Clustering Jerárquico

Objetivos

- Utilizar scikit-learn para el clustering jerárquico.
- Crear dendogramas para visualizar los datos.

Tabla de contenido

Clustering jerárquico- Aglomerativo

Recuerde que el aglomerativo es el enfoque bottom up. Utilizaremos Complete Linkage como criterio de enlace. **Nota**: Puede intentar utilizar Avergare Linkage donde está Complete Linkage para ver la diferencia.

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.samples_generator import make_blobs
%matplotlib inline
```

```
C:\Users\marco\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:143: FutureWar
ning: 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 instead
be imported from sklearn.datasets. Anything that cannot be imported from sklearn.datas
ets is now part of the private API.
  warnings.warn(message, FutureWarning)
```

Generando datos aleatorios

Generaremos un conjunto de datos usando la clase **make_blobs**. Los parámetros de entrada son:

- **n_samples**: El número de puntos igualmente divididos entre clusters
    - Seleccione un número entre 10 y 1500
- **centers**: El número de centros a generar, o las ubicaciones centrales fijas
    - Seleccione coordenadas x e y para generar los centros. Tenga 1-10 centros (ej. centers=[[1,1], [2,5]])
- **cluster_std**: La desviación estándar de los clusters.
    - Seleccione un número entre 0.5-1.5

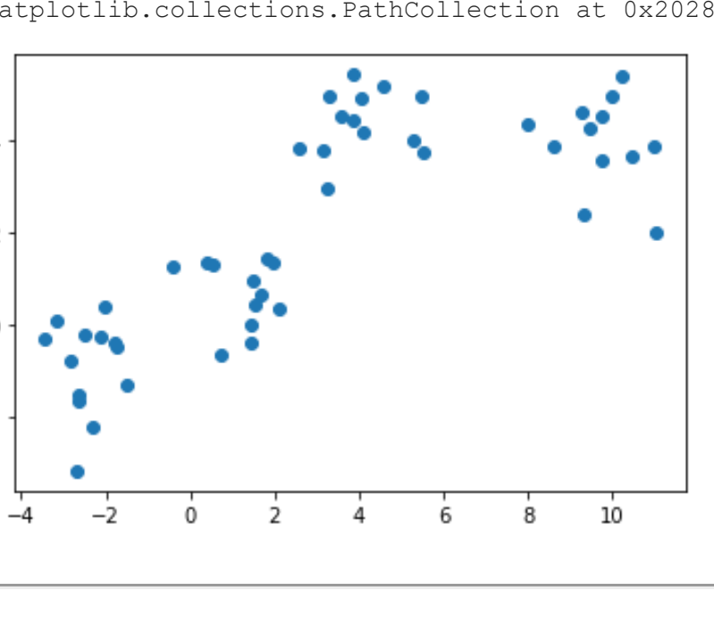
  Guarde el resultado en **X1** e **y1**.

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

Gráfico de dispersión de los datos generados

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

Out[3]:
```
<matplotlib.collections.PathCollection at 0x20288808700>
```



Clustering aglomerativo

La clase clustering aglomerativo requiere de 2 parámetros:

- **n_clusters**: El número de clusters a formar así como el número de centroides a generar
    - El valor será: 4
- **linkage**: Qué criterio de enlace se utilizará. El criterio de enlace determina qué distancia se usará entre conjuntos de observaciones. The algorithm will merge the pairs of cluster that minimize this criterion.
    - El valor será: 'complete'
    - **Nota**: Se recomienda probar también "Average"

  El resultado se guardará en una variable llamada **agglom**

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

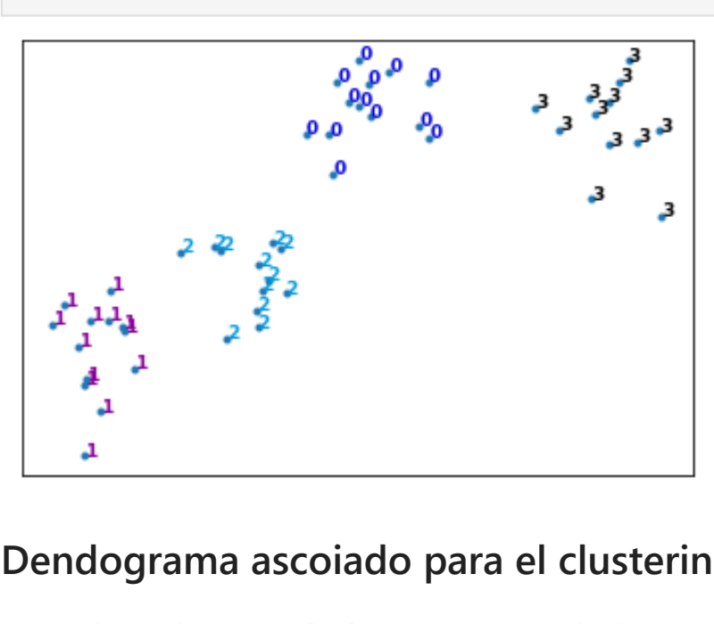
Ajustamos el modelo con X1 e y1 generados como se vio arriba.

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

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

Veamos el cluster.

In [6]:
```
# Creamos una figura de tamaño 6x4 pulgadas.
plt.figure(figsize=(6,4))

# Estas 2 lineas se utilizan para escalar los puntos de datos
# de otro modo los puntos quedarán muy dispersos entre sí

# Creamos un tango mínimo y máximo para X1
x_min, x_max = np.min(X1, axis=0), np.max(X1, axis=0)

# Obtenemos la distancia promedio para X1
X1 = (X1 - x_min) / (x_max - x_min)

# Este loop despliega los puntos de datos
for i in range(X1.shape[0]):
    # Reemplazamos los puntos de datos con sus respectivos valores de cluster
    # (ejemplo 0) y su color es codificado con un colormap
    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})

#
plt.xticks([])
plt.yticks([])
#plt.axis('off')



# Desplegamos el gráfico original antes del clustering
plt.scatter(X1[:, 0], X1[:, 1], marker='.')
# Desplegamos el gráfico
plt.show()
```



Dendograma ascoiado para el clustering jerárquico aglomerativo

Recuerde que la **matriz de distancia** contiene la distancia de cada punto a cada otro punto del dataset.

Utilice la función **distance_matrix** que requiere 2 entradas. Use la matriz de características, X1, como esas entradas y guarde la matriz de distancia en una variable llamada dist_matrix. Recuerde que la los valores de distancia son simétricos, con una diagonal de 0's. Esta es una forma de verificar que los valores son correctos.

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

```
[[0.         0.43459509 0.336276   ... 0.67127257 0.56005593 0.05435269]
 [0.43459509 0.         0.69952069 ... 0.56936145 0.99190959 0.38282201]
 [0.336276   0.69952069 0.         ... 1.00580006 0.38060122 0.38276295]
 ...
 [0.67127257 0.56936145 1.00580006 ... 0.         1.1226861  0.63105613]
 [0.56005593 0.99190959 0.38060122 ... 1.1226861  0.         0.61392395]
 [0.05435269 0.38282201 0.38276295 ... 0.63105613 0.61392395 0.        ]]
```

Utilice la clase linkage de hierarchy, pásele los parámetros:

- Matriz de distancia
- 'complete' para "Complete Linkage"

Guarde el resultado en la variable **Z**

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

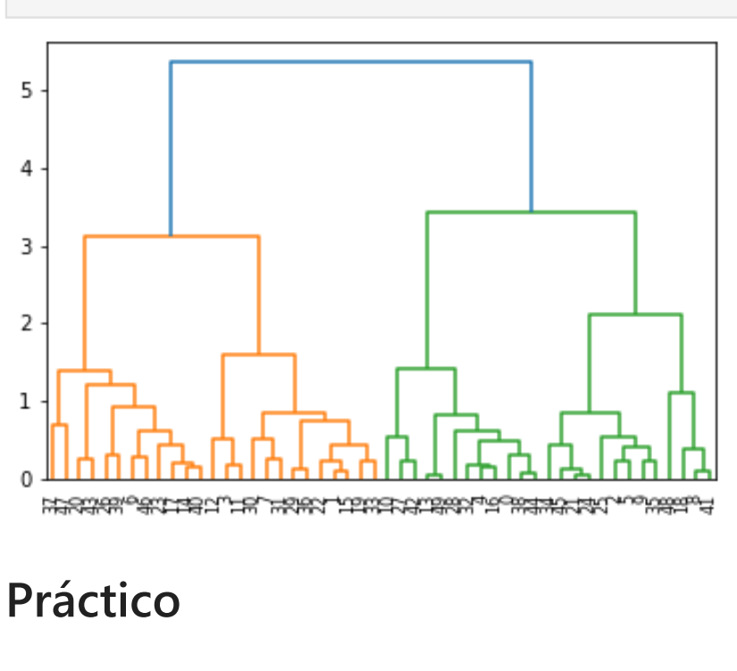
```
<ipython-input-8-3814b774a052>:1: ClusterWarning: scipy.cluster: The symmetric non-neg
ative hollow observation matrix looks suspiciously like an uncondensed distance matrix
  Z = hierarchy.linkage(dist_matrix, 'complete')
```

El clustering jerárquico es típicamente visualizado como un dendograma tal cual se muestra en la celda siguiente. Cada combinación es representada por una línea horizontal. La coordenada y de la línea horizontal es ls similaridad de los 2 clusters combinados, donde las ciudades son vistas como clusters singleton. Al movernos desde la capa inferior hacia el nodo superior, el dendograma nos permite reconstruir la historia de combinaciones que resultaron en el cluster mostrado.

Luego, guardaremos el dendograma en una variable llamada dendro. Al hacerlo, el dendograma será desplegado. Utilizaremos **dendogram** de la clase hierarchy y le pasaremos el parámetro:

- Z

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

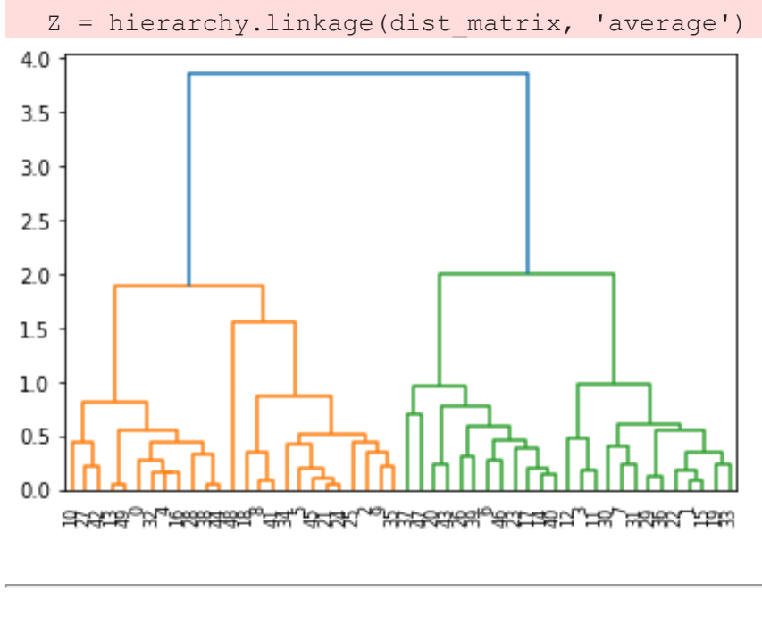


Práctico

Utilicemos **average** como linkage para ver cómo cambia el dendograma.

In [10]:
```
Z = hierarchy.linkage(dist_matrix, 'average')
dendro = hierarchy.dendrogram(Z)
```

```
<ipython-input-10-90bcb3750f27>:1: ClusterWarning: scipy.cluster: The symmetric non-ne
gative hollow observation matrix looks suspiciously like an uncondensed distance matri
x
  Z = hierarchy.linkage(dist_matrix, 'average')
```



Clustering en el dataset de vehículos

Imagine que un fabricante de automóviles ha desarrollado prototipos para un nuevo vehículo. Antes de introducir el modelo nuevo, el fabricante quiere determinar qué vehículos existentes en el mercado son los más parecidos al prototipo; esto es, cómo los vehículos pueden ser agrupados, qué grupo es el más similar a este modelo y por tanto contra qué modelos competirá.

Nuestro objetivo es usar métodos de clustering para encontrar los clusters más distintivos de vehículos. Resumirá los vehículos existentes y ayudará a los fabricantes a tomar una decisión sobre el suministro de nuevos modelos.

Descargando los datos

In [11]:
```
#!wget -O cars_clus.csv https://cf-courses-data.s3.us.cloud-object-storage.appdomain.c

import urllib.request
url = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloper
filename = 'cars_clus.csv'
urllib.request.urlretrieve(url, filename)
```

Out[11]:
```
('cars_clus.csv', <http.client.HTTPMessage at 0x20288d23f70>)
```

Leyendo los datos

In [12]:
```
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)
```

Out[12]:

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

Limpieza de datos

Limpiemos el dataset eliminando las filas que contienen valores null:

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

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

Selección de características

In [14]:
```
featureset = pdf[['engine_s',  'horsepow', 'wheelbas', 'width', 'length', 'curb_wgt',
```

Normalización

MinMaxScaler transforma características escalando cada característica en un rango dado. Es por defecto (0,1). Esto es, el estimador escala y traslada cada característica individualmente de modo que queda entre 0 y 1.

In [15]:
```
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[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 usando Scipy

En esta parte utilizaremos el paquete Scipy.

Primero, calculemos la matriz de distancia.

In [16]:
```
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])
```

```
<ipython-input-16-643b1eed25c3>:3: DeprecationWarning: scipy.zeros is deprecated and w
ill be removed in SciPy 2.0.0, use numpy.zeros instead
  D = scipy.zeros([leng,leng])
```

En el clustering aglomerativo, en cada iteración, el algoritmo debe actualizar la matriz de distancia para reflejar la distancia de los clusters recientemente formados con los restantes. Los siguientes métodos están soportados en Scipy para calcular las distancias entre clusters:

```
- single
- complete
- average
- weighted
- centroid
```

Utilizaremos **complete**

In [17]:
```
import pylab
import scipy.cluster.hierarchy
Z = hierarchy.linkage(D, 'complete')
```

```
<ipython-input-17-8655000d21de>:3: ClusterWarning: scipy.cluster: The symmetric non-ne
gative hollow observation matrix looks suspiciously like an uncondensed distance matri
x
  Z = hierarchy.linkage(D, 'complete')
```

Esencialmente, el clustering jerárquico NO requiere un número pre-establecido de clusteres. Sin embargo, en algunas aplicaciones, queremos una partición de clusteres disjuntos como en el "flat clustering". Entonces, puede utilizar una línea de corte:

In [18]:
```
from scipy.cluster.hierarchy import fcluster
max_d = 3
clusters = fcluster(Z, max_d, criterion='distance')
clusters
```

Out[18]:
```
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)
```

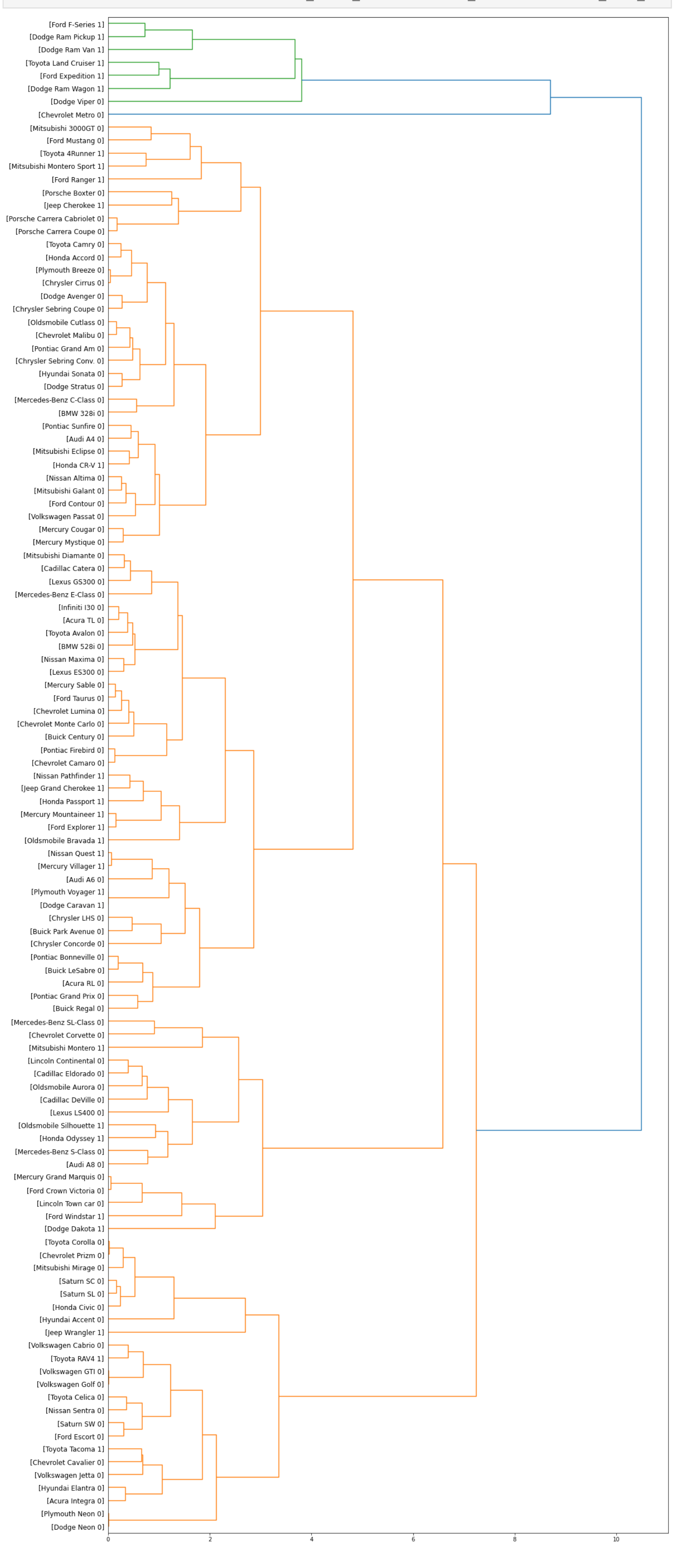
También puede determinar el número de clusters directamente:

In [19]:
```
from scipy.cluster.hierarchy import fcluster
k = 5
clusters = fcluster(Z, k, criterion='maxclust')
clusters
```

Out[19]:
```
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, 1, 3, 3, 1, 1, 1,
       3, 4, 1, 1, 3, 1, 1], dtype=int32)
```

Ahora, imprimimos el dendograma:

In [20]:
```
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,  leaf_label_func=llf, leaf_rotation=0, leaf_font_size
```



Clustering utilizando scikit-learn

Hagamos lo mismo, pero ahora con el paquete scikit-learn:

In [21]:
```
dist_matrix = distance_matrix(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.        ]]
```

Ahora podemos usar la función "AgglomerativeClustering" de scikit-learn que realiza un clustering jerárquico utilizando un enfoque bottom up. El criterio de enlace determina la métrica utilizada para la estrategia de combinación:

- Ward minimiza la suma de las diferencias cuadráticas entre todos los clusters. Es un enfoque de varianza mínima y en este sentido es similar a la función objetivo de k-means pero atacado con un enfoque de clustering aglomerativo.
- Maximun o complete linkage minimiza la distancia máxima entre observaciones de pares de clusters.
- Average minimiza el promedio de distancias entre observaciones de pares de clusters.

In [22]:
```
agglom = AgglomerativeClustering(n_clusters = 6, linkage = 'complete')
agglom.fit(feature_mtx)
agglom.labels_
```

Out[22]:
```
array([1, 2, 2, 1, 2, 3, 1, 2, 2, 2, 2, 2, 3, 3, 2, 1, 1, 2, 2, 2, 5, 1,
       4, 1, 1, 2, 1, 2, 1, 1, 1, 5, 0, 0, 0, 3, 2, 1, 2, 1, 2, 3, 2, 3,
       0, 3, 0, 1, 1, 1, 2, 3, 1, 1, 2, 1, 2, 2, 2, 3, 3, 3, 1, 1, 1,
       1, 2, 1, 2, 2, 1, 1, 2, 3, 2, 3, 1, 2, 3, 5, 1, 1, 2, 3, 2, 1, 3,
       2, 3, 1, 1, 2, 1, 1, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1,
       2, 0, 1, 1, 1, 1, 1], dtype=int64)
```

Agregamos un nuevo campo al dataframe para mostrar el cluster de cada fila:

In [23]:
```
pdf['cluster_'] = agglom.labels_
pdf.head()
```

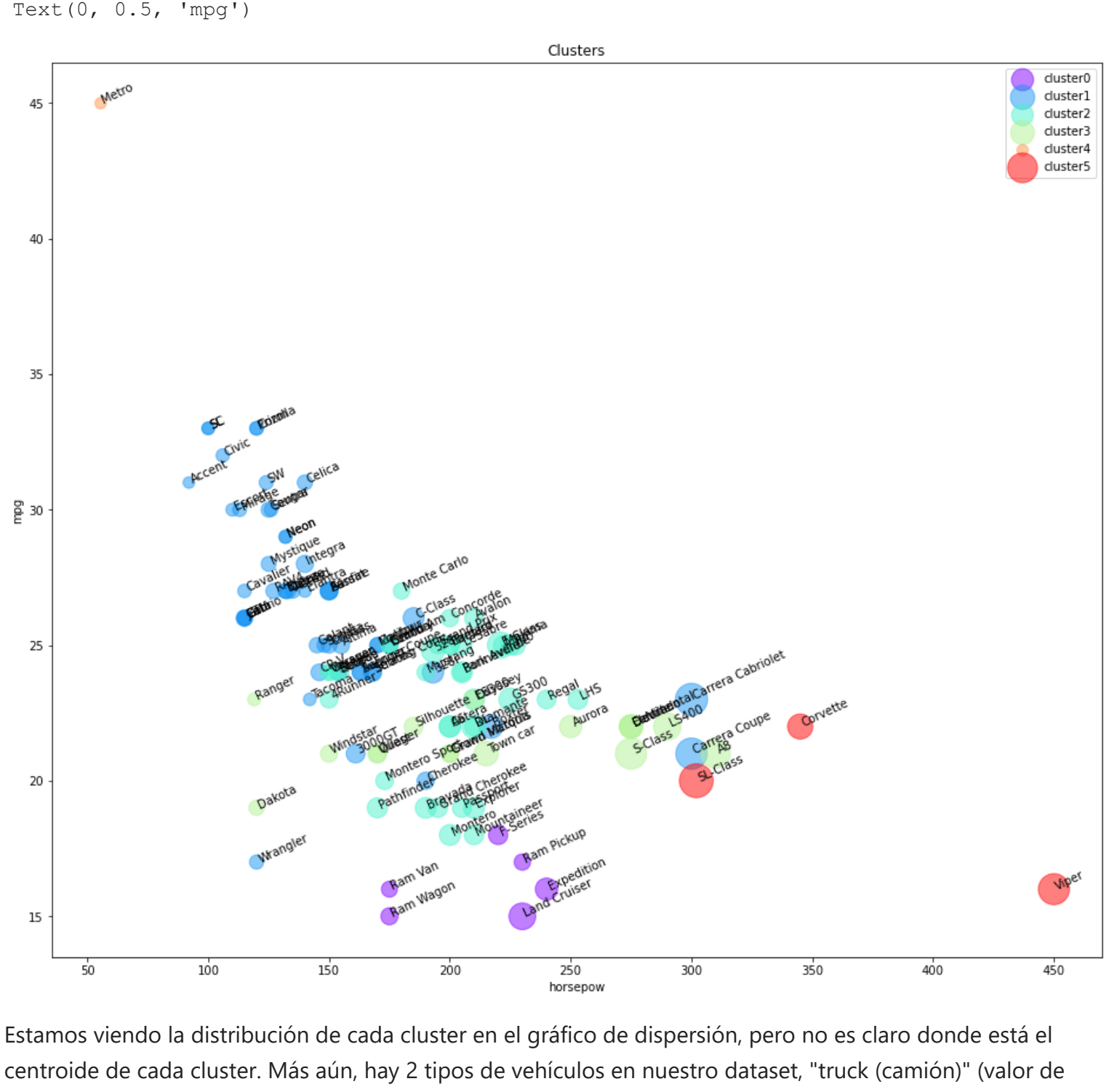
Out[23]:

	manufact	model	sales	resale	type	price	engine_s	horsepow	wheelbas	width	length	curb_wgt	fu
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 [24]:
```
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]), rotat
    plt.scatter(subset.horsepow, subset.mpg, s= subset.price*10, c=color, label='clust
#    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*.  Plea
se use the *color* keyword-argument or provide a 2-D array with a single row if you in
tend 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*.  Plea
se use the *color* keyword-argument or provide a 2-D array with a single row if you in
tend 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*.  Plea
se use the *color* keyword-argument or provide a 2-D array with a single row if you in
tend 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*.  Plea
se use the *color* keyword-argument or provide a 2-D array with a single row if you in
tend 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*.  Plea
se use the *color* keyword-argument or provide a 2-D array with a single row if you in
tend 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*.  Plea
se use the *color* keyword-argument or provide a 2-D array with a single row if you in
tend to specify the same RGB or RGBA value for all points.
```

Out[24]:
```
Text(0, 0.5, 'mpg')
```



Estamos viendo la distribución de cada cluster en el gráfico de dispersión, pero no es claro donde está el centroide de cada cluster. Más aún, hay 2 tipos de vehículos en nuestro dataset, "truck (camión)" (valor de 1 en la columna type) y "car". Por lo tanto, los usamos para distinguir las clases y resumir el clúster. Primero contamos el número de casos en cada grupo:

In [25]:
```
pdf.groupby(['cluster_','type'])['cluster_'].count()
```

Out[25]:
```
cluster_  type
0         1.0      6
1         0.0     47
          1.0      5
2         0.0     27
          1.0     11
3         0.0     10
          1.0      7
4         0.0      1
5         0.0      3
Name: cluster_, dtype: int64
```

Ahora vemos las características de cada cluster:

In [26]:
```
agg_cars = pdf.groupby(['cluster_','type'])['horsepow','engine_s','mpg','price'].mean
agg_cars
```

```
<ipython-input-26-fb63ecec89ff>:1: FutureWarning: Indexing with multiple keys (implici
tly converted to a tuple of keys) will be deprecated, use a list instead.
  agg_cars = pdf.groupby(['cluster_','type'])['horsepow','engine_s','mpg','price'].mea
n()
```

Out[26]:

		horsepow	engine_s	mpg	price
cluster_	**type**				
0	**1.0**	211.666667	4.483333	16.166667	29.024667
1	**0.0**	146.531915	2.246809	27.021277	20.306128
	1.0	145.000000	2.580000	22.200000	17.009200
2	**0.0**	203.111111	3.303704	24.214815	27.750593
	1.0	182.090909	3.345455	20.181818	26.265364
3	**0.0**	256.500000	4.410000	21.500000	42.870400
	1.0	160.571429	3.071429	21.428571	21.527714
4	**0.0**	55.000000	1.000000	45.000000	9.235000
5	**0.0**	365.666667	6.233333	19.333333	66.010000

Tenemos 3 clusters principales con la mayoría de los vehículos en ellos:

Cars:

- Cluster 1: mpg casi alto y baja potencia
- Cluster 2: buen mpg y horsepower, pero alto precio
- Cluster 3: mpg bajo, alto horsepower y alto precio

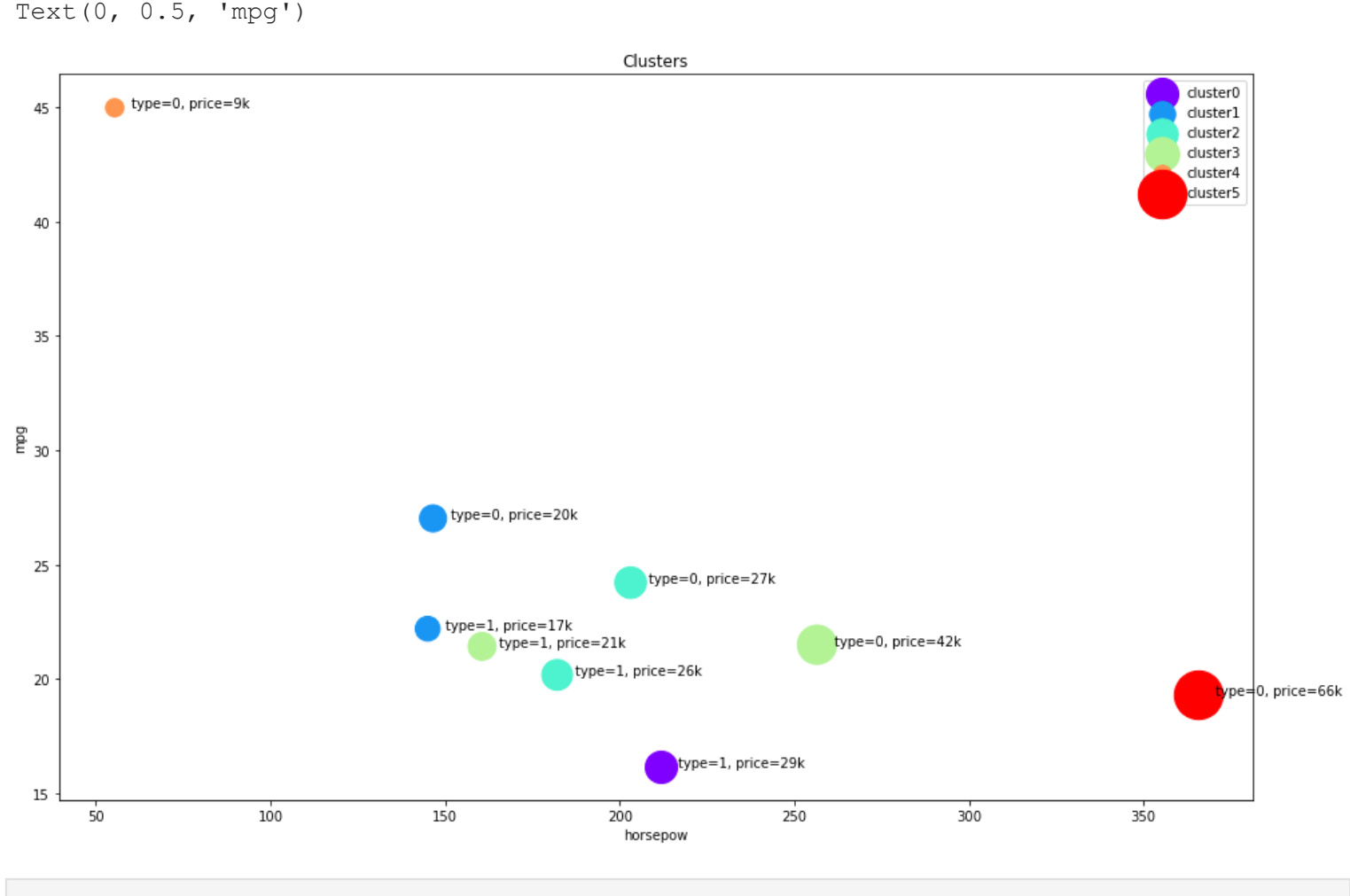
Trucks:

- Cluster 1: mpg casi alto dentro de los camiones, bajo horsepower y precio
- Cluster 2: mpg casi bajo, horsepower medio y alto precio
- Cluster 3: buen mpg y horsepower, bajo precio

In [28]:
```
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='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
as value-mapping will have precedence in case its length matches with *x* & *y*.  Plea
se use the *color* keyword-argument or provide a 2-D array with a single row if you in
tend 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*.  Plea
se use the *color* keyword-argument or provide a 2-D array with a single row if you in
tend 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*.  Plea
se use the *color* keyword-argument or provide a 2-D array with a single row if you in
tend 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*.  Plea
se use the *color* keyword-argument or provide a 2-D array with a single row if you in
tend 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*.  Plea
se use the *color* keyword-argument or provide a 2-D array with a single row if you in
tend 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*.  Plea
se use the *color* keyword-argument or provide a 2-D array with a single row if you in
tend to specify the same RGB or RGBA value for all points.
```

Out[28]:
```
Text(0, 0.5, 'mpg')
```



In [ ]: