

PART 01

The first problem was to identify trades and traders in the given trades which can be considered out of the norm when placing trades. This problem was tackled using an unsupervised learning technic called DBSCAN.

DBSCAN is a machine learning technic which is very suitable to identify the outliers. It has proven with even better results than most of the clustering techniques including **kmeans** . From the following figure 1 the ability for DBSCAN to identify the clusters correctly and thereby find the outliers is depicted vividly. This was one of the reasons for me to select this technique in tackling this problem.

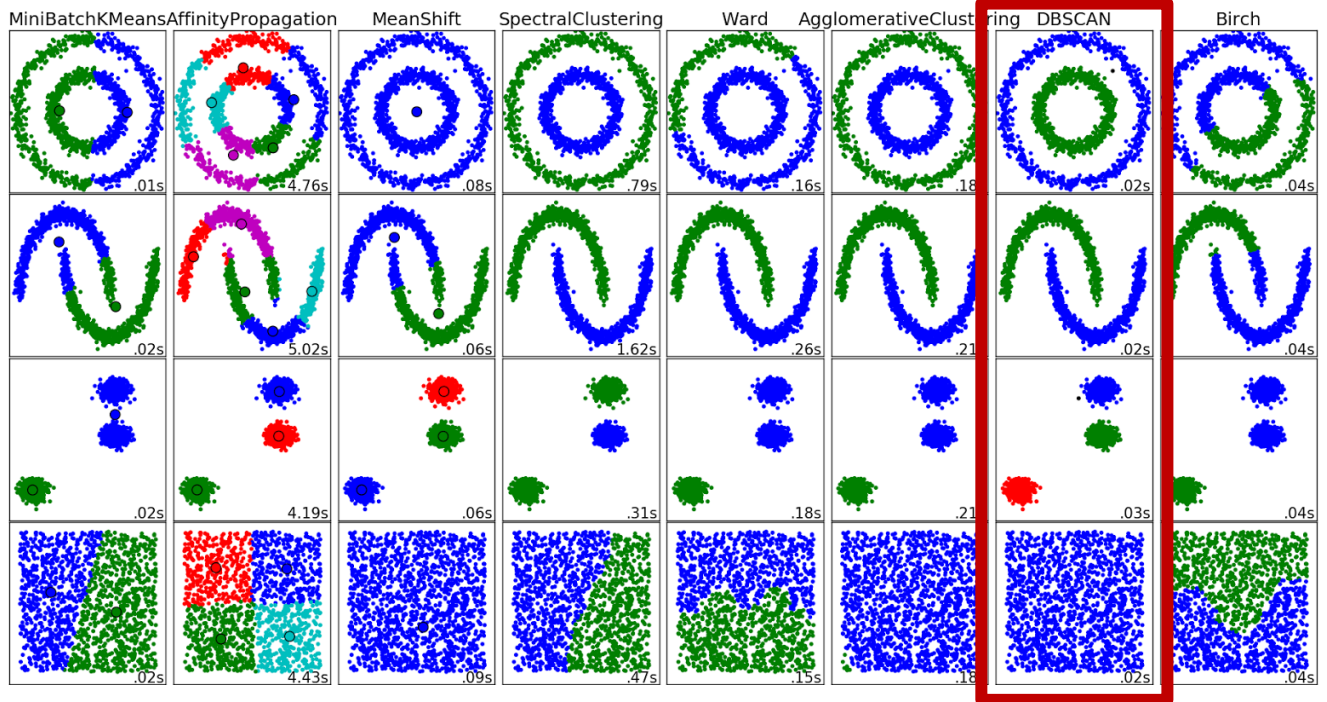


Figure 1: Comparison of DBSCAN on detecting clusters

Density-based spatial clustering of applications with noise (DBSCAN) works as follows. DBSCAN groups together points that are close to each other based on a distance measurement (usually Euclidean distance) and a minimum number of points. It also marks as outliers the points that are in low-density regions.

DBSCAN requires two parameters.

- **eps**: specifies how close points should be to each other to be considered a part of a cluster. It means that if the distance between two points is lower or equal to this value (eps), these points are considered neighbors.
- **minPoints**: the minimum number of points to form a dense region. For example, if we set the minPoints parameter as 5, then we need at least 5 points to form a dense region.

My Solution:

The provided data set had 8 columns meaning 8 attributes. As the problem definition says we needed only to consider the trade quantities and the traders. However this had to be limited to only one stock. So I selected the **stock with most number of trades** as the accuracy could be better. Also the brokers were taken from the **buy side as more information** is embedded in the buy trades.

Thus the Datafram I used only had the fields “Buy Broker” and the “Executed Qty” related to the stock number “ES0158252033”. Figure 2 shows a sample of data from the considered dataframe.

buy_target - DataFrame			
Index	Buy Broker ID	Executed Qty	
3	B204480	1	
4	A163878	1	
5	B204480	2	
6	B128778	6	
7	C439398190	12	
8	C424759231	18	
9	B128778	44	
10	B204480	75	
11	C11084986	14	
12	A8605026	64	

Figure 2: Dataframe under consideration

NOVELTY

Then the DBSCAN was applied to each broker separately and the outliers for each broker were calculated. However the **eps** and **min_samples** but as a novel approach I proposed two approaches in calculating those values. They are as follows.

1. Means square distance using KMeans algorithm
2. Standard deviation of the quantities.

Means square distance using KMeans algorithm:

In this method I mean square distance from each centroid of the kmeans algorithm clusters (1 or 2 clusters) and used it as the *eps*. The *min_samples* were considered to be one third of the total number of trades.

```
#Kmeans attribute calculation
#####
kmeans = KMeans(n_clusters = 2, init = 'k-means++', random_state = 40)
kmeans.fit(target)
sm = np.size(target, axis=0)*0.33
eps_ = np.sqrt(kmeans.inertia_/sm)
```

Standard deviation of quantities.

Here a constant multiplication of the standard deviation of quantities is used as the *eps* for the DBSCAN model and the division of the number of trades by the same constant is used as the *min_samples*.

```
#standard deviation attribute calculation
#####
n=1.5
eps_ = n*np.std(target[:,1])
sm = np.size(target, axis=0)/n
#####
```

Results

In the both of the above situations the results were quite similar. Also this approach gives a more meaning to setting values rather than doing it arbitrarily after trial and error. Interpretation of the results can be done very easily using the figure 3 and 4.

Here the red colour dots represent the outliers while the green colour dots represent the dense areas.

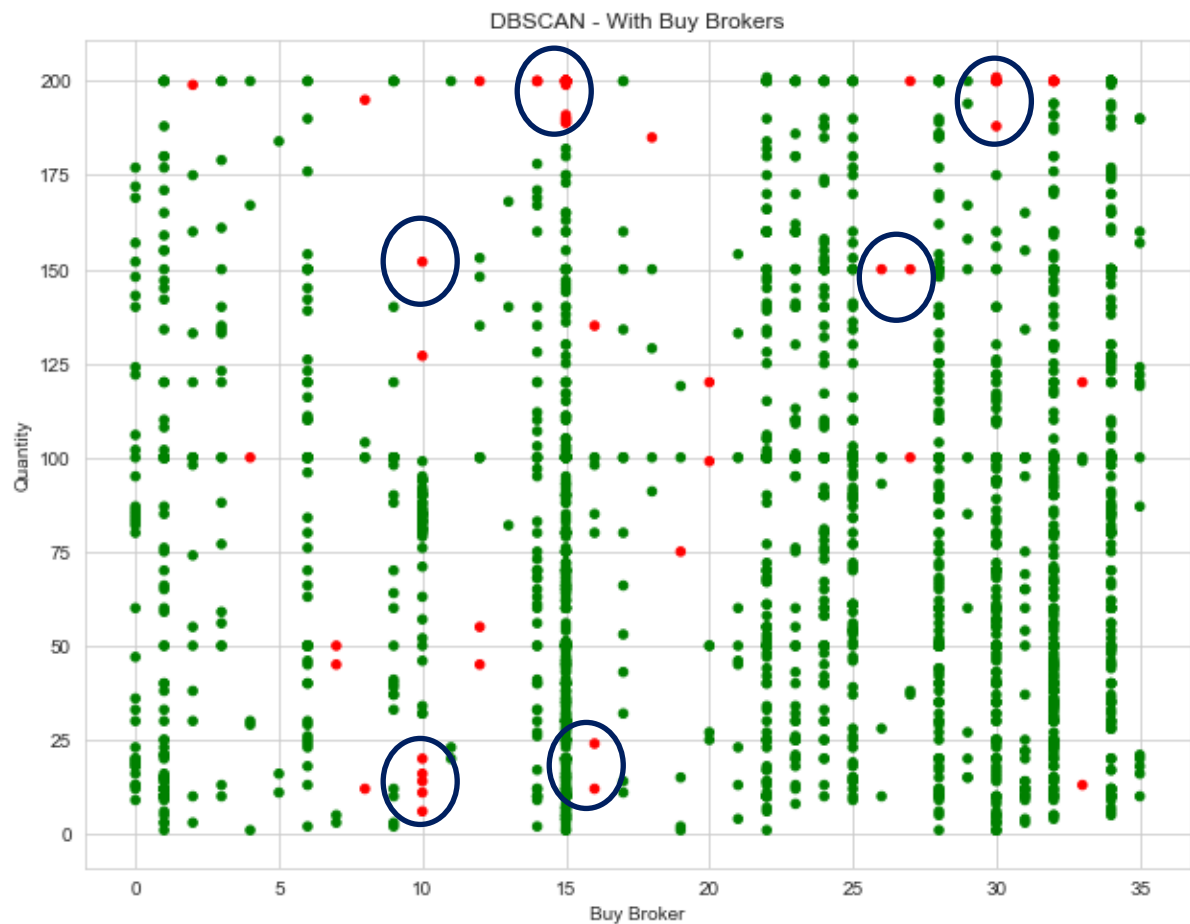


Figure 3: Results from KMeans approach where n_clusters =2 and multiplication factor = 0.315

In this KMeans approach we could clearly identify both high quantity trades and as well as small quantity trades which are out of the norm for each trader separately.

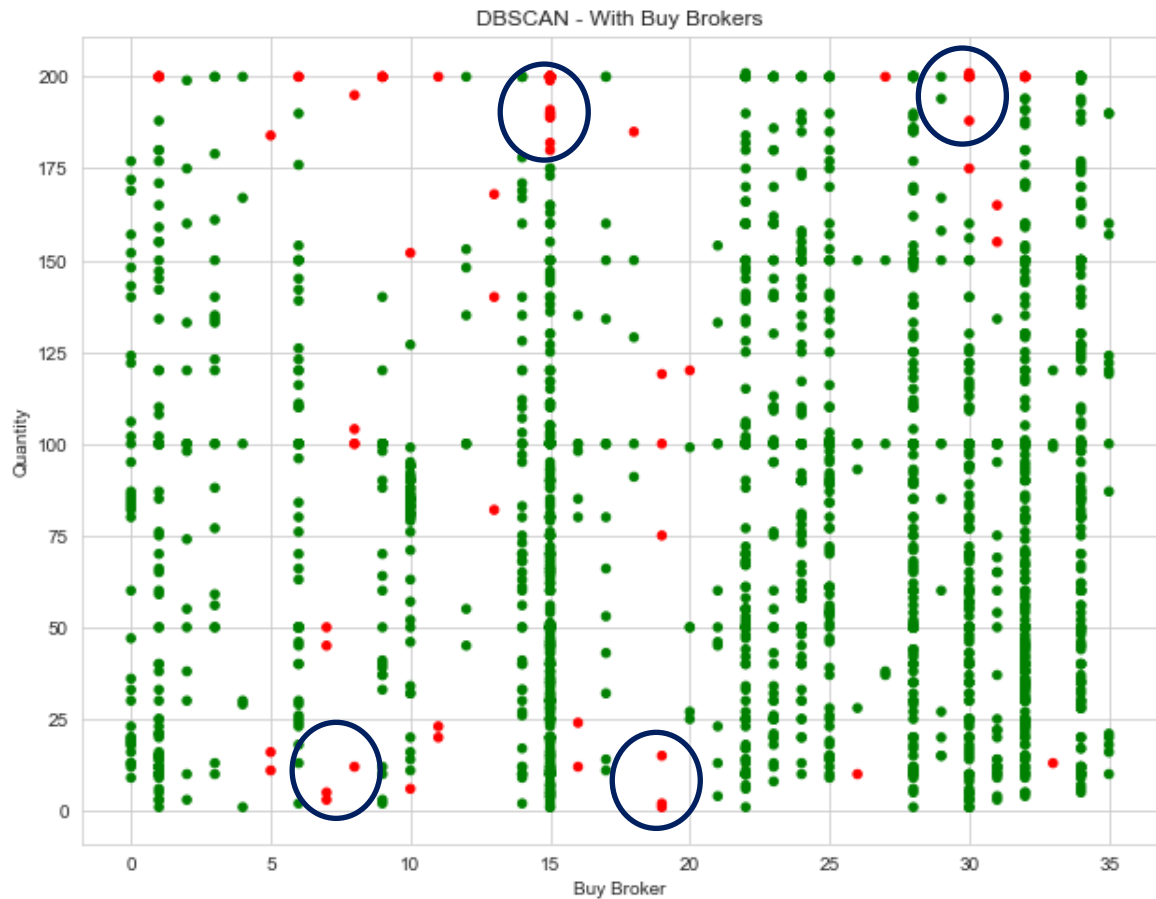


Figure 4: Results from standard deviation approach where the $n=1.45$

In this method the results are very similar except for some additional points which are not those much interpretable. Some of the brokers are completely marked as outlier brokers as they don't have enough trades to qualify as a cluster. But outliers are decently identified here as well.

CONCLUSION

The best solution can be considered as the **KMeans and DBSCAN** put together as an association to find outliers in the above dataset.

*PYTHON code for part 1 is **DBSCAN_Separate_OutlierDetection.py***

PART 02

The second part of the project actually made me think of so many ways to find a solution. But due to the time constraint finally I had to limit myself to only one approach which is not the best after all. However it can be used to get a rough understanding of the relationship between the brokers who work as a collusive trader group. Despite of that I tried on numerous other methods. But due to lack of time a complete implementation was not possible. If you request, I would be willing to send my attempts in a separate folder even though they are not fully complete. (email – thulithatheekshana.7@gmail.com)

APRIORI - ASSOCIATION RULE MINING

Apriori algorithm is used to identify relationship between items purchased by people in the markets. Here the algorithm considers three main parameters between different combinations of the same items. They are as follows.

$$\text{support}(\mathbf{M}) = \frac{\# \text{ user watchlists containing } \mathbf{M}}{\# \text{ user watchlists}}$$

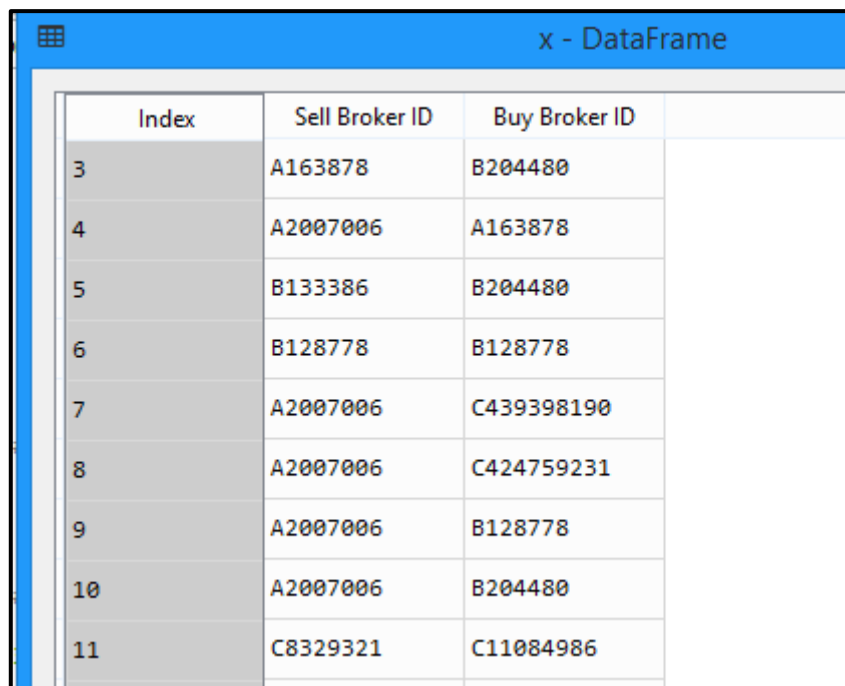
$$\text{confidence}(\mathbf{M}_1 \rightarrow \mathbf{M}_2) = \frac{\# \text{ user watchlists containing } \mathbf{M}_1 \text{ and } \mathbf{M}_2}{\# \text{ user watchlists containing } \mathbf{M}_1}$$

$$\text{lift}(\mathbf{M}_1 \rightarrow \mathbf{M}_2) = \frac{\text{confidence}(\mathbf{M}_1 \rightarrow \mathbf{M}_2)}{\text{support}(\mathbf{M}_2)}$$

So the dataset is reduced to only two columns as we would only be considering the buy and sell broker relationships.

MY SOLUTION

Here the expectation is to get the most related trader pairs that get together in selling and buying stocks. Here also I have limited my calculations for only one stock. But we wouldn't need a much higher "support" value as the threshold, as it is not related to our problem very much. However a rather higher "confidence" should be there to get the needed associations between the brokers. Thus a relative larger value for confidence was used.

The image shows a screenshot of a Jupyter Notebook interface. At the top, there is a blue header bar with a grid icon on the left and the text "x - DataFrame" on the right. Below this, a table is displayed with four columns: "Index", "Sell Broker ID", "Buy Broker ID", and an empty column. The table contains 11 rows of data, with the "Index" column ranging from 3 to 11. The "Sell Broker ID" and "Buy Broker ID" columns contain alphanumeric strings representing broker IDs.

Index	Sell Broker ID	Buy Broker ID	
3	A163878	B204480	
4	A2007006	A163878	
5	B133386	B204480	
6	B128778	B128778	
7	A2007006	C439398190	
8	A2007006	C424759231	
9	A2007006	B128778	
10	A2007006	B204480	
11	C8329321	C11084986	

Figure 5: Dataset used for the Arpiori algorithm

Implementation of the problem was rather simple. The library *Apyori* was used in python.

Results

The results can be very easily interpreted. Several instances of the algorithm with different confidence values can be used to get a clear understanding about the brokers.

With min_confidence = 0.2

```
Rule: A170820 -> A11174628
Support: 0.00101010101010101
Confidence: 0.39999999999999997
Lift: 8.336842105263157
=====
Rule: C129286 -> A11174628
Support: 0.00101010101010101
Confidence: 0.3333333333333333
Lift: 6.947368421052632
=====
Rule: A11288376 -> A550725
Support: 0.00101010101010101
Confidence: 0.39999999999999997
Lift: 3.52
=====
Rule: B228150 -> B8734110
Support: 0.00101010101010101
Confidence: 0.22222222222222224
Lift: 3.4920634920634925
=====
Rule: C11084986 -> C8489260
Support: 0.00101010101010101
Confidence: 0.2857142857142857
Lift: 5.096525096525096
=====
```

Here we can see that there are 5 trader pairs with 9 traders who had been selling and buying from each other. This could be considered as an associated collusive trader pairs doing buying and selling the stocks to each other to affect the price. However if we increase the min_confidence value we could obtain rather suspicious traders.

With min_confidence = 0.3

```
Rule: A170820 -> A11174628
Support: 0.00101010101010101
Confidence: 0.39999999999999997
Lift: 8.336842105263157
=====
Rule: C129286 -> A11174628
Support: 0.00101010101010101
Confidence: 0.3333333333333333
Lift: 6.947368421052632
=====
Rule: A11288376 -> A550725
Support: 0.00101010101010101
Confidence: 0.39999999999999997
Lift: 3.52
=====
```

As we can see in the above result the lift is very high in the first rule. This makes them a very suspicious trader group for doing such selling and buying actions. If we consider the second rule, we can observe the same trader from the rule 1 (**A11174628**) is also trading with another trader very often making it a suspicious behavior. We could be naïve and propose that traders **A170820**, **C129286**, **A11174628** as a collusive trader group.

But we should remember that this is a very primitive method of identifying collusive traders. The results can be slightly improved if we used a **time series anomaly detection algorithm such as Moving average or ARIMA before applying Apriori**. But there are even better approaches to identify stock price manipulations such as HMM and derivatives of HMM. Those approaches were studied but the time was a constraint for implementation.

PYTHON code for part 2 is `Apriori_CollusiveTraderGroups.py`

Bibliography

<https://www.edureka.co/blog/apriori-algorithm/>

<https://www.datacamp.com/community/tutorials/moving-averages-in-pandas>

<https://towardsdatascience.com/how-dbscan-works-and-why-should-i-use-it-443b4a191c80>

Python documentations