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!

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.

Importing required packages

In [28]:
```
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
from sklearn.preprocessing import MinMaxScaler
import scipy
from scipy.cluster.hierarchy import fcluster
import pylab
from sklearn.metrics.pairwise import euclidean_distances
%matplotlib inline
```

Let's download and import the data on vahicle data using *pandas* `read_csv()` method.

Download Dataset

Reading the data

In [3]:
```
df = pd.read_csv("vahicle_clus.csv")

# take a look at the dataset
df.head()
```

Out[3]:

	manufact	model	sales	resale	type	price	engine_s	horsepow	wheelbas	width	length	curb_wgt	fuel_cap	mpg	lnsales	par
0	Acura	Integra	16.919	16.360	0.000	21.500	1.800	140.000	101.200	67.300	172.400	2.639	13.200	28.000	2.828	
1	Acura	TL	39.384	19.875	0.000	28.400	3.200	225.000	108.100	70.300	192.900	3.517	17.200	25.000	3.673	
2	Acura	CL	14.114	18.225	0.000	*null*	3.200	225.000	106.900	70.600	192.000	3.470	17.200	26.000	2.647	
3	Acura	RL	8.588	29.725	0.000	42.000	3.500	210.000	114.600	71.400	196.600	3.850	18.000	22.000	2.150	
4	Audi	A4	20.397	22.255	0.000	23.990	1.800	150.000	102.600	68.200	178.000	2.998	16.400	27.000	3.015	

In [4]:
```
df.columns
```

Out[4]:
```
Index(['manufact', 'model', 'sales', 'resale', 'type', 'price', 'engine_s',
       'horsepow', 'wheelbas', 'width', 'length', 'curb_wgt', 'fuel_cap',
       'mpg', 'lnsales', 'partition'],
      dtype='object')
```

In [5]:
```
df.dtypes
```

Out[5]:
```
manufact      object
model         object
sales         object
resale        object
type          object
price         object
engine_s      object
horsepow      object
wheelbas      object
width         object
length        object
curb_wgt      object
fuel_cap      object
mpg           object
lnsales       object
partition    float64
dtype: object
```

In [12]:
```
df.isna().any()
```

Out[12]:
```
manufact     False
model        False
sales        False
resale       False
type         False
price        False
engine_s     False
horsepow     False
wheelbas     False
width        False
length       False
curb_wgt     False
fuel_cap     False
mpg          False
lnsales      False
partition    False
dtype: bool
```

Data Cleaning

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

In [11]:
```
print ("Shape of dataset before cleaning: ", df.size)
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)
print ("Shape of dataset after cleaning: ", df.size)
df.head(5)
```

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

Out[11]:

	manufact	model	sales	resale	type	price	engine_s	horsepow	wheelbas	width	length	curb_wgt	fuel_cap	mpg	lnsales	partition
0	Acura	Integra	16.919	16.360	0.0	21.50	1.8	140.0	101.2	67.3	172.4	2.639	13.2	28.0	2.828	0.0
1	Acura	TL	39.384	19.875	0.0	28.40	3.2	225.0	108.1	70.3	192.9	3.517	17.2	25.0	3.673	0.0
2	Acura	RL	8.588	29.725	0.0	42.00	3.5	210.0	114.6	71.4	196.6	3.850	18.0	22.0	2.150	0.0
3	Audi	A4	20.397	22.255	0.0	23.99	1.8	150.0	102.6	68.2	178.0	2.998	16.4	27.0	3.015	0.0
4	Audi	A6	18.780	23.555	0.0	33.95	2.8	200.0	108.7	76.1	192.0	3.561	18.5	22.0	2.933	0.0

Feature selection

Let's select our feature set:

In [13]:
```
feature_set = df[['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.

In [15]:
```
x = feature_set.values #returns a numpy array
min_max_scaler = MinMaxScaler()
feature_mtx = min_max_scaler.fit_transform(x)
feature_mtx [0:5]
```

Out[15]:
```
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.

