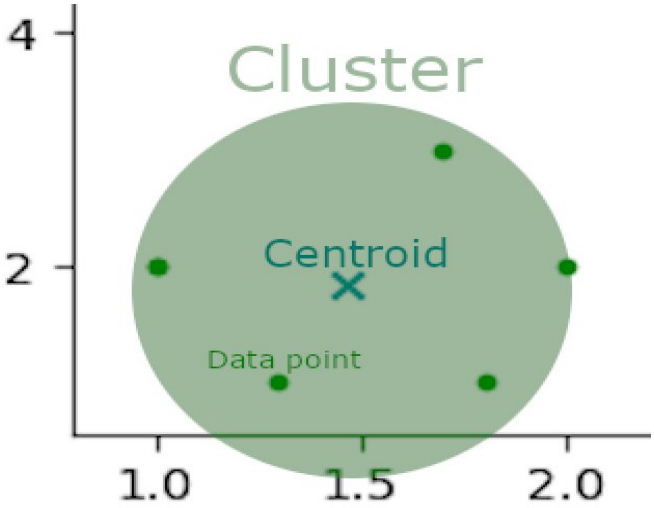
Clustering with K-means

Clustering is an algorithm that is hepful to grouping data due to some characteristics. There are a few ways to make clustering, in this case we are going to use the K-Means, witch is a **centroid** based clustering, if you do not undestand what is centroid or clustering do not worry we are going to explain that to you.

An easy way to understand the concepts

Cluster can be described as a group of points or features.
Centroid works as the center of a cluster or group.
The number of centroids wanted is the **K** in the K-means, in other words, if you want to label your data in two groups, K = 2, and there will be two clusters, and 2 centroids.
The number of clusters must be less or equal to the number of data points, this means, if you are going to use 10 data points at most you can have 10 clusters (K = 10).



How does the K-Means work?

The steps of this algorithm are:

1) Randomly pick k-centroids from the sample points as initial cluster centers.
2) Assign each sample to the nearest centroid.
3) Move the centroids to the center of the samples that were assigned to it.
4) Repeat steps 2 and 3 until the cluster assignments do not change or a user-defined tolerance or maximum number of iterations is reached.

The way to do this is using either Lloyd's or Elkan's algorithm.
If you want a visualization of Lloyd's algorithm please see [3].

Simple example

The best way to understand this is with an example.
First let's import the modules and functions required:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

Numpy will allow to transform all data in array form, this is due to efficency and as requirement in future functions. **Matplotlib** will help to visualize everything with simple graphs. **Sklearn** contains the machine learning algorithm that we are going to use, in this case, Kmeans.

Now we are going to create random data that will help us to understand how to use this algorithm, in this case x is a list of x coordenates but it can represent any kind of data. For example, can be daily returns of a certain stock in the market or the size of some product.
Is the same for y, in this case is just a list of y coordenates, but it can be any kind of information that we want, such as the variance of the return of some stock in the market, or the sales of some product.

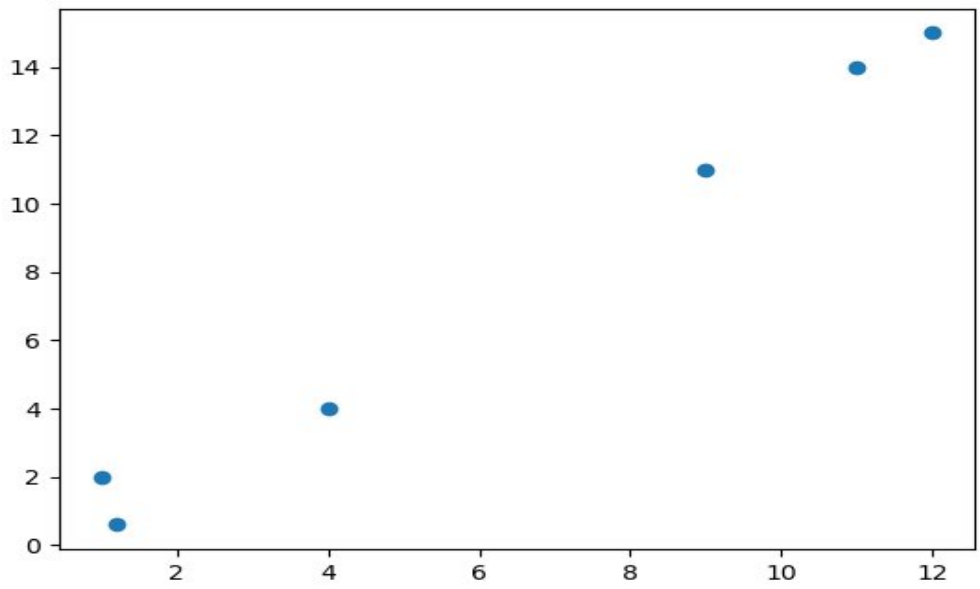
In the end, this vectors are the features that we want our algorithm to use to make groups or clusters.

```
x = [ 1, 4, 1.2,  9, 12, 11 ]
y = [ 2, 4, 0.6, 11, 15, 14 ]
```

Let's plot this coordenates to see them in a simple graph.

```
plt.scatter( x , y )
plt.show()
```

And we will see something like:



In this case for simplicity and to visualize better the algorithm we are using just two vectors, x and y, but we can use as much vectors as we want.

Now that we know how this look like, we can decide the number of clusters that we want. There are some few ways to select the best number for K, but, that is for future examples, by now let's select the number of clusters manually, we can say that this algorithm is semi-unsupervised.

Ok, let's think that we want two clusters (*image how this is going to be grouped*), so we write:

```
kmeans = KMeans(n_clusters= 2)
```

This line will create an empy object, with two clusters, centroids, labels, etc., but without info, so every variable in this object is empty as well, now we need to fit the info into this empty space, but we cannot put the vectors just like that, we need a numpy array (this is a requirement).

For this exaple we are going to create the numpy array from scratch but it can be created from the last vectors.

```
X = np.array([[  1, 2],
              [  4, 4],
              [1.2, 0.6],
              [  9, 11],
              [ 12, 15],
              [ 11, 14]])
```

This **X** is an array that contains the same information that before, the first column is the **x** vector and the second column is the **y** vector.

The next step is to fit this information into the empy object named **kmeans**, so let's write:

```
kmeans.fit(X)
```

Now this object will create everything for us, this means that now we can see in which cluster is each point, and the coordenates of each centroids. To extract and plot this in a friendly way, let's write this:

```
centroids = kmeans.cluster_centers_
labels = kmeans.labels_
colors = ['g.','r.','c.','y.']
```

Cluster_centers_ is an array that contains the coordenates of each centroind.
Labels_ is a list that contains the labeling of each point (in order) created by the algorithm.
Colors is a list createrd by us to distinguish each cluster.

We can print the this data to visualize it with:

```
print(centroids)
print(labels)
```

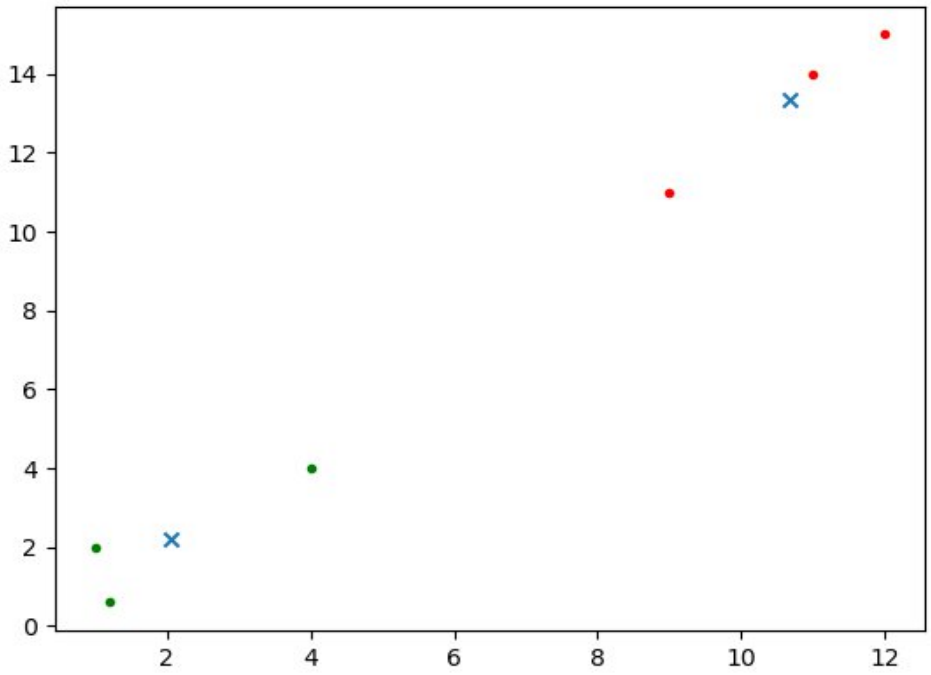
and we will see something like this:

```
[[ 2.06666667  2.2       ]
 [10.66666667 13.33333333]]
[0 0 0 1 1 1]
```

It's quite difficult to imagine how this clusters look like, so lets create a graphic that will help us to see it, the next code will help:

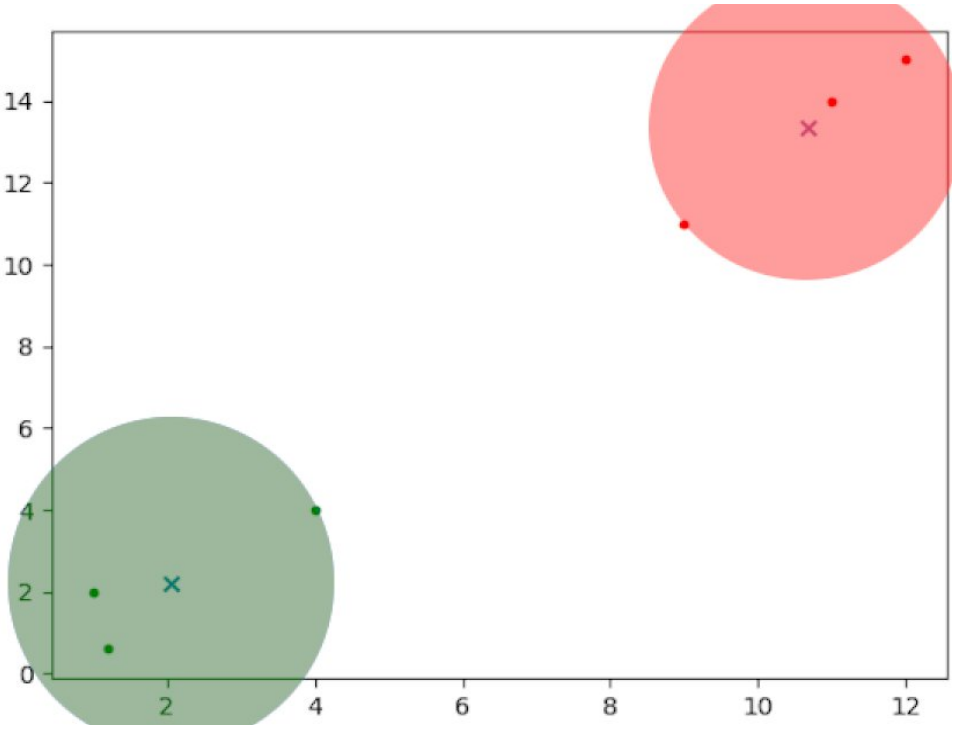
```
'Plots every dot and the centroids'
for i in range(len(X)):
    'Shows the i-th dot and his label'
    print('Cordinate:',X[i]," label:",labels[i])
    'Plots the point with the corresponding color'
    plt.plot( X[i][0], X[i][1], colors[labels[i]] )
'Plots the centroids as a x '
plt.scatter( centroids[:,0] , centroids[:,1], marker = 'x')
plt.show()
```

and the result is:



Now we can see that each cluster corresponds to each cluster and the x's are the centroids created by the algorithm.

So, the firs cluster is the green one, the second cluster is de red one.
If we could see the clusters like the algorithm does, we could see something like:

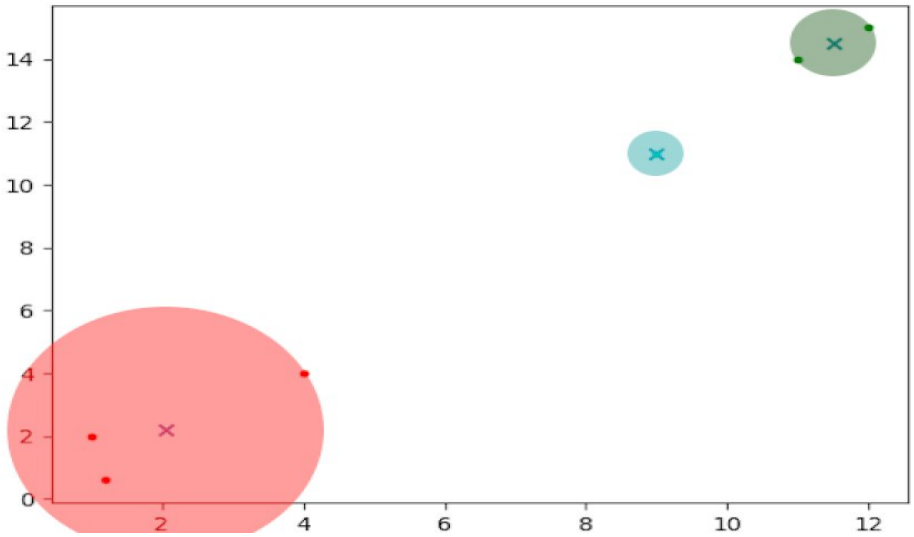


But, what happens when we put 3 clusters?

We, just need to change the n_clusters in:

```
kmeans = KMeans(n_clusters= 3)
```

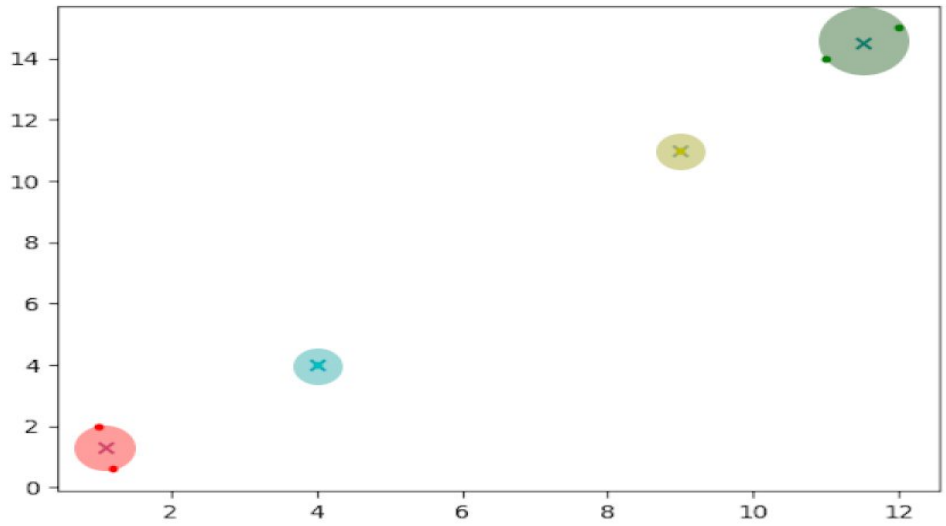
And run the code again...



Now we can see that one of the points is alone, which means that the distance from that blue point to the nearest centroid of a cluster (green) is higher than the distance of the red point in the frontier of the red circle to the centroid of the red cluster.

So if we change the number of clusters to four, we can assume that the point in the frontier of the red circle is going to be a new cluster.

Making the corresponding changes and re-running the code it returns:



It's important to mention that it doesn't matter if the colors of the clusters change, this is because the algorithm found this clusters in different order depending of the solutions that finds each time the programs runs a minimize optimization.

Finally an important note from the module sklearn to know is: If the algorithm stops before fully converging (because of tol or max_iter), labels_ and cluster_centers_ will not be consistent, i.e. the cluster_centers_ will not be the means of the points in each cluster. Also, the estimator will reassign labels_ after the last iteration to make labels_ consistent with predict on the training set.

Conclusions

This algorithm is useful to create groups that can be difficult for a human due, maybe, the size of the data (to much info, and maybe not easily to group), the number to vectors that will be used (More than 4 vectors it's difficult to visualize because is a 4-dimension problem), or simply due the distribution of the points.

However, if you are interested in learning how to select the best K number, or another ways to cluster, or want to see an example in finance, feel free to check out our Github.

Code

```
#Libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

#Data
'Some data that will help as example'
x = [ 1, 4, 1.2,  9, 12, 11 ]
y = [ 2, 4, 0.6, 11, 15, 14 ]

'Shows the raw data'
plt.scatter( x , y )
plt.show()

'Transform the data into a numpy array'
X = np.array([     [  1, 2],
                   [  4, 4],
                   [1.2, 0.6],
                   [  9, 11],
                   [ 12, 15],
                   [ 11, 14]])

# Kmeans algorithm
'Select the number of clusters'
kmeans = KMeans(n_clusters= 4)

'Fit the data into the algorithm and the object kmeans is created'
kmeans.fit(X)

'Extract the centro'
centroids = kmeans.cluster_centers_
labels = kmeans.labels_
colors = ['g.','r.','c.','y.']

'Print the centroids and the labels'
print(centroids)
print(labels)

'Plots every dot and the centroids'
for i in range(len(X)):
   'Shows the i-th dot and his label'
   print('Cordinate:',X[i]," label:",labels[i])
   'Plots the point with the corresponding color'
   plt.plot( X[i][0], X[i][1], colors[labels[i]] )
'Plots the centroids as a x '
plt.scatter( centroids[:,0] , centroids[:,1], marker = 'x')
plt.show()
```

Real example

Now we can make the next example, clustering some countries by returns and volatility.
So, we need the file called ' Mapping Countries.xslx' where we are going to have the tickers of some ETF's that could represent those countries.

	A	B	C	D
1	Ticker	Country		
2	ARGT	Argentina		
3	EWA	Australia		
4	EWO	Austria		
5	EWK	Belgium		
6	EWZ	Brazil		
7	EWC	Canada		
8	ECH	Chile		
9	MCHI	China		
10	ICOL	Colombia		

This is what we are going to need to get all the data required from Yahoo finance.
First, let's take a loot at the libraries.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pandas_datareader as dr
from math import sqrt
from sklearn.cluster import KMeans
from scipy.cluster.vq import kmeans,vq
from numpy.core.shape_base import vstack
from scipy.cluster.hierarchy import dendrogram, linkage
```

We can see the same libraries than before, but, this time we include some other functions, just to make a more complete analysis, we will explained at the moment.

Next we use the datareader library to get the data and saving it into the prices_df variable.

```
'Get the data from yahoo and put it in DataFrame'
for ticker in tickers:
    try:
        prices = dr.DataReader(ticker,'yahoo','01/01/2018')['Adj Close']
        prices = pd.DataFrame(prices)
        prices.columns = [ticker]
        prices_list.append(prices)
    except:
        pass
    prices_df = pd.concat(prices_list,axis=1)
prices_df.sort_index(inplace=True)
print(prices_df)
```

Now let's calculate the average percentage return and volatilities.

```
returns = prices_df.pct_change().mean() * 252
returns = pd.DataFrame(returns)
returns.columns = ['Returns']
returns['Volatility'] = prices_df.pct_change().std() * sqrt(252)
```

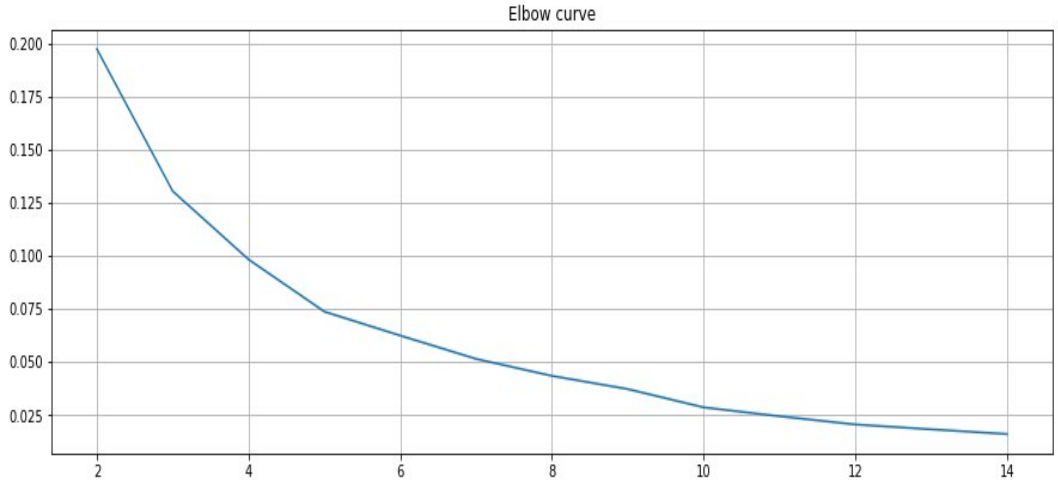
We format the data as a numpy array to feed the K-means algorithm

```
data = np.asarray([np.asarray(returns['Returns']),np.asarray(returns['Volatility'])]).T
X = data
distorsions = []
```

Next we plot the elbow curve, this help us to know the number of clusters that we need.
In this case we make the cases from 2 to 15 clusters.

```
for k in range(2, 15):
    k_means = KMeans(n_clusters=k)
    k_means.fit(X)
    distorsions.append(k_means.inertia_)
fig = plt.figure(figsize=(15, 5))
plt.plot(range(2, 15), distorsions)
plt.grid(True)
plt.title('Elbow curve')
plt.show()
```

The Elbow curve look like this.



In the X axis we find the number of clusters and in the Y axis we find the percentage of information lose using that clusters. Just for this example we will use 5 clusters.

```
centroids,_ = kmeans(data,5)
```

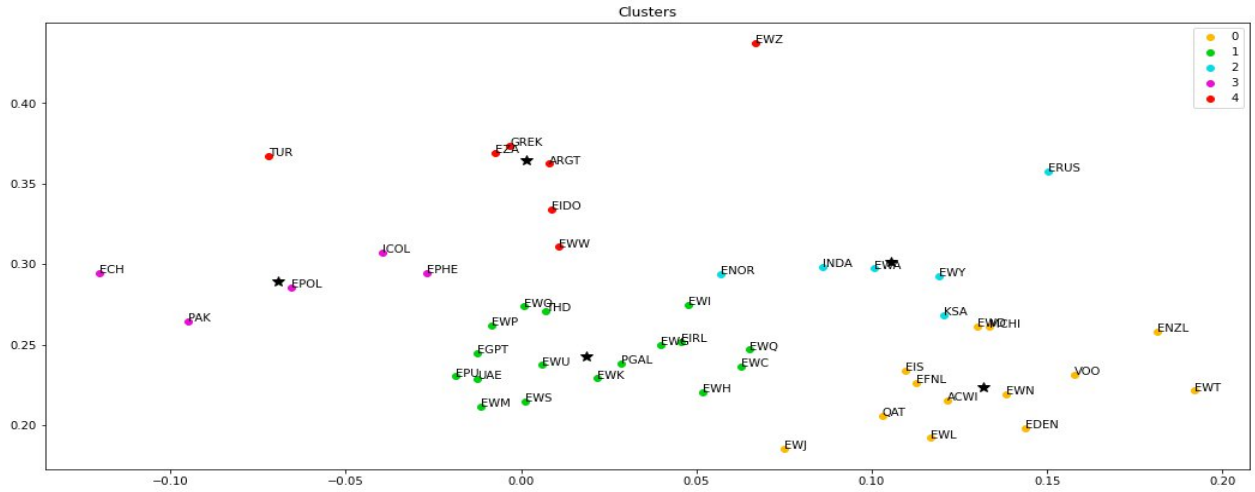
This line will give us the centroids of each cluster and we will save it into the centroids variable.
Then we assign each sample to a cluster, also we create a dataframe with the information to plot everything and a list with the colors that we want for each cluster.

```
idx,_ = vq(data,centroids)
Cluster_points = pd.DataFrame(data=data)
Cluster_points.columns = ['Ret','Vol']
Cluster_points['Cluster'] = idx
colors = ['#fcba03', '#02cf10', '#07dce3', '#e810d2', '#ff0800']
```

Let's plot every point with the corresponding color and coluster with this block of code.

```
plt.figure(figsize=(18,8))
for i in range(len(idx)):
    'Plots the name of each ticker'
    plt.text( data[i,0], data[i,1], tickers[i], size=10)
for i in range(len(colors)):
    'Plot the points of each ticker'
    plt.scatter(x = Cluster_points.loc[Cluster_points['Cluster']==i, 'Ret'],
                y = Cluster_points.loc[Cluster_points['Cluster']==i,'Vol'],
                color = colors[i], label=i)
    'Plots the centroids as stars'
    plt.plot(centroids[i,0],centroids[i,1],'*',color='black',markersize=10)
plt.legend()
plt.title('Clusters')
plt.show()
```

The scatter looks like this.



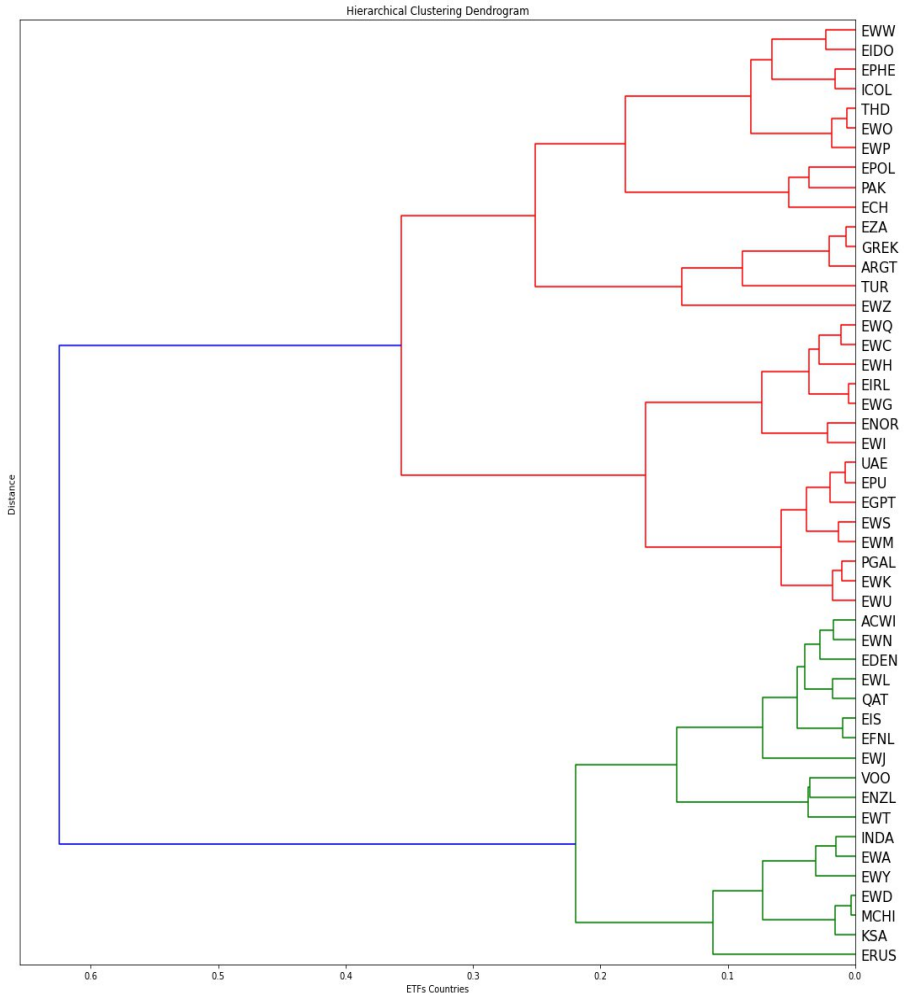
We will save everything into a excel file to futures references.

```
details = [(name,cluster) for name, cluster in zip(returns.index,idx)]
Det = pd.DataFrame(details)
Det.columns = ['Ticker','Cluster']
Det.to_excel("Ticker_Cluster.xlsx")
```

Finally we make a hierarchical clustering dendrogram to show how the clusters are builted.

```
plt.figure(figsize=(15, 15))
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('ETFs Countries')
plt.ylabel('Distance')
dendrogram(linkage(vstack([Cluster_points['Vol'],Cluster_points['Ret']]).T,
                   'ward'),
           orientation='left',
           leaf_rotation=0.,
           leaf_font_size=16.,
           labels=returns.index )
plt.tight_layout()
plt.show()
```

The graphic tell us wich stock goes into wich cluster and looks like this.



References

https://www.analytixlabs.co.in/blog/types-of-clustering-algorithms/
https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
https://datasciencelab.wordpress.com/tag/lloyds-algorithm/
https://github.com/ArturoAguilar87/Pynance_plus_Workshops