



```
import numpy as np
import pandas as pd
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
```

Read data

```
dataset=pd.read_csv('/content/drive/MyDrive/cars_clus.csv')
dataset.head()
```



	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	0.000	21.500	1.800	140.000	101.200	67.300	172.400	2.639	13.200	28.000	2.828	0.0
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	0.0
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	0.0
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	0.0
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	0.0



```
dataset.shape

(159, 16)
```

Data Cleaning

drop the rows that have null value

```
print ("Shape of dataset before cleaning: ", dataset.size)
dataset[[ 'sales', 'resale', 'type', 'price', 'engine_s',
        'horsepow', 'wheelbas', 'width', 'length', 'curb_wgt', 'fuel_cap',
        'mpg', 'lnsales']] = dataset[['sales', 'resale', 'type', 'price', 'engine_s',
        'horsepow', 'wheelbas', 'width', 'length', 'curb_wgt', 'fuel_cap',
        'mpg', 'lnsales']].apply(pd.to_numeric, errors='coerce')
dataset = dataset.dropna() # drop null values
dataset = dataset.reset_index(drop=True)
print ("Shape of dataset after cleaning: ", dataset.size)
dataset.head(5)
```

Feature selection

```
featureset = dataset[['engine_s', 'horsepow', 'wheelbas', 'width', 'length', 'curb_wgt', 'fuel_cap', 'mpg']] # select feature set
```

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.

```
from sklearn.preprocessing import MinMaxScaler
x = featureset.values #returns a numpy array
min_max_scaler = MinMaxScaler() # use 'MinMaxScaler' function to scale each feature to a range
feature_mtx = min_max_scaler.fit_transform(x) # fit MinMaxScaler
feature_mtx [0:5]

array([[0.11428571, 0.21518987, 0.18655098, 0.28143713, 0.30625832,
        0.23105559 , 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.

```
import scipy
leng = feature_mtx.shape[0] #shape of feature_mtx
D = scipy.zeros([leng,leng]) # returns a new array of given 'leng', with zeros
for i in range(leng):
```

```
for j in range(leng):
    D[i,j] = scipy.spatial.distance.euclidean(feature_mtx[i], feature_mtx[j]) # calculate the distance matrix using scipy's euclidean

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DeprecationWarning: scipy.zeros is deprecated and will be removed in SciPy 2.0.0, use numpy.zeros instead
This is separate from the ipykernel package so we can avoid doing imports until
```

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

Using the **linkage** class from hierarchy, pass in the parameters:The distance matrix 'D' and 'complete' for complete linkage

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

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: ClusterWarning: scipy.cluster: The symmetric non-negative hollow observation matrix looks suspiciously like an uncondensed distanc
This is separate from the ipykernel package so we can avoid doing imports until
```

Essentially, Hierarchical clustering does not require a pre-specified number of clusters. However, in some applications we want a partition of disjoint clusters just as in flat clustering. So you can use a cutting line:

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

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)

from scipy.cluster.hierarchy import fcluster
k = 5 # you can determine the number of clusters directly
clusters = fcluster(Z, k, criterion='maxclust')
clusters

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

Now, plot the dendrogram:

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

dendro = hierarchy.dendrogram(Z, leaf_label_func=llf, leaf_rotation=0, leaf_font_size =12, orientation = 'right') # Using the dendrogram class from hierarchy, pass in the parameter Z
```


Clustering using scikit-learn

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 as both inputs and save the distance matrix to a variable called `dist_matrix`

```
dist_matrix = distance_matrix(feature_mtx,feature_mtx) #calculate diatance matrix using feature matrix
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.          ]]
```

The Agglomerative Clustering class will require two inputs: **n_clusters** value will be 6

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'

```
agglom = AgglomerativeClustering(n_clusters = 6, linkage = 'complete') # 'AgglomerativeClustering' function to cluster the dataset
agglom.fit(feature_mtx)# fit the model
agglom.labels_
```

```
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, 1, 2, 1, 1, 2, 2, 2, 3, 3, 3, 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])
```

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

```
dataset['cluster_'] = agglom.labels_      # add cluster_ column
dataset.head()
```

```
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 = dataset[dataset.cluster_ == label]
    for i in subset.index:
        plt.text(subset.horsepow[i], subset.mpg[i],str(subset['model'][i]), rotation=25)
    plt.scatter(subset.horsepow, subset.mpg, s= subset.price*10, c=color, label='cluster'+str(label),alpha=0.5)
plt.legend()
plt.title('Clusters')
plt.xlabel('horsepow')
plt.ylabel('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.

```
dataset.groupby(['cluster_', 'type'])['cluster_'].count() # count the number of cases in each group
```

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		

Now we can look at the characterestics of each cluster:

```
agg_cars = dataset.groupby(['cluster_', 'type'])['horsepow', 'engine_s', 'mpg', 'price'].mean() # find the mean of 'horsepow', 'engine_s', 'mpg', 'price' in each type of different clusters
agg_cars
```

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.

```
plt.figure(figsize=(16,10))
for color, label in zip(colors, cluster_labels):
    subset = agg_cars.loc[(label,),]
    for i in subset.index:
        plt.text(subset.loc[i][0]+5, subset.loc[i][2], 'type='+str(int(i)) + ', price='+str(int(subset.loc[i][3]))+'k')
    plt.scatter(subset.horsepow, subset.mpg, s=subset.price*20, c=color, label='cluster'+str(label))
plt.legend()
plt.title('Clusters')
plt.xlabel('horsepow')
plt.ylabel('mpg')
```


✓ 0s completed at 3:17 PM

