Animated k-Means Examples

This simple notebook show how to generate a sequence of frames to produce an animation of k-Means.

In [2]:
```
import warnings
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.metrics import silhouette_score
from sklearn import cluster, datasets, mixture
from sklearn.cluster import KMeans
from sklearn.neighbors import kneighbors_graph

from matplotlib.colors import ListedColormap

import math

warnings.simplefilter(action='ignore', category=FutureWarning)

np.random.seed(844)
```

Utility Functions

We define the utility functions we need to

- define the plotting parameters so that if we want to change our the figures are saved we change all the parameters in one place
- plot the current status of k-means
- run k-means step by step for a given number of iterations

In [3]:
```
def GetParameters(dataset):
    parameters = {'figsize':(10,10), 'xlabel':'x', 'ylabel':'y'}
    parameters['xmax']=np.round(np.max(dataset[:,0]),0)+0.5
    parameters['xmin']=np.round(np.min(dataset[:,0]),0)-0.5
    parameters['ymax']=np.round(np.max(dataset[:,1]),0)+0.5
    parameters['ymin']=np.round(np.min(dataset[:,1]),0)-0.5
    parameters['alpha']=1.0
    parameters['point-color']='grey'
    parameters['cluster-color']=['#a6cdf6','#b2d0b7','#f98ea1']
    parameters['centroid-color']=['#1b80e8','#599062','#e20c32']
    return parameters
```

In [4]:
```
def PlotClusters(parameters, X, centroids=[], clusters_id=[], label=''):
    plt.figure(figsize=parameters['figsize'])

    plt.rc('font', **{'family' : 'sans', 'size'   : 16})
    plt.rc('xtick', labelsize=20)
    plt.rc('ytick', labelsize=20)

    if (clusters_id==[]):
        plt.scatter(X[:, 0], X[:, 1], s=50, alpha=parameters['alpha'], c=parameters['point-color'])
    else:
        plt.scatter(X[:, 0], X[:, 1], s=50, alpha=parameters['alpha'], c=[parameters['cluster-color'][x] for x in clusters_id])

    if (centroids!=[]):
        plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=400, color=parameters['centroid-color'][:len(centroids)])

    plt.xlim([parameters['xmin'],parameters['xmax']])
    plt.ylim([parameters['ymin'],parameters['ymax']])
    plt.xlabel(parameters['xlabel'])
    plt.ylabel(parameters['ylabel'])

    if 'title' in parameters:
        plt.title(parameters['title'])

    if (label!=''):
        plt.savefig(label)
    plt.show();
```

In [5]:
```
def StepByStepKMeans(dataset,centroids,iterations,filename):

    parameters = GetParameters(dataset)

    k = len(centroids)

    PlotClusters(parameters,dataset,centroids,label=filename+'-start.png');

    for i in range(iterations):

        model = KMeans(n_clusters=k, init=centroids, max_iter=1)

        model.fit(dataset)

        clusters_id = model.predict(dataset)

        parameters['title'] = 'Iteration '+str(i)+" - Assign Cluster"
        parameters['title'] = ''
        PlotClusters(parameters,dataset,centroids,clusters_id,label=filename+'-'+str(i)+'-assign.png')

        centroids = model.cluster_centers_

        parameters['title'] = 'Iteration '+str(i)+" - Update Centroid Cluster"
        parameters['title'] = ''
        PlotClusters(parameters,dataset,centroids,clusters_id,label=filename+'-'+str(i)+'-update.png')

        if (model.n_iter_==0):
            return
```

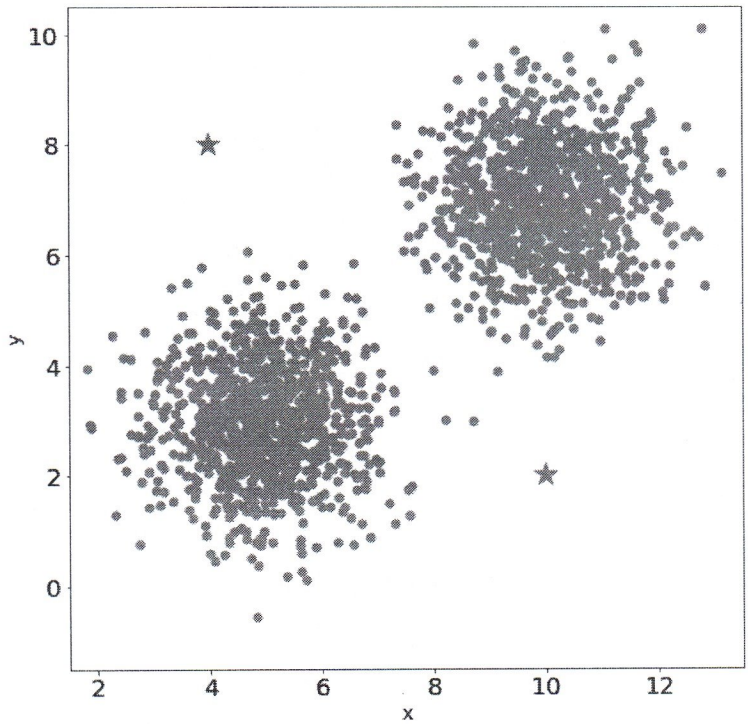
Example Dataset

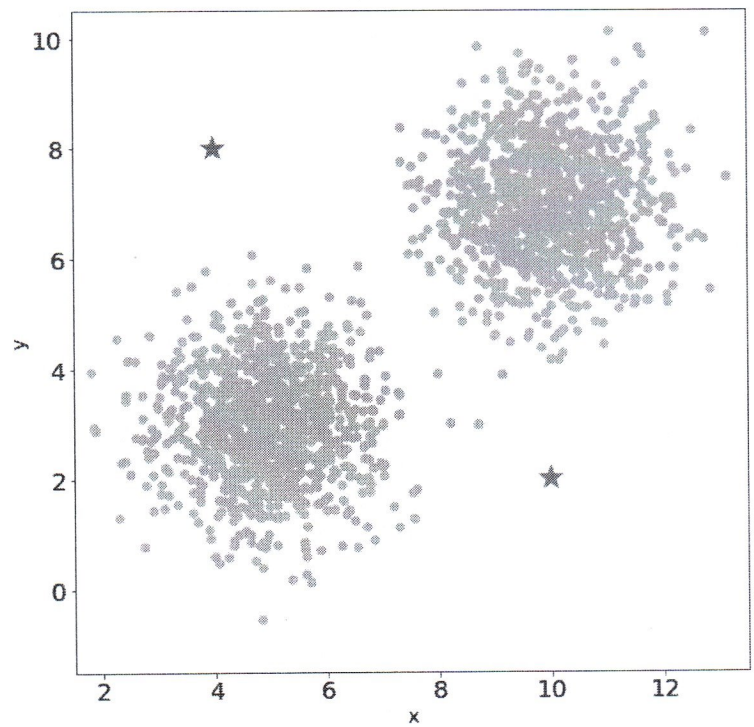
First, we define a simple dataset with three clusters.

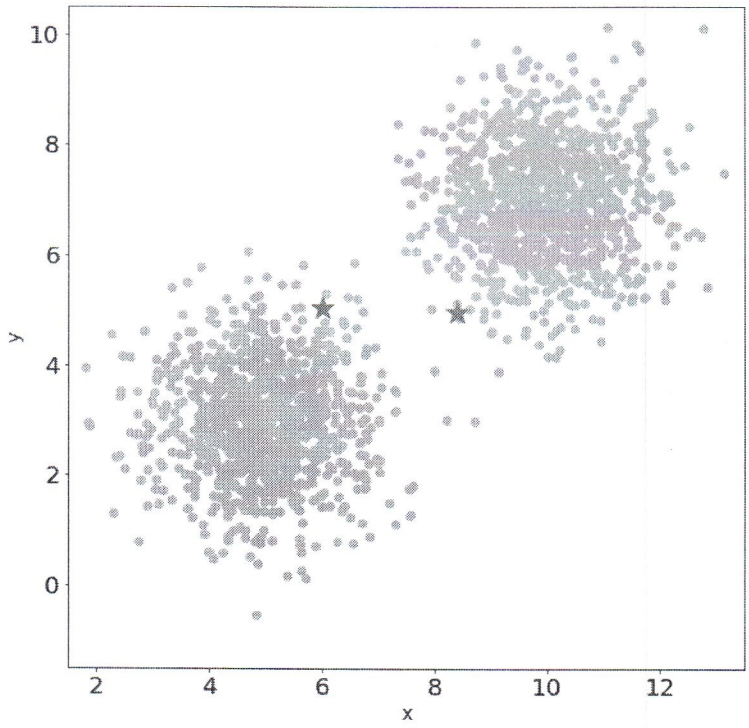
In [6]:
```
cluster1 = np.random.normal(5, 1, (1000,2))
cluster2 = np.random.normal(15, 1, (1000,2))
cluster3 = np.random.multivariate_normal([17,3], [[1,0],[0,1]], 1000)
cluster4 = np.random.multivariate_normal([2,16], [[.5,0],[0,.5]], 1000)
# da un errore
dataset1 = np.concatenate((cluster1, cluster2))
```

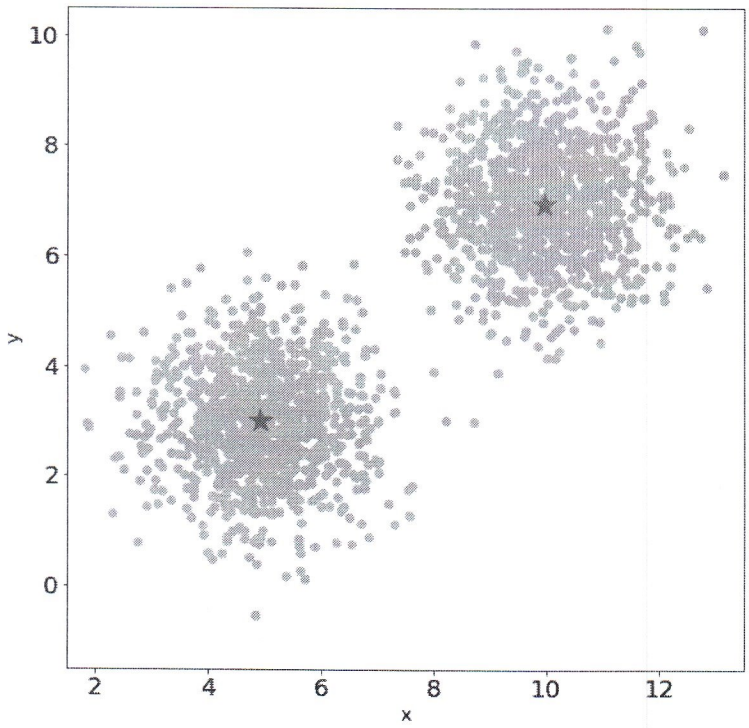
Define the plot parameters and plot the data

In [7]:
```
cluster3A = np.random.normal([5,3], 1, (1000,2))
cluster3B = np.random.normal([10,7], 1, (1000,2))
dataset3 = np.concatenate((cluster3A, cluster3B))

parameters = GetParameters(dataset1)
centroids = np.array([[4,8], [10,2]], np.float64)
StepByStepKMeans(dataset3,centroids,10,'example3');
```

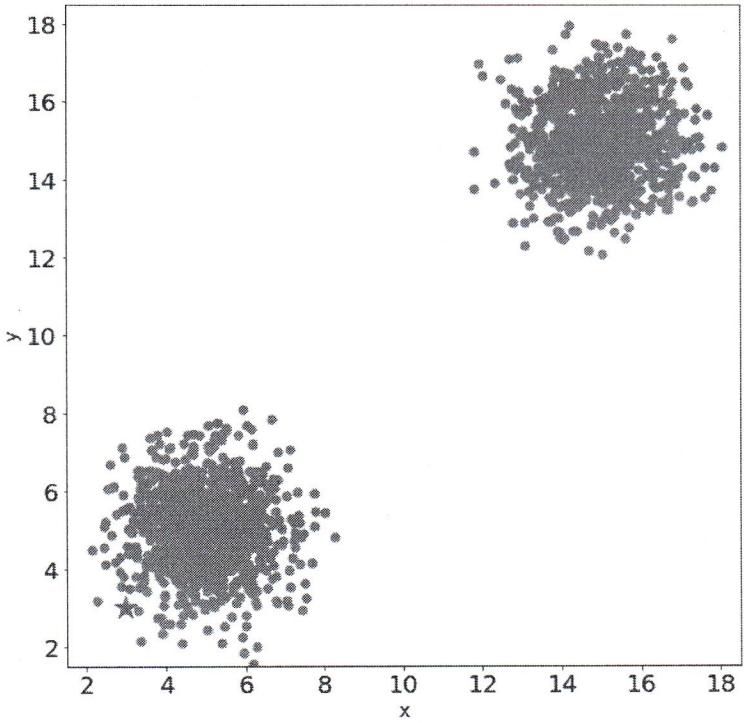


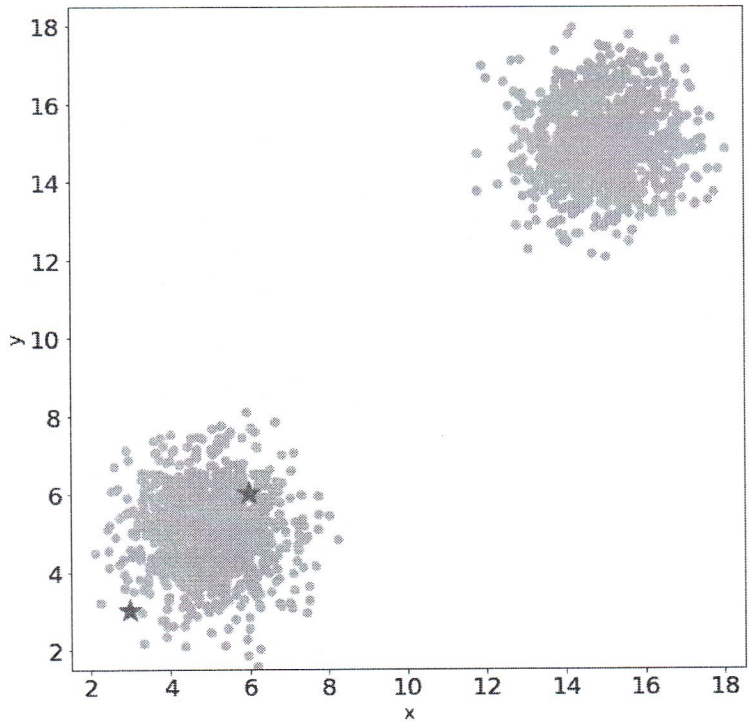


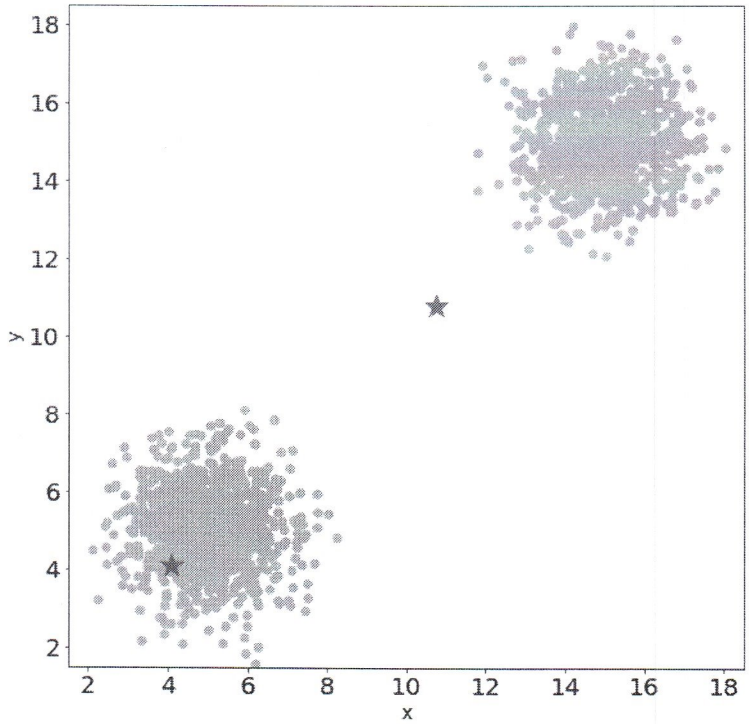


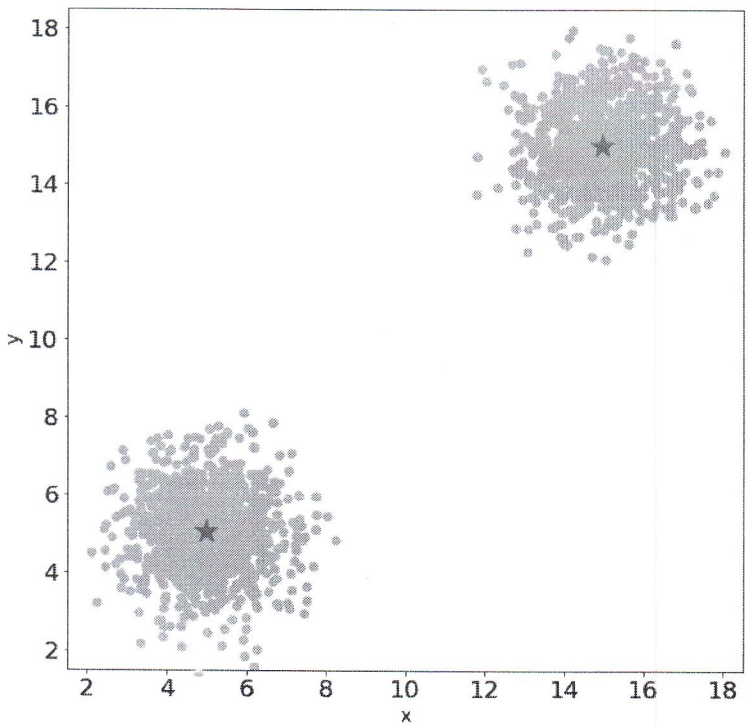


In [8]:
```
centroids = np.array([[3,3], [6,6]], np.float64)
StepByStepKMeans(dataset1,centroids,10,"example2")
```


18
16
14
12
y 10
8
6
4
2
2 4 6 8 10 12 14 16 18
x

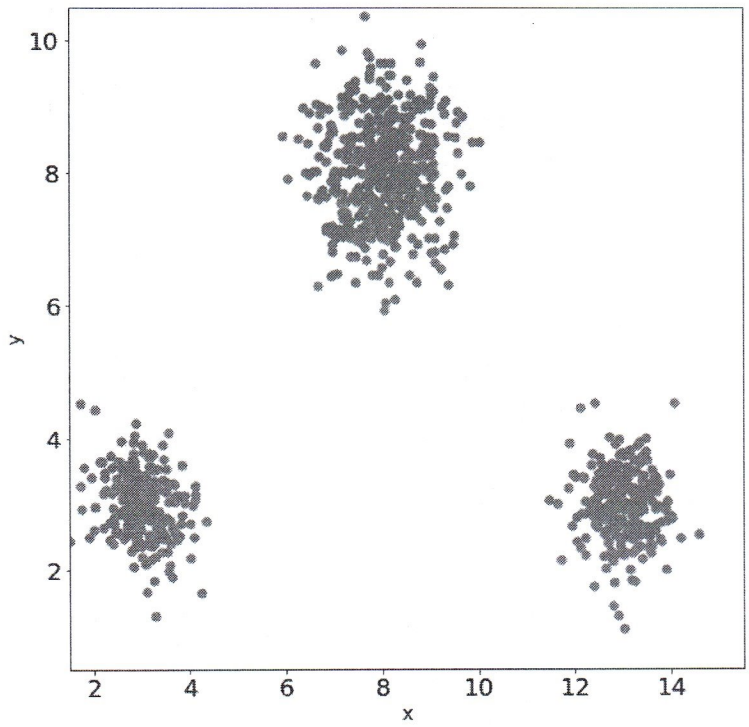

18
16
14
12
y 10
8
6
4
2
2 4 6 8 10 12 14 16 18
x



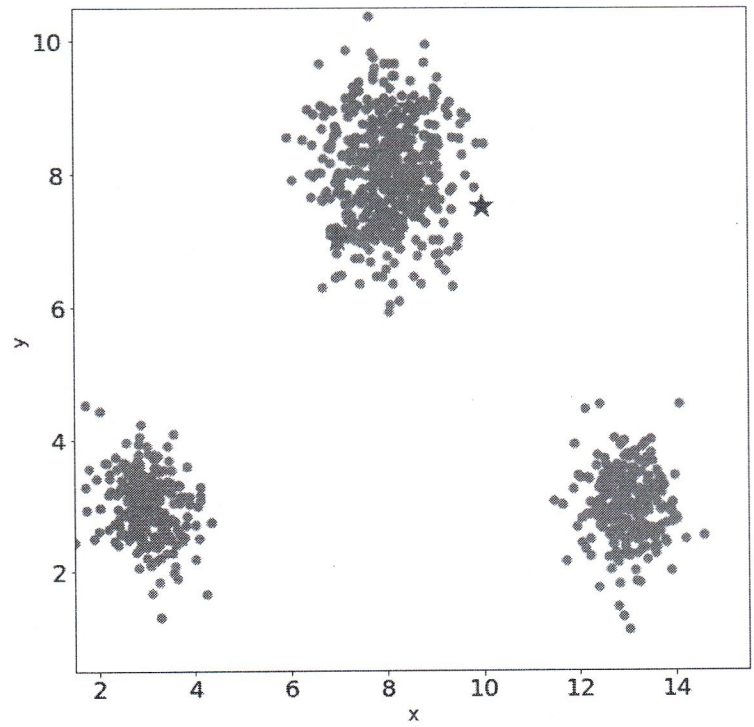


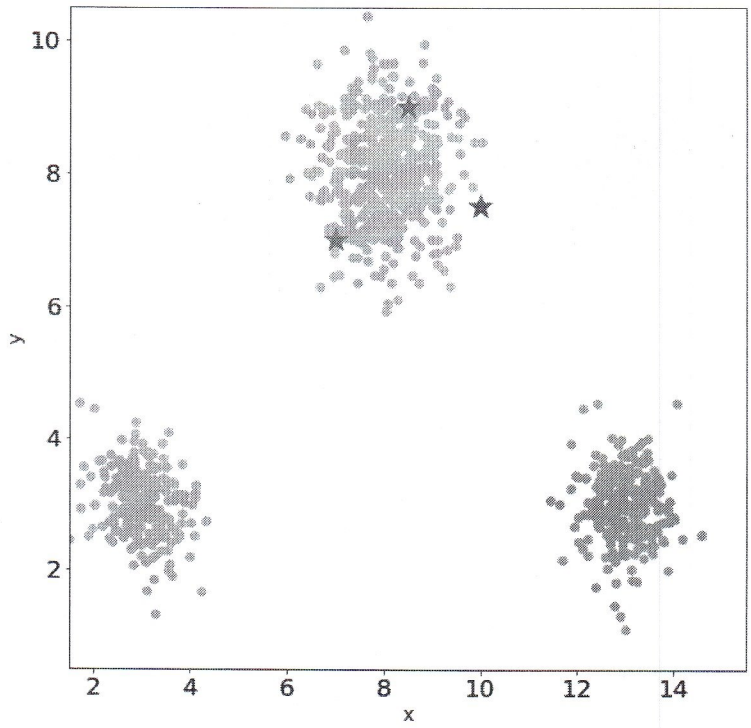
In [19]:
```
cluster4A = np.random.normal([8,8], .75, (500,2))
cluster4B = np.random.normal([3,3], .5, (250,2))
cluster4C = np.random.normal([13,3], .5, (250,2))

dataset4 = np.concatenate((cluster4A, cluster4B, cluster4C))

parameters = GetParameters(dataset4)
```

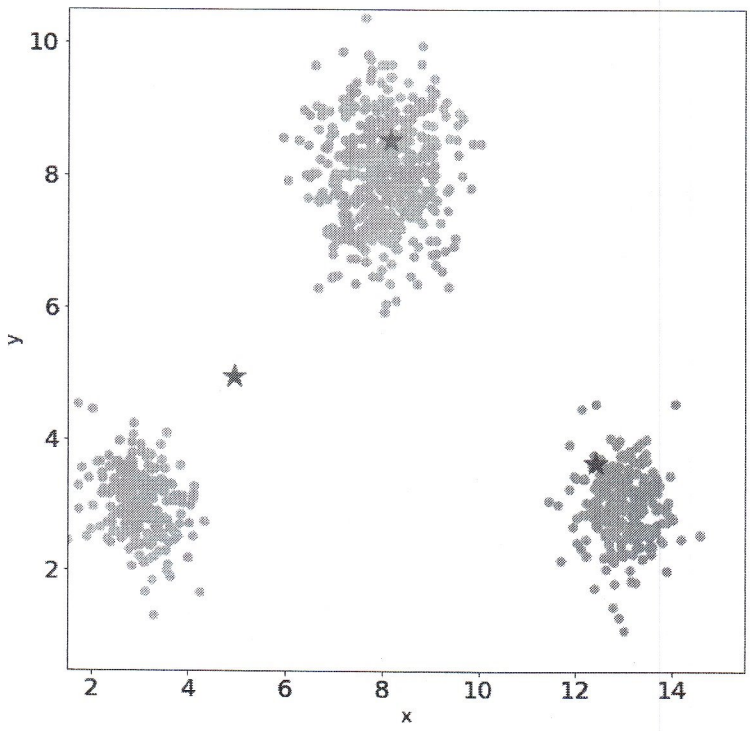
In [20]:
```
PlotClusters(parameters, dataset4, centroids=[], clusters_id=[], label='example4-data.png');
```

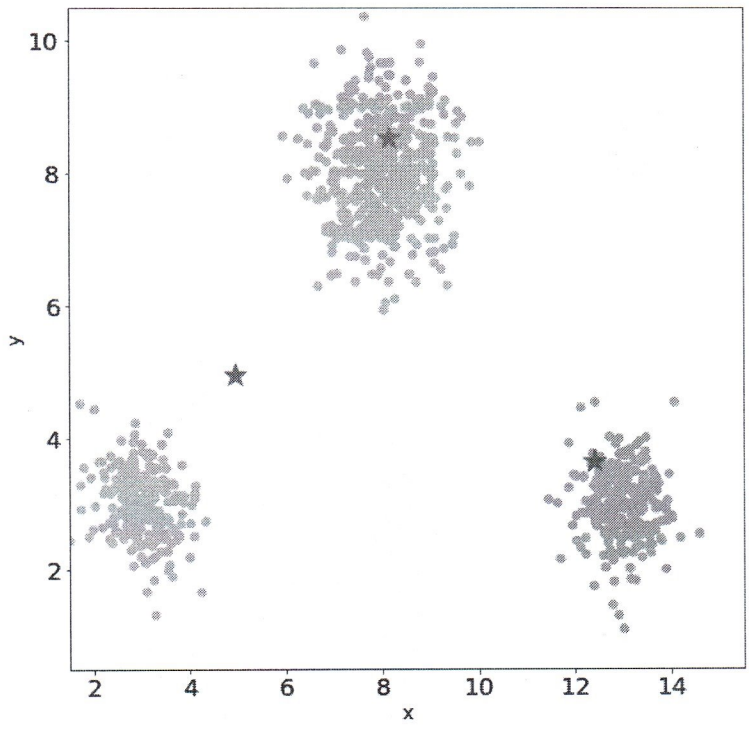


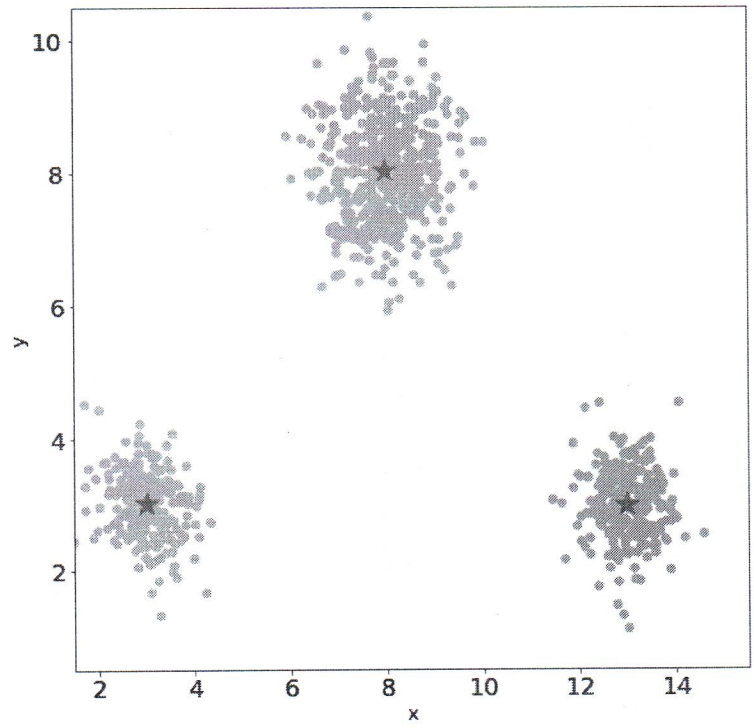
In [34]:
```
# centroids = np.array([[4,8], [10,2]], np.float64)
np.random.seed(1231231231)
centroids = np.random.normal([8,8], [1,1], (3,2))
# centroids
StepByStepKMeans(dataset4,np.array([[7,7],[8.5,9],[10,7.5]]),10,'example4');
```



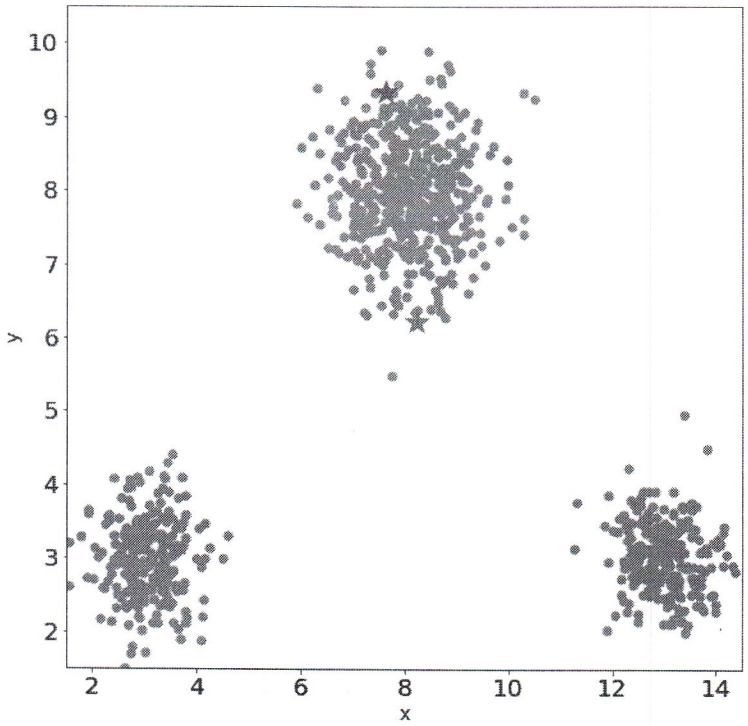

10
8
6
4
2
y
2
4
6
8
10
12
14
x

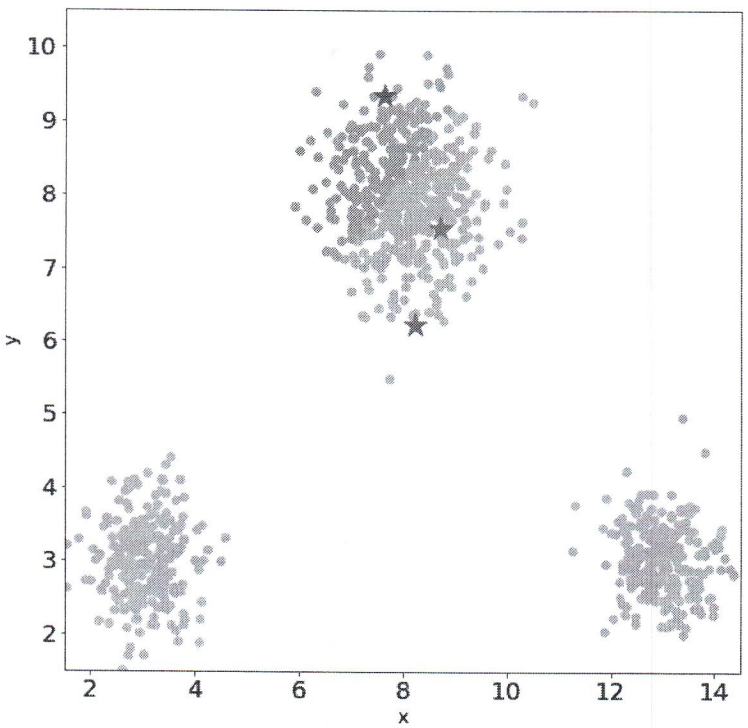

10
8
6
4
2
y
2
4
6
8
10
12
14
x

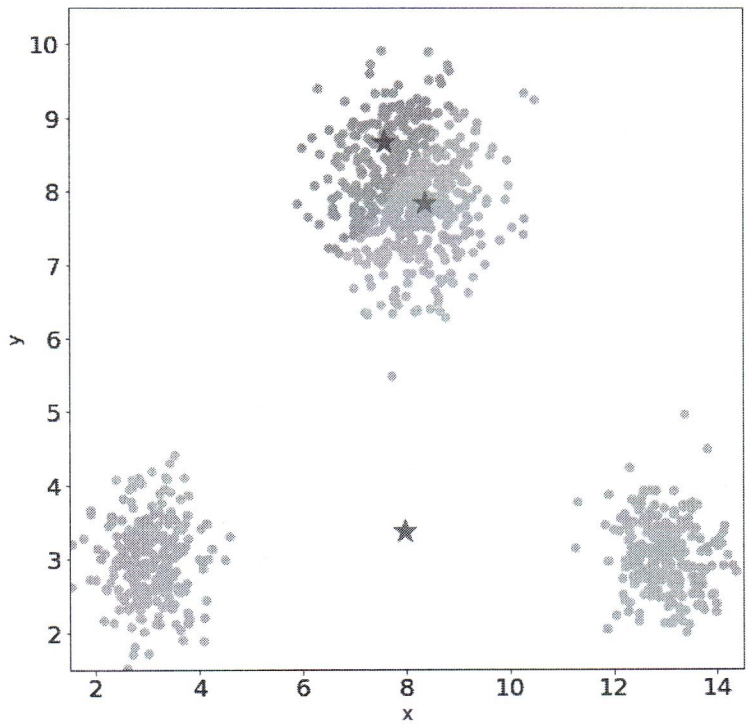


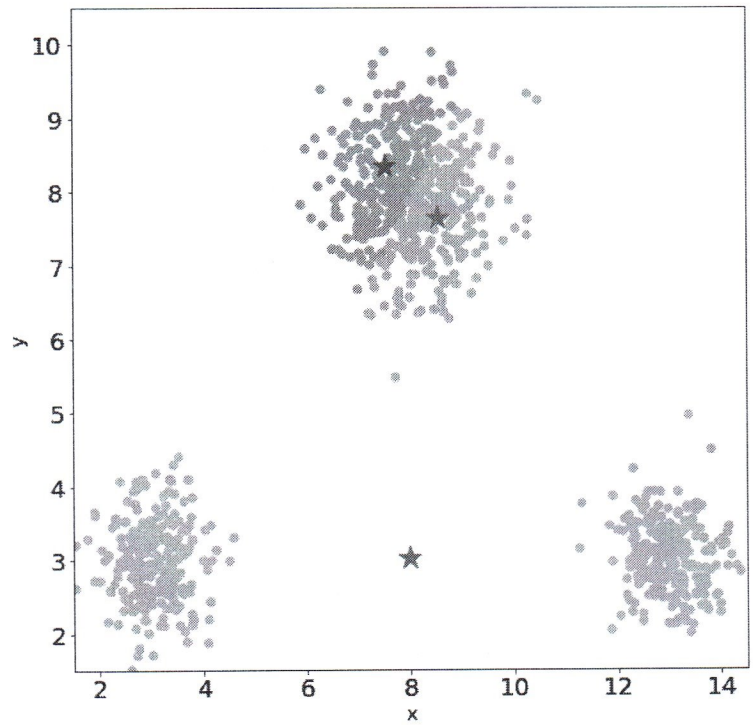


In [12]:
```
parameters = GetParameters(dataset4)
np.random.seed(1234)
centroids = np.random.normal([8,8], [.5,1.5], (3,2))
StepByStepKMeans(dataset4,centroids,5,'example5');
```


10
9
8
7
6
5
4
3
2
y
2
4
6
8
10
12
14
x


10
9
8
7
6
5
4
3
2
y
2
4
6
8
10
12
14
x





In [ ]:

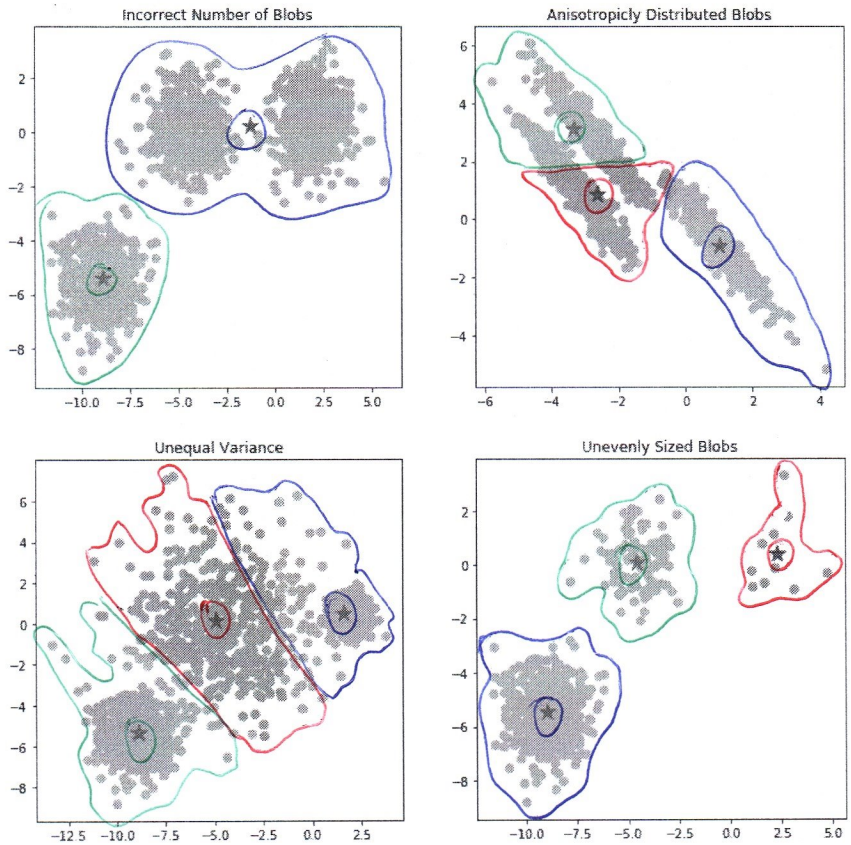
Demonstration of k-means assumptions

This example is meant to illustrate situations where k-means will produce unintuitive and possibly unexpected clusters. In the first three plots, the input data does not conform to some implicit assumption that k-means makes and undesirable clusters are produced as a result. In the last plot, k-means returns intuitive clusters despite unevenly sized blobs.

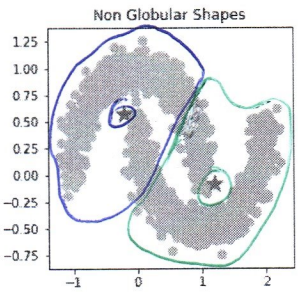
In [23]:
```
print(__doc__)

# Author: Phil Roth <mr.phil.roth@gmail.com>
# License: BSD 3 clause

import numpy as np
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
from sklearn.datasets import make_moons
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
%matplotlib inline

# define the color palettes
data_colors = ['#a6cdf6','#b2d0b7','#f98ea1']
background_cmap3 = ListedColormap(['#a6cdf6','#b2d0b7','#f98ea1'])
centroid_colors = ['#1b80e8','#599062','#e20c32']
centroid_cmap = ListedColormap(centroid_colors)
plt.register_cmap(cmap=background_cmap3)
# plt.register_cmap(cmap=dots_cmap)


plt.figure(figsize=(12, 12))
n_samples = 1500
random_state = 170

##############################################
# Incorrect number of clusters
##############################################
X, y = make_blobs(n_samples=n_samples, random_state=random_state)

model = KMeans(n_clusters=2, random_state=random_state).fit(X)
y_pred = model.predict(X)

plt.subplot(221)
plt.scatter(X[:, 0], X[:, 1], s=50, c=[data_colors[y] for y in y_pred], cmap=background_cmap3)
plt.scatter(model.cluster_centers_[:,0], model.cluster_centers_[:,1], marker='*', s=200, c=centroid_colors[:len(model.cluster_c
plt.title("Incorrect Number of Blobs")


##############################################
# Anisotropicly distributed data
##############################################

transformation = [[0.60834549, -0.63667341], [-0.40887718, 0.85253229]]
X_aniso = np.dot(X, transformation)

model = KMeans(n_clusters=3, random_state=random_state).fit(X_aniso)
y_pred = model.predict(X_aniso)

plt.subplot(222)
plt.scatter(X_aniso[:, 0], X_aniso[:, 1], s=50, c=[data_colors[y] for y in y_pred], cmap=background_cmap3)
plt.scatter(model.cluster_centers_[:,0], model.cluster_centers_[:,1], marker='*', s=200, c=centroid_colors[:len(model.cluster_c
plt.title("Anisotropicly Distributed Blobs")

##############################################
# Different variance
##############################################

X_varied, y_varied = make_blobs(n_samples=n_samples,
                                cluster_std=[1.0, 2.5, 0.5],
                                random_state=random_state)
model = KMeans(n_clusters=3, random_state=random_state).fit(X_varied)
y_pred = model.predict(X_varied)

plt.subplot(223)
# plt.scatter(X_varied[:, 0], X_varied[:, 1], c=y_pred, cmap=dots_cmap)
plt.scatter(X_varied[:, 0], X_varied[:, 1], s=50, c=[data_colors[y] for y in y_pred], cmap=background_cmap3)
plt.scatter(model.cluster_centers_[:,0], model.cluster_centers_[:,1], marker='*', s=200, c=centroid_colors[:len(model.cluster_c
plt.title("Unequal Variance")


##############################################
# Unevenly sized blobs
##############################################

X_filtered = np.vstack((X[y == 0][:500], X[y == 1][:100], X[y == 2][:10]))
model = KMeans(n_clusters=3, random_state=random_state).fit(X_filtered)
y_pred = model.predict(X_filtered)

plt.subplot(224)
# plt.scatter(X_filtered[:, 0], X_filtered[:, 1], c=y_pred, cmap=dots_cmap)
plt.scatter(X_filtered[:, 0], X_filtered[:, 1], s=50, c=[data_colors[y] for y in y_pred], cmap=background_cmap3)
plt.scatter(model.cluster_centers_[:,0], model.cluster_centers_[:,1], marker='*', s=200, c=centroid_colors[:len(model.cluster_c

plt.title("Unevenly Sized Blobs")
```

```
Automatically created module for IPython interactive environment
```

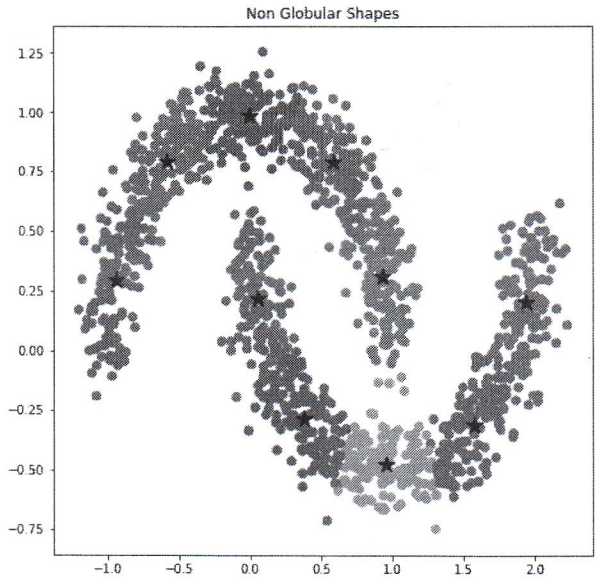
Out[23]:
```
Text(0.5, 1.0, 'Unevenly Sized Blobs')
```



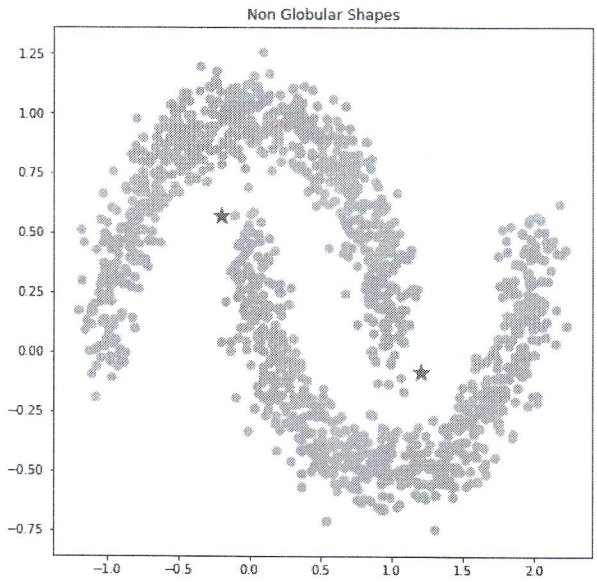
In [31]:
```
plt.figure(figsize=(8, 8))
n_samples = 1500
random_state = 170
X, y = make_moons(n_samples=n_samples, random_state=random_state, noise=0.1)

# Incorrect number of clusters
model = KMeans(n_clusters=2, random_state=random_state).fit(X)
y_pred = model.predict(X)

plt.subplot(221)
plt.scatter(X[:, 0], X[:, 1], s=50, c=[data_colors[y] for y in y_pred], cmap=background_cmap3)
plt.scatter(model.cluster_centers_[:,0], model.cluster_centers_[:,1], marker='*', s=200, c=centroid_colors[:len(model.cluster_c
plt.title("Non Globular Shapes")
plt.savefig("kMeans-NonGlobularShapes-k2.png")
plt.show()
```



In [35]:
```
plt.figure(figsize=(8, 8))
n_samples = 1500
random_state = 170
X, y = make_moons(n_samples=n_samples, random_state=random_state, noise=0.1)

# Incorrect number of clusters
model = KMeans(n_clusters=10, random_state=random_state).fit(X)
y_pred = model.predict(X)

# plt.subplot(221)
plt.scatter(X[:, 0], X[:, 1], s=50, c=y_pred, cmap='Dark2')
plt.scatter(model.cluster_centers_[:,0], model.cluster_centers_[:,1], marker='*', s=200, c='black',)
plt.title("Non Globular Shapes")
plt.savefig("kMeans-NonGlobularShapes-k10.png")
plt.show()
```



In [36]:
```
plt.figure(figsize=(8, 8))
n_samples = 1500
random_state = 170
X, y = make_moons(n_samples=n_samples, random_state=random_state, noise=0.1)

# Incorrect number of clusters
model = KMeans(n_clusters=2, random_state=random_state).fit(X)
y_pred = model.predict(X)

# plt.subplot(221)
plt.scatter(X[:, 0], X[:, 1], s=50, c=[data_colors[y] for y in y_pred], cmap=background_cmap3)
plt.scatter(model.cluster_centers_[:,0], model.cluster_centers_[:,1], marker='*', s=200, c=centroid_colors[:len(model.cluster_c
plt.title("Non Globular Shapes")
plt.savefig("kMeans-NonGlobularShapes-k2.png")
plt.show()
```



In [ ]:

k-Means Clustering

This notebook shows simple examples of k-means clustering, the elbow/knee analysis used to select the most adequate number of clusters, and some examples of the known limitations of k-means.

We start by importing the required libraries

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

from scipy.spatial.distance import cdist, pdist

import matplotlib
%matplotlib inline

# color palette
color_palette1 = matplotlib.colors.ListedColormap([plt.cm.Paired.colors[0],plt.cm.Paired.colors[2],plt.cm.Paired.colors[4]], na
color_palette2 = matplotlib.colors.ListedColormap([plt.cm.Paired.colors[1],plt.cm.Paired.colors[3],plt.cm.Paired.colors[5]], na
```

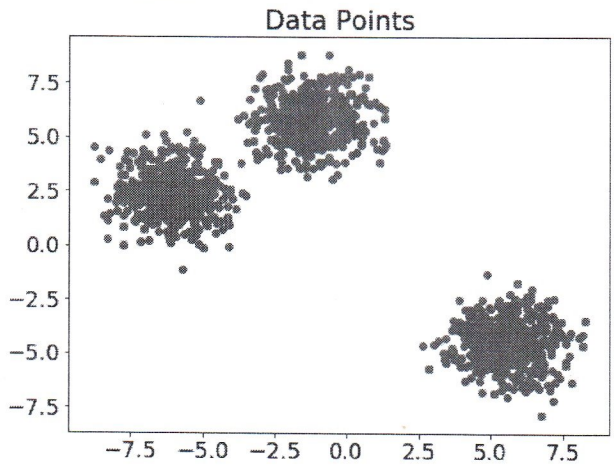
Simple k-Means

To show some examples of k-means we create an artificial dataset comprising three blobs.

In [2]:
```
random_state = 1234 ## another interesting example can be generated using the seed 36
no_clusters = 3
no_samples = 1500

x, y = make_blobs(centers=no_clusters, n_samples=no_samples, random_state=random_state)
```

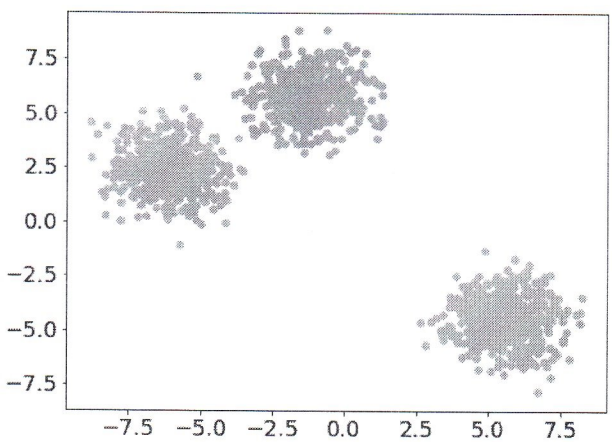
Let's plot the blobs!

In [3]:
```
plt.figure(figsize=(8, 6));
font = {'family' : 'sans', 'size'   : 18}
plt.rc('font', **font)
plt.scatter(x[:,0],x[:,1], cmap=color_palette1);
plt.title("Data Points");
```



First, we apply k-means with the correct number of clusters (k=3)

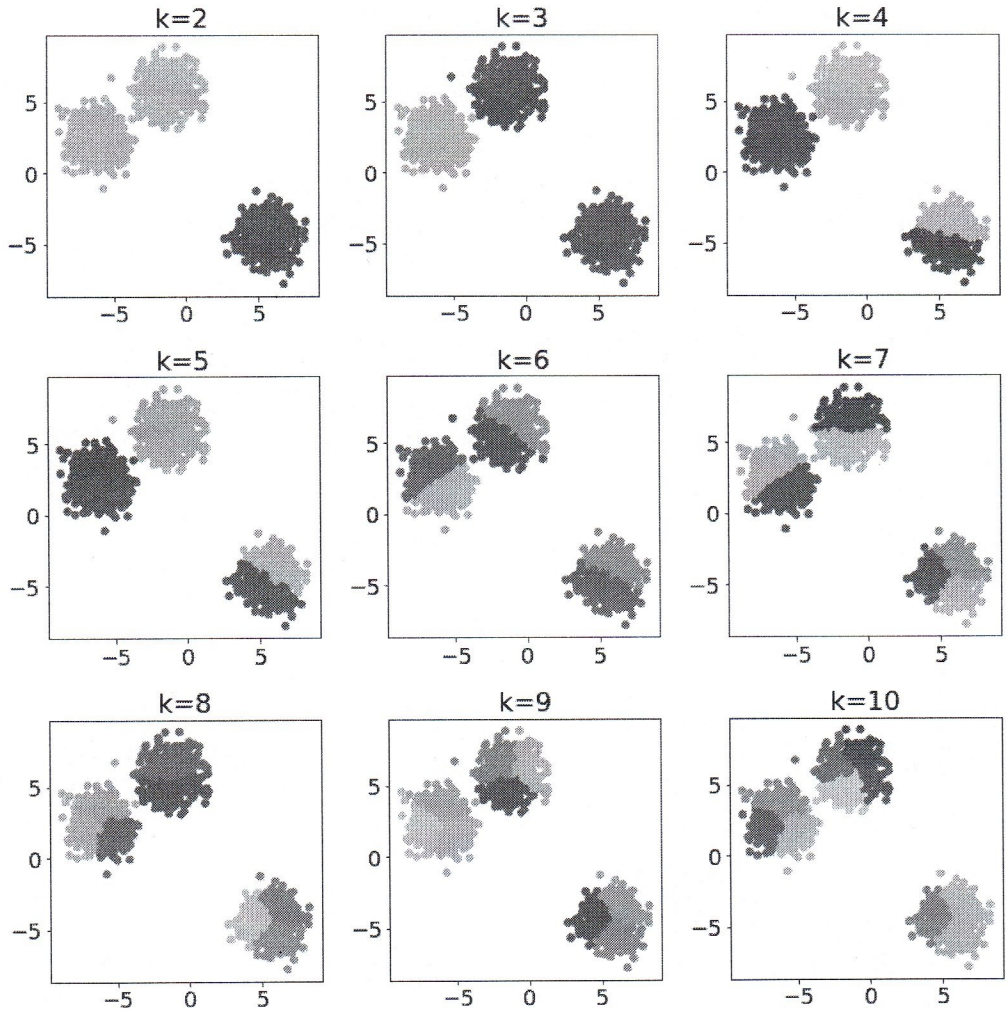
In [4]:
```
yp = KMeans(n_clusters=3).fit_predict(x)
plt.figure(figsize=(8, 6));
font = {'family' : 'sans', 'size'   : 18}
plt.rc('font', **font)
plt.scatter(x[:,0],x[:,1],c=yp,cmap=color_palette1);
```



Next, we apply k-means with different values of k and plot the result.

In [5]:
```
plt.figure(figsize=(12, 12));

for i in range(9):
```

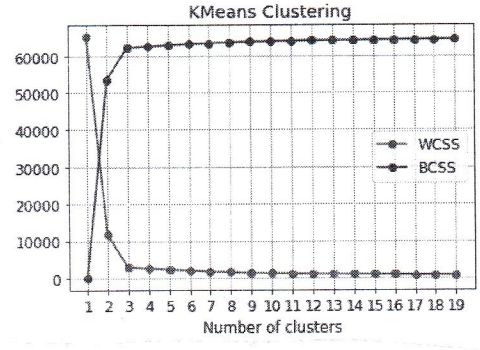
```
    yp = KMeans(n_clusters=(i+2)).fit_predict(x)
    plt.subplot(330+(i+1))
    plt.title('k='+str(i+2))
    plt.scatter(x[:, 0], x[:, 1], c=yp,cmap=plt.get_cmap('Vega20'))

plt.tight_layout()
```



Knee/Elbow Analysis

In [6]:
```
def KneeElbowAnalysis(x,max_k=20):
    k_values = range(1,max_k)
    clusterings = [KMeans(n_clusters=k, random_state=random_state).fit(x) for k in k_values]
    centroids = [clustering.cluster_centers_ for clustering in clusterings]

    D_k = [cdist(x, cent, 'euclidean') for cent in centroids]
    cIdx = [np.argmin(D,axis=1) for D in D_k]
    dist = [np.min(D,axis=1) for D in D_k]
    avgWithinSS = [sum(d)/x.shape[0] for d in dist]

    # Total with-in sum of square
    wcss = [sum(d**2) for d in dist]

    tss = sum(pdist(x)**2)/x.shape[0]
    bss = tss-wcss

    kIdx = 10-1

    #
    # elbow curve
    #
    fig = plt.figure()
    font = {'family' : 'sans', 'size'   : 12}
    plt.rc('font', **font)
    plt.xticks(k_values)
    plt.plot(k_values, wcss, 'bo-', color='red', label='WCSS')
    plt.plot(k_values, bss, 'bo-', color='blue', label='BCSS')
    plt.grid(True)
    plt.xlabel('Number of clusters')
    plt.legend()
    plt.title('KMeans Clustering');
```

In [7]:
```
KneeElbowAnalysis(x)
```



Comparing different clustering algorithms on toy datasets

This example shows characteristics of different clustering algorithms on datasets that are "interesting" but still in 2D. With the exception of the last dataset, the parameters of each of these dataset-algorithm pairs has been tuned to produce good clustering results. Some algorithms are more sensitive to parameter values than others.

The last dataset is an example of a 'null' situation for clustering: the data is homogeneous, and there is no good clustering. For this example, the null dataset uses the same parameters as the dataset in the row above it, which represents a mismatch in the parameter values and the data structure.

While these examples give some intuition about the algorithms, this intuition might not apply to very high dimensional data.

You must have the latest version of sklearn installed. Some versions don't recognize "single" linkage methods.

Connectivity constraints

- constraints that we can put on the merging on hierarchical agglomerative clustering } In this case we put the constraint that a point can be merged only with points that are in the 10 nearest neigh. This prevents to merge clusters that are too far away.

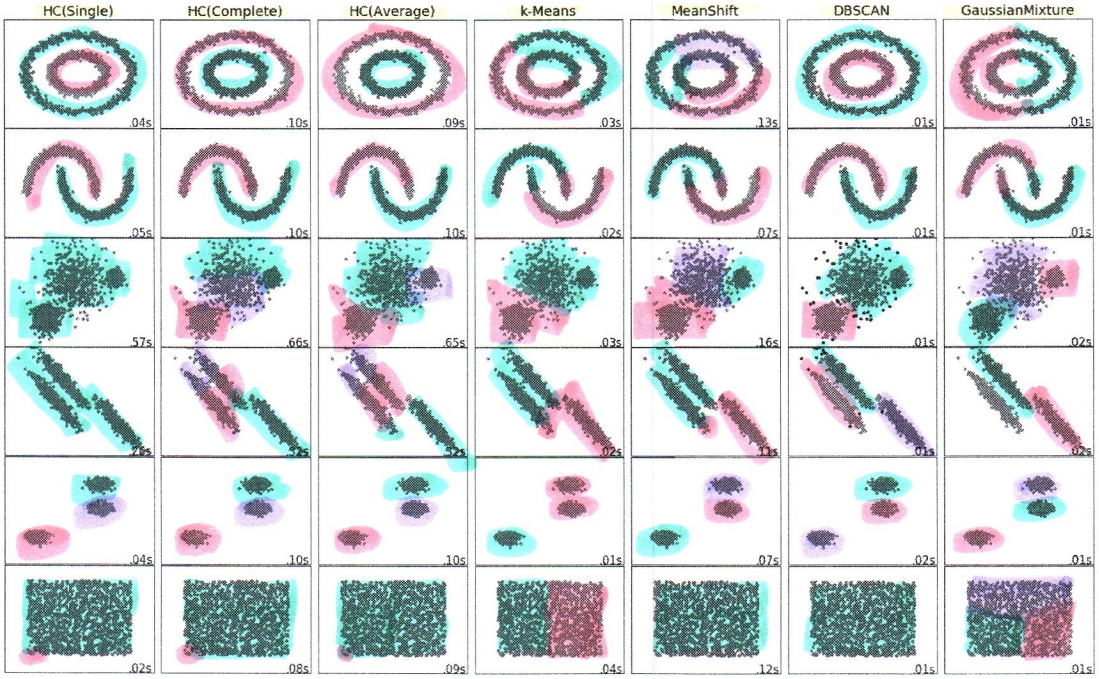
In agglomerative clustering, we can restrict which clusters to join by adding connectivity constraints. These constraints specify which examples are considered connected and only clusters with connected examples, from one cluster to the other, can be joined into larger clusters.

Check the example with and without connectivity. With connectivity, hierarchical clustering with average and complete clustering cannot correctly separate the first two second datasets (the donuts and the half moons). When a connectivity constraint that restricts the connection of each example only to the 10 nearest neighbours (by calling the **kneighbors_graph** function, creates a graph of connections that respects the structure of the data and prevents these inadequate clusters from forming.

In [2]:
```
print(__doc__)

import time
import warnings

import numpy as np
import matplotlib.pyplot as plt

from sklearn import cluster, datasets, mixture
from sklearn.neighbors import kneighbors_graph
from sklearn.preprocessing import StandardScaler
from itertools import cycle, islice
%matplotlib inline

np.random.seed(0)

# ============
# Generate datasets. We choose the size big enough to see the scalability
# of the algorithms, but not too big to avoid too long running times
# ============
n_samples = 1500
noisy_circles = datasets.make_circles(n_samples=n_samples, factor=.5,
                                      noise=.05)
noisy_moons = datasets.make_moons(n_samples=n_samples, noise=.05)
blobs = datasets.make_blobs(n_samples=n_samples, random_state=8)
no_structure = np.random.rand(n_samples, 2), None

# Anisotropicly distributed data
random_state = 170
X, y = datasets.make_blobs(n_samples=n_samples, random_state=random_state)
transformation = [[0.6, -0.6], [-0.4, 0.8]]
X_aniso = np.dot(X, transformation)
aniso = (X_aniso, y)

# blobs with varied variances
varied = datasets.make_blobs(n_samples=n_samples,
                             cluster_std=[1.0, 2.5, 0.5],
                             random_state=random_state)

# ============
# Set up cluster parameters
# ============
plt.figure(figsize=(9 * 2 + 3, 12.5))
plt.subplots_adjust(left=.02, right=.98, bottom=.001, top=.96, wspace=.05,
                    hspace=.01)

plot_num = 1

default_base = {'quantile': .3,
                'eps': .3,
                'damping': .9,
                'preference': -200,
                'n_neighbors': 10,
                'n_clusters': 3}

datasets = [
    (noisy_circles, {'damping': .77, 'preference': -240,
                     'quantile': .2, 'n_clusters': 2}),
    (noisy_moons, {'damping': .75, 'preference': -220, 'n_clusters': 2}),
    (varied, {'eps': .18, 'n_neighbors': 2}),
    (aniso, {'eps': .15, 'n_neighbors': 2}),
    (blobs, {}),
    (no_structure, {})]

for i_dataset, (dataset, algo_params) in enumerate(datasets):
    # update parameters with dataset-specific values
    params = default_base.copy()
    params.update(algo_params)

    X, y = dataset

    # normalize dataset for easier parameter selection
    X = StandardScaler().fit_transform(X)
```

```
    # estimate bandwidth for mean shift
    bandwidth = cluster.estimate_bandwidth(X, quantile=params['quantile'])

    # connectivity matrix for structured Ward
    connectivity = kneighbors_graph(
        X, n_neighbors=params['n_neighbors'], include_self=False)
    # make connectivity symmetric
    connectivity = 0.5 * (connectivity + connectivity.T)

    # ============
    # Create cluster objects
    # ============

    use_connectivity = True

    if (use_connectivity):
        single_linkage = cluster.AgglomerativeClustering(
            linkage="single",
            connectivity = connectivity,
            n_clusters=params['n_clusters'])
        complete_linkage = cluster.AgglomerativeClustering(
            linkage="complete",
            connectivity = connectivity,
            n_clusters=params['n_clusters'])
        average_linkage = cluster.AgglomerativeClustering(
            linkage="average",
            connectivity = connectivity,
            n_clusters=params['n_clusters'])
    else:
        single_linkage = cluster.AgglomerativeClustering(
            linkage="single",
            n_clusters=params['n_clusters'])
        complete_linkage = cluster.AgglomerativeClustering(
            linkage="complete",
            n_clusters=params['n_clusters'])
        average_linkage = cluster.AgglomerativeClustering(
            linkage="average",
            n_clusters=params['n_clusters'])
    kmeans = cluster.KMeans(n_clusters=2, init='random')
    ms = cluster.MeanShift(bandwidth=bandwidth, bin_seeding=True)
    dbscan = cluster.DBSCAN(eps=params['eps'])
    gmm = mixture.GaussianMixture(
        n_components=params['n_clusters'], covariance_type='full')



    clustering_algorithms = (
        ('HC(Single)', single_linkage),
        ('HC(Complete)', complete_linkage),
        ('HC(Average)', average_linkage),
        ('k-Means', kmeans),
        ('MeanShift', ms),
        ('DBSCAN', dbscan),
        ('GaussianMixture', gmm)
    )

    for name, algorithm in clustering_algorithms:
        t0 = time.time()

        # catch warnings related to kneighbors_graph
        with warnings.catch_warnings():
            warnings.filterwarnings(
                "ignore",
                message="the number of connected components of the " +
                "connectivity matrix is [0-9]{1,2}" +
                " > 1. Completing it to avoid stopping the tree early.",
                category=UserWarning)
            warnings.filterwarnings(
                "ignore",
                message="Graph is not fully connected, spectral embedding" +
                " may not work as expected.",
                category=UserWarning)
            algorithm.fit(X)

        t1 = time.time()
        if hasattr(algorithm, 'labels_'):
            y_pred = algorithm.labels_.astype(np.int)
        else:
            y_pred = algorithm.predict(X)

        plt.subplot(len(datasets), len(clustering_algorithms), plot_num)
        if i_dataset == 0:
            plt.title(name, size=18)

        colors = np.array(list(islice(cycle(['#377eb8', '#ff7f00', '#4daf4a',
                                             '#f781bf', '#a65628', '#984ea3',
                                             '#999999', '#e41a1c', '#dede00']),
                                      int(max(y_pred) + 1))))
        # add black color for outliers (if any)
        colors = np.append(colors, ["#000000"])
        plt.scatter(X[:, 0], X[:, 1], s=10, color=colors[y_pred])

        plt.xlim(-2.5, 2.5)
        plt.ylim(-2.5, 2.5)
        plt.xticks(())
        plt.yticks(())
        plt.text(.99, .01, ('%.2fs' % (t1 - t0)).lstrip('0'),
                 transform=plt.gca().transAxes, size=15,
                 horizontalalignment='right')
        plot_num += 1

plt.show()
```

```
Automatically created module for IPython interactive environment
```


HC(Single)
HC(Complete)
HC(Average)
k-Means
MeanShift
DBSCAN
GaussianMixture
.04s
.10s
.09s
.03s
.13s
.01s
.01s
.05s
.10s
.10s
.02s
.07s
.01s
.01s
.57s
.66s
.65s
.03s
.16s
.01s
.02s
.26s
.32s
.32s
.02s
.11s
.01s
.02s
.04s
.10s
.10s
.01s
.07s
.02s
.01s
.02s
.08s
.09s
.04s
.12s
.01s
.01s

In [ ]:

Silhouette Analysis

Silhouette is both a measure of cohesion and separation of clusters. It is based on the difference between the average distance to points in the closest cluster and to points in the same cluster. For each point x_i we calculate its silhouette coefficient s_i as,

$$s(x_i) = \frac{b(x_i) - a(x_i)}{max\{a(x_i), b(x_i)\}}$$

Where $a(x_i)$ is the average distance between item x_i and all other data within the same cluster; $b(x_i)$ is the mean distances from x_i to points in the closest cluster.

The s_i value of a point x_i lies in the interval [−1,+1]:

- A value close to +1 indicates that x_i is much closer to points in its own cluster and is far from other clusters.
- A value close to zero indicates that x_i is close to the boundary between two clusters.
- A value close to −1 indicates that x_i is much closer to another cluster than its own cluster, and therefore, the point may be mis-clustered.

Silhouette Coefficient

The Silhouette Coefficient is defined as the mean s_i value across all the points. A value close to +1 indicates a good clustering.

$$SC = \frac{1}{n} \sum_i s_i$$

Drawbacks: note that the Silhouette Coefficient is generally higher for convex clusters than other concepts of clusters, such as density based clusters like those obtained through DBSCAN.

Example of silhouette analysis on k-Means clustering

Silhouette analysis can be used to study the separation distance between the resulting clusters. The silhouette plot displays a measure of how close each point in one cluster is to points in the neighboring clusters and thus provides a way to assess parameters like number of clusters visually. This measure has a range of [-1, 1].

Silhouette coefficients (as these values are referred to as) near +1 indicate that the sample is far away from the neighboring clusters. A value of 0 indicates that the sample is on or very close to the decision boundary between two neighboring clusters and negative values indicate that those samples might have been assigned to the wrong cluster.

In this example the silhouette analysis is used to choose an optimal value for `n_clusters` .

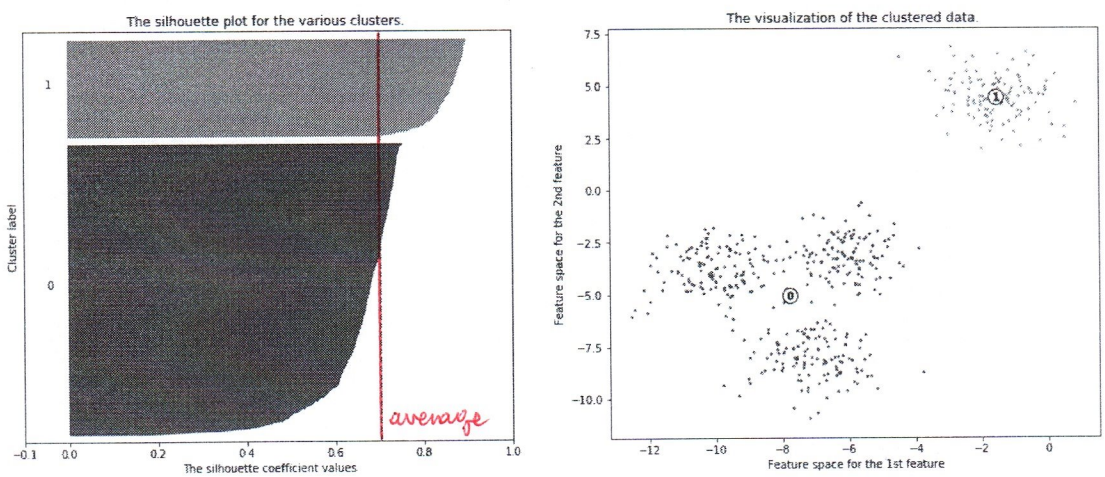
In [3]:
```
from __future__ import print_function

from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score

import matplotlib.pyplot as plt
import matplotlib.cm as cm
%matplotlib inline
import numpy as np

print(__doc__)

# Generating the sample data from make_blobs
# This particular setting has one distinct cluster and 3 clusters placed close
# together.
X, y = make_blobs(n_samples=500,
                  n_features=2,
                  centers=4,
                  cluster_std=1,
                  center_box=(-10.0, 10.0),
                  shuffle=True,
                  random_state=1)  # For reproducibility

range_n_clusters = [2, 3, 4, 5, 6]

for n_clusters in range_n_clusters:
    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -1, 1 but in this example all
    # lie within [-0.1, 1]
    ax1.set_xlim([-0.1, 1])
    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])

    # Initialize the clusterer with n_clusters value and a random generator
    # seed of 10 for reproducibility.
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict(X)

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg = silhouette_score(X, cluster_labels)
    print("For n_clusters =", n_clusters,
          "The average silhouette_score is :", silhouette_avg)

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(X, cluster_labels)

    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
```

```
        # cluster i, and sort them
        ith_cluster_silhouette_values = \
            sample_silhouette_values[cluster_labels == i]

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
                          0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.7)

        # Label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples

    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    # The vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

    # 2nd Plot showing the actual clusters formed
    colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
    ax2.scatter(X[:, 0], X[:, 1], marker='.', s=30, lw=0, alpha=0.7,
                c=colors, edgecolor='k')

    # Labeling the clusters
    centers = clusterer.cluster_centers_
    # Draw white circles at cluster centers
    ax2.scatter(centers[:, 0], centers[:, 1], marker='o',
                c="white", alpha=1, s=200, edgecolor='k')

    for i, c in enumerate(centers):
        ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1,
                    s=50, edgecolor='k')

    ax2.set_title("The visualization of the clustered data.")
    ax2.set_xlabel("Feature space for the 1st feature")
    ax2.set_ylabel("Feature space for the 2nd feature")

    plt.suptitle(("Silhouette analysis for KMeans clustering on sample data "
                  "with n_clusters = %d" % n_clusters),
                 fontsize=14, fontweight='bold')

plt.show()
```

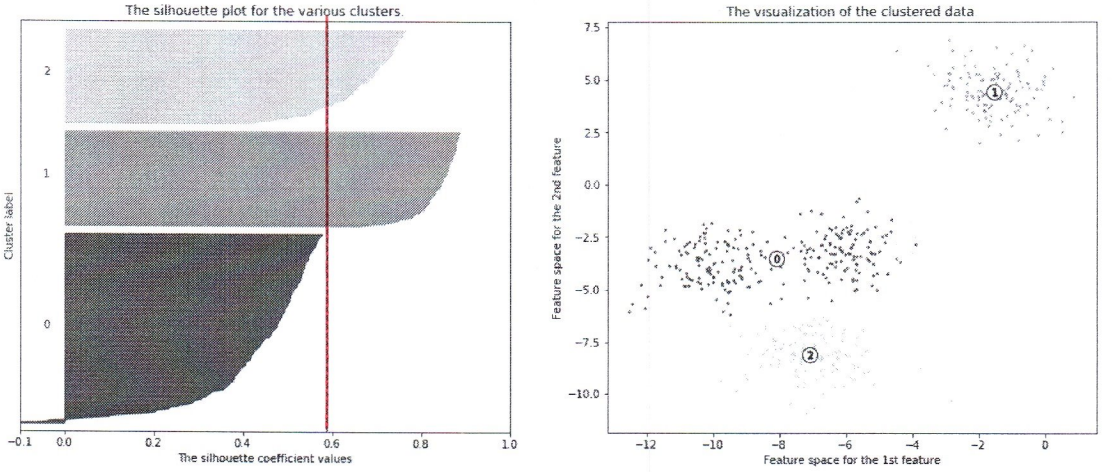
```
Automatically created module for IPython interactive environment
For n_clusters = 2 The average silhouette_score is : 0.7049787496083262
For n_clusters = 3 The average silhouette_score is : 0.5882004012129721
For n_clusters = 4 The average silhouette_score is : 0.6505186632729437
For n_clusters = 5 The average silhouette_score is : 0.56376469026194
For n_clusters = 6 The average silhouette_score is : 0.4504666294372765
```

highest

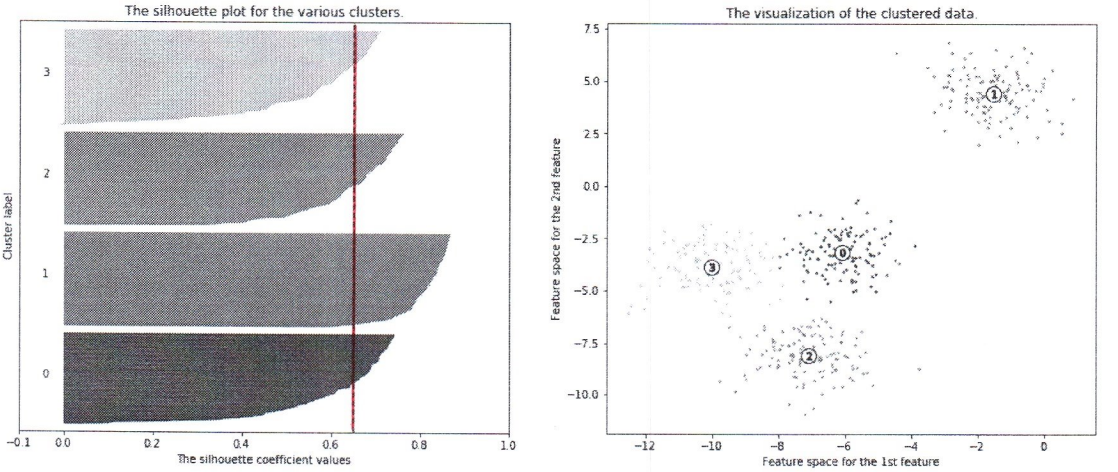
we know that are 4 clusters



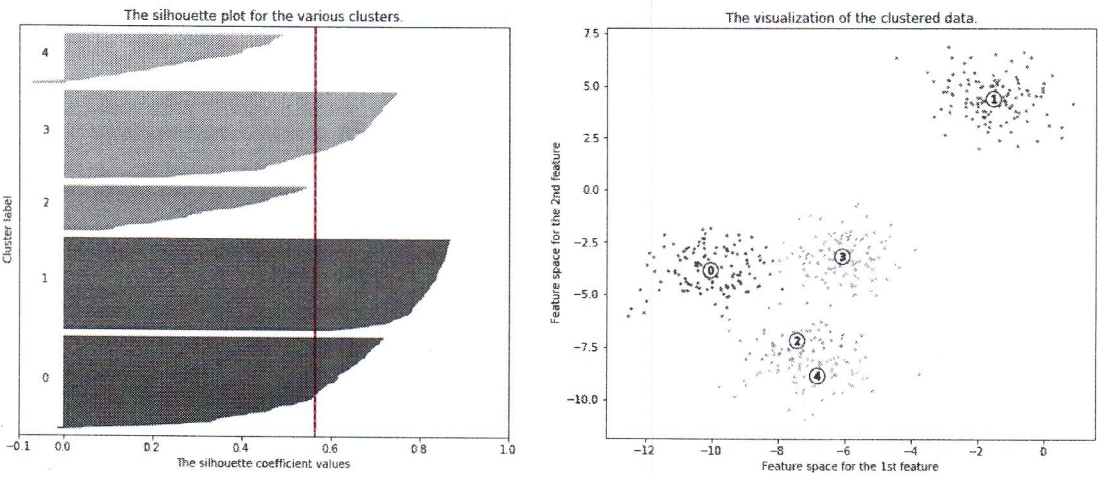
Silhouette analysis for KMeans clustering on sample data with n_clusters = 3



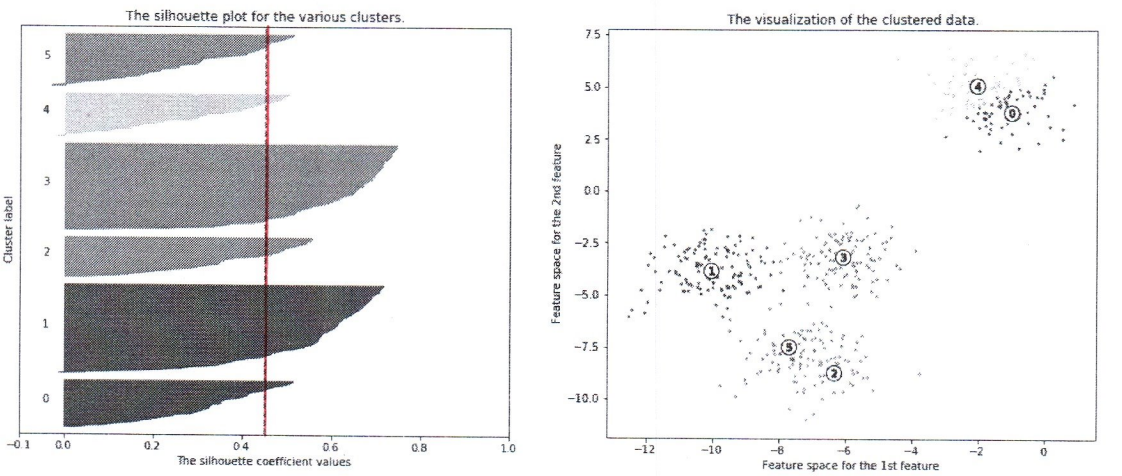
Silhouette analysis for KMeans clustering on sample data with n_clusters = 4



Silhouette analysis for KMeans clustering on sample data with n_clusters = 5



Silhouette analysis for KMeans clustering on sample data with n_clusters = 6



The silhouette plot shows that the `n_clusters` value of 3, 5 and 6 are a bad pick for the given data due to the presence of clusters with below

average silhouette scores and also due to wide fluctuations in the size of the silhouette plots. Silhouette analysis is more ambivalent in deciding between 2 and 4.

Also from the thickness of the silhouette plot the cluster size can be visualized. The silhouette plot for cluster 0 when `n_clusters` is equal to 2, is bigger in size owing to the grouping of the 3 sub clusters into one big cluster. However when the `n_clusters` is equal to 4, all the plots are more or less of similar thickness and hence are of similar sizes as can be also verified from the labelled scatter plot on the right.

In [ ]:

Silhouette Analysis for Iris

We apply the silhouette analysis to the Iris dataset.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn import datasets
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from scipy.spatial.distance import cdist, pdist

import matplotlib.pyplot as plt
import matplotlib.cm as cm
%matplotlib inline
import numpy as np



np.set_printoptions(precision=5, suppress=True)  # suppress scientific float notation
```

In [2]:
```
iris = datasets.load_iris()
target = np.array(iris.target)

print("Number of examples: ", iris.data.shape[0])
print("Number of variables:", iris.data.shape[0])
print("Variable names:     ", iris.feature_names)
print("Target values:      ", iris.target_names)
print("Class Distribution  ", [(x,sum(target==x)) for x in np.unique(target)])
```

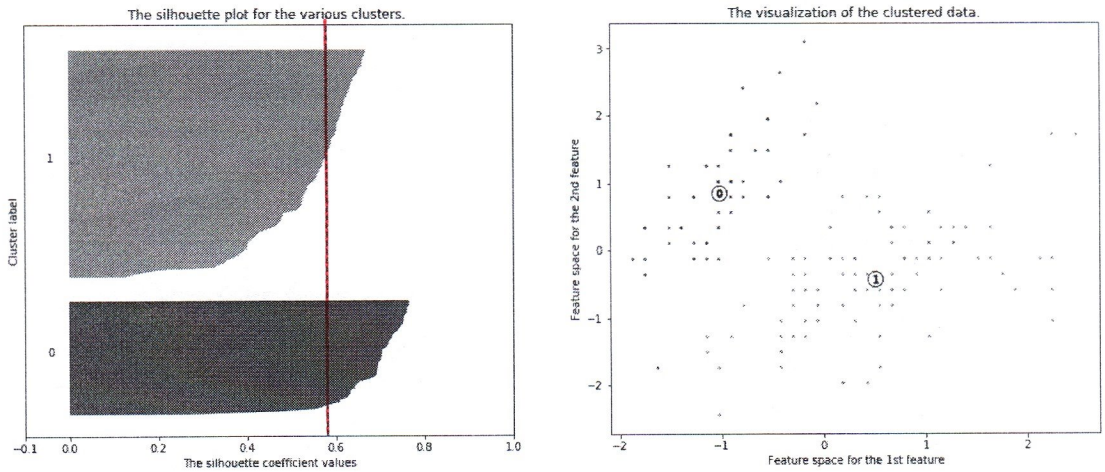
```
Number of examples:  150
Number of variables: 150
Variable names:      ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Target values:       ['setosa' 'versicolor' 'virginica']
Class Distribution   [(0, 50), (1, 50), (2, 50)]
```

In [3]:
```
scaler = StandardScaler()
X = scaler.fit_transform(iris.data)

range_n_clusters = [2, 3, 4, 5, 6]

for n_clusters in range_n_clusters:
    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -1, 1 but in this example all
    # lie within [-0.1, 1]
    ax1.set_xlim([-0.1, 1])
    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])

    # Initialize the clusterer with n_clusters value and a random generator
    # seed of 10 for reproducibility.
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict(X)

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg = silhouette_score(X, cluster_labels)
    print("For n_clusters =", n_clusters,
          "The average silhouette_score is :", silhouette_avg)

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(X, cluster_labels)

    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = \
            sample_silhouette_values[cluster_labels == i]

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
                          0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.7)

        # Label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples

    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    # The vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])
```

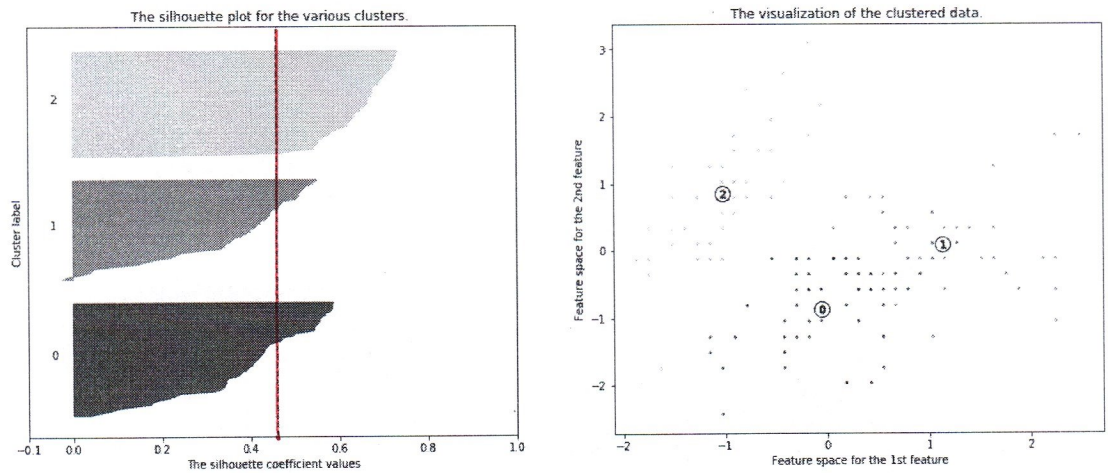
```
    # 2nd Plot showing the actual clusters formed
    colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
    ax2.scatter(X[:, 0], X[:, 1], marker='.', s=30, lw=0, alpha=0.7,
                c=colors, edgecolor='k')

    # Labeling the clusters
    centers = clusterer.cluster_centers_
    # Draw white circles at cluster centers
    ax2.scatter(centers[:, 0], centers[:, 1], marker='o',
                c="white", alpha=1, s=200, edgecolor='k')

    for i, c in enumerate(centers):
        ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1,
                    s=50, edgecolor='k')

    ax2.set_title("The visualization of the clustered data.")
    ax2.set_xlabel("Feature space for the 1st feature")
    ax2.set_ylabel("Feature space for the 2nd feature")

    plt.suptitle(("Silhouette analysis for KMeans clustering on sample data "
                  "with n_clusters = %d" % n_clusters),
                 fontsize=14, fontweight='bold')

plt.show()
```

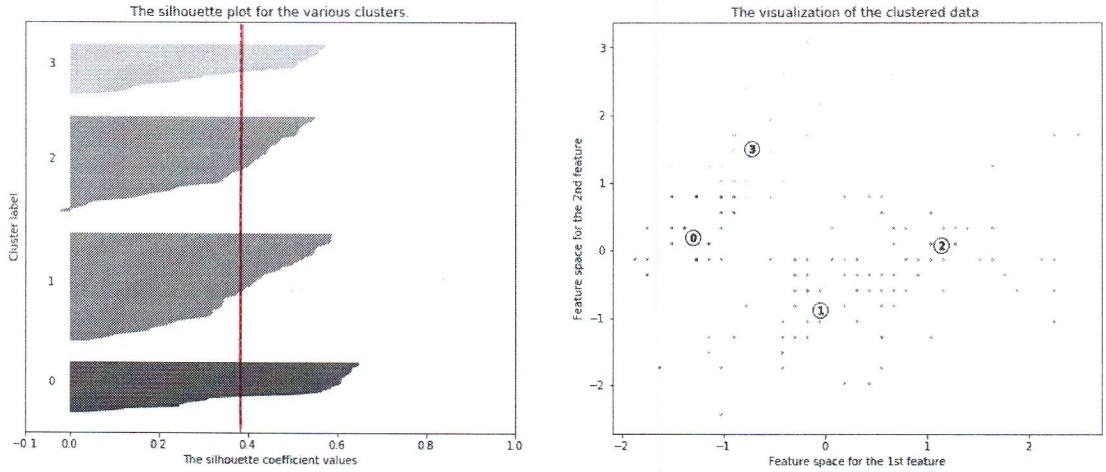
```
For n_clusters = 2 The average silhouette_score is : 0.5817500491982808
For n_clusters = 3 The average silhouette_score is : 0.45994823920518635
For n_clusters = 4 The average silhouette_score is : 0.383850922475103
For n_clusters = 5 The average silhouette_score is : 0.34273996820787694
For n_clusters = 6 The average silhouette_score is : 0.3239092632329394
```

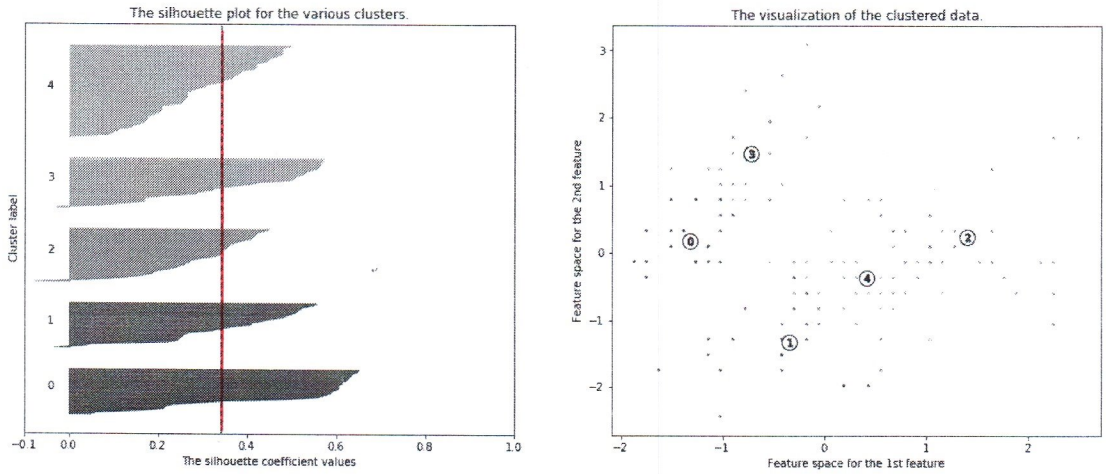
Silhouette analysis for KMeans clustering on sample data with n_clusters = 2

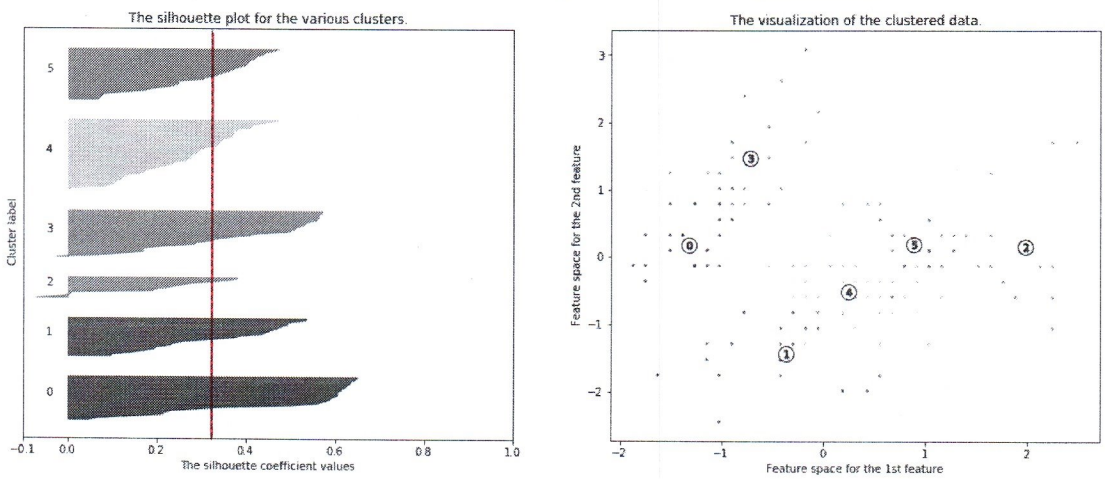


Silhouette analysis for KMeans clustering on sample data with n_clusters = 3









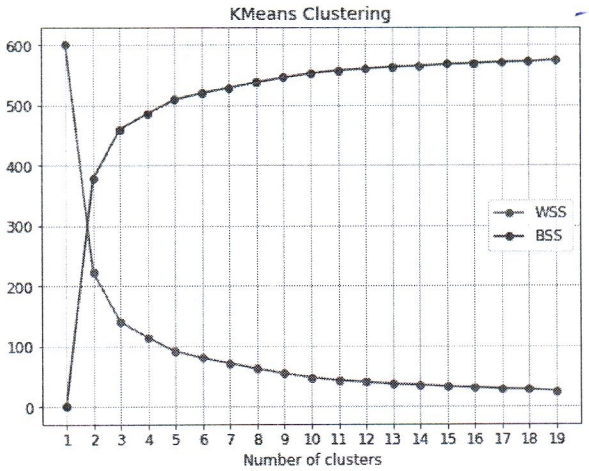
How many clusters should we choose?

Silhouette analysis provides some guidelines but it is just one perspective and we should try more approaches and looks for some sort of agreement. For instance, we might check what the elbow and knee analysis has to say about this.

In [4]:
```
def KneeElbowAnalysis(x,k_values=range(1,20)):
    clusterings = [KMeans(n_clusters=k, random_state=random_state).fit(x) for k in k_values]
    centroids = [clustering.cluster_centers_ for clustering in clusterings]

    D_k = [cdist(x, cent, 'euclidean') for cent in centroids]
    cIdx = [np.argmin(D,axis=1) for D in D_k]
    dist = [np.min(D,axis=1) for D in D_k]
    avgWithinSS = [sum(d)/x.shape[0] for d in dist]

    # Total with-in sum of square
    wss = [sum(d**2) for d in dist]

    tss = sum(pdist(x)**2)/x.shape[0]
    bss = tss-wss

    return wss, bss
```

In [7]:
```
random_state=1234
k_values = range(1,20)
wss,bss = KneeElbowAnalysis(X,k_values)
```

In [8]:
```
fig = plt.figure(figsize=(8,6))
font = {'family' : 'sans', 'size'   : 12}
plt.rc('font', **font)
plt.xticks(k_values)
plt.plot(k_values, wss, 'bo-', color='red', label='WSS')
plt.plot(k_values, bss, 'bo-', color='blue', label='BSS')
plt.grid(True)
plt.xlabel('Number of clusters')
plt.legend()
plt.title('KMeans Clustering');
```



- Not so efficient, we should consider other method of clustering (also the silhouette analysis suggests it). There is **no** clear elbow.
Notice that Silhouette is a complementary analysis, not a substitutive for the elbow analysis. !