Clustering analysis: Identifying groups of best and worst performing S&P500 stocks - Karl Merisalu

Background: Index tracker funds are one of the most common ways to invest one's assets, because they are well diversified, low cost and as such offer a good return to risk profile for investors. One of the most well known indexes in the world is S&P500, which tracks the performance of 500 large companies listed on stock exchanges in the United States.

Every year, some of the 500 companies in S&P500 index perform better and some perform worse. If a client is exposed to S&P500 index returns, knowing which companies performed better or worse and why makes an interesting conversation topic for clients.

Problem: How can I explain why some companies in S&P500 index have performed better and some have performed worse?

I could look into each company separately and identify the cause for performance, but that would be a very time consuming task and also a very long conversation with a client. Instead, it would probably be better for me to summarize what's happening on the market on a higher level. One way to do this would be to divide stocks by similar performance (or by other characteristics) into "buckets" and characterise these clusters. This could be an efficient way to explain our clients S&P500 performance.

Task: I will work together with my group members (Colleague X, Colleague Y and Colleague Z) to develop a clustering algorithm to divide stocks in S&P500 index into clusters by their past performance.

The project will be conducted in Python ecosystem to ensure the analysis would be fast and replicable on other collections of financial instruments. Python also has good tools/libraries to conduct clustering analysis, which may be less intuitive and customisable in other ecosystems.

In this group project, in addition to specific analysis tasks, my role is also to act as a project manager: to coordinate the workflow, allocation of tasks. Coordinating this project is somewhat similar to my day to day job, where I frequently interact with all other departments as part of cross-functional projects and tasks in my role as a Business Manager.

I am also responsible for the analysis post "elbow curve", interpretation and suggestions for further research.

All group members are responsible for reviewing all of the below and making sure they understand and are able to reproduce everything individually.

Given that all group members work in different departments of UBS Asset Management, this project will be a great example of project management, collaborating across teams/departments and making sure that everyone understand all aspects of the work.

As a result of this analysis, I will be able to identify and divide stocks of S&P500 index (or any other collection of stocks) into clusters based on past performance.

Limitations: The analysis is only based on past performance, which may not be the best way to describe stock performances. Stock performances could be impacted by industry effects, country political risks and many other factors. As such, the results of this analysis should be considered in combination with other views to form a full picture for clients.

Summary of actions: To summarize, this is a clustering group project, where I'm going to divide stocks of S&P500 index into groups based on their past performance. This project will be useful for my firm because this script can help identifying groups of similar stocks out of a large collection, which could take a very long time doing it manually. This would be helpful to our research analysts and client advisors for conducting conversations with UBS clients. In the end of the project I will conclude findings and make suggestions for further research.

Following is a step by step guide on how I'm going to accomplish it:

1) Use case *(for real life perspective)*
2) Setting up the environment *(enables me to use pre-developed complex functions on data)*
3) Dataset *(import and cleaning of the data)*
4) K-means clustering analysis *(creating and optimising the model, results)*
5) Conclusion *(interpretation and suggestions for further research)*

1) Use Case

To start with and put our analysis into perspective:

An investor wishes to create an investment portfolio by selecting a number of stocks from the universe of 500 large cap stocks contained in the S&P 500 Index. I'm applying cluster analysis to the stock universe to see which groups may form and based on which characteristics. Ideally, the results will give the investor valuable information for the stock selection process. The cluster analysis is run on risk and return data derived from the initial data set of daily historical stock prices.

2) Setting up the environment

I start with importing required libraries, which will help me to conduct the analysis. Each library is commented in the code below:

In [1]:

```
# Import the pandas library in order to read, transform and manipulate the data
import pandas as pd

# Import matplotlib to visualise the data
import matplotlib.pyplot as plt

# Import the sklearn library in order to implement machine learning algorithms. Note, t
hat we only import the
# functions we'll need from sklearn (cluster & preprocessing).
from sklearn.cluster import KMeans

# From sklearn import a function that allows us to scale the our data
from sklearn.preprocessing import scale

# Command that enables us to display the graphs in our Jupyter Notebook (right here!)
%matplotlib inline

from pylab import plot,show
from numpy import vstack,array
from numpy.random import rand
import numpy as np
from scipy.cluster.vq import kmeans,vq
from math import sqrt
```

3) Dataset

Daily stock prices of S&P500 Index members from 30 Nov 2010 to 30 Nov 2018 (8 years). This data was sourced by one of our project group members.

UBS has 4 data confidentiality classes:
- **Strictly confidential** – not to be further distributed
- **Confidential** – only for targeted audience
- **Internal** – can be shared with colleagues internally
- **Public** – can be shared internally and externally

This S&P500 stock price data qualifies as public data, so there is no need to hide/transform any of the data

In [2]:

```
# We import the historical stock prices of the S&P 500 Index members via the pandas rea
d_csv
# function into a DataFrame
df_stockprices = pd.read_csv("public_stock_data3.csv")
```

In [3]:

```
# We take a first look at our Data Frame
df_stockprices.head()
```

Out[3]:

	Date	LYB UN Equity	AXP UN Equity	VZ UN Equity	AVGO UW Equity	BA UN Equity	CAT UN Equity	JPM UN Equity	CVX UN Equity	KO UN Equity	...	M Equ
0	30.11.2010	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	...	100
1	01.12.2010	102.29	102.50	101.06	102.87	103.06	103.37	102.06	102.14	102.42	...	100
2	02.12.2010	104.38	104.07	102.28	108.81	104.42	104.75	105.16	104.36	102.74	...	103
3	03.12.2010	104.48	103.84	102.78	101.69	104.34	105.65	105.97	104.84	102.11	...	104
4	06.12.2010	106.40	104.12	102.75	102.80	104.42	105.44	106.69	104.92	101.30	...	103

5 rows × 407 columns

Comment: The data looks well formatted for our needs. Let's take a quick look how many missing values I have.

In [4]:

```
# We use the isnull() function to find missing values in the DataFrame
df_stockprices.isnull().sum().sum()
```

Out[4]:

```
0
```

I have no missing values - great! Let's move on with adjusting the dataset to our needs.

Next, I drop date values as I don't need them:

In [5]:

```
# I create a new DataFrame based on the previous one,
# but drop the date column to be able to perform calculations on returns only.
prices = df_stockprices.drop(['Date'], axis=1)
```

Calculating annual returns and volatilities next. Because I have daily prices, I do the following steps:

1) Calculate daily percentage changes for returns
2) Then calculate mean value of percentage changes and multiply with number of trading days (252) in a year to get annual return
3) I make sure the result is stored as a dataframe and call the column as "Returns"
4) Finally I calculate annual volatility as using (daily standard deviation) * √252 formula and save it as "Volatility" column

In [6]:

```
# I calculate average percentage return p.a. and volatilities p.a. over a theoretical o
ne year period
returns = prices.pct_change().mean() * 252
returns = pd.DataFrame(returns)
returns.columns = ['Returns']
returns['Volatility'] = prices.pct_change().std() * sqrt(252)
```

In [7]:

```
# First look at our new DataFrame containing returns and volatilities of our stock univ
erse
returns.head()
```

Out[7]:

	Returns	Volatility
LYB UN Equity	0.244836	0.318487
AXP UN Equity	0.151902	0.212115
VZ UN Equity	0.135083	0.164739
AVGO UW Equity	0.339874	0.337397
BA UN Equity	0.254090	0.227732

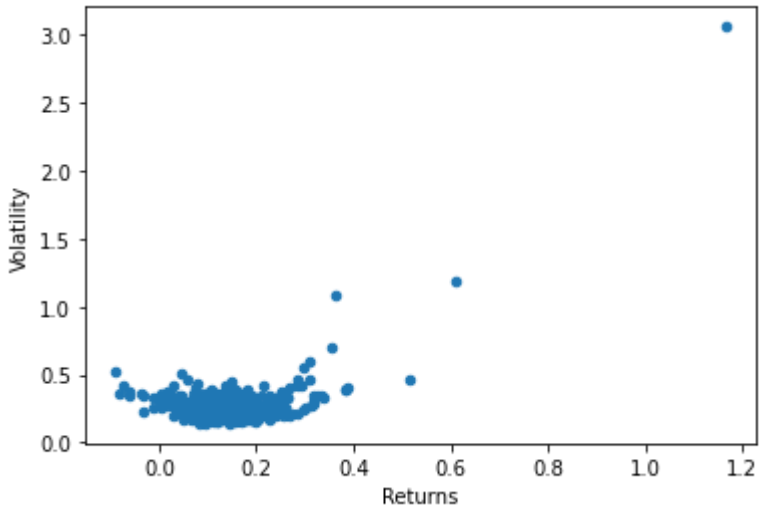
Comment: It looks like I have the data we need for cluster analysis. Before I start with clustering, let's take a quick look at how the data looks like on a chart

In [8]:

```
# We plot the return and volatility combinations for each stock on a scatter graph
returns.plot.scatter(x='Returns', y='Volatility')
```

Out[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d412b98>
```



Comment: Interestingly I seem to have some very extreme outliers. This would complicate clustering analysis and make it less reliable. I'm going to find out which stocks are these outliers and drop them, until I have a more consistent dataset. Alternatively, I could use clustering techniques to identify outliers.

In [9]:

```
# We identify the stock outliers (and we subsequently remove them).
# We understand that this is a meaningful change to the data set and keep this in mind.
print(returns.idxmax())
```

```
Returns       C UN Equity
Volatility    C UN Equity
dtype: object
```

In [10]:

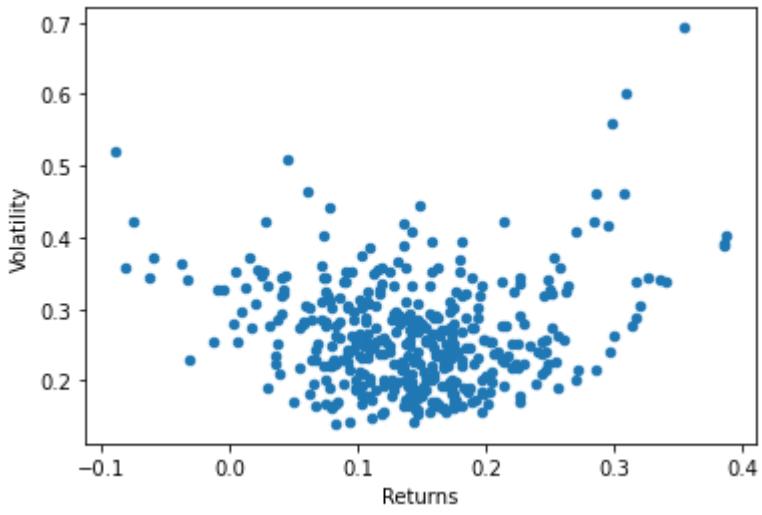
```
# drop outlier stocks from the data set
returns.drop('C UN Equity',inplace=True)

# the following other outliers were identified by running the above code again:
returns.drop('MSI UN Equity',inplace=True)
returns.drop('ABMD UW Equity',inplace=True)
returns.drop('XRX UN Equity',inplace=True)
#recreate data to feed into the algorithm
data = np.asarray([np.asarray(returns['Returns']),np.asarray(returns['Volatility'])]).T
```

In [11]:

```
returns.plot.scatter(x='Returns', y='Volatility')
```

Out[11]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1d4a7bb0>
```



Comment: Great, the data looks much more consistent! Now it's time for clustering.

4) K-means clustering analysis

K-means clustering is one of the most common clustering algorithms. More detailed information about K-means can be found here: https://en.wikipedia.org/wiki/K-means_clustering (https://en.wikipedia.org/wiki/K-means_clustering)

Shortly, k-means clustering works like this:
1) I select a predetermined number of centroids *(definition: average position of all the points of an object)* to be randomly placed on a chart among all datapoints. Let's call these centroids "k" or "k-points".
2) All datapoints are then associated with the "k-point" that they're closest distance to.
3) The algorithm calculates a mean location for datapoints associated with their closest "k-point". "K-point" location is then updated with this mean value.
4) We go back to step #2, and repeat the process of adjusting "k-point" for a several times until it reaches its equilibrium location.
5) We have now identified centers of clusters, which are "k-points" and clusters themselves, which are the closest points surrounding these "k-points", respectively.

To start with, I begin modelling with with 5 clusters:

In [12]:

```
# I run the kmeans function and store the result in our variable, 'model'
model = KMeans(n_clusters=5)

# Now I fit our kmeans model to our data.
model.fit(scale(returns))
```

Out[12]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=5, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

Comment: I now have the clusters. Time to label them next:

In [13]:

```
# Next we apply labels to each cluster for better tracking and creation of summary statistics
returns['cluster'] = model.labels_
```

In [14]:

```
returns.head()
```

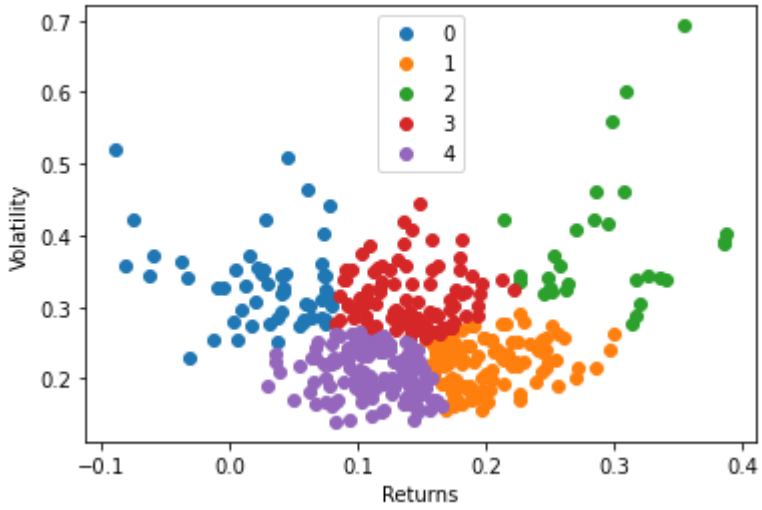
Out[14]:

	Returns	Volatility	cluster
LYB UN Equity	0.244836	0.318487	2
AXP UN Equity	0.151902	0.212115	4
VZ UN Equity	0.135083	0.164739	4
AVGO UW Equity	0.339874	0.337397	2
BA UN Equity	0.254090	0.227732	1

Comment: each S&P500 stock is assigned a cluster number as seen above. Let's portray all clusters colorfully on a chart next:

In [15]:

```
# We continue with a colourful visualisation of our clusters, after the most extreme ou
tliers have been dropped

# First we create a group for each cluster using the groupby function
groups = returns.groupby('cluster')

# Plotting the clusters!
fig, ax = plt.subplots()
for name, group in groups:
    ax.plot(group.Returns, group.Volatility, marker='o', linestyle='', label=name)

# Add axis labels appropriately
plt.xlabel('Returns')
plt.ylabel('Volatility')
ax.legend()
plt.show()
```

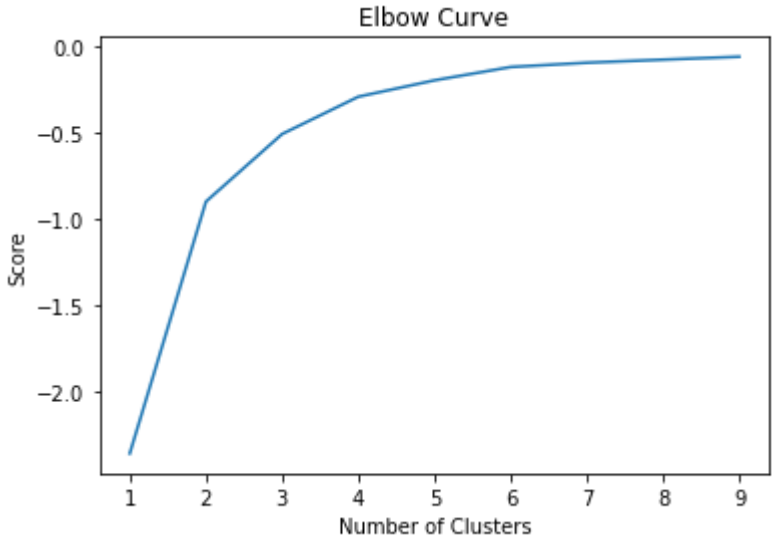


Comment: Voila! I have successfully created 5 clusters based on their return and volatility characteristics.

Would it make sense to have more or less clusters than 5? Let's find out by analyse this with an elbow test next.

In [16]:

```
# Next we run the "elbow test", so see what might be a good number of clusters to work
 with
# WSS elbow test

# First specify what will go on our X and Y axes
X = returns[['Returns']]
Y = returns[['Volatility']]

#
Nc = range(1, 10)
kmeans = [KMeans(n_clusters=i) for i in Nc]
score = [kmeans[i].fit(Y).score(Y) for i in range(len(kmeans))]
plt.plot(Nc,score)
plt.xlabel('Number of Clusters')
plt.ylabel('Score')
plt.title('Elbow Curve')
plt.show()
```



Comment: The fewer clusters I can use to model datapoints well the better the outcome, because it shows me how much I can summarise the data with clusters. Otherwise I could add 500 clusters for 500 stocks, which would add not much...

Elbow curve shows me how much additional "gain" I am getting by adding another cluster. The number of clusters should be chosen at the point where the curve starts flattening out. In this case 5 seems like a good number of clusters, but I could also consider running the algorithm with 4 clusters. Elbow curve is a good indicator, but not a definitive answer and as such I may need to exercise personal judgement.

In this case I choose to proceed with 5 clusters.

Let's check how many stocks we have in each cluster next:

In [17]:

```
# We review how many stocks are in each cluster
returns["cluster"].value_counts()
```

Out[17]:

```
4    125
1    102
3     92
0     54
2     29
Name: cluster, dtype: int64
```

Comment: good news is that we have a good amount of datapoints in each cluster as also evidenced by the chart above.

I can also run descriptive statistics on clusters, which provides a statistical summary that can be used for further analysis.

In [18]:

```
# To better understand the properties each cluster, we run some descriptive statistics
 and write them into a DataFrame
cluster_stats = returns.groupby('cluster')[['Returns', 'Volatility']].describe()
```

In [19]:

```
cluster_stats
```

Out[19]:

	Returns								Volatil
	count	mean	std	min	25%	50%	75%	max	count
cluster									
0	54.0	0.027919	0.045111	-0.088518	0.006877	0.038633	0.062974	0.079920	54.0
1	102.0	0.201753	0.034228	0.152513	0.173189	0.196381	0.224423	0.300159	102.0
2	29.0	0.293972	0.048351	0.214493	0.252861	0.294882	0.319528	0.386720	29.0
3	92.0	0.140621	0.032308	0.083159	0.114317	0.140659	0.162789	0.221240	92.0
4	125.0	0.112228	0.031396	0.030245	0.091672	0.112526	0.139814	0.164865	125.0

Comment: Probably the best way for me to summarize clusters, however, is to calculate a return to risk ratio. This ratio will show me how much return given cluster is offering for a unit of risk on average. The higher the ratio the better the investment is considered generally (it translates to more return with more certainty).

In [20]:

```
# Risk-adjusted returns are often used as an evaluation input.
# We therefore calculate a simple metric which is return by unit of risk. The new metri
c is added into the DataFrame.
cluster_stats['Return/Volatility Ratio'] = (cluster_stats['Returns']['mean'])/(cluster_
stats)['Volatility']['mean']
```

In [21]:

```
# Following output shows the average "Return by unit of risk" for each cluster.
# Typically a higher ratio is deemed more attractive then a lower ratio,
# as the investor receives a higher return versus the amount of risk taken.
cluster_stats['Return/Volatility Ratio']
```

Out[21]:

```
cluster
0    0.084214
1    0.911700
2    0.758610
3    0.448783
4    0.547031
Name: Return/Volatility Ratio, dtype: float64
```

Comment: as expected, cluster #1 (orange) has the highest ratio and cluster #0 (blue) the lowest ratio.

As a side note, interestingly, there seem to be rather big differences in cluster return/volatility ratios, which means that it would be difficult to model return on volatility with a linear regression for all clusters together, should I want to do it in the future.

As an example for next steps, I can quickly peak into cluster #1, to see which stocks it consists of, however, this is already outside of this project's scope.

In [23]:

```
returns.loc[returns['cluster'] == 1]
```

Out[23]:

	Returns	Volatility	cluster
BA UN Equity	0.254090	0.227732	1
JPM UN Equity	0.186449	0.244872	1
DIS UN Equity	0.173328	0.199785	1
EXR UN Equity	0.271375	0.216126	1
HD UN Equity	0.255554	0.190896	1
...	...	...	...
MXIM UW Equity	0.169215	0.252436	1
DRE UN Equity	0.181261	0.228075	1
ADS UN Equity	0.174277	0.251886	1
EQIX UW Equity	0.246245	0.256666	1
DLR UN Equity	0.164797	0.231108	1

102 rows × 3 columns

5) Conclusion

Interpretation: The investor can now easily identify the cluster containing a selection of stocks of the S&P500 universe with the highest historical average return-to-risk ratio (or the lowest). We have achieved a classification/grouping of a large stock universe into meaningful clusters.

The investor might find this information useful to support the stock selection and portfolio construction process.

Based on the above, investors would generally prefer to invest in stocks which are in cluster 1, because these have historically provided biggest return with least amount of uncertainty. Cluster 1 is followed by cluster 2, 4 and then 3 from attractiveness perspective.

Cluster 0 has a very low return to volatiliy ratio. Investing in stocks in this cluster may not be a good idea for a simple buy and hold strategy. However, such a cluster may be attractive for investors who benefit from stock price decrease or volatility.

This is a good example of how such an analysis can benefit UBS. We could use clustering to make tailored investment recommendations to to UBS clients or create indexes or funds (combinations of different stocks) based on cluster characteristics and then sell these to our customers.

Further research: It would be interesting to see if these stocks clusters that have formed above share futher characteristics beyond return and risk properties. For example, further analysis can be done, to see whether stocks clustering, as they do above, belong to a specific industry sector such as Health Care or Consumer Staples. Do they share a geographical footprint or something else.

The above analysis above was done using the full time horizon available to calculate the return and volatility statistics for each stock. It might be intersting to see what happens when we apply the cluster analysis to historical sub periods, thereby adding another dimension (3D).

It would be interesting to see how persistent the clusters are, i.e. how strongly individual stocks move between clusters and to what extent the clusters change entirely.

In [ ]: