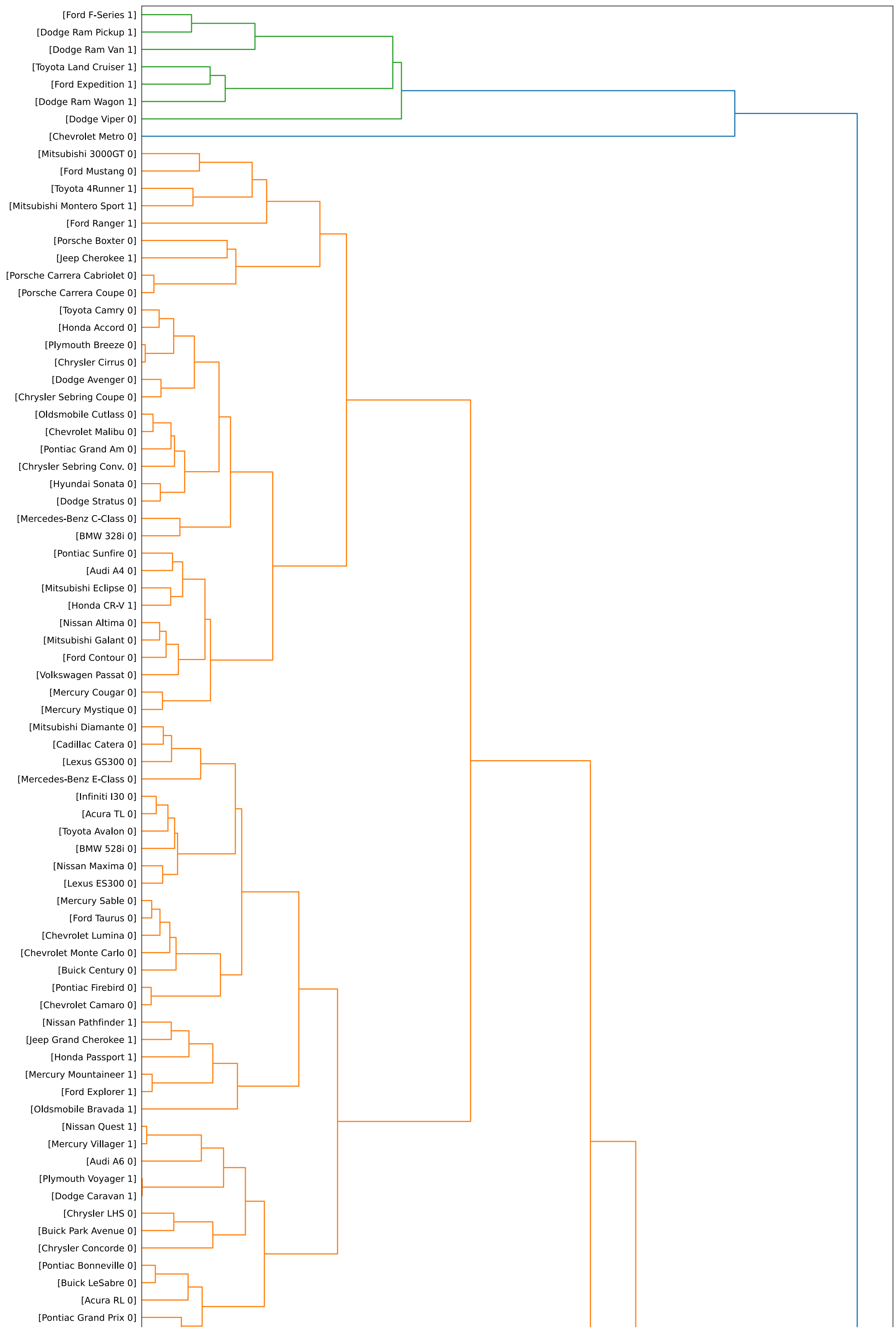
In [13]:
```python
from scipy.cluster.hierarchy import fcluster
k = 5
clusters = fcluster(Z, k, criterion='maxclust')
clusters
```

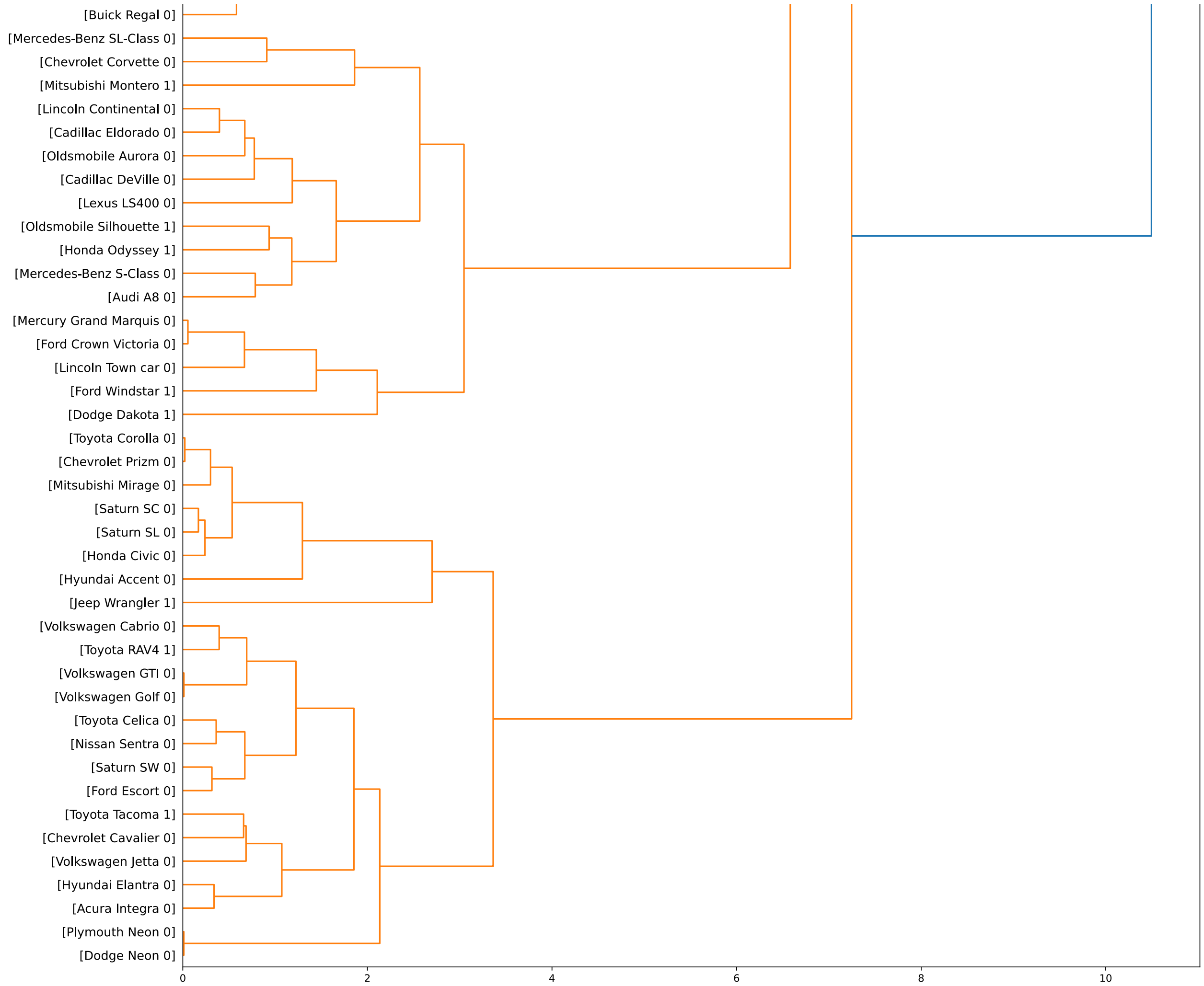
Out[13]:
```
array([1, 3, 3, 3, 3, 2, 3, 3, 3, 3, 3, 3, 2, 2, 3, 1, 3, 3, 3, 3, 2, 1,
       5, 3, 3, 3, 3, 3, 1, 3, 3, 4, 4, 4, 4, 2, 3, 1, 3, 3, 3, 2, 3, 2,
       4, 3, 4, 1, 3, 3, 3, 2, 1, 1, 3, 3, 1, 3, 3, 3, 3, 2, 2, 2, 1, 3,
       3, 3, 3, 2, 3, 3, 3, 3, 2, 3, 3, 3, 3, 2, 2, 1, 3, 3, 3, 3, 3, 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)
```

Visualizing data

In [14]:
```python
# Plotting the Model
fig = pylab.figure(figsize=(18,50))
def llf(id):
    return '[%s %s %s]' % (df['manufact'][id], df['model'][id], int(float(df['type'][id
```

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





Clustering using scikit-learn

In [15]:

```
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.        ]]
```

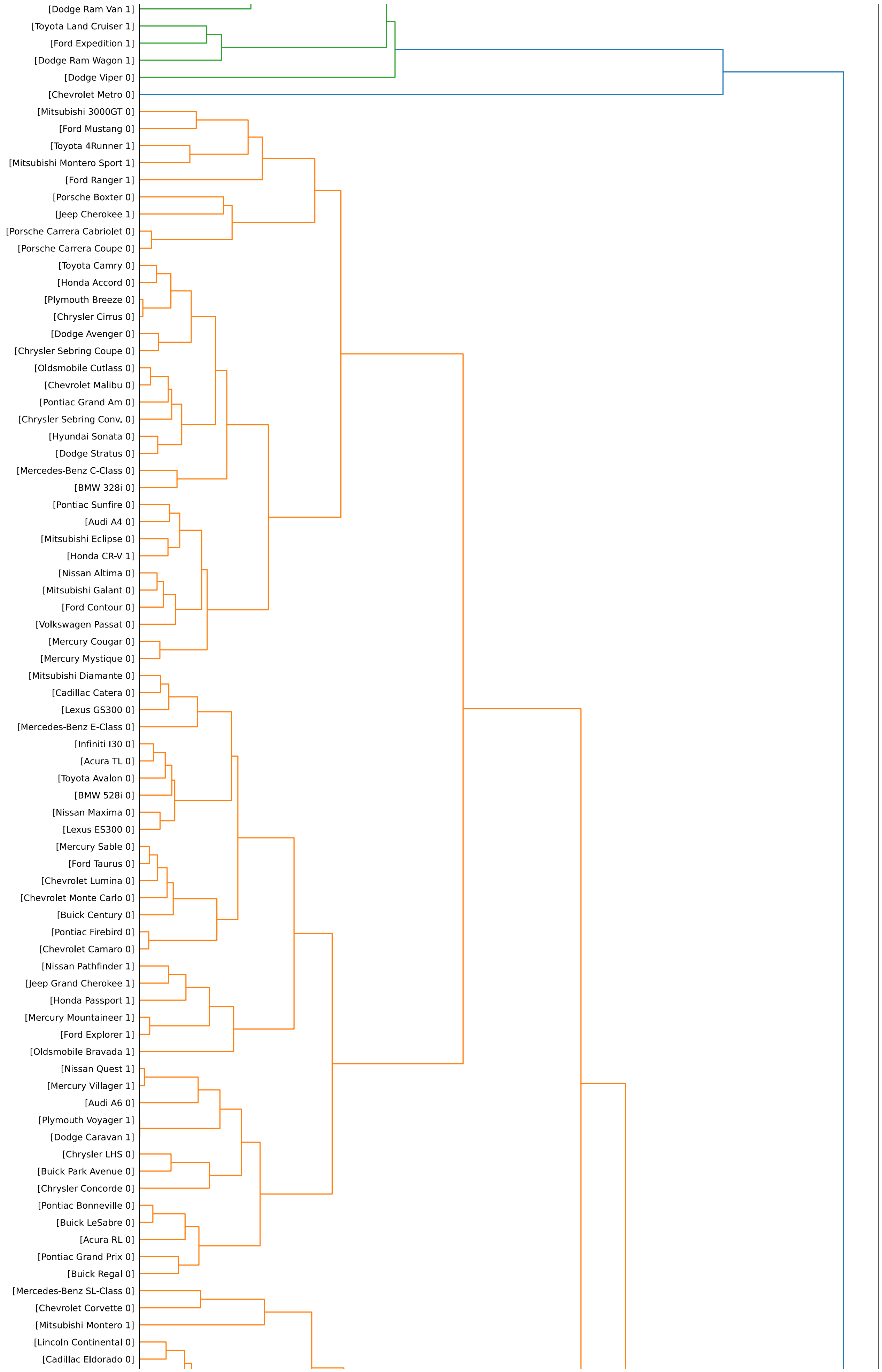
In [18]:

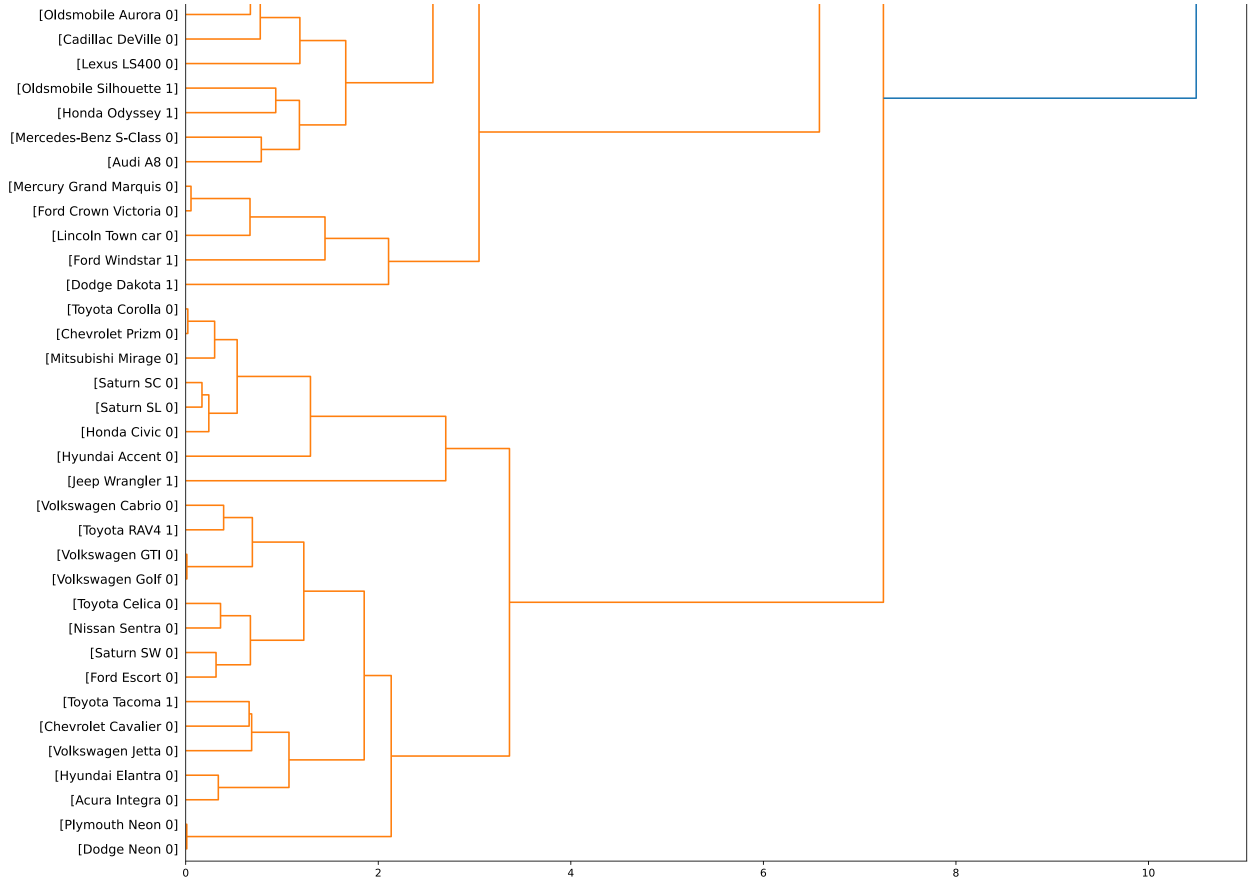
```
Z_using_dist_matrix = hierarchy.linkage(dist_matrix, 'complete')
```

```
<ipython-input-18-bf9ca02f569b>:1: ClusterWarning: scipy.cluster: The symmetric non-nega
tive hollow observation matrix looks suspiciously like an uncondensed 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]' % (df['manufact'][id], df['model'][id], int(float(df['type'][id

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


[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]



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.

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

agglom.labels_
```

```
C:\Users\Faraz Khoubsirat\anaconda3\lib\site-packages\sklearn\cluster\_agglomerative.py:
492: 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)
```

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]:
```
df['cluster_'] = agglom.labels_
```

```
df.head()
```

Out[21]:

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

Visualizing data

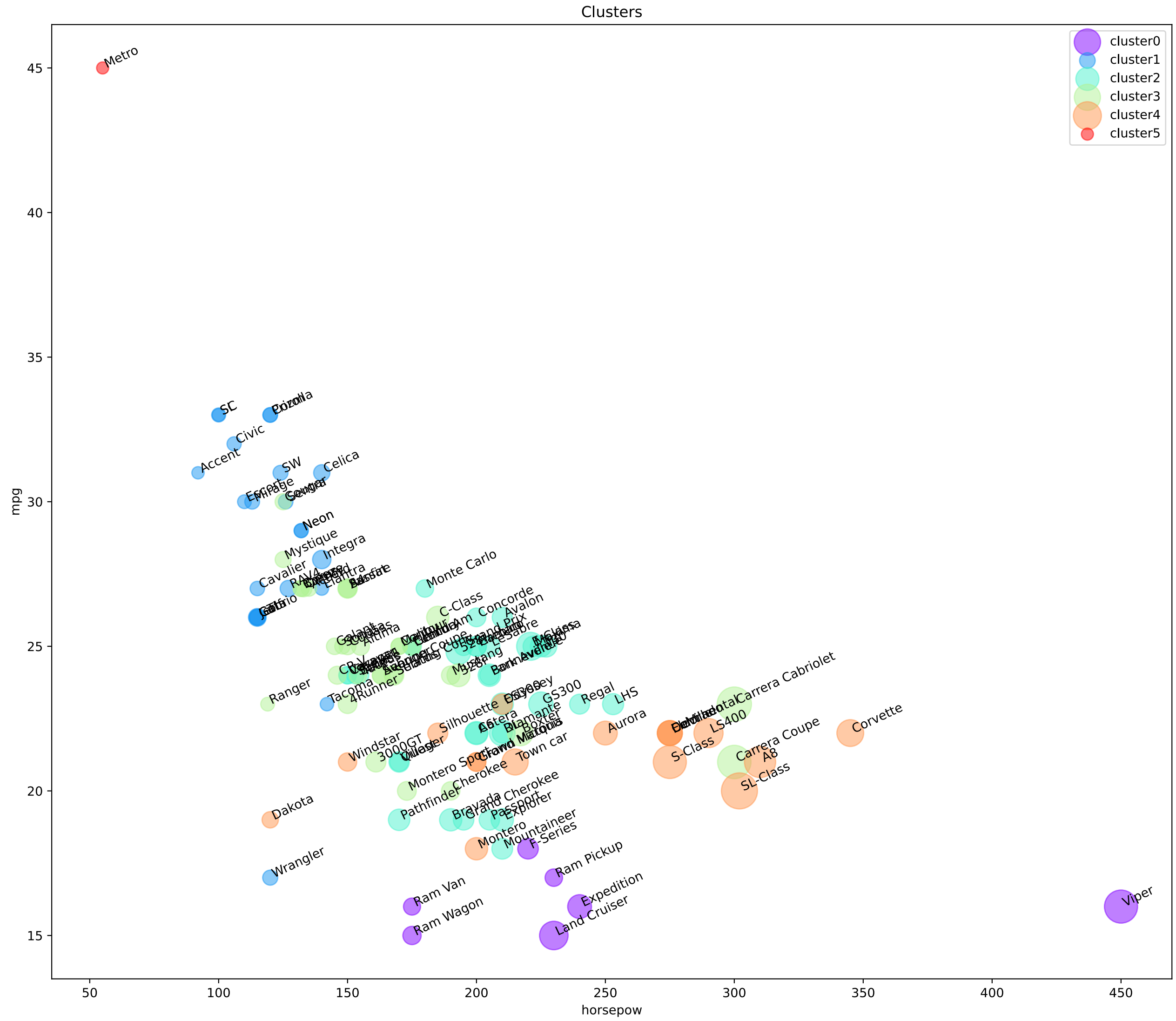
In [28]:

```
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 = df[df.cluster_ == label]
    for i in subset.index:
            plt.text(subset.horsepow[i], subset.mpg[i],str(subset['model'][i]), rotatio
    plt.scatter(subset.horsepow, subset.mpg, s= subset.price*10, c=color, label='cluste
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 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 a
s 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 a
s 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 a
s 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 a
s 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 a
s 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.
```

Out[28]:

```
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 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:

In [24]:
```
df.groupby(['cluster_','type'])['cluster_'].count()
```

Out[24]:
```
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 [25]:
```
agg_cars = df.groupby(['cluster_','type'])['horsepow','engine_s','mpg','price'].mean()
agg_cars
```

```
<ipython-input-25-a9701cdb999c>:1: FutureWarning: Indexing with multiple keys (implicitl
y converted to a tuple of keys) will be deprecated, use a list instead.
  agg_cars = df.groupby(['cluster_','type'])['horsepow','engine_s','mpg','price'].mean()
```

Out[25]:

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

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.

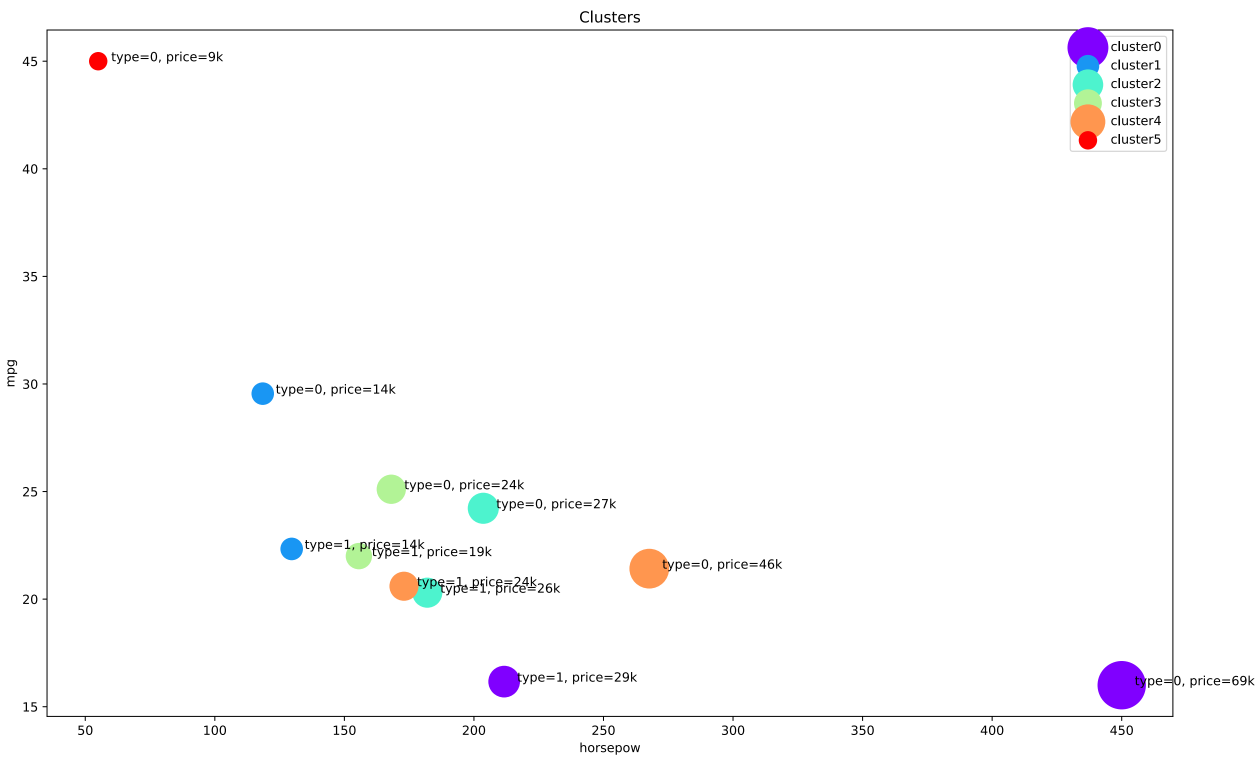
In [26]:

```
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 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 a
s 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 a
s 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 a
s 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 a
s 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 a
s 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.
```

Out[26]: `Text(0, 0.5, 'mpg')`



In [ ]: