Hierarchical Clustering Part 1 - Single Linkage

This notebook shows simple examples of hierarchical clustering and the elbow/knee analysis used to select the most adequate number of clusters.

The notebook is an adaptation of

https://joernhees.de/blog/2015/08/26/scipy-hierarchical-clustering-and-dendrogram-tutorial/

We start by importing the required libraries.

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

from sklearn.datasets import make_blobs

# we are using the scipy implementation
from scipy.spatial.distance import cdist, pdist
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster, inconsistent

%matplotlib inline

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

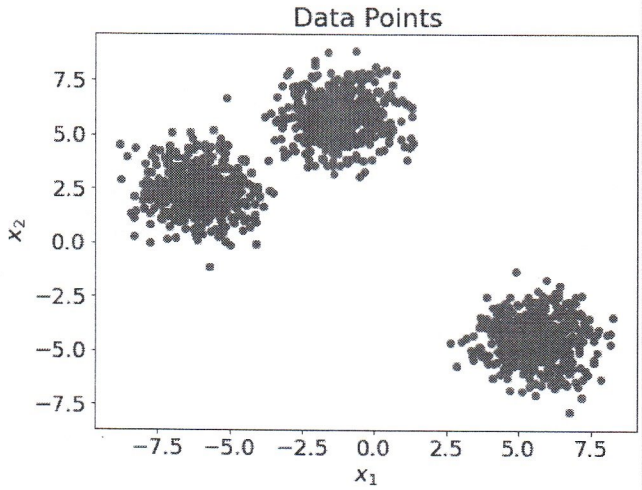
Example Data

We generate some example data with 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=plt.get_cmap('Pastel1'));
plt.xlabel("$x_1$")
plt.ylabel("$x_2$")
plt.title("Data Points");
```



First, we generate the dendrogram using single linkage. The matrix z contains one raw for every merge operation.

In [4]:
```
Z = linkage(X, 'single')
print('Z has %d rows'%(len(Z)))
```

Z = vector of merges (1499 merges)

```
Z has 1499 rows
```

In [5]:
```
print('The first merge was done between %d and %d based on a distance of %.5f '%(Z[0][0],Z[0][1],Z[0][2]))
```

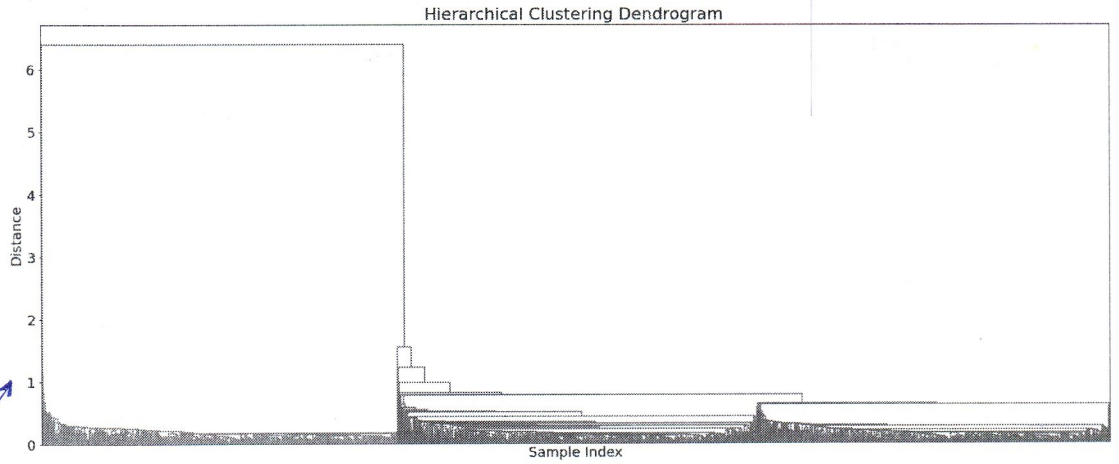
```
The first merge was done between 1247 and 1466 based on a distance of 0.00100
```

the two datapoints which were merged

Now let's plot the full dendrogram.

In [6]:
```
plt.figure(figsize=(25, 10))
font = {'family' : 'sans', 'size'   : 18}
plt.rc('font', **font)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
dendrogram(Z,
    leaf_rotation=90.,  # rotates the x axis labels
    leaf_font_size=8.,  # font size for the x axis labels
)
plt.xticks([])
plt.show()
```

Dendrogram plot



(It's colored)

Number of Cluster Selection using "Inconsistency"

The colors identify the clusters that have been automatically selected by SciPy using "inconsistency that works by identifying the number of clusters by comparing the current height of the merge with the average height of the previous merges, normalized by their standard deviation.

$$inconsistency = \frac{h - avg}{std}$$

where h is the current height, avg is the average merge height over a window of previous merges (identified by a parameter d or depth), and std is the standard deviation of the previous merge heights over a window of size d. Links that join distinct clusters have a high inconsistency coefficient; links that join indistinct clusters have a low inconsistency coefficient.

As an example we can print the inconsistency of the final merges based on a window size of 5:

In [7]:
```
inconsistency = inconsistent(Z, d=5)

for i in reversed(range(1,15)):
    print("from %d to %d => Inconsistency %.3f"%(i,i+1,inconsistency[-i][3]))
```

```
from 14 to 15 => Inconsistency 0.752
from 13 to 14 => Inconsistency 1.026
from 12 to 13 => Inconsistency 0.886
from 11 to 12 => Inconsistency 1.755
from 10 to 11 => Inconsistency 1.204
from 9 to 10 => Inconsistency 2.500
from 8 to 9 => Inconsistency 1.690
from 7 to 8 => Inconsistency 1.427
from 6 to 7 => Inconsistency 1.464
from 5 to 6 => Inconsistency 1.627
from 4 to 5 => Inconsistency 1.615
from 3 to 4 => Inconsistency 1.599
from 2 to 3 => Inconsistency 1.501
from 1 to 2 => Inconsistency 2.635
```

As can be noted we have a higher inconsistendy between k=1 and k=2 but also for the merge between k=9 and k=10. This might suggest k=2 as a possible partition (as suggested in the dendrogram). However, inconsistency values depend on the size of the window (the depth used). So if we print the inconsistency using a higher depth value, the values change. Still the higher inconsistency coefficient is higher for the merge between 2 and 1, but its values for the previous merges are higher than the one for the 9 to 10 clusters merge.

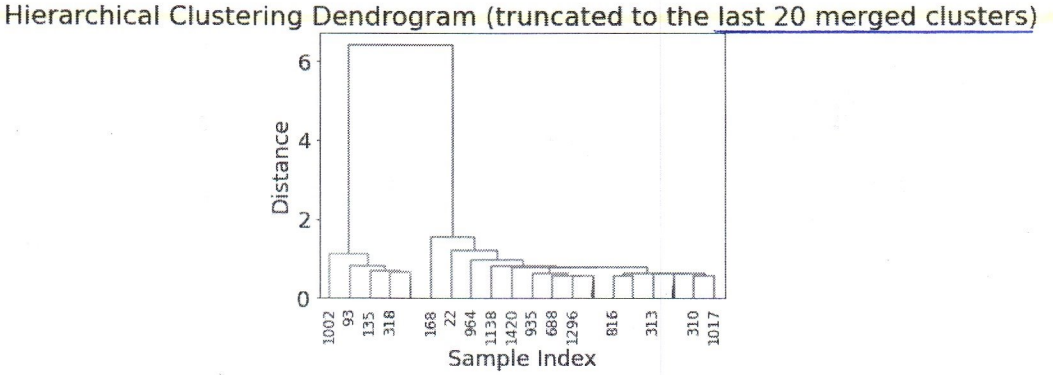
In [8]:
```
inconsistency = inconsistent(Z, d=10)

for i in reversed(range(1,15)):
    print("from %d to %d => Inconsistency %.3f"%(i,i+1,inconsistency[-i][3]))
```

```
from 14 to 15 => Inconsistency 1.393
from 13 to 14 => Inconsistency 1.479
from 12 to 13 => Inconsistency 1.210
from 11 to 12 => Inconsistency 1.697
from 10 to 11 => Inconsistency 1.423
from 9 to 10 => Inconsistency 2.022
from 8 to 9 => Inconsistency 1.734
from 7 to 8 => Inconsistency 1.751
from 6 to 7 => Inconsistency 1.832
from 5 to 6 => Inconsistency 2.318
from 4 to 5 => Inconsistency 2.259
from 3 to 4 => Inconsistency 2.679
from 2 to 3 => Inconsistency 2.910
from 1 to 2 => Inconsistency 4.675
```

Analyzing the Final Merges

To select the value of k we focus on the final part of the dendrogram which confirm that the last merges involve clusters of single values.

In [9]:
```
plt.title('Hierarchical Clustering Dendrogram (truncated to the last 20 merged clusters)')
font = {'family' : 'sans', 'size'   : 18}
plt.rc('font', **font)
plt.xlabel('Sample Index')
plt.ylabel('Distance')
dendrogram(
    Z,
    truncate_mode='lastp',  # show only the last p merged clusters
    p=20,  # show only the last p merged clusters
    show_leaf_counts=False,  # otherwise numbers in brackets are counts
    leaf_rotation=90.,
    leaf_font_size=12.,
    show_contracted=True,  # to get a distribution impression in truncated branches
)
plt.show()
```

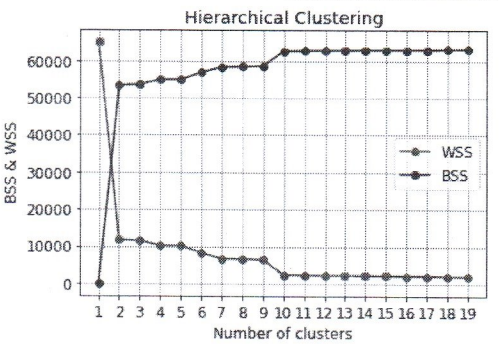


Knee/Elbow Analysis

To decide the number of cluster, let's plot the WSS and BSS curves and perform a knee/elbow analysis. For every k, we compute the clustering from the dendrogram z and compute WSS and BSS.

In [10]:
```
wss_values = []
bss_values = []
k_values = range(1,20)

for k in k_values:
    clustering = fcluster(Z, k, criterion='maxclust')
    centroids = [np.mean(X[clustering==c],axis=0) for c in range(1,k+1)]
    cdist(X, centroids, 'euclidean')
    D = cdist(X, centroids, 'euclidean')
    cIdx = np.argmin(D,axis=1)
    d = np.min(D,axis=1)

    avgWithinSS = sum(d)/len(X)

    # Total with-in sum of square
    wss = sum(d**2)

    tss = sum(pdist(X)**2)/len(X)

    bss = tss-wss

    wss_values += [wss]
    bss_values += [bss]
```

In [11]:
```
fig = plt.figure()
font = {'family' : 'sans', 'size'   : 12}
plt.rc('font', **font)
plt.plot(k_values, wss_values, 'bo-', color='red', label='WSS')
plt.plot(k_values, bss_values, 'bo-', color='blue', label='BSS')
plt.grid(True)
plt.xlabel('Number of clusters')
plt.ylabel('BSS & WSS')
plt.xticks(k_values)
plt.legend()
plt.title('Hierarchical Clustering');
```

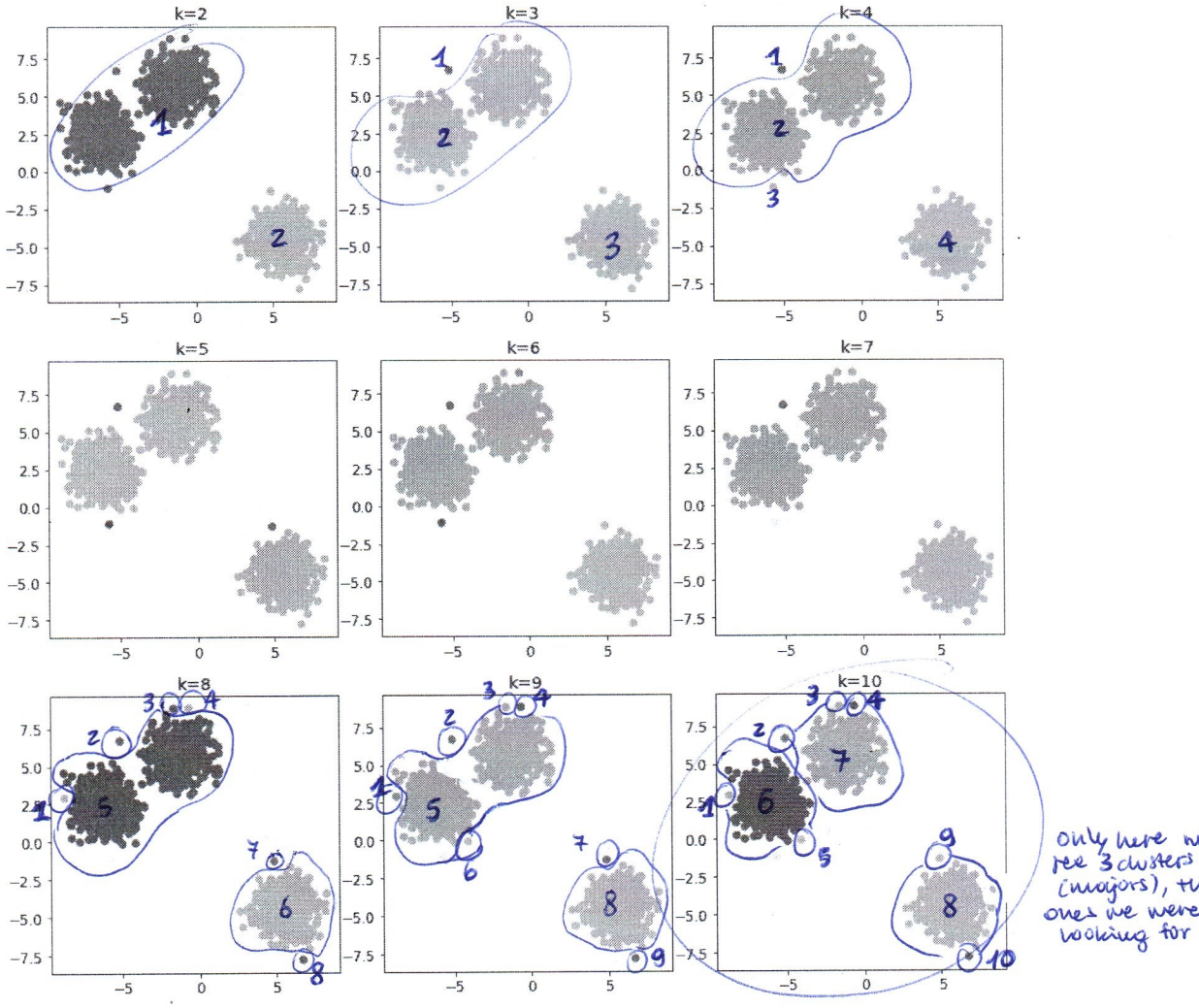


From the chart we note that there is a big drop for k=2 and then WSS and BSS smoothly decrease/increase until k=10. So we might decide to either choose k=2 or k=10.

Visualization of Clustering Solutions

To better understand what happened, let's check the clusterings for all the values of k from 2 to 10.

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

for k in range(2,11):
    yp = fcluster(Z, k, criterion='maxclust')

    plt.subplot(330+(k-1))
    plt.title('k='+str(k))
    plt.scatter(X[:, 0], X[:, 1], c=yp,cmap=plt.get_cmap('Paired'))

plt.tight_layout()
```



Note that, when k=2 the algorithm has separated the two nearby cluster from the very distant one. The next merge operations regard points around the main clusters that are typically nearby existing clusters and thus don't generate significant changes of BSS and WSS until k=10 when the three big clusters and a pletora of one point clusters are found.

To identify this we can analyze the size of the clusters produced for every value of k. As can be noted between 2 and 9 all the clusters that are generated contains one single item. For k=10, the largest cluster splits and afterwards the merges continue to involve clusters of very few elements.

In [13]:
```
for k in range(2,11):
    print ("Clustering using k="+str(k))
    clustering = fcluster(Z, k, criterion='maxclust')
    frequency = np.bincount(clustering)
    index = np.nonzero(frequency)[0]
    for i in zip(index,frequency[index]):
        print("\t"+str(i))
    print("-"*22)
```

```
Clustering using k=2
        (1, 500)
        (2, 1000)
----------------------
Clustering using k=3
        (1, 500)
        (2, 999)
        (3, 1)
----------------------
Clustering using k=4
        (1, 500)
        (2, 998)
        (3, 1)
        (4, 1)
----------------------
Clustering using k=5
        (1, 499)
        (2, 1)
        (3, 998)
        (4, 1)
        (5, 1)
----------------------
Clustering using k=6
        (1, 499)
        (2, 1)
        (3, 997)
        (4, 1)
        (5, 1)
        (6, 1)
----------------------
Clustering using k=7
        (1, 499)
        (2, 1)
        (3, 996)
        (4, 1)
        (5, 1)
        (6, 1)
        (7, 1)
```

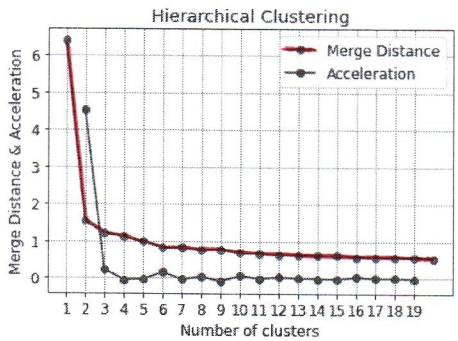
```
----------------------
Clustering using k=8
        (1, 498)
        (2, 1)
        (3, 1)
        (4, 996)
        (5, 1)
        (6, 1)
        (7, 1)
        (8, 1)
----------------------
Clustering using k=9
        (1, 498)
        (2, 1)
        (3, 1)
        (4, 995)
        (5, 1)
        (6, 1)
        (7, 1)
        (8, 1)
        (9, 1)
----------------------
Clustering using k=10
        (1, 498)
        (2, 1)
        (3, 1)
        (4, 499)
        (5, 496)
        (6, 1)
        (7, 1)
        (8, 1)
        (9, 1)
        (10, 1)
----------------------
```

Acceleration (in terms of distance)

The elbow methods can be applied also with other metrics. For instance, we can use it to search for the highest acceleration of merge distance growth. In the example below, we compute such difference from the dendrogam and plot it again the distance of each merge. Then, we look for the largest value and use it to select the number of clusters.

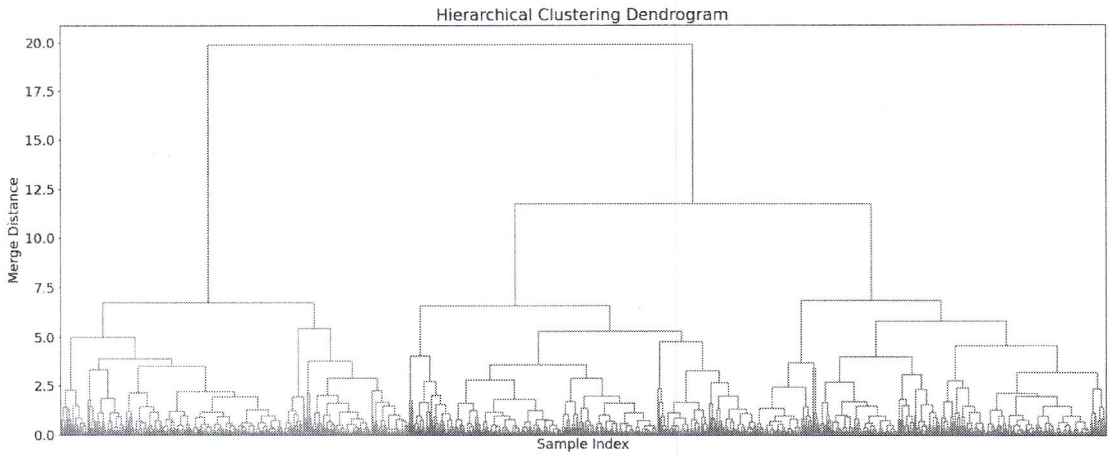
In [14]:
```
merge_distance = Z[-20:, 2]
reverse_merge_distance = merge_distance[::-1]
index = np.arange(1,len(merge_distance)+1)

acceleration = np.diff(merge_distance, 2)  # 2nd derivative of the distances
reversed_acceleration = acceleration[::-1]
```

In [15]:
```
fig = plt.figure()
font = {'family' : 'sans', 'size'   : 12}
plt.rc('font', **font)
plt.plot(index, reverse_merge_distance, 'bo-', color='red', label='Merge Distance')
plt.plot(index[:-2]+1, reversed_acceleration, 'bo-', color='blue', label='Acceleration')
plt.grid(True)
plt.xlabel('Number of clusters')
plt.ylabel('Merge Distance & Acceleration')
plt.xticks(k_values)
plt.legend()
plt.title('Hierarchical Clustering');
```



In [16]:
```
print("The number of cluster suggested using acceleration is %d"%(reversed_acceleration.argmax() + 2))
```

```
The number of cluster suggested using acceleration is 2
```

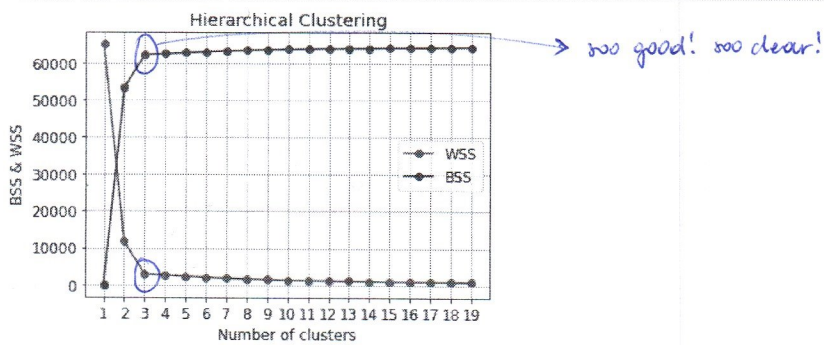


Knee/Elbow Analysis

To decide the number of cluster, let's plot the WSS and BSS curves and perform a knee/elbow analysis. For every k, we compute the clustering from the dendrogram z and compute WSS and BSS.

In [6]:
```
wss_values = []
bss_values = []
k_values = range(1,20)

for k in k_values:
    clustering = fcluster(Z, k, criterion='maxclust')
    centroids = [np.mean(X[clustering==c],axis=0) for c in range(1,k+1)]
    cdist(X, centroids, 'euclidean')
    D = cdist(X, centroids, 'euclidean')
    cIdx = np.argmin(D,axis=1)
    d = np.min(D,axis=1)

    avgWithinSS = sum(d)/X.shape[0]

    # Total with-in sum of square
    wss = sum(d**2)

    tss = sum(pdist(X)**2)/X.shape[0]

    bss = tss-wss

    wss_values += [wss]
    bss_values += [bss]
```

In [7]:
```
fig = plt.figure()
font = {'family' : 'sans', 'size'   : 12}
plt.rc('font', **font)
plt.plot(k_values, wss_values, 'bo-', color='red', label='WSS')
plt.plot(k_values, bss_values, 'bo-', color='blue', label='BSS')
plt.grid(True)
plt.xlabel('Number of clusters')
plt.ylabel('BSS & WSS')
plt.xticks(k_values)
plt.legend()
plt.title('Hierarchical Clustering');
```



The plot for complete linkage shows a clear knee/elbow for three clusters. Let's check it also using the inconsistency coefficient with a depth of 5 and the acceleration. The former has a higher value for two clusters whereas acceleration confirm the elbow found with BSS/WSS for three clusters.

Inconsistency Coefficient

In [8]:
```
inconsistency = inconsistent(Z, d=5)

for i in reversed(range(1,15)):
    print("from %d to %d => Inconsistency %.3f"%(i,i+1,inconsistency[-i][3]))
```

```
from 14 to 15 => Inconsistency 2.768
from 13 to 14 => Inconsistency 3.134
from 12 to 13 => Inconsistency 3.203
from 11 to 12 => Inconsistency 3.431
from 10 to 11 => Inconsistency 3.173
from 9 to 10 => Inconsistency 2.960
from 8 to 9 => Inconsistency 2.926
from 7 to 8 => Inconsistency 3.377
from 6 to 7 => Inconsistency 3.267
from 5 to 6 => Inconsistency 2.915
```

Hierarchical Clustering Part 2 - Complete Linkage and Characterization

In this second notebook, we apply hierarchical clustering on the same data but use complete linkage we discuss how we can characterize/describe the clusters we computed.

The notebook is an adaptation of

https://joernhees.de/blog/2015/08/26/scipy-hierarchical-clustering-and-dendrogram-tutorial/

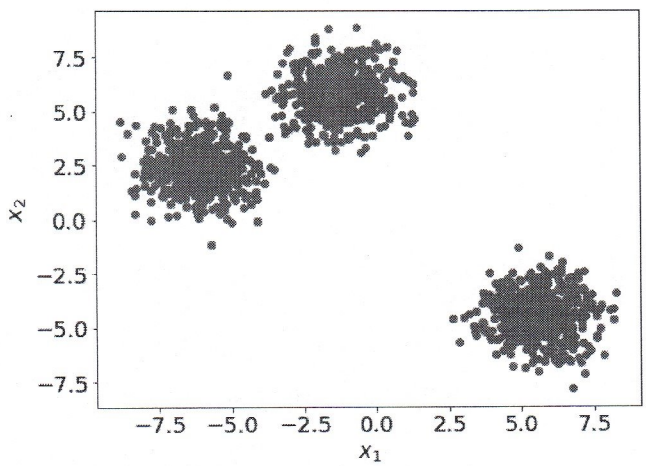
As before, we start by importing the required libraries and generate the same data we used in the previous notebook.

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

from sklearn.datasets import make_blobs

# we are using the scipy implementation
from scipy.spatial.distance import cdist, pdist
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster, inconsistent

%matplotlib inline

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

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=plt.get_cmap('Pastel1'));
plt.xlabel("$x_1$");
plt.ylabel("$x_2$");
# plt.title("Data Points");
```



First, we generate the dendrogram using complete linkage and then plot the full dendrogram.

In [4]:
```
Z = linkage(X, 'complete')
```

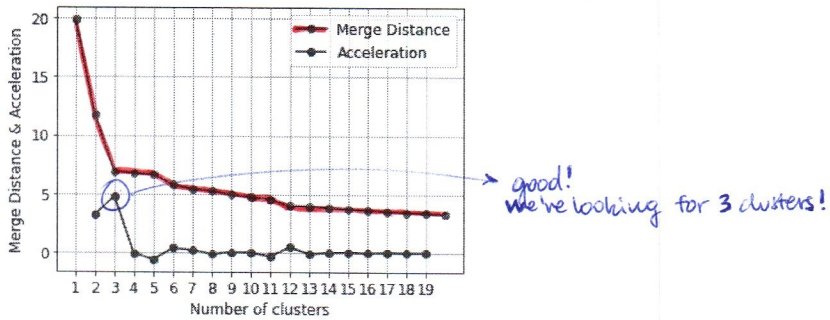
In [5]:
```
plt.figure(figsize=(25, 10))
font = {'family' : 'sans', 'size'   : 18}
plt.rc('font', **font)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Merge Distance')
dendrogram(Z,
    leaf_rotation=90.,  # rotates the x axis labels
    leaf_font_size=8.,  # font size for the x axis labels
)
plt.xticks([])
plt.show()
```

```
from 4 to 5 => Inconsistency 2.987
from 3 to 4 => Inconsistency 2.879
from 2 to 3 => Inconsistency 3.737
from 1 to 2 => Inconsistency 4.294
```

still not good

Acceleration

In [9]:
```
merge_distance = Z[-20:, 2]
reverse_merge_distance = merge_distance[::-1]
index = np.arange(1,len(merge_distance)+1)

acceleration = np.diff(merge_distance, 2)  # 2nd derivative of the distances
reversed_acceleration = acceleration[::-1]

fig = plt.figure()
font = {'family' : 'sans', 'size'   : 12}
plt.rc('font', **font)
plt.plot(index, reverse_merge_distance, 'bo-', color='red', label='Merge Distance')
plt.plot(index[:-2]+1, reversed_acceleration, 'bo-', color='blue', label='Acceleration')
plt.grid(True)
plt.xlabel('Number of clusters')
plt.ylabel('Merge Distance & Acceleration')
plt.xticks(k_values)
plt.legend()
# plt.title('Hierarchical Clustering');
plt.show();
```



good!
We're looking for 3 clusters!

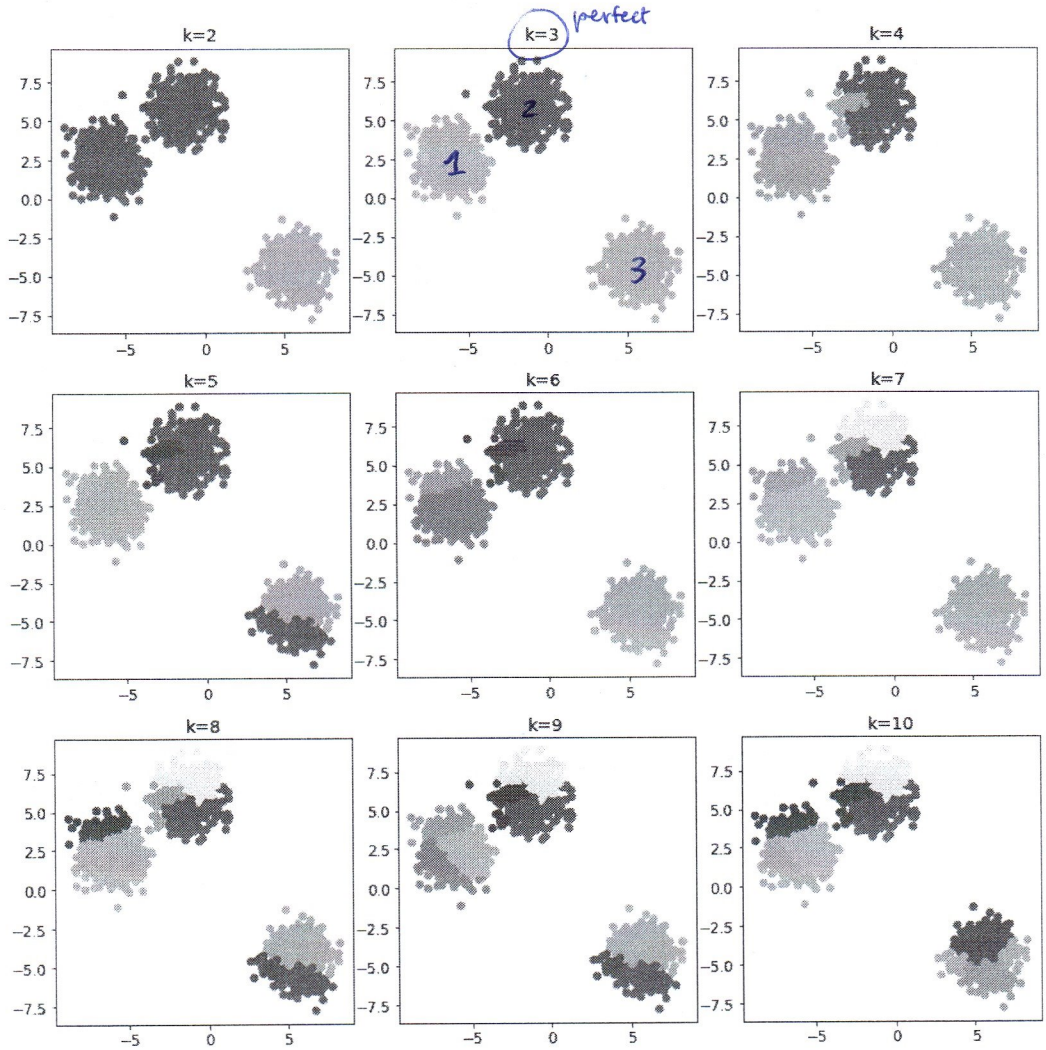
In [10]:
```
print("The number of cluster suggested using acceleration is %d"%(reversed_acceleration.argmax() + 2))
```

```
The number of cluster suggested using acceleration is 3
```

Visualization of Clustering Solutions

It is interesting to analyze the last nine clustering to analyze how the data points were partitioned.

In [11]:
```
plt.figure(figsize=(12, 12));

for k in range(2,11):
    yp = fcluster(Z, k, criterion='maxclust')

    plt.subplot(330+(k-1))
    plt.title('k='+str(k))
    plt.scatter(X[:, 0], X[:, 1], c=yp,cmap=plt.get_cmap('Paired'))

plt.tight_layout()
```



Describing the Clusters

Hierarchical clustering generates a hierarchy of possible clusterings. When we select k, we then generate the labels for all the data points. For instance, if we decide to use three clusters, our result will be a vector of 1500 labels. However, it would be difficult to understand the result of the clustering by looking at the raw labels and thus we need a way to describe those clusters in a succint way. Since the variables are numerical, for instance we can print for each cluster some summary statistics like mean, median, variance. So first, we create a data frame with the cluster information and then print some statistics by grouping the dataframe by the cluster label and then using the describe function.

In [12]:
```
cdf = pd.DataFrame(columns = ['x1','x2'], data = X)
cdf['cluster'] = fcluster(Z, 3, criterion='maxclust')
cdf.groupby(by=['cluster']).describe()
```

Out[12]:

	x1								x2							
	count	mean	std	min	25%	50%	75%	max	count	mean	std	min	25%	50%		
cluster																
1	500.0	5.614223	0.977901	2.654943	4.954818	5.640331	6.270446	8.279370	500.0	-4.482875	1.003710	-7.825452	-5.154613	-4.504309		
2	499.0	-6.114986	0.951013	-8.792615	-6.756957	-6.089769	-5.472167	-3.517612	499.0	2.401623	0.993789	-1.121341	1.784812	2.382643		
3	501.0	-1.254415	0.984865	-5.126344	-1.895958	-1.300397	-0.595947	1.332455	501.0	5.806578	1.001357	3.094381	5.060215	5.825887		

We note that the three clusters are characterized by rather distinct values of the variables x1 and x2 since the mean and the variance of the two variables is very different. They have the same standard deviation (they were created using the same standard deviation). So we can use these values to create a narrative that can characterize our result to an audience.

Hierarchical Clustering using the Weather Dataset

For more approaches to map categorical attributes to numerical ones, check out the tutorial at: https://pbpython.com/categorical-encoding.html

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

from sklearn.datasets import make_blobs

# we are using the scipy implementation
from scipy.spatial.distance import cdist, pdist
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster, inconsistent
from sklearn.preprocessing import LabelEncoder

%matplotlib inline

np.set_printoptions(precision=5, suppress=True)
```

In [80]:
```
def PlotDendrogram(Z, title=""):
    plt.figure(figsize=(25, 10))
    font = {'family' : 'sans', 'size'   : 18}
    plt.rc('font', **font)
    plt.title('Hierarchical Clustering Dendrogram')
    plt.xlabel('Sample Index')
    plt.ylabel('Merge Distance')
    dendrogram(Z,
        leaf_rotation=90.,  # rotates the x axis labels
        leaf_font_size=8.,  # font size for the x axis labels
    )
    plt.xticks([])
    plt.show()
```

In [109...
```
def ComputeWSSBSS(X,Z,k_values=range(1,20)):
    wss_values = []
    bss_values = []

    for k in k_values:
        clustering = fcluster(Z, k, criterion='maxclust')
        frequency = np.bincount(clustering)
        index = np.nonzero(frequency)[0]

        centroids = [np.mean(X[clustering==c],axis=0) for c in index]

        cdist(X, centroids, 'euclidean')
        D = cdist(X, centroids, 'euclidean')
        cIdx = np.argmin(D,axis=1)
        d = np.min(D,axis=1)

        avgWithinSS = sum(d)/len(X)

        # Total with-in sum of square
        wss = sum(d**2)

        tss = sum(pdist(X)**2)/len(X)

        bss = tss-wss

        wss_values += [wss]
        bss_values += [bss]
    return wss_values,bss_values
```

In [114...
```
def PlotKneeElbow(bss_values,wss_values,k_values,title=""):
    fig = plt.figure()
    font = {'family' : 'sans', 'size'   : 12}
    plt.rc('font', **font)
    plt.plot(k_values, wss_values, 'bo-', color='red', label='WSS')
    plt.plot(k_values, bss_values, 'bo-', color='blue', label='BSS')
    plt.grid(True)
    plt.xlabel('Number of clusters')
    plt.ylabel('BSS & WSS')
    plt.xticks(k_values)
    plt.legend()
    plt.title(title);
```

In [2]:
```
df = pd.read_csv("weather.csv")
df.describe()
```

Out[2]:

	Outlook	Temperature	Humidity	Windy	Play
count	14	14	14	14	14
unique	3	3	2	2	2
top	sunny	mild	high	False	yes
freq	5	6	7	8	9

In [3]:
```
target = "Play"
input_variables = df.columns[df.columns!=target]
```

we're performing clustering, we don't want the target variable to be considered (we eliminate it)

One Hot Encoding

We apply one-hot-encoding to the only true categorical variable. Note that, also Windy and Humidity might be considered categorical but since they have two values we just apply label encoder and maps the values in 0 and 1.

In [4]:
```
outlook_ohe = pd.get_dummies(df[['Outlook']])
df_numerical = outlook_ohe.copy()
```

we apply one hot encoding when we don't want to assign numerical values to categorical features with no sense. However, for example education high, medium or low can be associated to a numerical encoding (label encoder, ~~one hot~~).

"Windy" and "Humidity" can be considered as education

In [6]:
```
encoders = {}
for attribute in ['Humidity','Windy']:
    encoders[attribute] = LabelEncoder().fit(df[attribute])
    df_numerical[attribute] = encoders[attribute].transform(df[attribute])
```

In [7]:
```
df['Temperature'].value_counts()
```

Out[7]:
```
mild    6
hot     4
cool    4
Name: Temperature, dtype: int64
```

In [13]:
```
temperature_mapping = {'cool':0, 'mild':0.5, 'hot':1.0}
```

In [14]:
```
df_numerical['Temperature'] = df['Temperature'].replace(mapping)
```

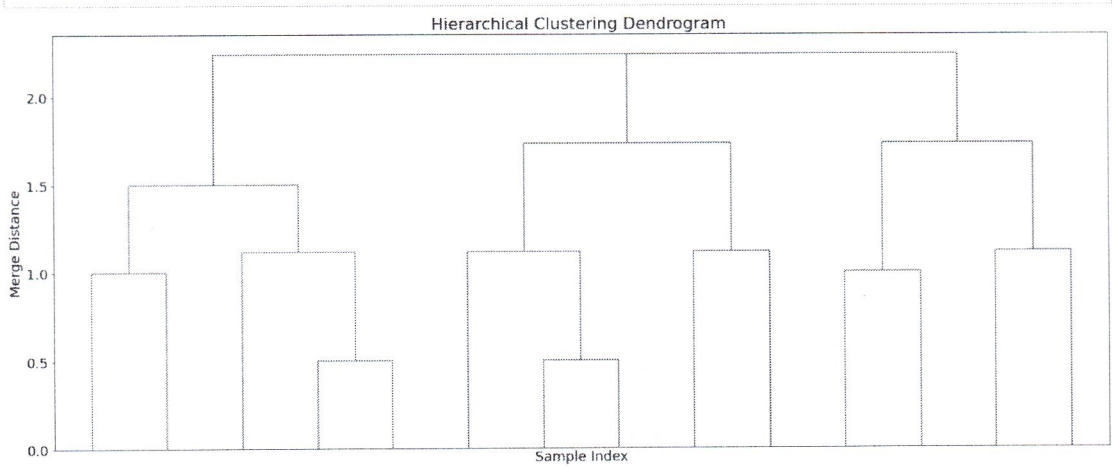
In [15]:
```
df_numerical
```

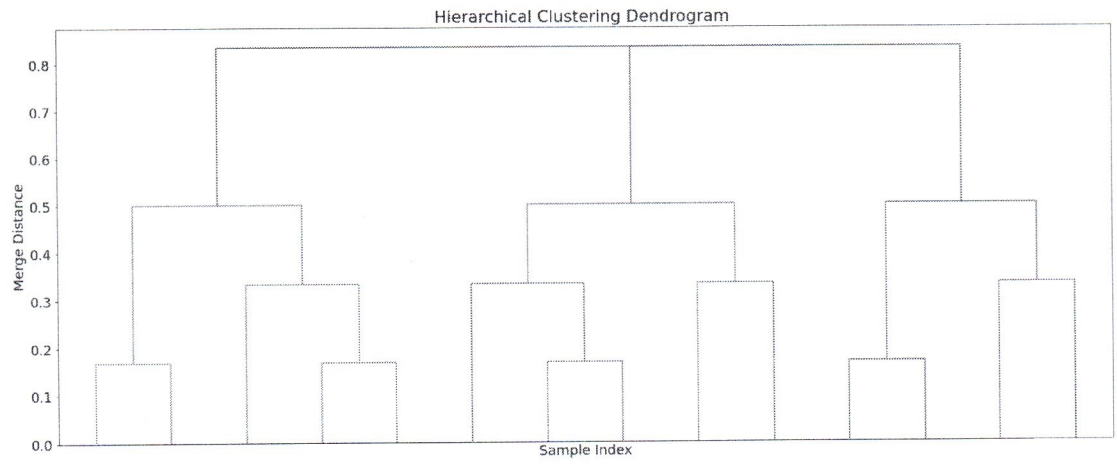
Out[15]:

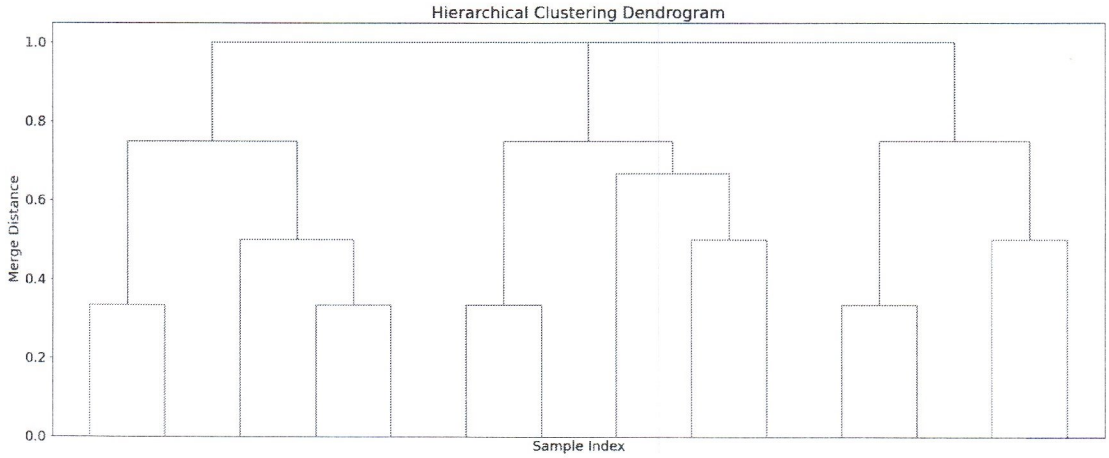
	Outlook_overcast	Outlook_rainy	Outlook_sunny	Humidity	Windy	Temperature
0	0	0	1	0	0	1.0
1	0	0	1	0	1	1.0
2	1	0	0	0	0	1.0
3	0	1	0	0	0	0.5
4	0	1	0	1	0	0.0
5	0	1	0	1	1	0.0
6	1	0	0	1	1	0.0
7	0	0	1	0	0	0.5
8	0	0	1	1	0	0.0
9	0	1	0	1	0	0.5
10	0	0	1	1	1	0.5
11	1	0	0	0	1	0.5
12	1	0	0	1	0	1.0
13	0	1	0	0	1	0.5

Hierarchical Clustering

In [81]:
```
Z = {}
Z['Euclidean'] = linkage(df_numerical, 'complete')
Z['Hamming'] = linkage(df_numerical, 'complete',metric='hamming')
Z['Jaccard'] = linkage(df_numerical, 'complete',metric='jaccard')

for key in Z.keys():
    PlotDendrogram(Z[key],key+' Distance')
```







Note that there are no cuts for two clusters so if we ask to produce the clustering for two clusters we will get only one cluster. This applies at other cut points so we can compute WSS and BSS only for some values of k (the missing ones are equivalent to the previous value).

In [104]:
```
for k in range(1,10):
    print(str(fcluster(Z['Euclidean'], k , criterion='maxclust')))
```

```
1 [1 1 1 1 1 1 1 1 1 1 1 1 1 1]
2 [1 1 1 1 1 1 1 1 1 1 1 1 1 1]
3 [2 2 3 1 1 1 3 2 2 1 2 3 3 1]
4 [2 2 3 1 1 1 3 2 2 1 2 3 3 1]
5 [2 2 4 1 1 1 5 2 3 1 3 5 4 1]
6 [3 3 5 1 2 2 6 3 4 2 4 6 5 1]
7 [3 3 5 1 2 2 6 3 4 2 4 6 5 1]
8 [3 3 5 1 2 2 6 3 4 2 4 6 5 1]
9 [3 3 5 1 2 2 6 3 4 2 4 6 5 1]
```

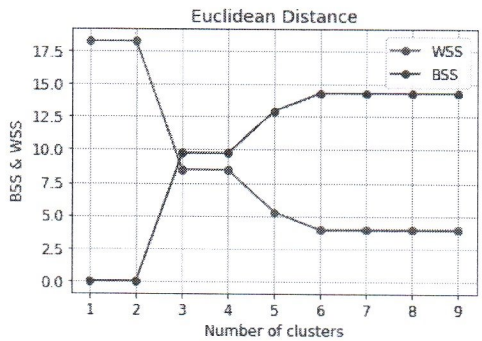
labels in the case k=1

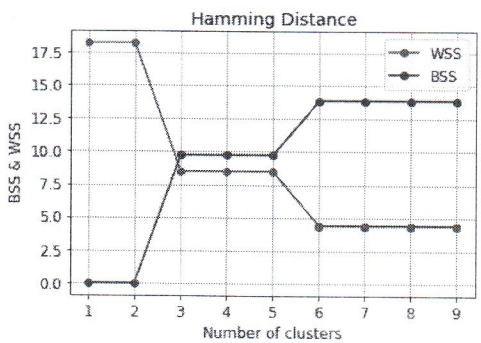
labels in the case k=2: so bad! there's not even a single 2

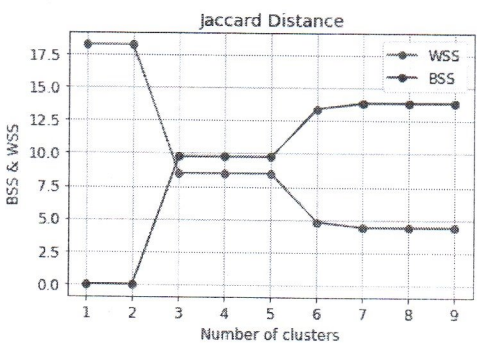
so bad! not even a 4

In [116]:
```
wss = {}
bss = {}

for key in Z.keys():
    wss[key],bss[key] = ComputeWSSBSS(df_numerical,Z[key],range(1,10))
```

In [118]:
```
for key in Z.keys():
    PlotKneeElbow(bss[key],wss[key],range(1,10),key+' Distance')
```







In [ ]:

Example of Clustering Validation using Internal and External Measures

In this example, we apply hierarchical clustering to the input variables of the Iris dataset. Then, we evaluate the solution using internal measures (by computing WSS and BSS) and external measures (by comparing the labels assigned by the clustering with the known labels).

We begin by importing the needed libraries.

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 scipy.spatial.distance import cdist, pdist

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

We load the Iris data set and print some statistics.

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[1])
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: 4
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)]
```

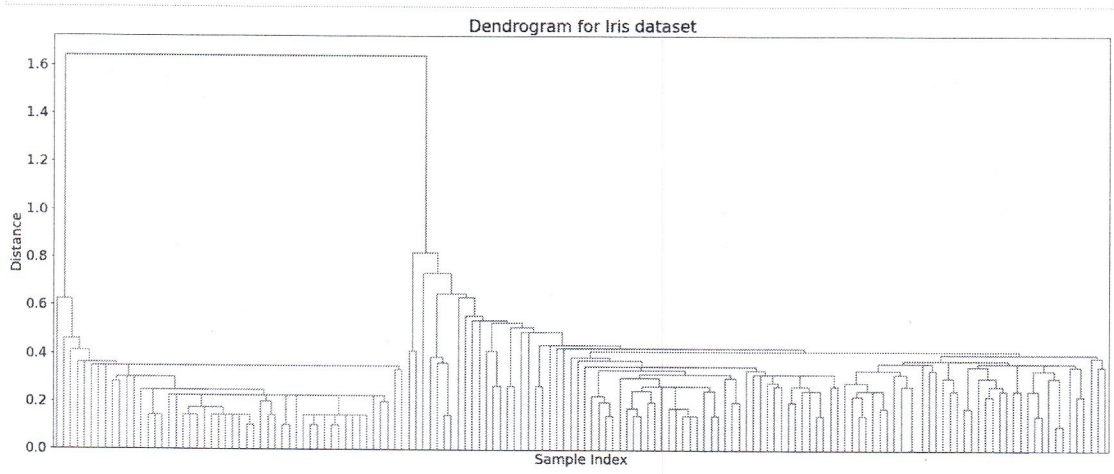
We apply hierarchical clustering using single linkage. Given that there are 150 cases, we have 149 merges.

In [3]:
```
merges = linkage(iris.data, method = 'single')
print('there have been %d merges'%(merges.shape[0]))
```

```
there have been 149 merges
```

Let's plot the dendrogram which shows two major clusters: a smaller one (in green) and a larger one.

In [4]:
```
plt.figure(figsize=(25, 10))
font = {'family' : 'sans', 'size'   : 18}
plt.rc('font', **font)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('sample index')
plt.ylabel('distance')
dendrogram(merges,
    leaf_rotation=90.,  # rotates the x axis labels
    leaf_font_size=8.,  # font size for the x axis labels
)
plt.title('Dendrogram for Iris dataset')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.xticks([])
plt.show()
```



Internal Measures

internal because we're using the characteristics of the clusters we built

We evaluate the possible clustering by analyzing the withing sum of squares (WSS) and the between sum of squares of the different clustering solutions.

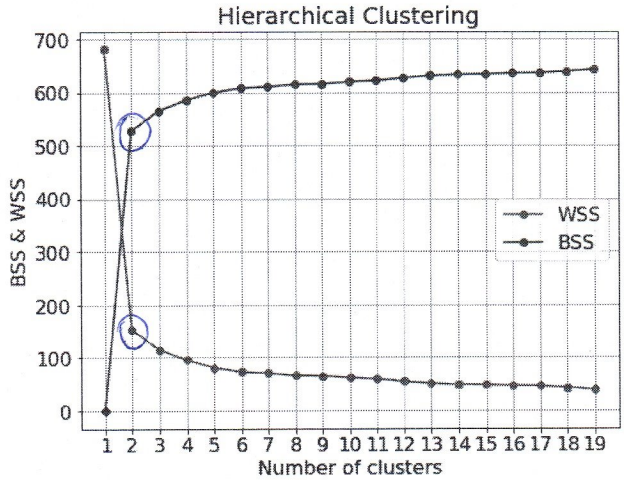
In [5]:
```
def ComputeInternalMeasures(x, merges, k_values):
    wss_values = {}
    bss_values = {}

    for k in k_values:
        clustering = fcluster(merges, k, criterion='maxclust')
        centroids = [np.mean(x[clustering==c],axis=0) for c in range(1,k+1)]
        cdist(x, centroids, 'euclidean')
        D = cdist(x, centroids, 'euclidean')
        cIdx = np.argmin(D,axis=1)
        d = np.min(D,axis=1)

        avgWithinSS = sum(d)/x.shape[0]
```

```
        # Total with-in sum of square
        wss = sum(d**2)

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

        bss = tss-wss

        wss_values[k] = wss
        bss_values[k] = bss
    return wss_values,bss_values
```

In [6]:
```
k_values = range(1,20)
wss_dict, bss_dict = ComputeInternalMeasures(iris.data, merges, k_values)
```

In [7]:
```
wss_values = [wss_dict[x] for x in range(1,20)]
bss_values = [bss_dict[x] for x in range(1,20)]
```

In [8]:
```
fig = plt.figure(figsize=(8,6))
font = {'family' : 'sans', 'size'   : 16}
plt.rc('font', **font)
plt.plot(k_values, wss_values, 'bo-', color='red', label='WSS')
plt.plot(k_values, bss_values, 'bo-', color='blue', label='BSS')
plt.grid(True)
plt.xlabel('Number of clusters')
plt.ylabel('BSS & WSS')
plt.xticks(k_values)
plt.legend()
plt.title('Hierarchical Clustering');
```



There is a clear elbow around k=2 but we cannot just select two clusters. We should use the elbow as a guide but test multiple solutions around that value.

External Meaures

This time we have the targets! We can compare targets and clusters

Let's evaluate the solution with three clusters and compute a table that report the distribution of original labels in each one of the three clusters so that we can compare the clustering solution against the known labels.

In [9]:
```
labels_single_k3 = fcluster(merges,3,criterion='maxclust')
# Create a DataFrame with labels and varieties as columns: df
df = pd.DataFrame({'labels': labels_single_k3, 'varieties': iris.target_names[target]})

# Create crosstab: ct
ct = pd.crosstab(df['labels'],df['varieties'])

# Print the table
print(ct)
```

targets

varieties	setosa	versicolor	virginica
labels			
1	50	0	0
2	0	0	2
3	0	50	48

our clusters (k=3)

cluster 1 contains only "setosa" elements (good!)

this is not a good behavior!

the labels "2" and "3" separated NOTHING!

In [10]:
```
labels_single_k4 = fcluster(merges,4,criterion='maxclust')
# Create a DataFrame with labels and varieties as columns: df
df = pd.DataFrame({'labels': labels_single_k4, 'varieties': iris.target_names[target]})

# Create crosstab: ct
ct = pd.crosstab(df['labels'],df['varieties'])

# Print the table
print(ct)
```

varieties	setosa	versicolor	virginica
labels			
1	50	0	0
2	0	0	2
3	0	50	47
4	0	0	1

The additional cluster? Useless! Let's change linkage!

Note that cluster 1 perfectly identifies all the examples labeled as "setosa", Cluster 2 contains just two examples (labeled "virginica"), while Cluster 3 contains all the remaining cases.

Complete Linkage

We repeat the analysis using complete linkage.

In [11]:
```
merges = linkage(iris.data, method = 'complete')
labels = fcluster(merges,3,criterion='maxclust')
# Create a DataFrame with labels and varieties as columns: df
df = pd.DataFrame({'labels': labels, 'varieties': iris.target_names[target]})

# Create crosstab: ct
ct = pd.crosstab(df['labels'],df['varieties'])

# Print the table
print(ct)
```

```
varieties  setosa  versicolor  virginica
labels
1               0          23         49
2               0          27          1
3              50           0          0
```

Still not so good! (however it's better than with single linkage)

perfect!

Average Linkage

We repeat the analysis using average linkage.

In [12]:
```
# Repeats the analysis using average linkage

merges = linkage(iris.data, method = 'average')
labels = fcluster(merges,3,criterion='maxclust')
# Create a DataFrame with labels and varieties as columns: df
df = pd.DataFrame({'labels': labels, 'varieties': iris.target_names[target]})

# Create crosstab: ct
ct = pd.crosstab(df['labels'],df['varieties'])

# Print the table
print(ct)
```

```
varieties  setosa  versicolor  virginica
labels
1              50           0          0
2               0           0         36
3               0          50         14
```

perfect!

still not perfect but always better than before!

How Can We Represent Clusters?

Hierarchical clustering produces as a result a hierarchy of partitions. Using either an internal or an external measure we can select the most appropriate number of clusters. Finally, we can assign the cluster labels to the examples.

But how can we present the clusters to our customers?

Hierarchical clustering produces a single column of cluster labels. But we cannot present just the labels we need to find a way to summarize the result. We need to build "cluster personas".

Suppose that we use the last clustering obtained using **single linkage** with three and four clusters. We can generate a summary of this solution by print the average values of all the features for each cluster.

In [13]:
```
df_iris = pd.DataFrame(iris.data,columns=iris.feature_names)
df_iris['cluster']=labels_single_k3
df_iris.groupby('cluster').describe()
```

Out[13]:

	sepal length (cm)								sepal width (cm)		...	petal length (cm)		petal width					
	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%	max	count	mean	std	min	25%	50%	75%
cluster																				
1	50.0	5.006000	0.352490	4.3	4.80	5.00	5.20	5.8	50.0	3.428000	...	1.575	1.9	50.0	0.246000	0.105386	0.1	0.20	0.2	0.30
2	2.0	7.800000	0.141421	7.7	7.75	7.80	7.85	7.9	2.0	3.800000	...	6.625	6.7	2.0	2.100000	0.141421	2.0	2.05	2.1	2.15
3	98.0	6.230612	0.631217	4.9	5.80	6.25	6.70	7.7	98.0	2.853061	...	5.500	6.9	98.0	1.667347	0.424452	1.0	1.30	1.6	2.00

3 rows × 32 columns

Given the statistics, we need to build a narrative that can characterize the clustering solution we selected. Such narrative is typically created using "cluster personas", profiles that describe the main features of the clusters.

Snake Plots

These are market research techniques used to compare different segments. They provide a visual representation of each segment attributes. First, the data are normalized (according to the mean and the standard deviation) then we plot each cluster's average normalized values of each attribute.

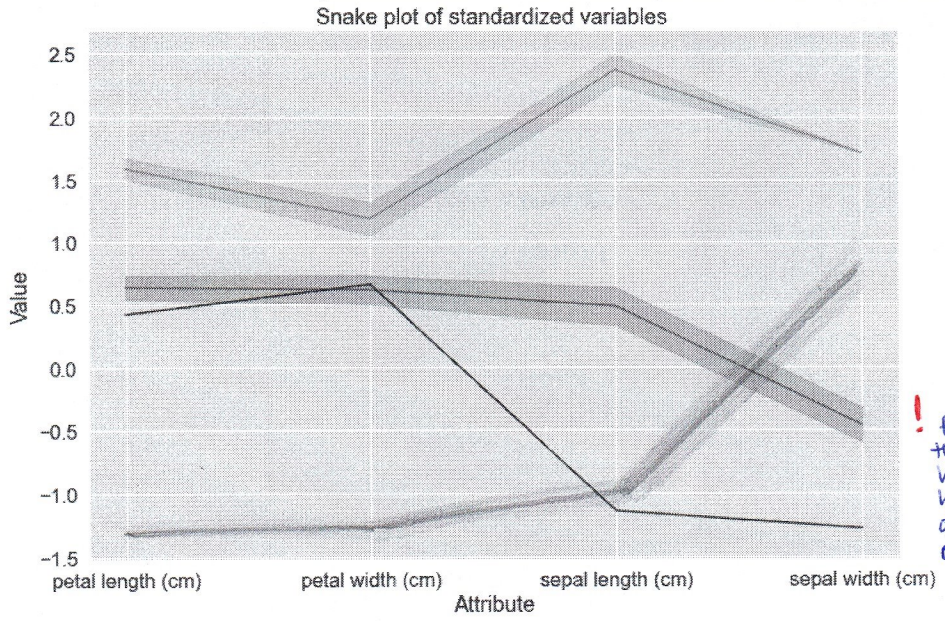
In [14]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

normalized_iris = scaler.fit_transform(iris.data)
df_iris_normalized = pd.DataFrame(normalized_iris,columns=iris.feature_names)
df_iris_normalized['cluster'] = labels_single_k4
```

In [15]:
```
df_iris_melt = pd.melt(df_iris_normalized,
                       id_vars=['cluster'],
                       value_vars=iris.feature_names,
                       var_name='Attribute',
                       value_name='Value')
```

In [16]:
```
import seaborn as sns
```

In [18]:
```
plt.title('Snake plot of standardized variables')
sns.set(rc={'figure.figsize':(12,8)})
sns.set(font_scale = 1.5)
```

```
sns.lineplot(x="Attribute", y="Value", hue='cluster', data=df_iris_melt)
plt.legend(loc='center right', bbox_to_anchor=(1.25, 0.5), ncol=1);
```



! From the surrounding of the lines we can understand how variates a cluster (or how many elements a cluster has (e.g. probably the cluster 4 has really few elem.))

In [ ]: