Data Analysis - Lab 4

M. Sébastien MASCHA & M. Sauvage Pierre

ISEP Paris – November 5th, 2019

Import of libraries

This document has been done using python on Jupyter Notebook with the librairies:

- maths for sqrt, pi, exp
- Numpy to manipulate arrays
- pandas to import csv
- matplotlib to plot graphics
- seaborn to make your charts prettier (built on top of Matplotlib)
- sklearn : tools for data mining and data analysis
- mlxtend : tools for ploting PCA

In [94]:

```
# coding: utf-8

import data

from math import sqrt,pi,exp
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns; sns.set()

import sklearn
# Normalize data
from sklearn.preprocessing import StandardScaler
# Dimension reduction
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as QDA
# Useful
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans
```

Exercice A - Analysing Fiher's Iris with the K-Means algorithm

In this exercice, we study one of the most famous dataset : Fisher's Iris , in which different characteristics of iris flower are studied.

Question 1 - Open the file

We use the comma separator because we saw in the text editor that the data was separated by commas.

In [95]:

```
iris_df = pd.read_csv("data/iris.csv", sep =';')


iris_df.head()
```

Out[95]:

	SepalLength	SepalWidth	PetalLength	PetalWidth	Class
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

As we can see from the command dataframe.head() & dataframe.shape(), our dataset contains attributes on 150 flowers : their sepals length and width and the same measure for their petals. Each flower is assign a class

Question 2 - Remove these labels from the main set and store them in another vector.

In [96]:

```
iris_class = iris_df['Class']
iris_dfcopy = iris_df.copy
iris_df = iris_df.drop(iris_df.columns[[4]], axis=1)
iris_df.head()
```

Out[96]:

	SepalLength	SepalWidth	PetalLength	PetalWidth
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Question 3 - Use the command PCA(·) from sklearn.decomposition to do a Principal Component Analysis on your data. Then use the following lines to retrieve the dataset projected on the two principal components:

In [181]:

```
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(scaled_df_features)

principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])
principalDf_iris_wth_class = pd.concat([principalDf, iris_class ],axis = 1)
```

Question 4 Use the K-Means algorithm (library sklearn.cluster) on your data df_iris_PCA to obtain a partition with 3 clusters and visualize your results

In [182]:

```
kmeans = KMeans(n_clusters = 3, n_init = 5, max_iter=300,random_state=232).fit
(principalDf)
kmeans.score(principalDf)
prediction = kmeans.predict(principalDf)
print(prediction)
print(kmeans.score(principalDf))
```

```
[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 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 0 0 0 2 0 0 0 0 0 0 0 2 0 0 0 0
2 0 0 0
 0 2 2 2 0 0 0 0 0 0 0 2 2 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0
2 0 2 2
 0 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 2 2 2 2 2 2 0 0 2 2 2 0 2 2 2 0 2
2 2 0 2
 2 0]
-116.24247259315806
```

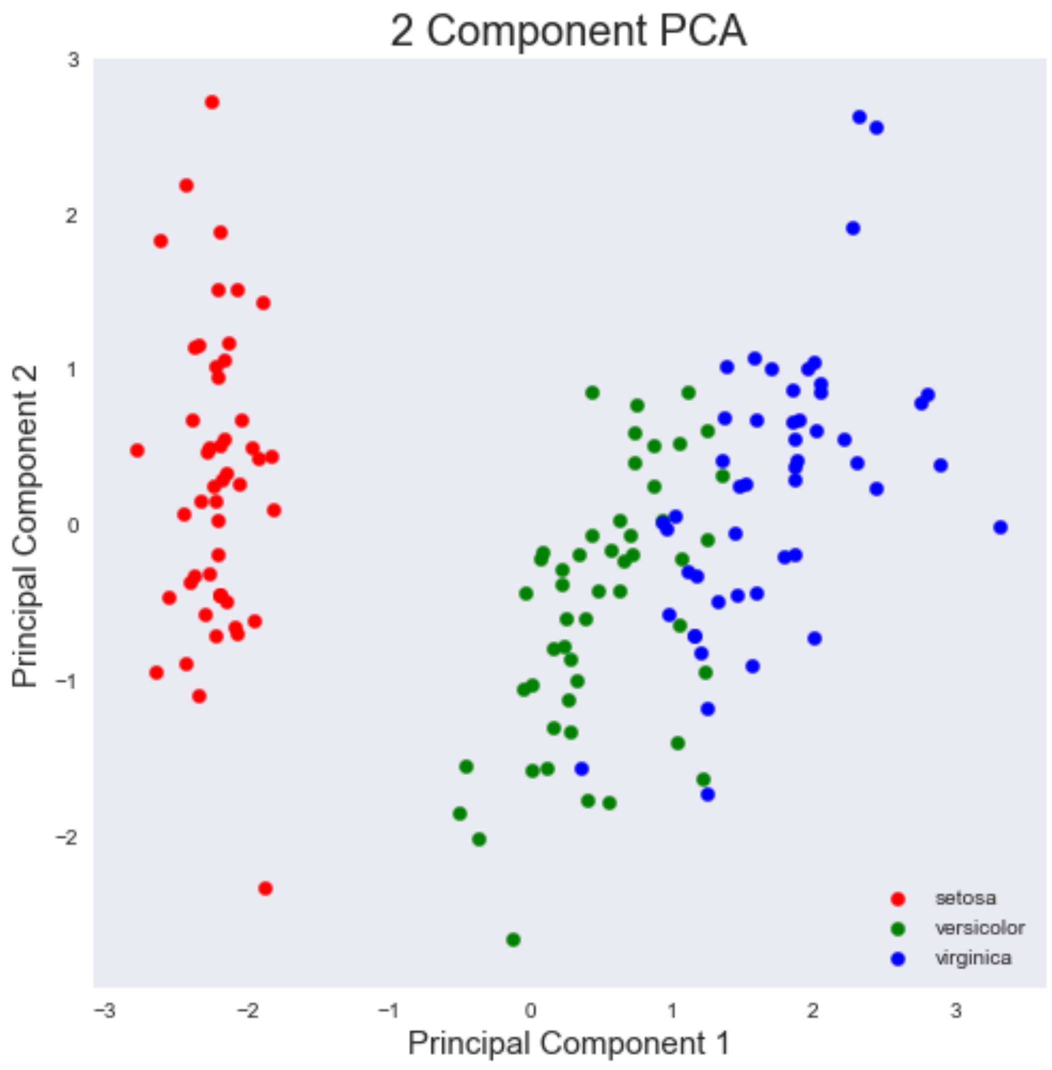
We use the code of lab 3 exercize A :

In [162]:

```
def associateColorToClass(iris_class):
    switcher = {
                    'setosa': "r",
                    'virginica': "b",
                    'versicolor': "g"
                }

    return switcher.get(iris_class)

principalDf_iris_wth_class['Color'] = principalDf_iris_wth_class.apply(lambda
x: associateColorToClass(x['Class']), axis=1)
```

In [163]:

```
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 Component PCA', fontsize = 20)

targets = principalDf_iris_wth_class.Class.unique()
colors = principalDf_iris_wth_class.Color.unique()
for target, color in zip(targets,colors):
    indicesToKeep = principalDf_iris_wth_class['Class'] == target
    ax.scatter(principalDf_iris_wth_class.loc[indicesToKeep, 'principal compon
ent 1']
               , principalDf_iris_wth_class.loc[indicesToKeep, 'principal comp
onent 2']
               , c = color
               , s = 40)
ax.legend(targets)
ax.grid()
```



Question 5- Repeat question 4) several times. What happens ? Comment.

We can see that each time we repeat the lines of code from the question 4, the values in the array prediction change : The K-Mean clustering prediction varies each times assigning a value to a cluster (0,1,2) and assigning this values to each flower the prediction of belonging to one cluster (and the value of kmeans.score(iris_df) also changes by the hundreth).

Question 6- Project the labels that you stored in a separate vector in question 2). Compare these results with the partitions from your K-Means experiments. Comment.

In [190]:

```
prediction_series = pd.Series(prediction)
prediction_str=np.empty((len(prediction))).astype(str)
prediction_str[prediction== 1]='setosa'
prediction_str[prediction== 0]='versicolor'
prediction_str[prediction== 2]='virginica'
prediction_series = pd.Series(prediction_str)
compare = pd.concat([iris_class,prediction_series], axis =1)
compare['equal']=(compare['Class']==compare[0])
compare['equal'].value_counts()
```

Out[190]:

```
True     123
False     27
Name: equal, dtype: int64
```

We can see that all the prediction are not valid, as some times virginica class has the corresponding value of 2 and less often the value of 1. We have 27 prediction that are false out of 123 representing 21,95% of prediction. It can be explain by the fact that two classes : versicolor and virginica are colliding when ploting the PCA visualization. it results that some flowers are consider versicolor instead of virginica and vice versa

Question 7 - Display the confusion matrix between your results with the theoretical labels.

In [166]:

```
prediction_str=np.empty((len(prediction))).astype(str)
prediction_str[prediction== 1]='setosa'
prediction_str[prediction== 0]='versicolor'
prediction_str[prediction== 2]='virginica'
#conf_pred = confusion_matrix(np.array(iris_class.values, dtype=int),np.array(
prediction, dtype=int))
conf_pred = confusion_matrix(np.array(iris_class.values, dtype=str),np.array(p
rediction_str))
np.unique(np.array(iris_class.values, dtype=str))
conf_pred
```

Out[166]:

```
array([[50,  0,  0],
       [ 0, 39, 11],
       [ 0, 14, 36]])
```

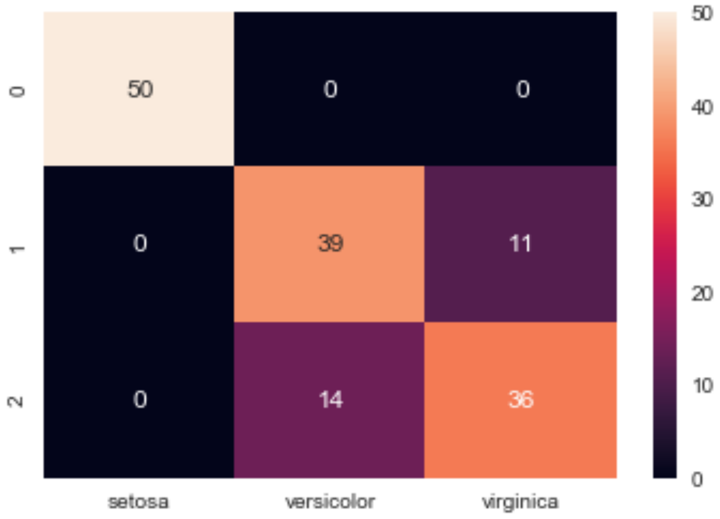
Before computing the confusion matrix, we have to change the values of prediction by assigning them

We have the confusion matrix between the original class and the prediction

```
In [167]:
```

```
conf_pred_df = pd.DataFrame(conf_pred, columns = ["setosa",'versicolor',"virgi
nica"])
sns.heatmap(conf_pred_df, annot=True)
```

```
Out[167]:

<matplotlib.axes._subplots.AxesSubplot at 0x10e0e39e8>
```



We try to find the best confusion matrix by changing the random state in the function KMeans(), and assigning the right number (0,1,2) to the corresponding , by analysis the PCA visualization and the prediction array

Question 8 - Choose a solution that seems good enough for you and compute the silhouette index (command silhouette_score() of sklearn.metrics). Comment.

```
In [168]:
```

```
score = silhouette_score(principalDf, prediction, metric='euclidean')
score
```

```
Out[168]:

0.5081546339516393
```

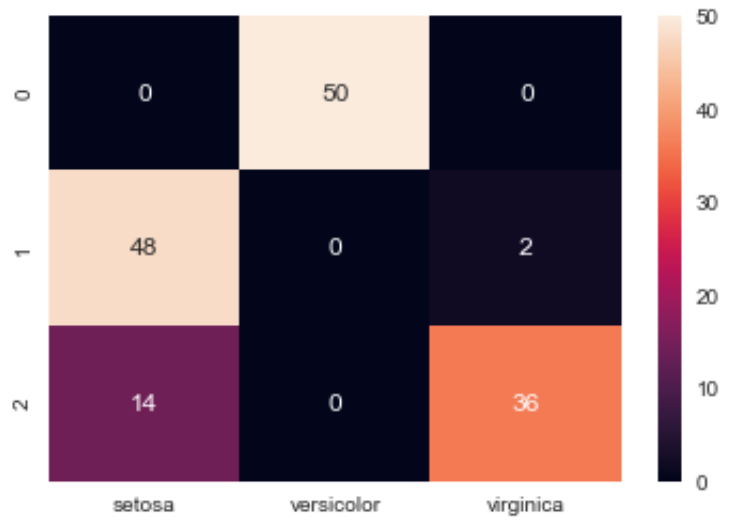
We would have an higher silhouette score we computed the KMeans with a number of cluster equals to 2 : as said before two classes are really similare , as seen on the PCA visualization, and could be consider as 1 cluster only. It would result in a higher silhouette score.

Question 9 - Start again questions 4) to 8) using the original data (with 4 variables) instead of the projected ones.

In [155]:

```
kmeans = KMeans(n_clusters = 3, n_init = 5, max_iter=300, random_state= 1).fit
(iris_df)
kmeans.score(iris_df)
prediction = kmeans.predict(iris_df)
print(prediction)
print(kmeans.score(iris_df))
prediction_str=np.empty((len(prediction))).astype(str)
prediction_str[prediction== 0]='setosa'
prediction_str[prediction== 1]='versicolor'
prediction_str[prediction== 2]='virginica'
#conf_pred = confusion_matrix(np.array(iris_class.values, dtype=int),np.array(
prediction, dtype=int))
conf_pred = confusion_matrix(np.array(iris_class.values, dtype=str),np.array(p
rediction_str))
np.unique(np.array(iris_class.values, dtype=str))
conf_pred_df = pd.DataFrame(conf_pred, columns = ["setosa",'versicolor',"virgi
nica"])
sns.heatmap(conf_pred_df, annot=True)
plt.show()
score = silhouette_score(iris_df, prediction, metric='euclidean')
print(score)
```

```
[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
1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 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 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0
2 2 2 2
 2 2 0 0 2 2 2 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]
-78.94084142614565
```



```
0.5525919445499757
```

We have a silhouette score closer to one (by approximatively 0.05) that we can explain by the use of more variables to categorize the data instead of the two principals components, meaning our model would be more precise. But we have to be careful if we use the original ones instead of the projected datas as sometimes some caracteristics doesn't have an importance in the classification of the data, and the model would be impacted as such in resulting in a lower silhouette score

Data Analysis - Lab 4

M. Sébastien MASCHA & M. Sauvage Pierre

ISEP Paris – September 24th, 2019

Exercice B - Hierarchical Clustering

This exercise is a tutorial on how to use scipy's hierarchical clustering.

Question 1 - Import of libraries

This document has been done using python on Jupyter Notebook with the librairies:

- maths for sqrt, pi, exp
- Numpy to manipulate arrays
- pandas to import csv
- matplotlib to plot graphics
- seaborn to make your charts prettier (built on top of Matplotlib)
- sklearn : tools for data mining and data analysis
- SciPy : a Python-based ecosystem of open-source software for mathematics, science, and engineering.

In [158]:

```
# coding: utf-8

import data

from math import sqrt,pi,exp
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns; sns.set()

import sklearn
# Normalize data
from sklearn.preprocessing import StandardScaler
# Dimension reduction
from sklearn.decomposition import PCA
# Useful
import scipy
from scipy.cluster.hierarchy import dendrogram , linkage , fcluster
from scipy.cluster.hierarchy import cophenet , inconsistent , maxRstat
from scipy.spatial.distance import pdist
```

Question 2 - Generate a sample of random data

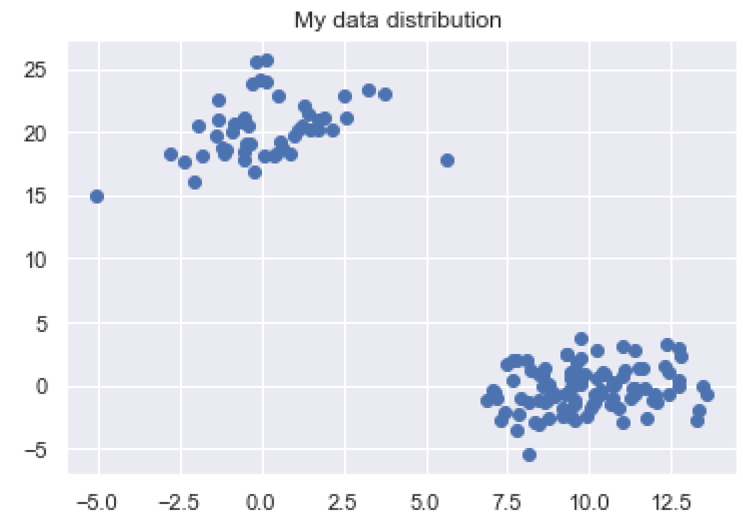
We will use numpy's multivariate_normal from random package

It draws random samples from a multivariate normal distribution.

The multivariate normal, multinormal or Gaussian distribution is a generalization of the one-dimensional normal distribution to higher dimensions. Such a distribution is specified by its mean and covariance matrix. These parameters are analogous to the mean (average or “center”) and variance (standard deviation, or “width,” squared) of the one-dimensional normal distribution.

In [159]:

```
np.random.seed(42) # for repeatability
a = np.random. multivariate_normal([10 , 0] , [[3 , 1] , [1 , 4]] , size =[100
,])
b = np.random.multivariate_normal([0, 20], [[3, 1], [1, 4]], size =[50 ,])
X=np.concatenate((a, b),)
plt.scatter(X[:,0], X[:,1])
plt.title ( 'My data distribution ')
plt.show()
```

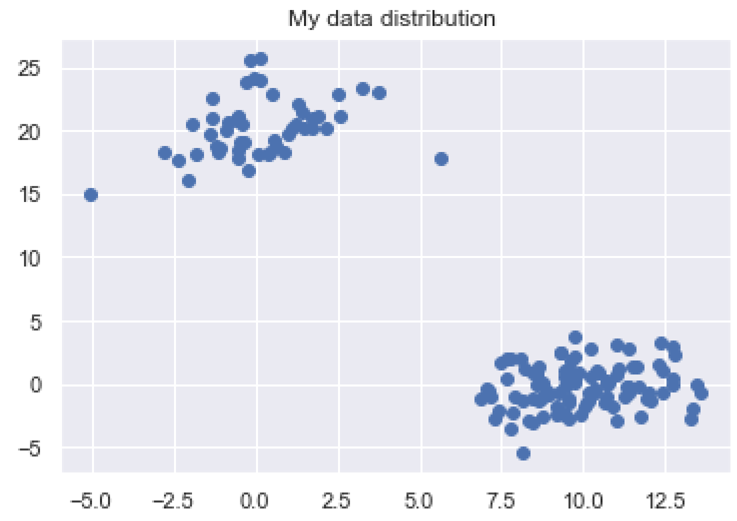


In [160]:

```
np.random.seed(42) # for repeatability
a = np.random. multivariate_normal([10 , 0] , [[3 , 1] , [1 , 4]] , size =[100
,])
b = np.random.multivariate_normal([0, 20], [[3, 1], [1, 4]], size =[50 ,])
X = np.concatenate((a, b),)
plt.scatter(X[:,0], X[:,1])
plt.title ( 'My data distribution ')
```

Out[160]:

```
Text(0.5, 1.0, 'My data distribution ')
```



Question 3 - Generate the linkage matrix and visualize the dendrogram

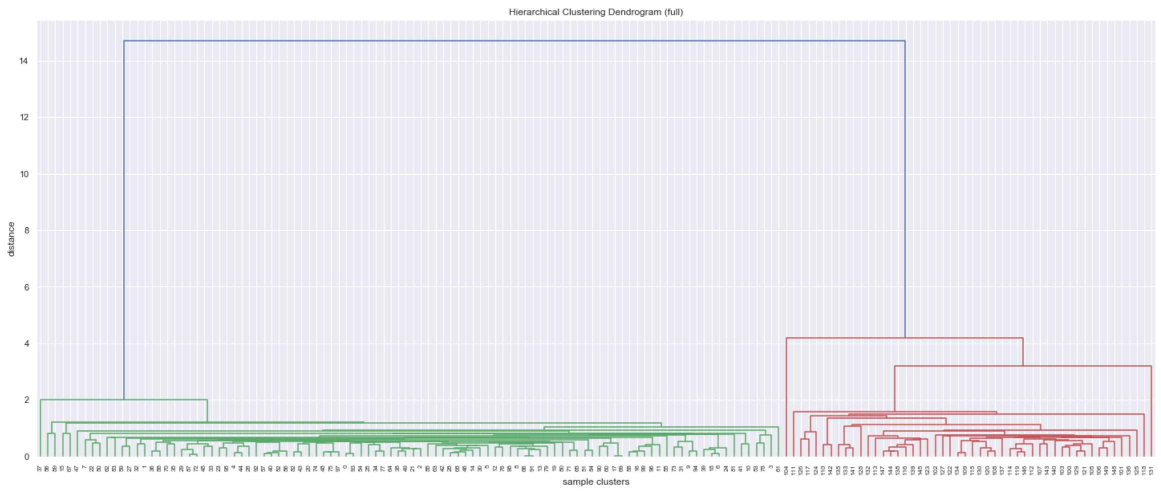
Testing different type of linkage

I.e. : 'single', 'complete', 'average', 'euclidean', 'cityblock', 'hamming',' cosine'

In [161]:

```
# alternative linkage methods: 'single ', 'complete ', 'average ', 'euclidean '
 ( default ) , ' cityblock ' aka Manhattan 'hamming ',' cosine'

Z = linkage(X, 'single' , optimal_ordering=True) # euclidean (default) would hav
e the same result

c, coph_dists = cophenet(Z, pdist(X))
print( 'Cophenetic Correlation : %1.2f ' % c)

plt.figure(figsize=(25, 10))
plt.title('Hierarchical Clustering Dendrogram (full)')
plt.xlabel('sample clusters')
plt.ylabel('distance')
dn = dendrogram( Z, leaf_rotation=90., leaf_font_size=8.,)
plt.show()
```

```
Cophenetic Correlation : 0.98
```

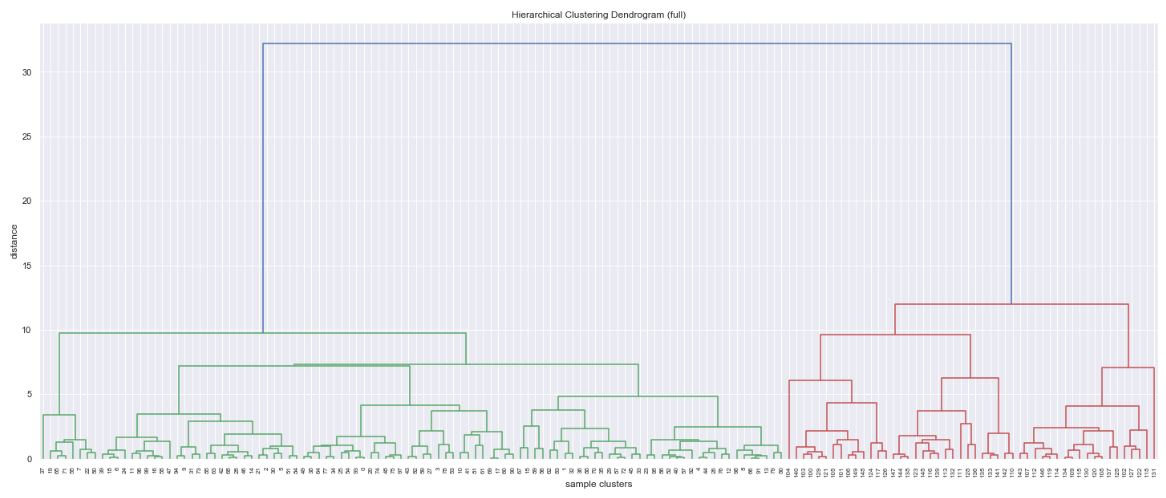


Euclidean-linkage (default) would have the same result than single.

In [162]:

```
Z = linkage(X, 'complete' , optimal_ordering=True) # average, weighted, centroi
d, median, would have the same result

c, coph_dists = cophenet(Z, pdist(X))
print( 'Cophenetic Correlation : %1.2f ' % c)

plt.figure(figsize=(25, 10))
plt.title('Hierarchical Clustering Dendrogram (full)')
plt.xlabel('sample clusters')
plt.ylabel('distance')
dn = dendrogram( Z, leaf_rotation=90., leaf_font_size=8.,)
plt.show()
```

```
Cophenetic Correlation : 0.98
```



Average-linkage, weighted, centroid, median, would have the same result than complete.

Question 4 - Comment the hierarchical clustering encoded as a linkage matrix.

Comprehensive visualization of the clusters

Tree's form remind us the history of the different aggregations. An agregation at the level i of the dendrogram means these two cluster have the shorter distance (based on the chosen one ; e.g. ward) between all clusters of level i.

Highlight hierarchical cluster structures

The length of the two legs of the U-link represents the distance between the child clusters.

Question 5 - Search for the definition and role of the cophenetic coefficient

The cophenetic coefficient is a mesure that shows how the goodness of fit of our clustering.

It is the intergroup dissimilarity at which the two observations are first combined into a single cluster.

A dendrogram is an appropriate summary of some data if the correlation between the original distances and the cophenetic distances is high

Question 6 - Simplified version of the dendrogram

Our above dendrogram is pretty big and pretty messy.

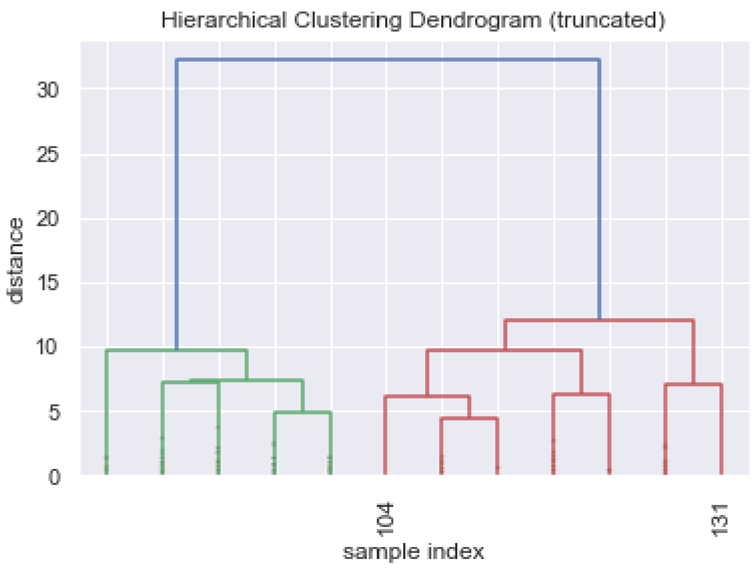
We can use different parameters of dendrogram function.

"truncate_mode" parameter defined the method used to truncate our dendrogram. E.g. : we will use 'lastp' to keep the last p merged clusters. "p" is defined as follow :

"p" parameter will defined the number of merged cluster to keep in our dendrogram. It will be the last p merged clusters.

In [163]:

```python
#display truncated dendrogram
plt.title('Hierarchical Clustering Dendrogram (truncated)')
plt.xlabel('sample index')
plt.ylabel('distance')
dendrogram(
    Z,
    truncate_mode='lastp' , #to explain
    p=12, #to explain
    show_leaf_counts=False ,
    leaf_rotation =90.,
    leaf_font_size=12.,
    show_contracted=True,
)
plt.show()
```



Question 7 - Cut the dendrogram to obtain the clusters

We want to argue for a certain number of clusters.

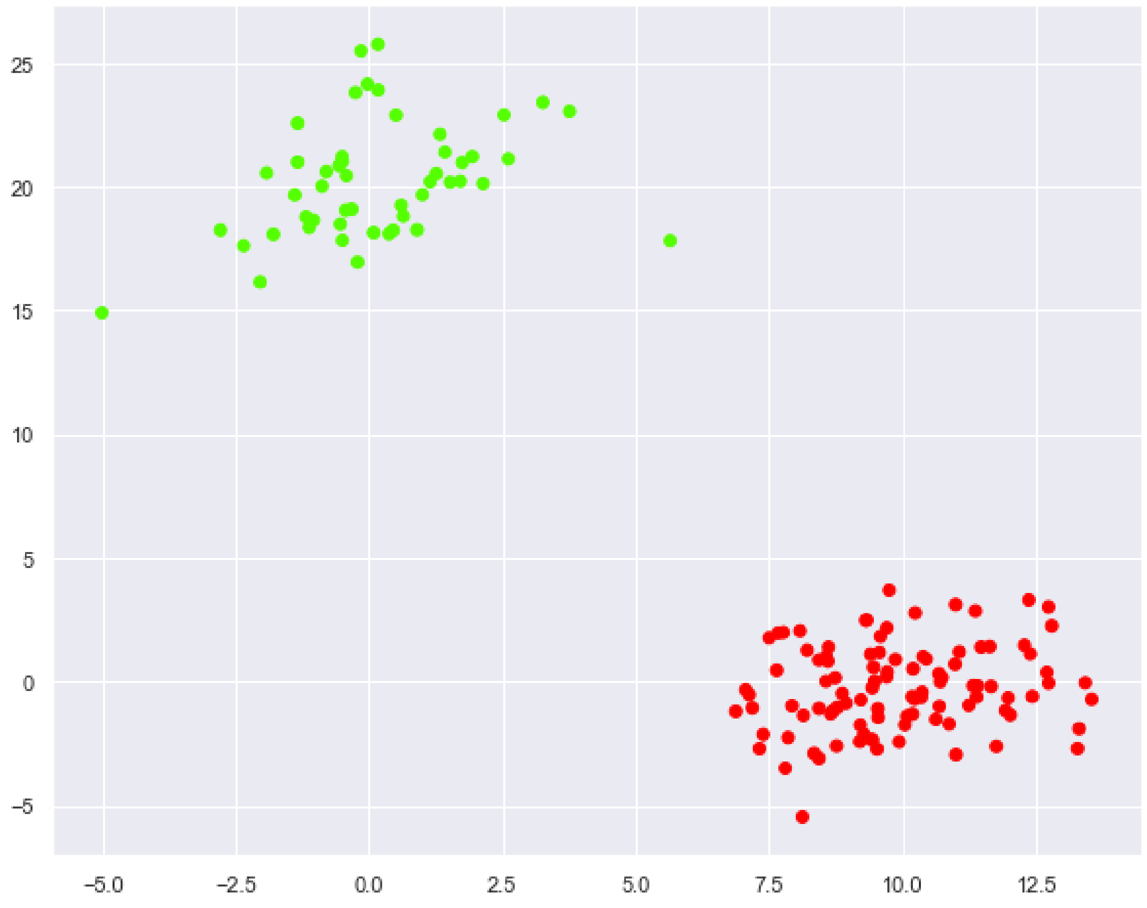
1) Method A : distance

Documentation's definition :

Forms flat clusters so that the original observations in each flat cluster have no greater a cophenetic distance than t.

In [164]:

```
max_d = 20
clusters = fcluster(Z, max_d, criterion='distance')

plt.figure(figsize=(10, 8))
plt.scatter(X[:,0], X[:,1], c=clusters , cmap='prism')
plt.show()
```



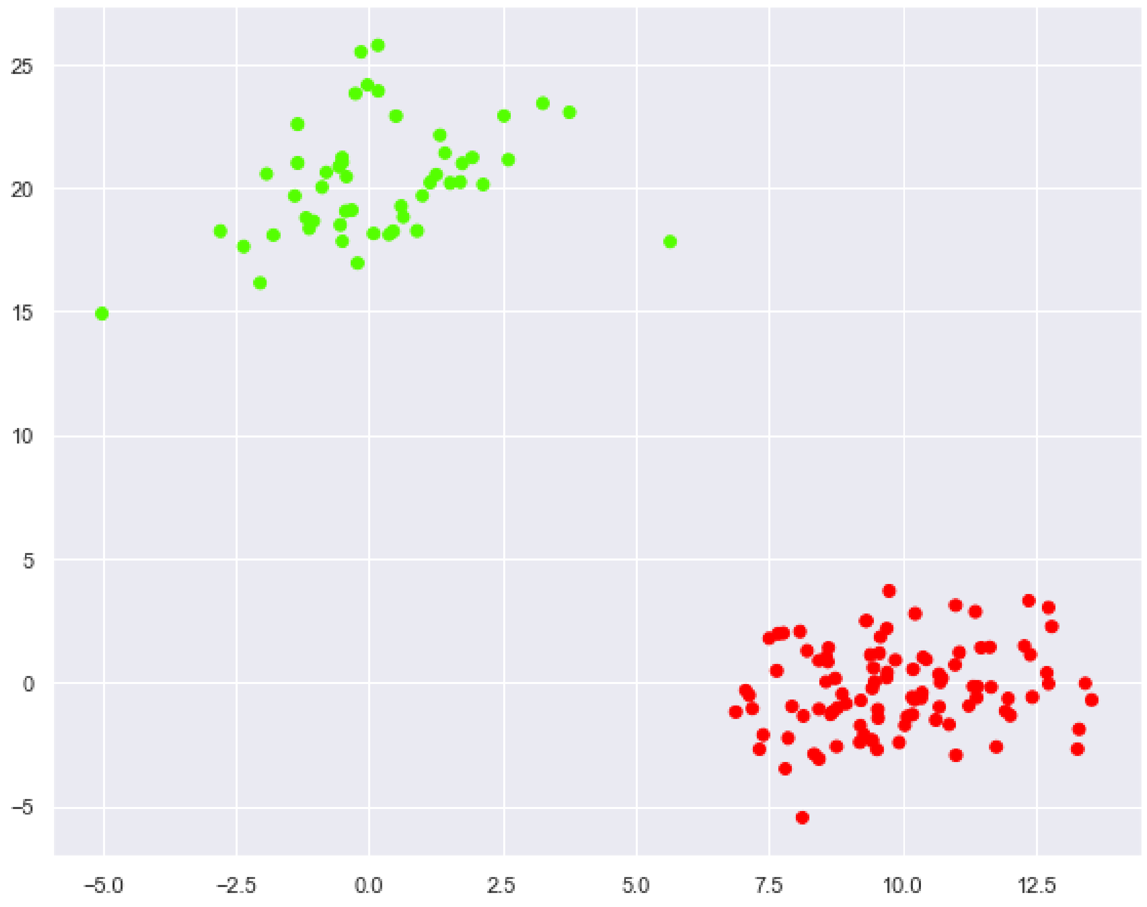
2) Method B : maxclust

Documentation's definition :

Finds a minimum threshold r so that the cophenetic distance between any two original observations in the same flat cluster is no more than r and no more than t flat clusters are formed.

In [165]:

```
k=2
clusters = fcluster(Z,k,criterion='maxclust')

plt.figure(figsize=(10, 8))
plt.scatter(X[:,0], X[:,1], c=clusters , cmap='prism')
plt.show()
```



3) Comparaison : distance vs maxclust

1. Difference

'distance' method : Thanks to dendrogram we have the distance between clusters. Then we can define a minimum distance to keep clusters. It will be the distance used to build Z with linkage function. We can vizualize this distance with the dendrogram. max_d is fixing the minimum distance (chosen during linkage call) to define a U-link as a cluster that we keep.

'maxclust' method : If we know how many cluster we should keep approximatly, then we'll certainly use this method. It keeps clusters based on dendrogram hierarchy (based on the tree). k is the maximum number of cluster we want to keep.

1. pros and cons

If we know how many clusters we should keep, then 'maxclust' method is definitly the best as it will keep the first k cluster (based on the distance we chose).

Nonetheless, if we prefer to focus on cophenetic coefficient and how well our clusters are representing our data, we should use 'distance' with an appropriate value.

Question 8 - Repeat the questions 3 to 7 with a different linkage

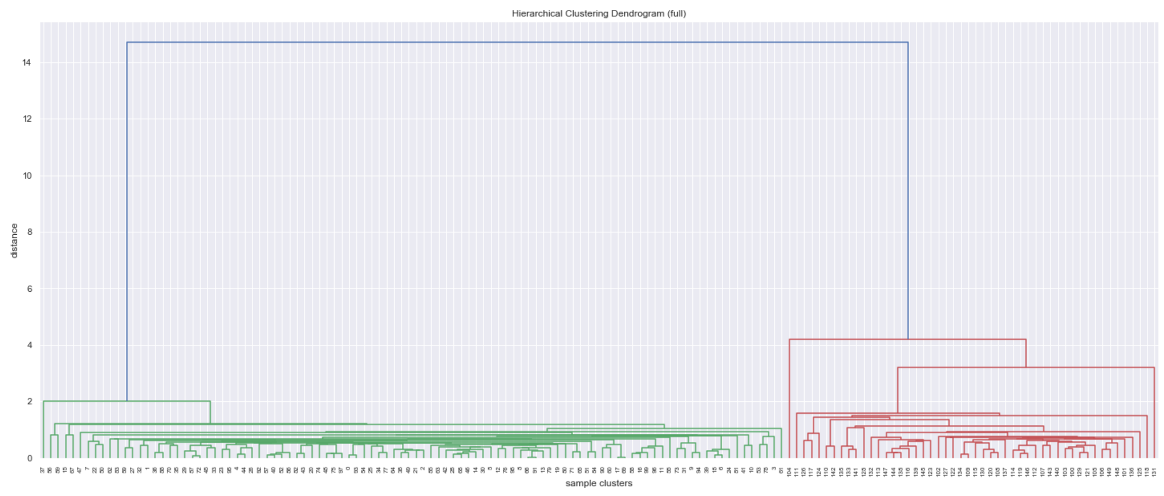
We will use single-linkage as there is a huge difference compare to complete-linkage used before.

Generate the linkage matrix and visualize the dendrogram

In [166]:

```
Z = linkage(X, 'single' , optimal_ordering=True) # euclidean (default) would hav
e the same result

c, coph_dists = cophenet(Z, pdist(X))
print( 'Cophenetic Correlation : %1.2f ' % c)

plt.figure(figsize=(25, 10))
plt.title('Hierarchical Clustering Dendrogram (full)')
plt.xlabel('sample clusters')
plt.ylabel('distance')
dn = dendrogram( Z, leaf_rotation=90., leaf_font_size=8.,)
plt.show()
```

```
Cophenetic Correlation : 0.98
```



Comment the hierarchical clustering encoded as a linkage matrix.

The hierarchical cluster structures is more complexe, the dendrogram formed many clusters with small distance.

Simplified version of the dendrogram

Our above dendrogram is pretty big and pretty messy.

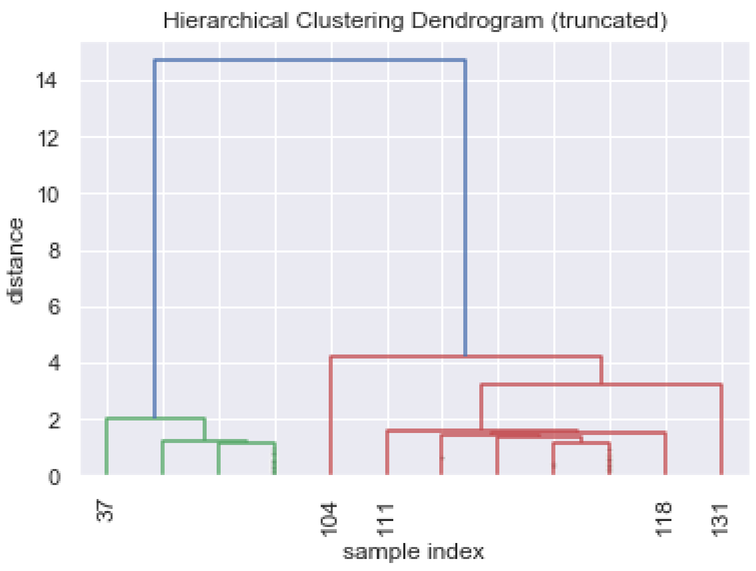
We can use different parameters of dendrogram function.

"truncate_mode" parameter defined the method used to truncate our dendrogram. E.g. : we will use 'lastp' to keep the last p merged clusters. "p" is defined as follow :

"p" parameter will defined the number of merged cluster to keep in our dendrogram. It will be the last p merged clusters.

In [167]:

```
#display truncated dendrogram
plt.title('Hierarchical Clustering Dendrogram (truncated)')
plt.xlabel('sample index')
plt.ylabel('distance')
dendrogram(
    Z,
    truncate_mode='lastp' , #to explain
    p=12, #to explain
    show_leaf_counts=False ,
    leaf_rotation =90.,
    leaf_font_size=12.,
    show_contracted=True,
)
plt.show()
```

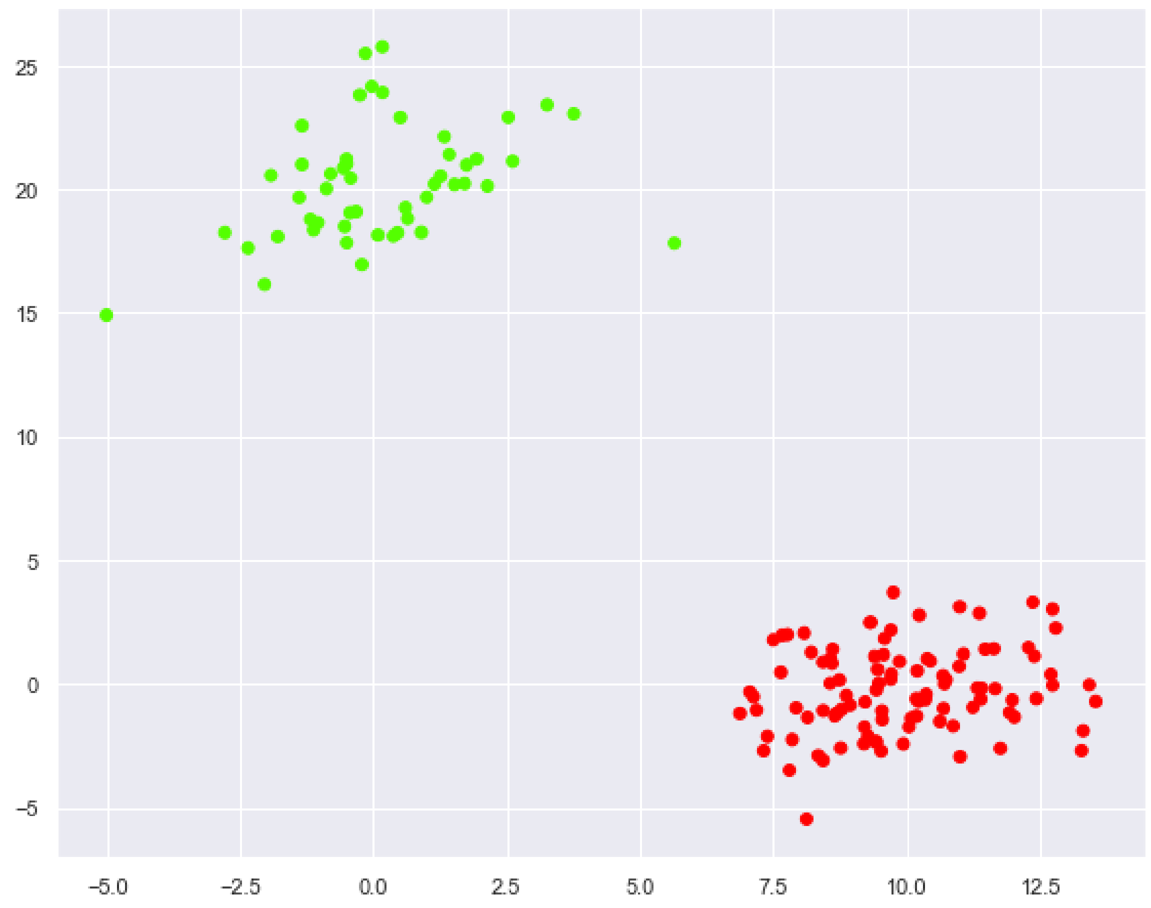


Cut the dendrogram to obtain the clusters

We want to argue for a certain number of clusters.

In [168]:

```
max_d = 14
clusters = fcluster(Z, max_d, criterion='distance')

plt.figure(figsize=(10, 8))
plt.scatter(X[:,0], X[:,1], c=clusters , cmap='prism')
plt.show()
```



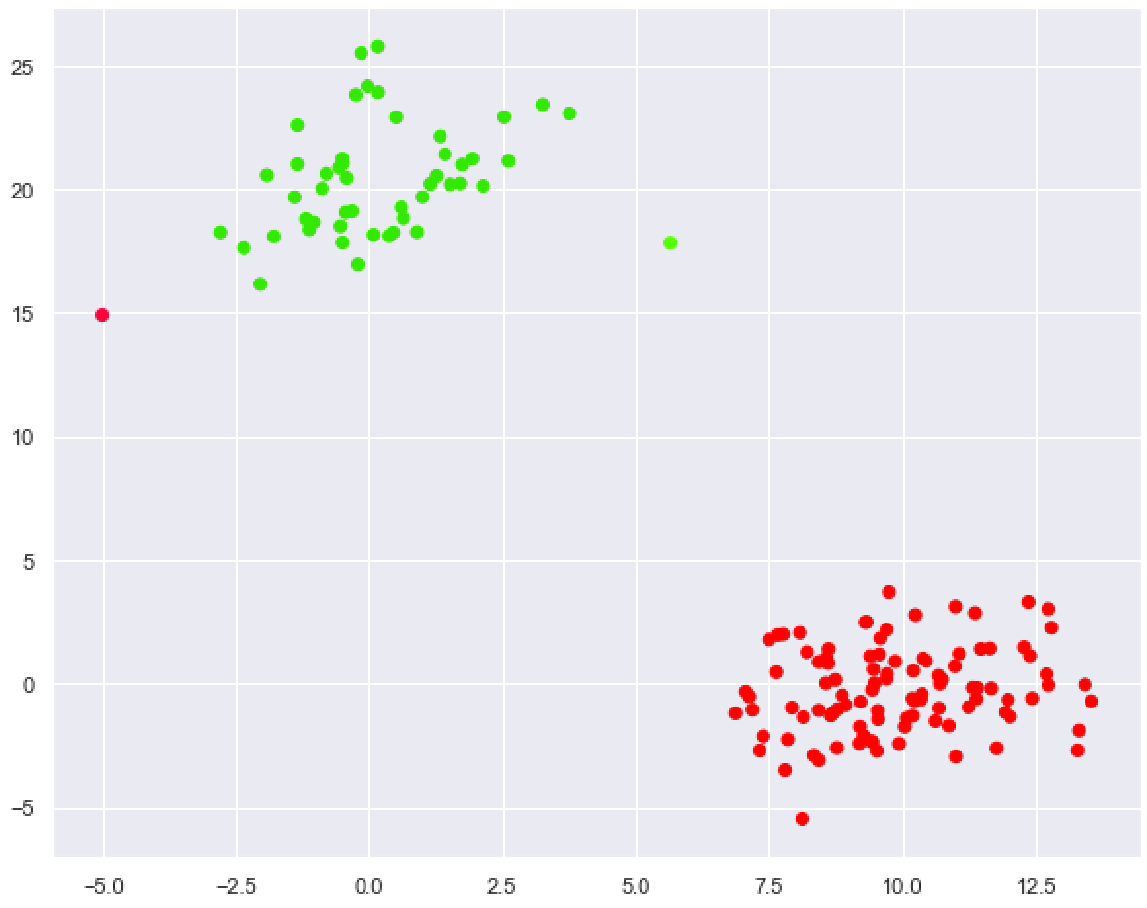
2) Method B : maxclust

Documentation's definition :

Finds a minimum threshold r so that the cophenetic distance between any two original observations in the same flat cluster is no more than r and no more than t flat clusters are formed.

In [169]:

```
k=4
clusters = fcluster(Z,k,criterion='maxclust')

plt.figure(figsize=(10, 8))
plt.scatter(X[:,0], X[:,1], c=clusters , cmap='prism')
plt.show()
```



We can see that single-linkage we quickly form clusters with a better cophenetic coefficient.

Cut-off using 'single-linkage' distance appear to form quickly our two cluster compare to 'complete-linkage'. We used max_d=14 ('single-linkage') compare to max_d=20 ('complete-linkage'). Same for maxclust method : k=2 ('single-linkage') compare to k=4 ('complete-linkage').

Data Analysis - Lab 4

M. Sébastien MASCHA & M. Sauvage Pierre

ISEP Paris – September 24th, 2019

Exercice C - Optimal cluster number in exoplanet data

In this exercice, we will try to guess the optimal number of clusters to be found in an arti□cial data set describing the atmospheric characteristics of exoplanets.

Import of libraries

This document has been done using python on Jupyter Notebook with the librairies:

- maths for sqrt, pi, exp
- Numpy to manipulate arrays
- pandas to import csv
- matplotlib to plot graphics
- seaborn to make your charts prettier (built on top of Matplotlib)
- sklearn : tools for data mining and data analysis
- mlxtend : tools for ploting PCA

In [13]:

```
# coding: utf-8

import data

from math import sqrt,pi,exp
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns; sns.set()

import sklearn
from sklearn import metrics
from sklearn.metrics import pairwise_distances
from sklearn.cluster import KMeans
```

Question 1 - Open the file

In [14]:

```
df = pd.read_csv("data/exo4_atm_extr.csv", sep =';')
df = df.drop(['Type'], axis=1)
print(df.shape)

df.head()
```

```
(1000, 11)
```

Out[14]:

	PH2O	PHe	PCH4	PH2	PN2	PNH3	PO2	PAr	PCO2	PSO2	PK
0	0.0	8.7	1.3	87.30	0.0	2.70	0.0	0.0	0.0	0.0	0.00
1	0.0	0.0	0.0	0.00	0.0	0.00	0.0	0.0	0.0	0.0	0.02
2	0.1	7.1	1.7	86.45	0.0	1.15	0.0	0.0	0.0	3.5	0.00
3	0.0	2.7	0.0	3.70	41.5	0.00	31.3	6.6	14.2	0.0	0.00
4	0.1	11.4	1.1	86.10	0.0	0.20	0.0	0.0	0.0	1.1	0.00

Question 2 - Write down the different properties of the Calinski-Harabasz and Davies- Bouldin indexes.

Calinski-Harabasz

- Not normalized
- Better when higher
- With balanced clusters, the CH index is generally a good criterion to indicate the correct number of clusters.

Davies- Bouldin indexes

- A lower DB value means a better clustering.
- This index is not normalized.
- It favors spherical clusters.
- It is biased so that it gives lower values with less clusters.

Question 3 - Calinski-Harabasz and Davies-Bouldin

In [19]:

```
kmeans_model = KMeans(n_clusters=3, random_state=1).fit(df.values)
labels = kmeans_model.labels_
print( "Calinski-Harabasz score is :" )
metrics.calinski_harabasz_score(df.values, labels)
```

```
Calinski-Harabasz score is :
```

Out[19]:

```
2527.9845312566085
```

In [20]:

```
print( "Davies-Bouldin score is :" )
metrics.davies_bouldin_score(df.values, labels)
```

```
Davies-Bouldin score is :
```

Out[20]:

```
0.3172965483768289
```

Question 4 - PCA

In [21]:

```
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(df)

principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])
principalDf.head(5)
```

Out[21]:

	principal component 1	principal component 2
0	62.601028	20.801273
1	-11.658641	-23.016774
2	61.723247	20.286628
3	-20.497593	-18.427451
4	61.875945	20.367130

In [ ]: