In this notebook we will use two data sets, the Boston Housing data and the Iris Plants data to illustrate the use of KMeans clustering technqiue. We will use both the KMeans clustering module from scikit-learn as well as a modified version of the KMeans impelementation from the Machine Learning in Action book.

In [1]:

```
import numpy as np
import pylab as pl
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.datasets import load_boston
boston = load_boston()
```

I. Clustering with Boston Housing Data

In [2]:

```
np.set_printoptions(suppress=True, precision=2, linewidth=120)
```

In [3]:

```
print(boston.feature_names)
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO' '
```

In [4]:

```
print(boston.data[:5])
```

```
[[  0.01  18.     2.31   0.     0.54   6.58  65.2    4.09   1.   296.    1
 [  0.03   0.     7.07   0.     0.47   6.42  78.9    4.97   2.   242.    1
 [  0.03   0.     7.07   0.     0.47   7.18  61.1    4.97   2.   242.    1
 [  0.03   0.     2.18   0.     0.46   7.    45.8    6.06   3.   222.    1
 [  0.07   0.     2.18   0.     0.46   7.15  54.2    6.06   3.   222.    1
```

In [5]:

```
data = pd.DataFrame(boston.data, columns=boston.feature_names)
data.head(10)
```

Out[5]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	L
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.12	
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395.60	
7	0.14455	12.5	7.87	0.0	0.524	6.172	96.1	5.9505	5.0	311.0	15.2	396.90	
8	0.21124	12.5	7.87	0.0	0.524	5.631	100.0	6.0821	5.0	311.0	15.2	386.63	
9	0.17004	12.5	7.87	0.0	0.524	6.004	85.9	6.5921	5.0	311.0	15.2	386.71	

Now we use KMeans algorithm of scikit-learn to perform the clustering.

In [6]:

```
kmeans = KMeans(n_clusters=5, max_iter=500, verbose=1) # initialization
```

In [7]:

```
kmeans.fit(data)
```

```
Initialization complete
start iteration
done sorting
end inner loop
Iteration 0, inertia 1490393.1841910556
start iteration
done sorting
end inner loop
Iteration 1, inertia 1444691.1079017124
start iteration
done sorting
end inner loop
Iteration 2, inertia 1444409.717605501
start iteration
done sorting
end inner loop
Iteration 3, inertia 1444245.771011614
start iteration
done sorting
end inner loop
Iteration 4, inertia 1443653.51838014
start iteration
done sorting
end inner loop
Iteration 5, inertia 1443511.995989992
start iteration
done sorting
end inner loop
Iteration 6, inertia 1443328.3423140734
start iteration
done sorting
end inner loop
Iteration 7, inertia 1443328.3423140734
center shift 0.000000e+00 within tolerance 2.942886e-01
Initialization complete
start iteration
done sorting
end inner loop
Iteration 0, inertia 1502224.5865587455
start iteration
done sorting
end inner loop
Iteration 1, inertia 1443743.8054449933
start iteration
done sorting
end inner loop
Iteration 2, inertia 1443462.415148782
start iteration
done sorting
end inner loop
Iteration 3, inertia 1443298.468554895
start iteration
done sorting
end inner loop
Iteration 4, inertia 1442706.215923421
```

```
start iteration
done sorting
end inner loop
Iteration 5, inertia 1442564.693533273
start iteration
done sorting
end inner loop
Iteration 6, inertia 1442381.0398573545
start iteration
done sorting
end inner loop
Iteration 7, inertia 1442381.0398573545
center shift 0.000000e+00 within tolerance 2.942886e-01
Initialization complete
start iteration
done sorting
end inner loop
Iteration 0, inertia 1503197.6672685954
start iteration
done sorting
end inner loop
Iteration 1, inertia 1484816.315690253
start iteration
done sorting
end inner loop
Iteration 2, inertia 1473794.743947951
start iteration
done sorting
end inner loop
Iteration 3, inertia 1470535.0614684885
start iteration
done sorting
end inner loop
Iteration 4, inertia 1470382.5683659036
start iteration
done sorting
end inner loop
Iteration 5, inertia 1470382.5683659036
center shift 0.000000e+00 within tolerance 2.942886e-01
Initialization complete
start iteration
done sorting
end inner loop
Iteration 0, inertia 1638034.4139285032
start iteration
done sorting
end inner loop
Iteration 1, inertia 1581754.0893802384
start iteration
done sorting
end inner loop
Iteration 2, inertia 1566013.9122332386
start iteration
done sorting
end inner loop
Iteration 3, inertia 1549050.1565243239
start iteration
```

```
done sorting
end inner loop
Iteration 4, inertia 1546640.1425491062
start iteration
done sorting
end inner loop
Iteration 5, inertia 1546640.1425491062
center shift 0.000000e+00 within tolerance 2.942886e-01
Initialization complete
start iteration
done sorting
end inner loop
Iteration 0, inertia 1861635.0536428196
start iteration
done sorting
end inner loop
Iteration 1, inertia 1706711.5945832497
start iteration
done sorting
end inner loop
Iteration 2, inertia 1633571.252481455
start iteration
done sorting
end inner loop
Iteration 3, inertia 1603298.5332338242
start iteration
done sorting
end inner loop
Iteration 4, inertia 1561841.787139927
start iteration
done sorting
end inner loop
Iteration 5, inertia 1470796.8643624666
start iteration
done sorting
end inner loop
Iteration 6, inertia 1463099.0474331595
start iteration
done sorting
end inner loop
Iteration 7, inertia 1451880.20042597
start iteration
done sorting
end inner loop
Iteration 8, inertia 1444582.7718663553
start iteration
done sorting
end inner loop
Iteration 9, inertia 1443328.3423140734
start iteration
done sorting
end inner loop
Iteration 10, inertia 1443328.3423140734
center shift 0.000000e+00 within tolerance 2.942886e-01
Initialization complete
start iteration
done sorting
```

```
end inner loop
Iteration 0, inertia 1517923.3279835347
start iteration
done sorting
end inner loop
Iteration 1, inertia 1483854.7628671615
start iteration
done sorting
end inner loop
Iteration 2, inertia 1469238.803570316
start iteration
done sorting
end inner loop
Iteration 3, inertia 1467603.8484652522
start iteration
done sorting
end inner loop
Iteration 4, inertia 1467603.8484652522
center shift 0.000000e+00 within tolerance 2.942886e-01
Initialization complete
start iteration
done sorting
end inner loop
Iteration 0, inertia 1585539.0056731445
start iteration
done sorting
end inner loop
Iteration 1, inertia 1546345.0951247003
start iteration
done sorting
end inner loop
Iteration 2, inertia 1497223.1139836025
start iteration
done sorting
end inner loop
Iteration 3, inertia 1470789.5779283214
start iteration
done sorting
end inner loop
Iteration 4, inertia 1468300.3782850797
start iteration
done sorting
end inner loop
Iteration 5, inertia 1467731.541312052
start iteration
done sorting
end inner loop
Iteration 6, inertia 1467603.8484652522
start iteration
done sorting
end inner loop
Iteration 7, inertia 1467603.8484652522
center shift 0.000000e+00 within tolerance 2.942886e-01
Initialization complete
start iteration
done sorting
end inner loop
```

```
Iteration 0, inertia 1530652.6214435766
start iteration
done sorting
end inner loop
Iteration 1, inertia 1474039.1874440382
start iteration
done sorting
end inner loop
Iteration 2, inertia 1470788.658537518
start iteration
done sorting
end inner loop
Iteration 3, inertia 1470382.5683659036
start iteration
done sorting
end inner loop
Iteration 4, inertia 1470382.5683659036
center shift 0.000000e+00 within tolerance 2.942886e-01
Initialization complete
start iteration
done sorting
end inner loop
Iteration 0, inertia 1833196.81070146
start iteration
done sorting
end inner loop
Iteration 1, inertia 1677745.904055242
start iteration
done sorting
end inner loop
Iteration 2, inertia 1670664.8073136113
start iteration
done sorting
end inner loop
Iteration 3, inertia 1670322.2747555608
start iteration
done sorting
end inner loop
Iteration 4, inertia 1670322.2747555608
center shift 0.000000e+00 within tolerance 2.942886e-01
Initialization complete
start iteration
done sorting
end inner loop
Iteration 0, inertia 1746070.6937772494
start iteration
done sorting
end inner loop
Iteration 1, inertia 1662405.8906899262
start iteration
done sorting
end inner loop
Iteration 2, inertia 1565605.0167736267
start iteration
done sorting
end inner loop
Iteration 3, inertia 1549784.641500654
```

```
start iteration
done sorting
end inner loop
Iteration 4, inertia 1544671.2012002505
start iteration
done sorting
end inner loop
Iteration 5, inertia 1543018.4493849452
start iteration
done sorting
end inner loop
Iteration 6, inertia 1542705.5106074852
start iteration
done sorting
end inner loop
Iteration 7, inertia 1542590.141332547
start iteration
done sorting
end inner loop
Iteration 8, inertia 1542590.141332547
center shift 0.000000e+00 within tolerance 2.942886e-01
```

Out[7]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=500,
    n_clusters=5, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=1)
```

In [8]:

```
clusters = kmeans.predict(data)
```

In [9]:

```
pd.DataFrame(clusters, columns=["Cluster"])
```

Out[9]:

	Cluster
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	0
17	0
18	0
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0
29	0
...	...
476	1
477	1
478	1

	Cluster
479	1
480	1
481	1
482	1
483	1
484	1
485	1
486	1
487	1
488	1
489	1
490	1
491	1
492	1
493	3
494	3
495	3
496	3
497	3
498	3
499	3
500	3
501	0
502	0
503	0
504	0
505	0

506 rows × 1 columns

The centroids provide an aggregate representation and a characterization of each cluster.

In [10]:

```
pd.options.display.float_format='{:,.2f}'.format

centroids = pd.DataFrame(kmeans.cluster_centers_, columns=boston.feature_nam
centroids
```

Out[10]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	L
0	0.24	17.26	6.71	0.08	0.48	6.47	56.07	4.84	4.34	274.69	17.86	388.78	
1	10.91	0.00	18.57	0.08	0.67	5.98	89.91	2.08	23.02	668.21	20.20	371.80	
2	16.06	-0.00	18.10	0.00	0.67	6.08	90.13	1.99	24.00	666.00	20.20	55.67	
3	0.62	12.88	12.03	0.06	0.56	6.21	69.25	3.63	4.73	402.31	17.76	382.25	
4	1.96	0.00	16.71	0.09	0.71	5.92	91.82	2.32	4.73	386.91	17.00	187.55	

In [11]:

```
def cluster_sizes(clusters):
    #clusters is an array of cluster labels for each instance in the data

    size = {}
    cluster_labels = np.unique(clusters)
    n_clusters = cluster_labels.shape[0]

    for c in cluster_labels:
        size[c] = len(data[clusters == c])
    return size
```

In [12]:

```
size = cluster_sizes(clusters)

for c in size.keys():
    print("Size of Cluster", c, "= ", size[c])
```

```
Size of Cluster 0 =  260
Size of Cluster 1 =  102
Size of Cluster 2 =  35
Size of Cluster 3 =  98
Size of Cluster 4 =  11
```

One way to measure the quality of clustering is to compute the Silhouette values for each instance in the data. The silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). It is the ratio of the difference between in-cluster dissimilarity and the closest out-of-cluster dissimilarity, and the maximum of these two values. The silhouette ranges from −1 to +1, where a high value indicates that the object is well matched to its own cluster and well separated from other clusters. If most objects have a high value, then the clustering configuration is appropriate. If many points have a low or negative value, then the clustering configuration may have too many or too few clusters. More details on the definition of [Silhouette measure (https://en.wikipedia.org/wiki/Silhouette_(clustering))](https://en.wikipedia.org/wiki/Silhouette_(clustering)).

In [13]:

```
from sklearn import metrics
```

In [14]:

```
silhouettes = metrics.silhouette_samples(data, clusters)
print(silhouettes[:20])
```

```
[0.53 0.61 0.63 0.61 0.61 0.61 0.42 0.34 0.31 0.38 0.35 0.39 0.41 0.45 0.41
```

In [15]:

```
print(silhouettes.mean())
```

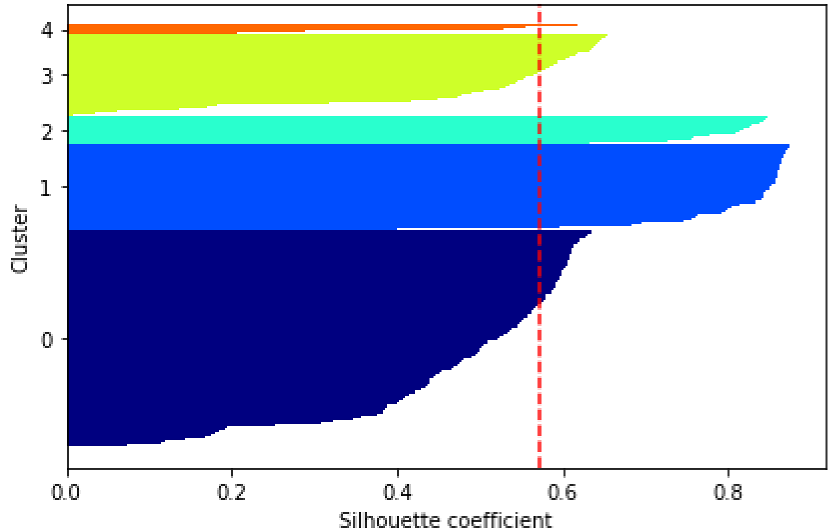
```
0.5707386655129686
```

In [16]:

```
def plot_silhouettes(data, clusters, metric='euclidean'):

    from matplotlib import cm
    from sklearn.metrics import silhouette_samples

    cluster_labels = np.unique(clusters)
    n_clusters = cluster_labels.shape[0]
    silhouette_vals = metrics.silhouette_samples(data, clusters, metric='euc
    c_ax_lower, c_ax_upper = 0, 0
    cticks = []
    for i, k in enumerate(cluster_labels):
        c_silhouette_vals = silhouette_vals[clusters == k]
        c_silhouette_vals.sort()
        c_ax_upper += len(c_silhouette_vals)
        color = cm.jet(float(i) / n_clusters)
        pl.barh(range(c_ax_lower, c_ax_upper), c_silhouette_vals, height=1.0
                      edgecolor='none', color=color)

        cticks.append((c_ax_lower + c_ax_upper) / 2)
        c_ax_lower += len(c_silhouette_vals)

    silhouette_avg = np.mean(silhouette_vals)
    pl.axvline(silhouette_avg, color="red", linestyle="--")

    pl.yticks(cticks, cluster_labels)
    pl.ylabel('Cluster')
    pl.xlabel('Silhouette coefficient')

    pl.tight_layout()
    #pl.savefig('images/11_04.png', dpi=300)
    pl.show()

    return
```

In [17]:

```
plot_silhouettes(data, clusters)
```



II. Clustering with Iris Plant Database

In [18]:

```
from sklearn.datasets import load_iris
iris = load_iris()
```

In [19]:
```
print(iris.DESCR)
```

```
Iris Plants Database
====================

Notes
-----
Data Set Characteristics:
    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica
    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20  0.76     0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

This is a copy of UCI ML iris datasets.
http://archive.ics.uci.edu/ml/datasets/Iris

The famous Iris database, first used by Sir R.A Fisher

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field an
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

References
----------
   - Fisher,R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda,R.O., & Hart,P.E. (1973) Pattern Classification and Scene Analysis
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
```

```
    Environments".  IEEE Transactions on Pattern Analysis and Machine
    Intelligence, Vol. PAMI-2, No. 1, 67-71.
  - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transacti
    on Information Theory, May 1972, 431-433.
  - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
    conceptual clustering system finds 3 classes in the data.
  - Many, many more ...
```

In [20]:

```
data = iris.data
target = iris.target
```

In [21]:

```
print(iris.feature_names)
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal widt
```

In [22]:

```
irisDF = pd.DataFrame(data, columns=iris.feature_names)
irisDF.head(10)
```

Out[22]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.10	3.50	1.40	0.20
1	4.90	3.00	1.40	0.20
2	4.70	3.20	1.30	0.20
3	4.60	3.10	1.50	0.20
4	5.00	3.60	1.40	0.20
5	5.40	3.90	1.70	0.40
6	4.60	3.40	1.40	0.30
7	5.00	3.40	1.50	0.20
8	4.40	2.90	1.40	0.20
9	4.90	3.10	1.50	0.10

In [23]:

```
print(iris.target)
```

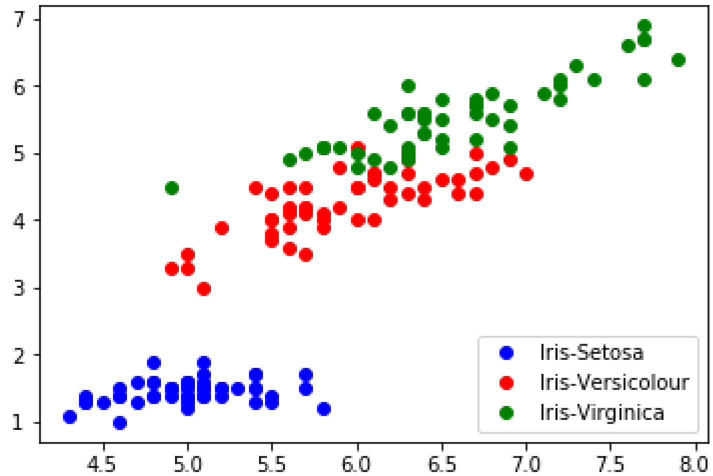
```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

This snippet uses the first and the third dimension (sepal length and sepal width) and the result is shown in the following figure.

Classes: 0 = Iris-Setosa, 1 = Iris-Versicolour, 2 = Iris-Virginica

In [24]:

```
pl.plot(data[target==0,0],data[target==0,2],'bo')
pl.plot(data[target==1,0],data[target==1,2],'ro')
pl.plot(data[target==2,0],data[target==2,2],'go')
pl.legend(('Iris-Setosa', 'Iris-Versicolour', 'Iris-Virginica'), loc=4)
pl.show()
```



In the graph we have 150 points and their color represents the class; the blue points represent the samples that belong to the specie setosa, the red ones represent versicolor and the green ones represent virginica. Next let's see if through clustering we can obtain the correct classes.

In [25]:

```
iris_kmeans = KMeans(n_clusters=3, max_iter=500, verbose=1, n_init=5) # init
iris_kmeans.fit(irisDF)
```

```
Initialization complete
start iteration
done sorting
end inner loop
Iteration 0, inertia 90.35420391545392
start iteration
done sorting
end inner loop
Iteration 1, inertia 82.0112981965174
start iteration
done sorting
end inner loop
Iteration 2, inertia 79.6309054945055
start iteration
done sorting
end inner loop
Iteration 3, inertia 78.94084142614602
start iteration
done sorting
end inner loop
Iteration 4, inertia 78.94084142614602
center shift 0.000000e+00 within tolerance 1.134707e-04
Initialization complete
start iteration
done sorting
end inner loop
Iteration 0, inertia 88.27962882882883
start iteration
done sorting
end inner loop
Iteration 1, inertia 81.26545514705883
start iteration
done sorting
end inner loop
Iteration 2, inertia 79.6309054945055
start iteration
done sorting
end inner loop
Iteration 3, inertia 78.94084142614602
start iteration
done sorting
end inner loop
Iteration 4, inertia 78.94084142614602
center shift 0.000000e+00 within tolerance 1.134707e-04
Initialization complete
start iteration
done sorting
end inner loop
Iteration 0, inertia 80.53182857142858
start iteration
done sorting
end inner loop
Iteration 1, inertia 78.94084142614602
```

```
start iteration
done sorting
end inner loop
Iteration 2, inertia 78.94084142614602
center shift 0.000000e+00 within tolerance 1.134707e-04
Initialization complete
start iteration
done sorting
end inner loop
Iteration 0, inertia 86.72070370370372
start iteration
done sorting
end inner loop
Iteration 1, inertia 79.4039
start iteration
done sorting
end inner loop
Iteration 2, inertia 78.94084142614602
start iteration
done sorting
end inner loop
Iteration 3, inertia 78.94084142614602
center shift 0.000000e+00 within tolerance 1.134707e-04
Initialization complete
start iteration
done sorting
end inner loop
Iteration 0, inertia 81.26545514705883
start iteration
done sorting
end inner loop
Iteration 1, inertia 79.6309054945055
start iteration
done sorting
end inner loop
Iteration 2, inertia 78.94084142614602
start iteration
done sorting
end inner loop
Iteration 3, inertia 78.94084142614602
center shift 0.000000e+00 within tolerance 1.134707e-04
```

Out[25]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=500,
    n_clusters=3, n_init=5, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=1)
```

In [26]:

```
c = iris_kmeans.predict(data)
```

In [27]:

```
c.shape
```

Out[27]:

```
(150,)
```

In [28]:
```
size = cluster_sizes(c)

for i in size.keys():
    print("Size of Cluster", i, "= ", size[i])
```

```
Size of Cluster 0 =  62
Size of Cluster 1 =  50
Size of Cluster 2 =  38
```

In [29]:
```
iris_centroids = pd.DataFrame(iris_kmeans.cluster_centers_, columns=iris.fea
iris_centroids
```

Out[29]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.90	2.75	4.39	1.43
1	5.01	3.42	1.46	0.24
2	6.85	3.07	5.74	2.07

Since we know what the actual classes are (in the target attribute), we can evaluate clustering performance by using metrics that compare our discovered cluster labels to the actual classes:

In [30]:
```
print(c)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2 2 2 0 2 2 0]
```

In [31]:
```
print(target)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

Homogeneity: each cluster contains only members of a single class. Completeness: all members of a given class are assigned to the same cluster.

In [32]:
```
from sklearn.metrics import completeness_score, homogeneity_score
```

In [33]:
```
print(completeness_score(target,c))
```

```
0.7649861514489816
```

In [34]:
```
print(homogeneity_score(target,c))
```

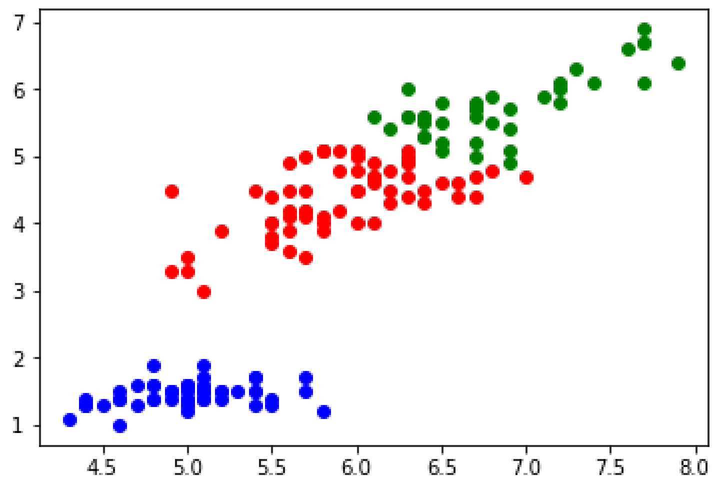
```
0.7514854021988339
```

The completeness score approaches 1 when most of the data points that are members of a given class are elements of the same cluster while the homogeneity score approaches 1 when all the clusters contain almost only data points that are member of a single class.

Let's again plot sepal length against sepal width, but this time we'll use our cluster labels instead of the actual class labels from the target attribute.

In [35]:

```
pl.plot(data[c==0,0],data[c==0,2],'ro')
pl.plot(data[c==1,0],data[c==1,2],'bo')
pl.plot(data[c==2,0],data[c==2,2],'go')
pl.show()
```



Let's also do some silhouette analysis on the Iris clusters:

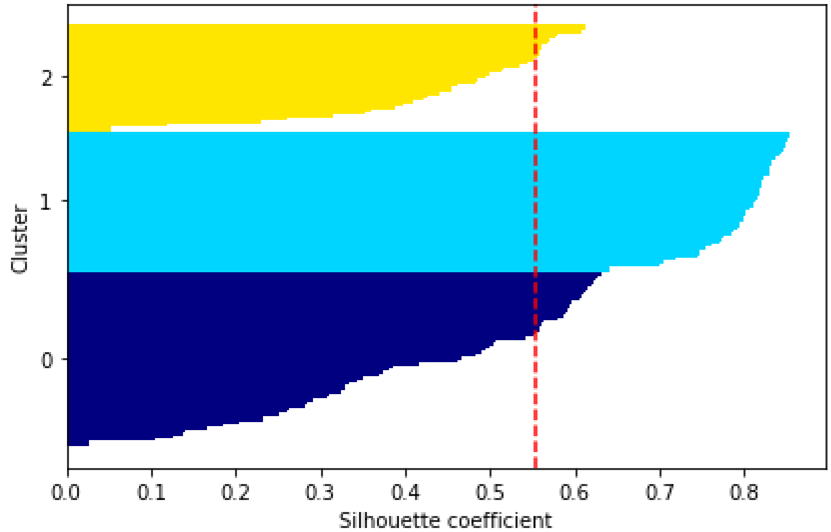
In [36]:

```
from sklearn.metrics import silhouette_samples
iris_silhouettes = metrics.silhouette_samples(iris.data, c)
print(iris_silhouettes[:20])
print("\n Mean Silhoouette Value: ", iris_silhouettes.mean())
```

```
[0.85 0.82 0.83 0.81 0.85 0.75 0.82 0.85 0.75 0.83 0.8  0.84 0.81 0.75 0.7

 Mean Silhoouette Value:  0.5525919445309032
```

In [37]:

```
plot_silhouettes(data, c)
```



Let's now use the [kMeans clustering implementation (http://facweb.cs.depaul.edu/mobasher/classes/CSC478/Data/kMeans.zip)](http://facweb.cs.depaul.edu/mobasher/classes/CSC478/Data/kMeans.zip) from Machine Learning in Action, Ch. 10:

In [38]:

```
import kMeans

data = np.array(data)
```

Note: in the MLA kMeans module only a Euclidean distance function "distEuclid" is provided which is passed to the kMeans function. For this example, we have added another distance function based on the Cosine Similarity measure to the kMeans module and this function is used in the example below.

In [39]:

```
centroids, clusters = kMeans.kMeans(data, 3, kMeans.distCosine, kMeans.randC
```

In [40]:

```
pd.DataFrame(centroids, columns=iris.feature_names)
```

Out[40]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.94	2.76	4.21	1.30
1	6.54	2.96	5.50	1.99
2	5.01	3.42	1.46	0.24

In [41]:

```
print(clusters[:10,:])
```

```
[[2. 0.]
 [2. 0.]
 [2. 0.]
 [2. 0.]
 [2. 0.]
 [2. 0.]
 [2. 0.]
 [2. 0.]
 [2. 0.]
 [2. 0.]]
```

In [42]:

```
iris_clusters = pd.DataFrame(clusters, columns=["Cluster", "MinDistance**2"]
iris_clusters.head(10)
```

Out[42]:

	Cluster	MinDistance**2
0	2.00	0.00
1	2.00	0.00
2	2.00	0.00
3	2.00	0.00
4	2.00	0.00
5	2.00	0.00
6	2.00	0.00
7	2.00	0.00
8	2.00	0.00
9	2.00	0.00

In [43]:

```
newC = iris_clusters["Cluster"].astype(int)
print(newC)
```

```
0      2
1      2
2      2
3      2
4      2
5      2
6      2
7      2
8      2
9      2
10     2
11     2
12     2
13     2
14     2
15     2
16     2
17     2
18     2
19     2
20     2
21     2
22     2
23     2
24     2
25     2
26     2
27     2
28     2
29     2
      ..
120    1
121    1
122    1
123    1
124    1
125    1
126    1
127    1
128    1
129    1
130    1
131    1
132    1
133    1
134    1
135    1
136    1
137    1
138    1
139    1
140    1
141    1
142    1
```

```
143    1
144    1
145    1
146    1
147    1
148    1
149    1
Name: Cluster, Length: 150, dtype: int32
```

In [44]:
```
print(completeness_score(target,newC))
```

```
0.9152529861036721
```

In [45]:
```
print(homogeneity_score(target,newC))
```

```
0.9134738072405234
```

Let's now try the Bisection kMeans algorithm also provided in the MLA kMeans module.

In [46]:
```
centroids_bk, clusters_bk = kMeans.biKmeans(data, 3, kMeans.distEuclid)
```

```
sseSplit, and notSplit:  152.36870647733906 0.0
the bestCentToSplit is:  0
the len of bestClustAss is:  150
sseSplit, and notSplit:  55.651677074041025 28.572830188679244
sseSplit, and notSplit:  15.347066666666667 123.79587628865981
the bestCentToSplit is:  0
the len of bestClustAss is:  97
```

In [47]:
```
print(centroids_bk)
```

```
[[6.85 5.01 5.95]]
```

In [48]:
```
bkC = clusters_bk.T[0]
bkC = bkC.astype(int)
```

In [49]:
```
print(bkC)
```

```
[[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1
  0 0 0 2 0 2 0 2 0 0 2 2 0 0 0 0 0 2 0 0 0 0 2 0 0 0 2 0 0 0 2 0 0 2]]
```

In [50]:
```
bkC = np.ravel(bkC)
print(bkC)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2
 0 2 0 2 0 2 0 0 2 2 0 0 0 0 0 2 0 0 0 0 2 0 0 0 2 0 0 0 2 0 0 2]
```

In [51]:

```
print(completeness_score(target,bkC))
```

```
0.6965705987251742
```

In [52]:

```
print(homogeneity_score(target,bkC))
```

```
0.686320173948772
```

In [ ]:

```

```