In [17]:
```
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\Meer Moazzam\AppData\Local\Temp\ipykernel_10212\1252677751.py:2: DeprecationWarning: scipy.zeros is de
precated and will be removed in SciPy 2.0.0, use numpy.zeros instead
  D = scipy.zeros([leng,leng])
```

Out[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.

In [21]:
```
Z = hierarchy.linkage(D, 'complete')
```

```
C:\Users\Meer Moazzam\AppData\Local\Temp\ipykernel_10212\2406838188.py:1: ClusterWarning: scipy.cluster: The sy
mmetric non-negative 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 [22]:
```
max_d = 3
clusters = fcluster(Z, max_d, criterion='distance')
clusters
```

Out[22]:
```
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)
```

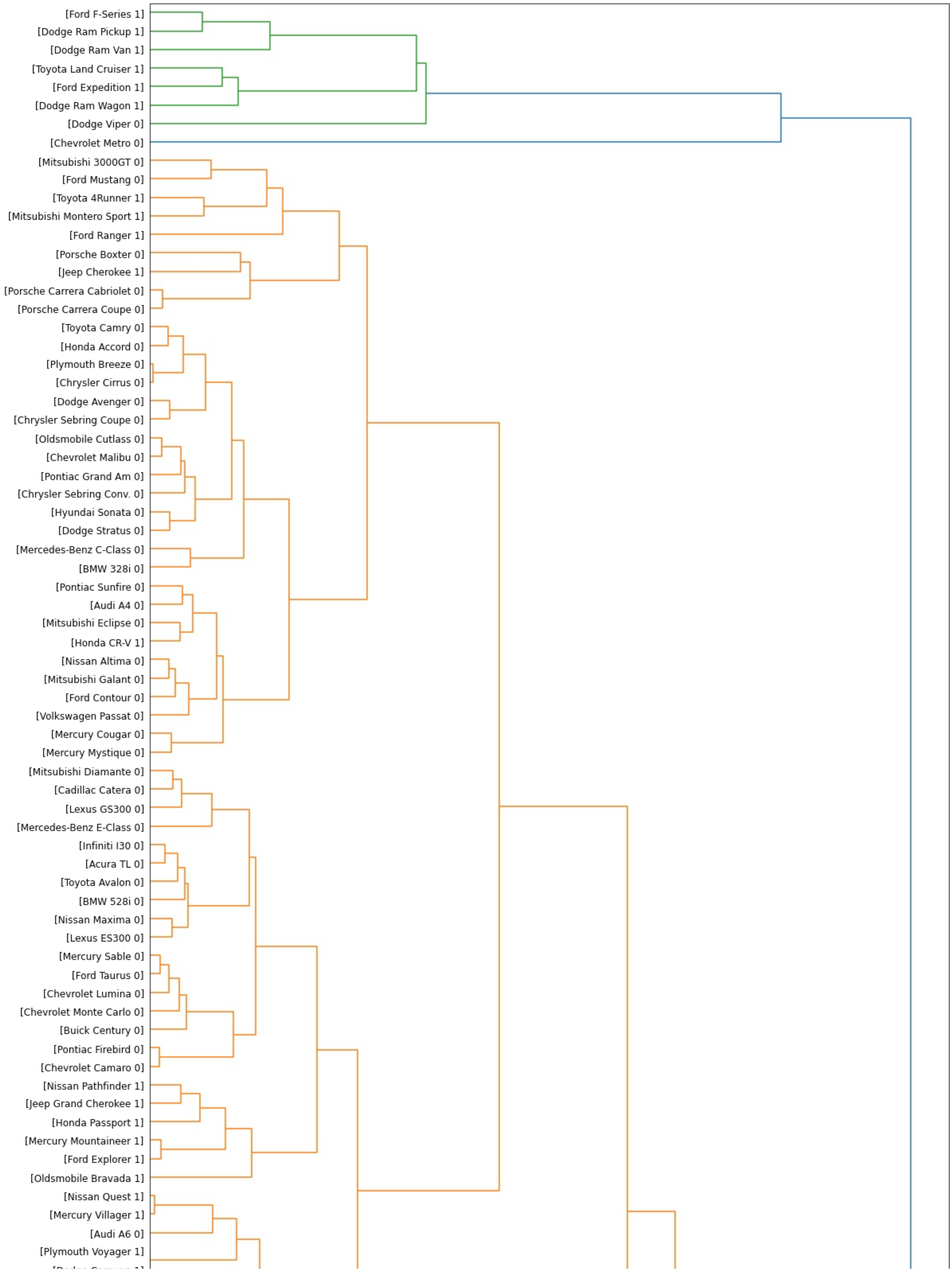
Also, you can determine the number of clusters directly:

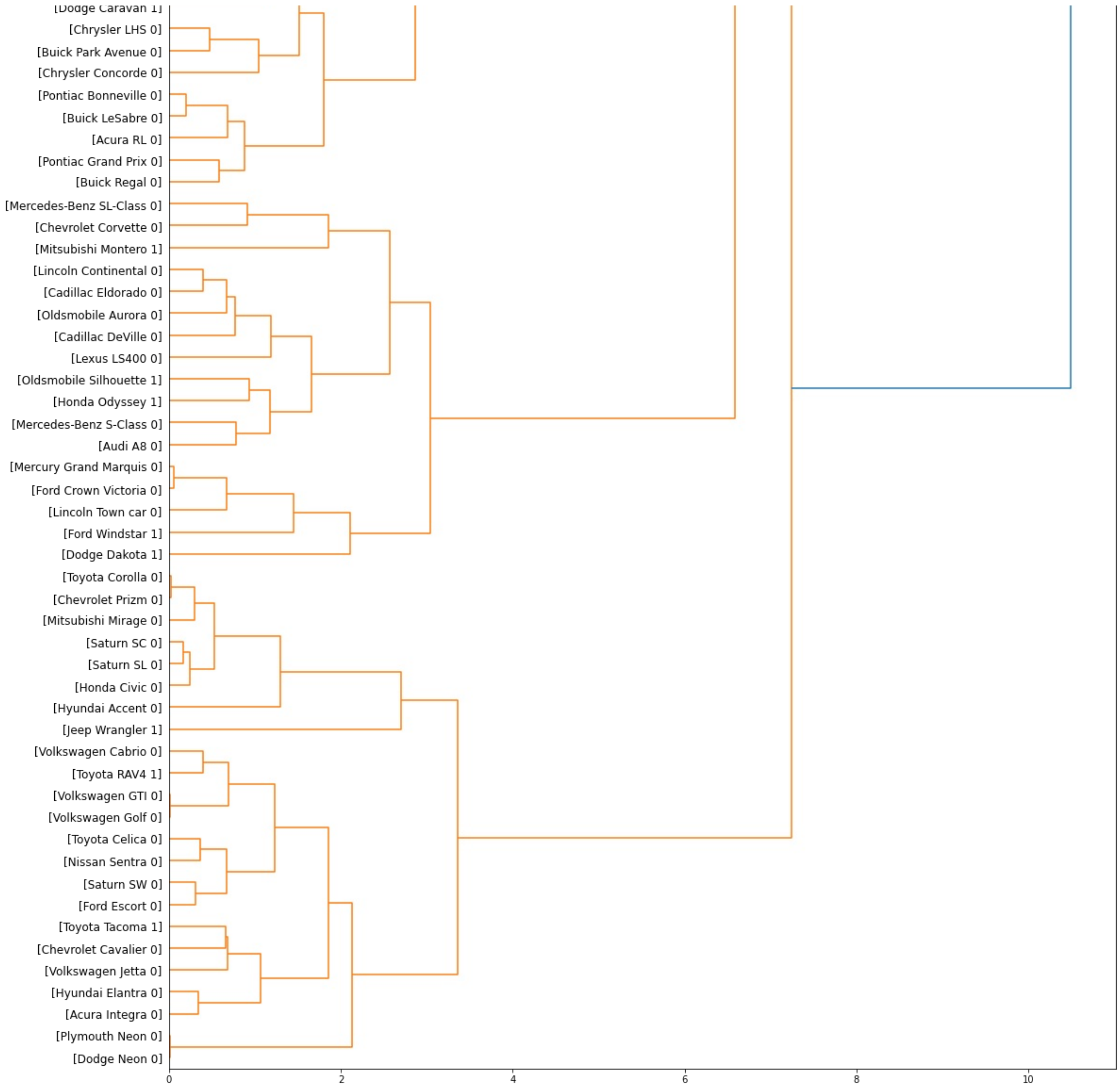
In [23]:
```
k = 5
clusters = fcluster(Z, k, criterion='maxclust')
clusters
```

Out[23]:
```
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)
```

Plot Dendrogram

In [27]:
```python
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 =12, orientation = 'righ
```





Clustering using scikit-learn

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

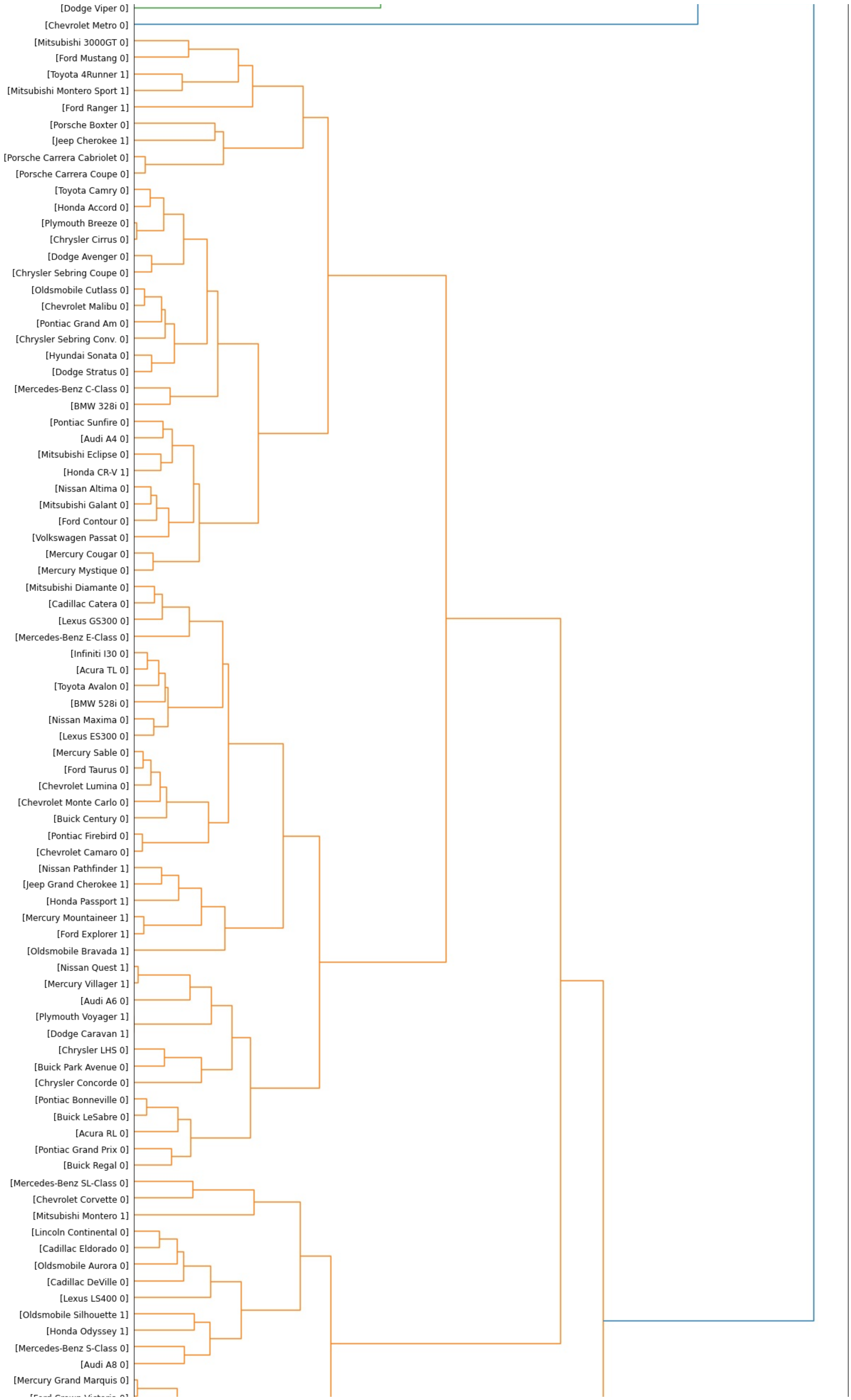
In [29]:
```
dist_matrix = euclidean_distances(feature_mtx,feature_mtx)
print(dist_matrix)
```

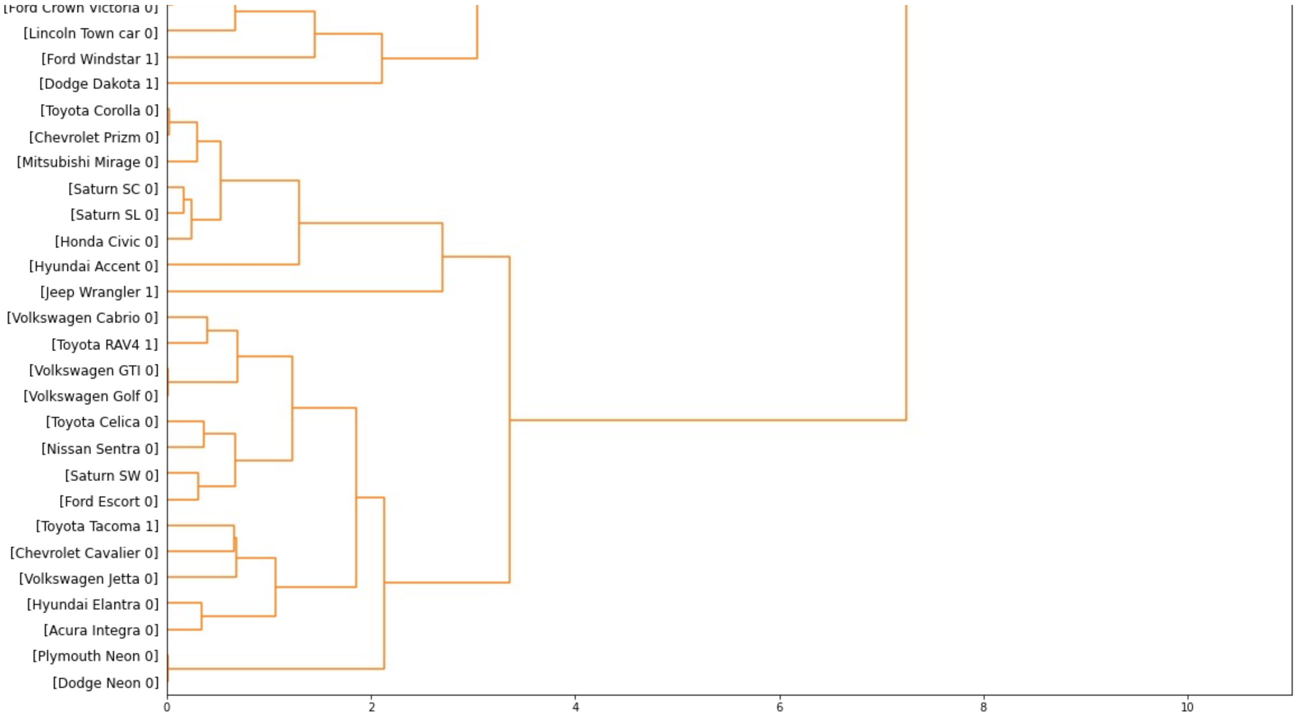
```
[[0.         0.57777143 0.75455727 ... 0.28530295 0.24917241 0.18879995]
 [0.57777143 0.         0.22798938 ... 0.36087756 0.66346677 0.62201282]
 [0.75455727 0.22798938 0.         ... 0.51727787 0.81786095 0.77930119]
 ...
 [0.28530295 0.36087756 0.51727787 ... 0.         0.41797928 0.35720492]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

In [30]:
```
Z_using_dist_matrix = hierarchy.linkage(dist_matrix, 'complete')
```

```
C:\Users\Meer Moazzam\AppData\Local\Temp\ipykernel_10212\1633147189.py:1: ClusterWarning: scipy.cluster: The sy
mmetric non-negative hollow observation matrix looks suspiciously like an uncondensed distance matrix
  Z_using_dist_matrix = hierarchy.linkage(dist_matrix, 'complete')
```

In [31]:
```
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=0, leaf_font_size =12, o
```


[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]



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 [32]:

```
agglom = AgglomerativeClustering(n_clusters = 6, linkage = 'complete')
agglom.fit(dist_matrix)

agglom.labels_
```

```
C:\Users\Meer Moazzam\Anaconda3\lib\site-packages\sklearn\cluster\_agglomerative.py:542: 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[32]:

```
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)
```

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

In [35]:

```
df['cluster_label'] = agglom.labels_
df.head()
```

Out[35]:

	manufact	model	sales	resale	type	price	engine_s	horsepow	wheelbas	width	length	curb_wgt	fuel_cap	mpg	lnsales	partition
0	Acura	Integra	16.919	16.360	0.0	21.50	1.8	140.0	101.2	67.3	172.4	2.639	13.2	28.0	2.828	0.0
1	Acura	TL	39.384	19.875	0.0	28.40	3.2	225.0	108.1	70.3	192.9	3.517	17.2	25.0	3.673	0.0
2	Acura	RL	8.588	29.725	0.0	42.00	3.5	210.0	114.6	71.4	196.6	3.850	18.0	22.0	2.150	0.0
3	Audi	A4	20.397	22.255	0.0	23.99	1.8	150.0	102.6	68.2	178.0	2.998	16.4	27.0	3.015	0.0
4	Audi	A6	18.780	23.555	0.0	33.95	2.8	200.0	108.7	76.1	192.0	3.561	18.5	22.0	2.933	0.0

In [39]:

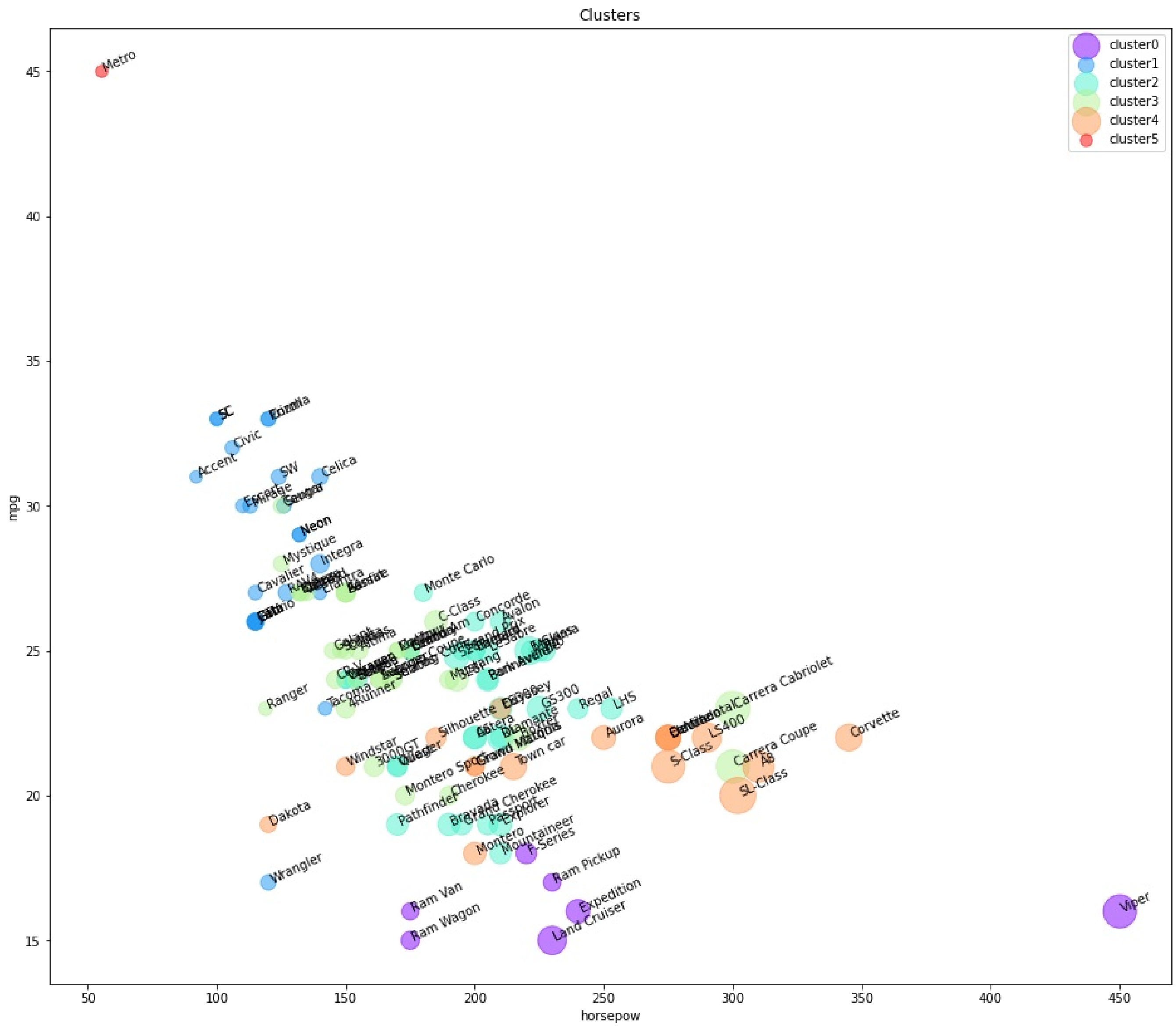
```
import matplotlib.cm as cm
n_clusters = max(agglom.labels_)+1
colors = cm.rainbow(np.linspace(0, 1, n_clusters))
cluster_labels = list(range(0, n_clusters))

# 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 == 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)
```

```python
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 ha
ve 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 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 ha
ve 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 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 ha
ve 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 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 ha
ve 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 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 ha
ve 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 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 ha
ve 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 to specify the same RGB or RGBA value for all points.
```

Out[39]:

```
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 [43]:

```python
df.groupby(['cluster_label','type'])['cluster_label'].count()
```

Out[43]:
```
cluster_label  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_label, dtype: int64
```

Now we can look at the characteristics of each cluster:

In [45]:
```
agg_vahicles = df.groupby(['cluster_label','type'])['horsepow','engine_s','mpg','price'].mean()
agg_vahicles
```

```
C:\Users\Meer Moazzam\AppData\Local\Temp\ipykernel_10212\3385367551.py:1: FutureWarning: Indexing with multiple
keys (implicitly converted to a tuple of keys) will be deprecated, use a list instead.
  agg_vahicles = df.groupby(['cluster_label','type'])['horsepow','engine_s','mpg','price'].mean()
```

Out[45]:

cluster_label	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.

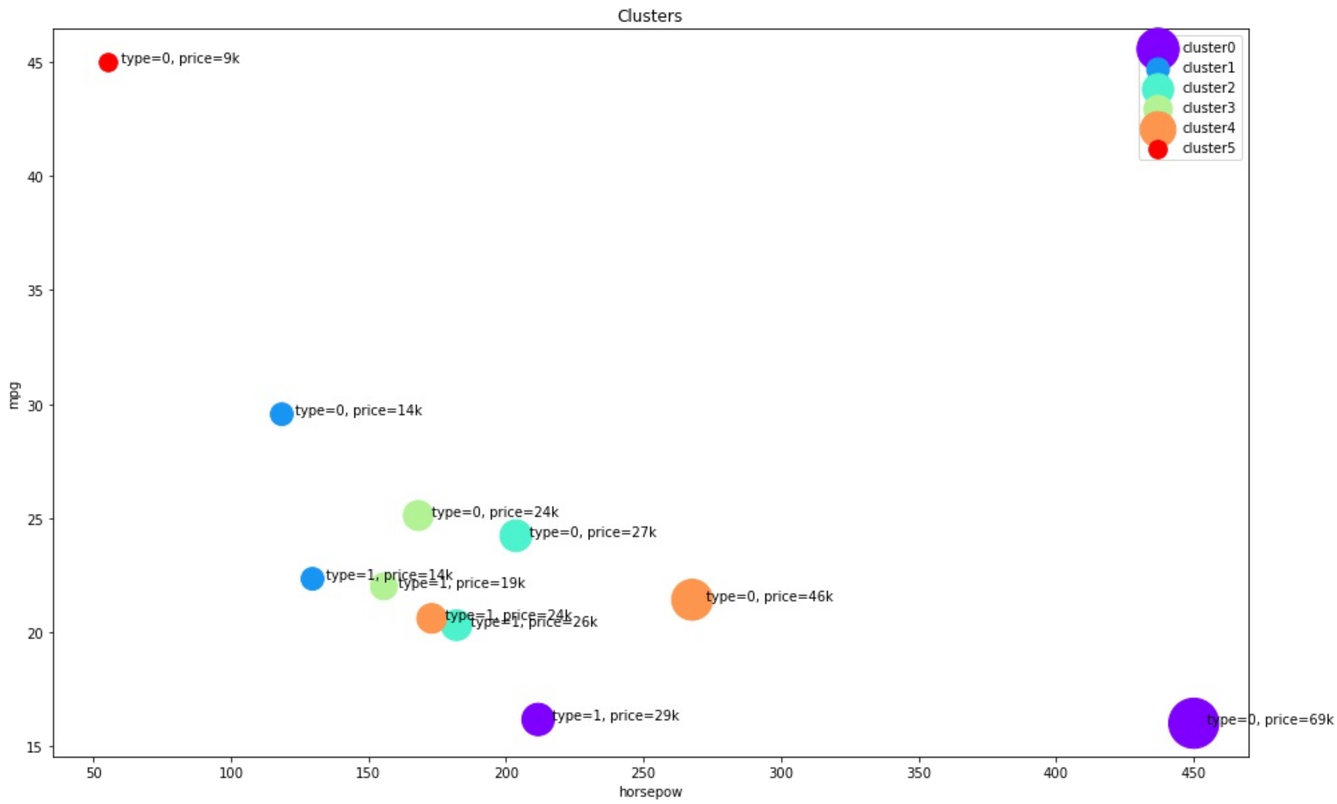
Please notice that we did not use **type** and **price** of cars in the clustering process, but Hierarchical clustering could forge the clusters and discriminate them with quite a high accuracy.

In [46]:
```
plt.figure(figsize=(16,10))
for color, label in zip(colors, cluster_labels):
    subset = agg_vahicles.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
    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 ha
ve 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 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 ha
ve 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 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 ha
ve 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 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 ha
ve 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 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 ha
ve 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 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 ha
ve 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 to specify the same RGB or RGBA value for all points.
```

Out[46]:

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



Thank you

Author

Moazzam Ali
