EE7206 MACHINE LEARNING Project Report

September 20, 2019

NAME : SANKALPA V.L.T.
REG No. : EG/2015/2757
E-mail : wltsankalpa@gmail.com
Contact No : 077-4588562

---

1 Q1

1.1 part 1 :- Identifing outlier trades based on Executed Price & Executed Qty using Hierarchical Clustering

An outlier is a data point that differs significantly from other observations. Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar and data points in different groups are more differs. Therefore, when clusters form outlier data points will be cluster in to an outlier clusters.

[0]:
```
# Importing the libraries

import matplotlib.pyplot as plt
import pandas as pd
import scipy.cluster.hierarchy as sch
from apyori import apriori
from sklearn.cluster import AgglomerativeClustering
# Upload Trade.csv file to google colab
from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>


Saving apyori.py to apyori (1).py
Saving Trades.csv to Trades (3).csv
```

1.1.1 Data preprocessing

[0]:
```
# Importing the trades data as pandas DataFrame
dataset = pd.read_csv('Trades.csv')
dataset
```

[0]:

	Trade Date	Executed Qty	...	Sell Broker ID	Buy Broker ID
0	01JUL2014:09:00:00	1	...	B11293128	B11293128
1	01JUL2014:09:00:00	1	...	A170820	A170820
2	01JUL2014:09:00:00	1	...	B169653	B169653
3	01JUL2014:09:00:00	1	...	A163878	B204480
4	01JUL2014:09:00:00	1	...	A2007006	A163878
5	01JUL2014:09:00:00	2	...	B133386	B204480
6	01JUL2014:09:00:00	6	...	B128778	B128778
7	01JUL2014:09:00:00	12	...	A2007006	C439398190
8	01JUL2014:09:00:00	18	...	A2007006	C424759231
9	01JUL2014:09:00:00	44	...	A2007006	B128778
10	01JUL2014:09:00:00	75	...	A2007006	B204480
11	01JUL2014:09:00:03	14	...	C8329321	C11084986
12	01JUL2014:09:00:03	64	...	C9324721	A8605026
13	01JUL2014:09:00:03	70	...	C11084986	A8605026
14	01JUL2014:09:00:03	136	...	C9324721	C11084986
15	01JUL2014:09:04:25	60	...	A8605026	A11174628
16	01JUL2014:09:09:03	450	...	C8329321	A2007006
17	01JUL2014:09:14:29	4	...	B128778	C439398190
18	01JUL2014:09:14:29	16	...	B128778	A11174628
19	01JUL2014:09:29:05	9	...	C9677800	A11174628
20	01JUL2014:09:32:38	3	...	C424759231	C439398190
21	01JUL2014:09:32:38	14	...	C424759231	C439398190
22	01JUL2014:09:32:38	133	...	C424759231	A128271
23	01JUL2014:09:36:15	200	...	C424759231	A163878
24	01JUL2014:09:43:08	13	...	C8329321	A11174628
25	01JUL2014:09:43:08	29	...	C8329321	A163878
26	01JUL2014:09:44:24	55	...	C439398190	A125250
27	01JUL2014:09:44:24	74	...	C9943570	A125250
28	01JUL2014:09:44:24	98	...	A11174628	A125250
29	01JUL2014:09:44:24	100	...	A8605026	A125250
...	...	...	...	...	...
1970	01JUL2014:17:28:57	91	...	C8329321	A8982441
1971	01JUL2014:17:28:57	100	...	C8329321	B128778
1972	01JUL2014:17:28:57	190	...	C8329321	B128778
1973	01JUL2014:17:29:19	152	...	A11174628	A8982441
1974	01JUL2014:17:29:32	83	...	B128778	A8982441
1975	01JUL2014:17:29:32	100	...	B128778	A8605026
1976	01JUL2014:17:29:32	133	...	B128778	B429816540
1977	01JUL2014:17:29:50	200	...	C156520	A8605026
1978	01JUL2014:17:29:52	32	...	A11174628	A8982441
1979	01JUL2014:17:29:52	46	...	A11174628	A8982441

```
1980  01JUL2014:17:29:52           63  ...       A11174628      A8982441
1981  01JUL2014:17:29:56          100  ...       A11174628     C11084986
1982  01JUL2014:17:35:00            5  ...        A2007006       A170820
1983  01JUL2014:17:35:00           10  ...         B169653       B128778
1984  01JUL2014:17:35:00           27  ...         B169653       B128778
1985  01JUL2014:17:35:00           30  ...         B169653       B128778
1986  01JUL2014:17:35:00           46  ...         B169653      C8329321
1987  01JUL2014:17:35:00           50  ...        A2007006      C8329321
1988  01JUL2014:17:35:00           51  ...         B133386       B128778
1989  01JUL2014:17:35:00           60  ...         B169653      C9324721
1990  01JUL2014:17:35:00          100  ...         C156520       C156520
1991  01JUL2014:17:35:00          100  ...         B169653       B128778
1992  01JUL2014:17:35:00          100  ...      C424759231       B128778
1993  01JUL2014:17:35:00          105  ...         B169653       B128778
1994  01JUL2014:17:35:00          140  ...         A128271       A128271
1995  01JUL2014:17:35:00          150  ...         B128778       B128778
1996  01JUL2014:17:35:00          191  ...         B128778       B128778
1997  01JUL2014:17:35:00          200  ...         C156520       C156520
1998  01JUL2014:17:35:00          200  ...         B169653       B128778
1999  01JUL2014:17:35:10          179  ...         A128271       A128271

[2000 rows x 8 columns]
```

[0]:
```
# Filtering the trades of stock ES0158252033 and create a new DataFrame
Trades = dataset.loc[dataset.Stock=='ES0158252033', :]
Trades
```

[0]:
```
              Trade Date  Executed Qty  ...  Sell Broker ID Buy Broker ID
3     01JUL2014:09:00:00             1  ...         A163878       B204480
4     01JUL2014:09:00:00             1  ...        A2007006       A163878
5     01JUL2014:09:00:00             2  ...         B133386       B204480
6     01JUL2014:09:00:00             6  ...         B128778       B128778
7     01JUL2014:09:00:00            12  ...        A2007006    C439398190
8     01JUL2014:09:00:00            18  ...        A2007006    C424759231
9     01JUL2014:09:00:00            44  ...        A2007006       B128778
10    01JUL2014:09:00:00            75  ...        A2007006       B204480
11    01JUL2014:09:00:03            14  ...        C8329321     C11084986
12    01JUL2014:09:00:03            64  ...        C9324721      A8605026
13    01JUL2014:09:00:03            70  ...       C11084986      A8605026
14    01JUL2014:09:00:03           136  ...        C9324721     C11084986
15    01JUL2014:09:04:25            60  ...        A8605026     A11174628
17    01JUL2014:09:14:29             4  ...         B128778    C439398190
18    01JUL2014:09:14:29            16  ...         B128778     A11174628
19    01JUL2014:09:29:05             9  ...        C9677800     A11174628
20    01JUL2014:09:32:38             3  ...      C424759231    C439398190
21    01JUL2014:09:32:38            14  ...      C424759231    C439398190
22    01JUL2014:09:32:38           133  ...      C424759231       A128271
23    01JUL2014:09:36:15           200  ...      C424759231       A163878
```

```
24    01JUL2014:09:43:08           13  ...       C8329321     A11174628
25    01JUL2014:09:43:08           29  ...       C8329321       A163878
26    01JUL2014:09:44:24           55  ...     C439398190       A125250
27    01JUL2014:09:44:24           74  ...       C9943570       A125250
28    01JUL2014:09:44:24           98  ...      A11174628       A125250
29    01JUL2014:09:44:24          100  ...       A8605026       A125250
30    01JUL2014:09:44:24          133  ...        A128271       A125250
31    01JUL2014:09:44:24          199  ...     C439398190       A125250
32    01JUL2014:09:49:47           10  ...       B128778    C439398190
38    01JUL2014:10:03:00           11  ...       A8605026     C8329321
...                  ...          ...  ...            ...          ...
1969  01JUL2014:17:28:57           82  ...       C8329321     A8982441
1970  01JUL2014:17:28:57           91  ...       C8329321     A8982441
1971  01JUL2014:17:28:57          100  ...       C8329321      B128778
1972  01JUL2014:17:28:57          190  ...       C8329321      B128778
1973  01JUL2014:17:29:19          152  ...      A11174628     A8982441
1974  01JUL2014:17:29:32           83  ...       B128778     A8982441
1975  01JUL2014:17:29:32          100  ...       B128778     A8605026
1976  01JUL2014:17:29:32          133  ...       B128778    B429816540
1977  01JUL2014:17:29:50          200  ...       C156520     A8605026
1978  01JUL2014:17:29:52           32  ...      A11174628     A8982441
1979  01JUL2014:17:29:52           46  ...      A11174628     A8982441
1980  01JUL2014:17:29:52           63  ...      A11174628     A8982441
1981  01JUL2014:17:29:56          100  ...      A11174628    C11084986
1983  01JUL2014:17:35:00           10  ...       B169653      B128778
1984  01JUL2014:17:35:00           27  ...       B169653      B128778
1985  01JUL2014:17:35:00           30  ...       B169653      B128778
1986  01JUL2014:17:35:00           46  ...       B169653     C8329321
1987  01JUL2014:17:35:00           50  ...      A2007006     C8329321
1988  01JUL2014:17:35:00           51  ...       B133386      B128778
1989  01JUL2014:17:35:00           60  ...       B169653     C9324721
1990  01JUL2014:17:35:00          100  ...       C156520      C156520
1991  01JUL2014:17:35:00          100  ...       B169653      B128778
1992  01JUL2014:17:35:00          100  ...     C424759231      B128778
1993  01JUL2014:17:35:00          105  ...       B169653      B128778
1994  01JUL2014:17:35:00          140  ...        A128271      A128271
1995  01JUL2014:17:35:00          150  ...       B128778      B128778
1996  01JUL2014:17:35:00          191  ...       B128778      B128778
1997  01JUL2014:17:35:00          200  ...       C156520      C156520
1998  01JUL2014:17:35:00          200  ...       B169653      B128778
1999  01JUL2014:17:35:10          179  ...        A128271      A128271

[1980 rows x 8 columns]
```

[0]:
```
# Add indexing column to Dataframe (Lets consider this index represent the␣
 ↪Trade Date)
Trades['Index'] = range(1, len(Trades) + 1)
```

```
Trades
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

[0]:

	Trade Date	Executed Qty	...	Buy Broker ID	Index
3	01JUL2014:09:00:00	1	...	B204480	1
4	01JUL2014:09:00:00	1	...	A163878	2
5	01JUL2014:09:00:00	2	...	B204480	3
6	01JUL2014:09:00:00	6	...	B128778	4
7	01JUL2014:09:00:00	12	...	C439398190	5
8	01JUL2014:09:00:00	18	...	C424759231	6
9	01JUL2014:09:00:00	44	...	B128778	7
10	01JUL2014:09:00:00	75	...	B204480	8
11	01JUL2014:09:00:03	14	...	C11084986	9
12	01JUL2014:09:00:03	64	...	A8605026	10
13	01JUL2014:09:00:03	70	...	A8605026	11
14	01JUL2014:09:00:03	136	...	C11084986	12
15	01JUL2014:09:04:25	60	...	A11174628	13
17	01JUL2014:09:14:29	4	...	C439398190	14
18	01JUL2014:09:14:29	16	...	A11174628	15
19	01JUL2014:09:29:05	9	...	A11174628	16
20	01JUL2014:09:32:38	3	...	C439398190	17
21	01JUL2014:09:32:38	14	...	C439398190	18
22	01JUL2014:09:32:38	133	...	A128271	19
23	01JUL2014:09:36:15	200	...	A163878	20
24	01JUL2014:09:43:08	13	...	A11174628	21
25	01JUL2014:09:43:08	29	...	A163878	22
26	01JUL2014:09:44:24	55	...	A125250	23
27	01JUL2014:09:44:24	74	...	A125250	24
28	01JUL2014:09:44:24	98	...	A125250	25
29	01JUL2014:09:44:24	100	...	A125250	26
30	01JUL2014:09:44:24	133	...	A125250	27
31	01JUL2014:09:44:24	199	...	A125250	28
32	01JUL2014:09:49:47	10	...	C439398190	29
38	01JUL2014:10:03:00	11	...	C8329321	30
...	...	...	...	...	...
1969	01JUL2014:17:28:57	82	...	A8982441	1951
1970	01JUL2014:17:28:57	91	...	A8982441	1952
1971	01JUL2014:17:28:57	100	...	B128778	1953
1972	01JUL2014:17:28:57	190	...	B128778	1954

```
1973  01JUL2014:17:29:19          152  ...       A8982441  1955
1974  01JUL2014:17:29:32           83  ...       A8982441  1956
1975  01JUL2014:17:29:32          100  ...       A8605026  1957
1976  01JUL2014:17:29:32          133  ...     B429816540  1958
1977  01JUL2014:17:29:50          200  ...       A8605026  1959
1978  01JUL2014:17:29:52           32  ...       A8982441  1960
1979  01JUL2014:17:29:52           46  ...       A8982441  1961
1980  01JUL2014:17:29:52           63  ...       A8982441  1962
1981  01JUL2014:17:29:56          100  ...      C11084986  1963
1983  01JUL2014:17:35:00           10  ...        B128778  1964
1984  01JUL2014:17:35:00           27  ...        B128778  1965
1985  01JUL2014:17:35:00           30  ...        B128778  1966
1986  01JUL2014:17:35:00           46  ...       C8329321  1967
1987  01JUL2014:17:35:00           50  ...       C8329321  1968
1988  01JUL2014:17:35:00           51  ...        B128778  1969
1989  01JUL2014:17:35:00           60  ...       C9324721  1970
1990  01JUL2014:17:35:00          100  ...        C156520  1971
1991  01JUL2014:17:35:00          100  ...        B128778  1972
1992  01JUL2014:17:35:00          100  ...        B128778  1973
1993  01JUL2014:17:35:00          105  ...        B128778  1974
1994  01JUL2014:17:35:00          140  ...        A128271  1975
1995  01JUL2014:17:35:00          150  ...        B128778  1976
1996  01JUL2014:17:35:00          191  ...        B128778  1977
1997  01JUL2014:17:35:00          200  ...        C156520  1978
1998  01JUL2014:17:35:00          200  ...        B128778  1979
1999  01JUL2014:17:35:10          179  ...        A128271  1980

[1980 rows x 9 columns]
```

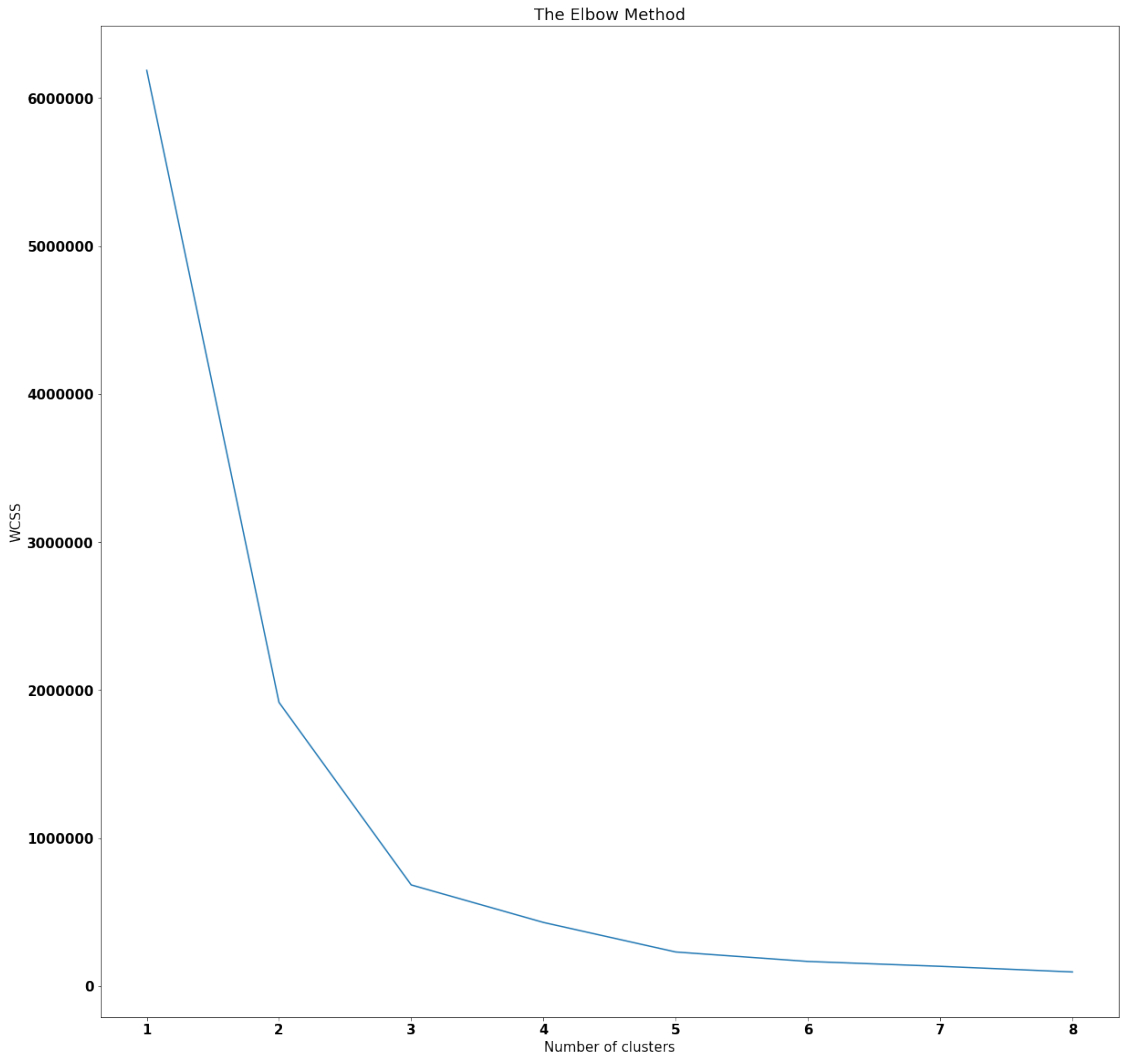
[0]:
```
# Create NumPy array with Execute Qty and Execute Price of Trades
X = Trades.iloc[:, [ 1, 2]].values
X
```

[0]:
```
array([[  1.  ,  19.71],
       [  1.  ,  19.71],
       [  2.  ,  19.71],
       ...,
       [200.  ,  10.71],
       [200.  ,  10.71],
       [179.  ,  10.71]])
```

1.1.2 Using the elbow method to find the optimal number of clusters suitable to identify outliners

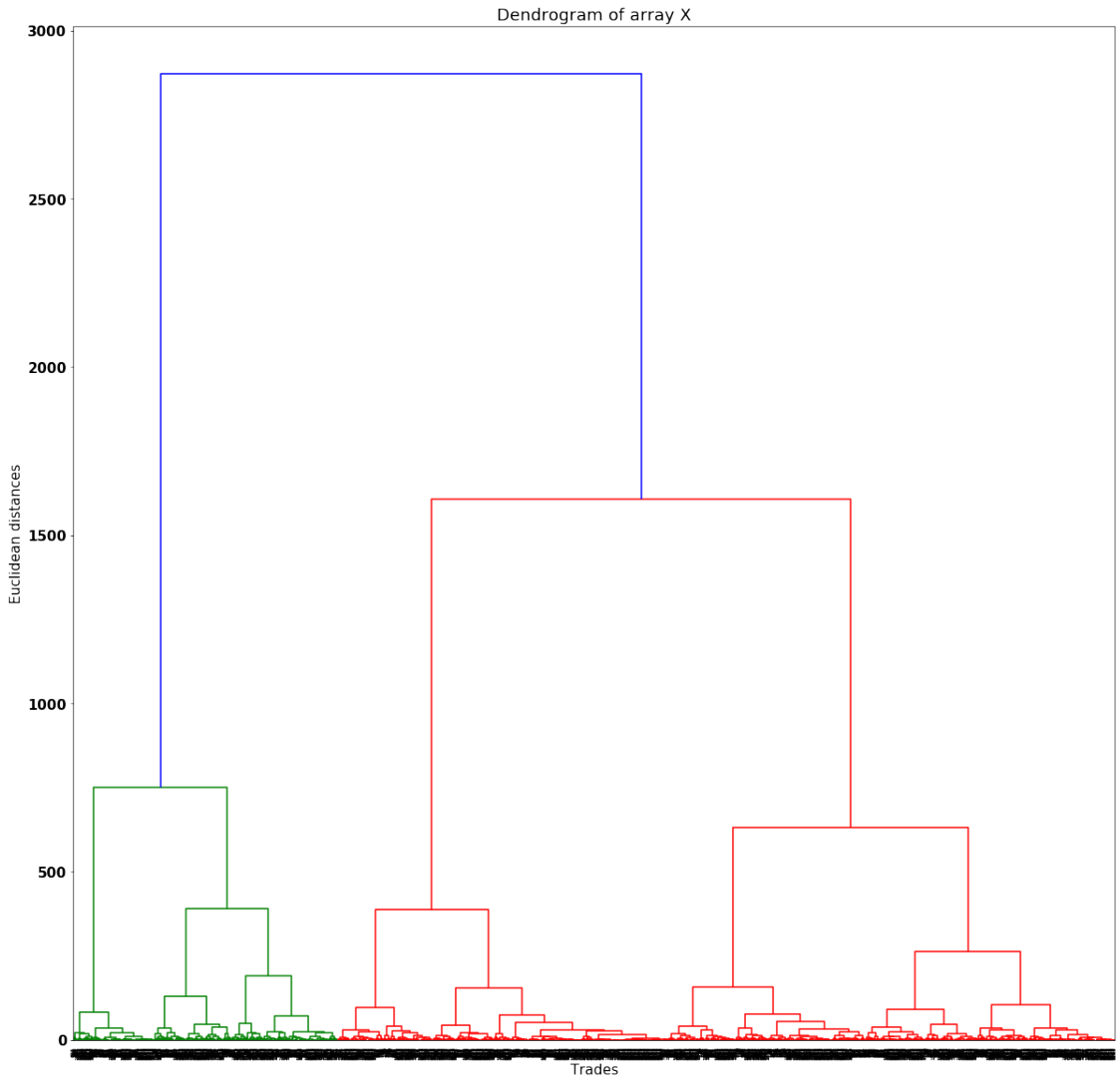
The elbow method plot graph between the number of clusters and **Within-Cluster-Sum-of-Squares (WCSS)**. When the gradient of graph become constant corresponding number of clusters can be consider as the optimal number of clusters.

[0]:
```
# change the font siXe on a matplotlib plot
font = {'family' : 'normal','weight' : 'bold','size'   : 15}
plt.rc('font', **font)

# change the graph siXe of a matplotlib plot
plt.rcParams['figure.figsize'] = (20, 20)

from sklearn.cluster import KMeans
wcss = []
for i in range(1, 9):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 9), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



According to above graph we can identify optimal number of clusters as five for this problem. ### Using the dendrogram to find the optimal number of clusters suitable to identify outliners Since the main point of Hierarchical Clustering is to make the dendrogram, because dendrogram contain the memory of Hierarchical Clustering algorithm, then work your way down to see the dierent combinations of clusters until having a single large cluter. Therefore in Hierarchical Clustering to double check the optimal number of clusters its possible to use **dendrogram**.The dendrogram itself that allows to nd the best clustering conguration.

[0]:
```
# plot dendrogram for X array
dendrogram_X = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram of array X')
plt.xlabel('Trades')
plt.ylabel('Euclidean distances')
plt.show()
```



by drawing line across 500 euclidean distance it possible to determine the optimal number of clusters as five clusters

1.1.3 Apply Hierarchical Clustering

[0]:
```
# Fitting Hierarchical Clustering to the array X
hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage =␣
 ↪'ward')
# Create Array with clusters
X_hc = hc.fit_predict(X)
X_hc
```

[0]:
```
array([4, 4, 4, ..., 3, 3, 0])
```

[0]:
```
# Add a X_hc array as Cluster column to a Trades dataframe
Trades['Cluster'] = X_hc
Trades
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

[0]:

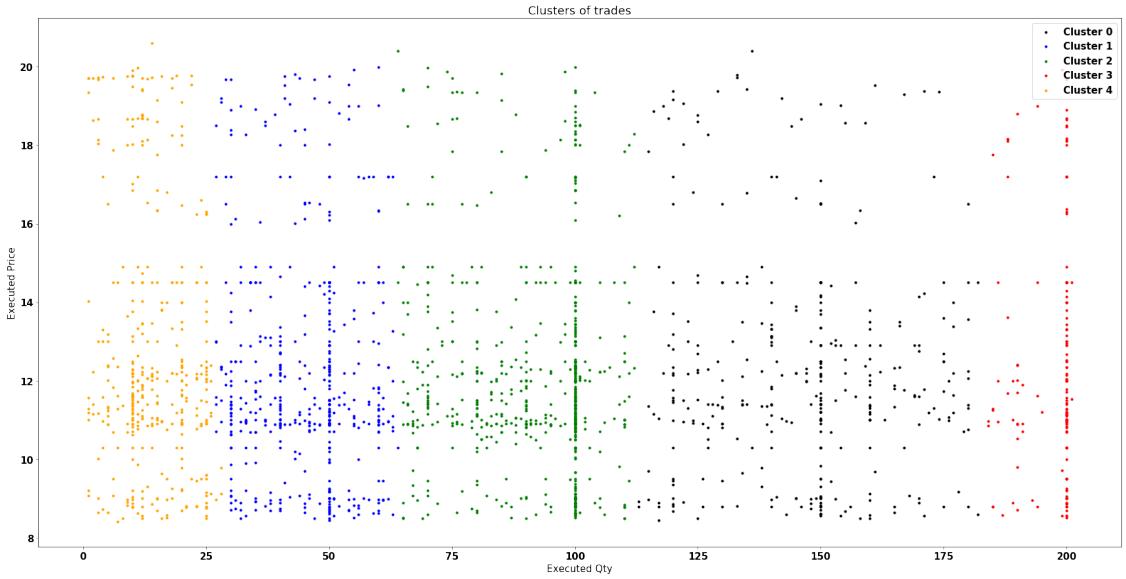
	Trade Date	Executed Qty	...	Index	Cluster
3	01JUL2014:09:00:00	1	...	1	4
4	01JUL2014:09:00:00	1	...	2	4
5	01JUL2014:09:00:00	2	...	3	4
6	01JUL2014:09:00:00	6	...	4	4
7	01JUL2014:09:00:00	12	...	5	4
8	01JUL2014:09:00:00	18	...	6	4
9	01JUL2014:09:00:00	44	...	7	1
10	01JUL2014:09:00:00	75	...	8	2
11	01JUL2014:09:00:03	14	...	9	4
12	01JUL2014:09:00:03	64	...	10	2
13	01JUL2014:09:00:03	70	...	11	2
14	01JUL2014:09:00:03	136	...	12	0
15	01JUL2014:09:04:25	60	...	13	1
17	01JUL2014:09:14:29	4	...	14	4
18	01JUL2014:09:14:29	16	...	15	4
19	01JUL2014:09:29:05	9	...	16	4
20	01JUL2014:09:32:38	3	...	17	4
21	01JUL2014:09:32:38	14	...	18	4
22	01JUL2014:09:32:38	133	...	19	0
23	01JUL2014:09:36:15	200	...	20	3
24	01JUL2014:09:43:08	13	...	21	4
25	01JUL2014:09:43:08	29	...	22	1
26	01JUL2014:09:44:24	55	...	23	1
27	01JUL2014:09:44:24	74	...	24	2
28	01JUL2014:09:44:24	98	...	25	2
29	01JUL2014:09:44:24	100	...	26	2
30	01JUL2014:09:44:24	133	...	27	0
31	01JUL2014:09:44:24	199	...	28	3
32	01JUL2014:09:49:47	10	...	29	4
38	01JUL2014:10:03:00	11	...	30	4
...	...	...	...	...	...
1969	01JUL2014:17:28:57	82	...	1951	2
1970	01JUL2014:17:28:57	91	...	1952	2

```
1971  01JUL2014:17:28:57          100  ...   1953       2
1972  01JUL2014:17:28:57          190  ...   1954       3
1973  01JUL2014:17:29:19          152  ...   1955       0
1974  01JUL2014:17:29:32           83  ...   1956       2
1975  01JUL2014:17:29:32          100  ...   1957       2
1976  01JUL2014:17:29:32          133  ...   1958       0
1977  01JUL2014:17:29:50          200  ...   1959       3
1978  01JUL2014:17:29:52           32  ...   1960       1
1979  01JUL2014:17:29:52           46  ...   1961       1
1980  01JUL2014:17:29:52           63  ...   1962       1
1981  01JUL2014:17:29:56          100  ...   1963       2
1983  01JUL2014:17:35:00           10  ...   1964       4
1984  01JUL2014:17:35:00           27  ...   1965       1
1985  01JUL2014:17:35:00           30  ...   1966       1
1986  01JUL2014:17:35:00           46  ...   1967       1
1987  01JUL2014:17:35:00           50  ...   1968       1
1988  01JUL2014:17:35:00           51  ...   1969       1
1989  01JUL2014:17:35:00           60  ...   1970       1
1990  01JUL2014:17:35:00          100  ...   1971       2
1991  01JUL2014:17:35:00          100  ...   1972       2
1992  01JUL2014:17:35:00          100  ...   1973       2
1993  01JUL2014:17:35:00          105  ...   1974       2
1994  01JUL2014:17:35:00          140  ...   1975       0
1995  01JUL2014:17:35:00          150  ...   1976       0
1996  01JUL2014:17:35:00          191  ...   1977       3
1997  01JUL2014:17:35:00          200  ...   1978       3
1998  01JUL2014:17:35:00          200  ...   1979       3
1999  01JUL2014:17:35:10          179  ...   1980       0

[1980 rows x 10 columns]
```

1.1.4 Visualising the clusters

[0]:
```
# change the graph size of a matplotlib plot
plt.rcParams['figure.figsize'] = (30, 15)
# Visualising the clusters
plt.scatter(X[X_hc == 0, 0], X[X_hc == 0, 1], s = 10, c = 'black', label =␣
 ↪'Cluster 0')
plt.scatter(X[X_hc == 1, 0], X[X_hc == 1, 1], s = 10, c = 'blue', label =␣
 ↪'Cluster 1')
plt.scatter(X[X_hc == 2, 0], X[X_hc == 2, 1], s = 10, c = 'green', label =␣
 ↪'Cluster 2')
plt.scatter(X[X_hc == 3, 0], X[X_hc == 3, 1], s = 10, c = 'red', label =␣
 ↪'Cluster 3')
plt.scatter(X[X_hc == 4, 0], X[X_hc == 4, 1], s = 10, c = 'orange', label =␣
 ↪'Cluster 4')
```

```
plt.title('Clusters of trades')
plt.xlabel('Executed Qty')
plt.ylabel('Executed Price')
plt.legend()
plt.show()
```



By observing cluster visualization scatter plot, it's possible to notice that cluster 3 (brown colour in scatter plot) has lowest data-point count comparing with other clusters. ### Calculating statistical information of clusters Use group by pandas aggregate to calculate statistical information of clusters based on Executed Qty, Executed price and index (this index represent the Trade date)

[0]:
```
# Calculate statistical information of clusters based on Executed Qty of trades
Cluster_Statistics_Qty = Trades.groupby("Cluster")['Executed Qty'].describe()
Cluster_Statistics_Qty
```

[0]:

Cluster	count	mean	std	min	25%	50%	75%	max
0	349.0	145.919771	17.744495	113.0	130.0	150.0	159.0	182.0
1	474.0	45.008439	10.001053	27.0	36.0	47.0	50.0	64.0
2	628.0	91.237261	12.464409	64.0	81.0	100.0	100.0	112.0
3	154.0	197.201299	5.036312	184.0	196.0	200.0	200.0	201.0
4	375.0	14.141333	6.815661	1.0	10.0	13.0	20.0	28.0

When observing above cluster statistical information of executed Qty of trades. All five clusters contain trades that has different statistical parameters. Especially cluster its possible to observe,

1. Trades which has executed Qty between **1 to 28** clustered to cluster **4**
2. Trades which has executed Qty between **27 to 64** clustered to cluster **1**

3. Trades which has executed Qty between **64 to 112** clustered to cluster **2**
4. Trades which has executed Qty between **113 to 182** clustered to cluster **0**
5. Trades which has executed Qty between **183 to 201** clustered to cluster **3**

Therefore, it possible to come to conclusion that, executed Qty do contribute to form an outlier trades.

[0]:
```
# Calculate statistical information of clusters based on Executed Price of␣
 ↪trades
Cluster_Statistics_Price = Trades.groupby("Cluster")['Executed Price'].
 ↪describe()
Cluster_Statistics_Price
```

[0]:

Cluster	count	mean	std	min	25%	50%	75%	max
0	349.0	12.273496	2.785368	8.45	10.71	11.77	13.4000	20.40
1	474.0	12.269156	2.840596	8.45	10.71	11.45	13.5075	19.99
2	628.0	12.174220	2.628834	8.50	10.82	11.55	13.0000	20.40
3	154.0	12.382532	2.946642	8.51	10.71	11.55	14.0000	19.92
4	375.0	12.752480	3.198436	8.41	10.90	11.89	14.2750	20.60

When observing above cluster statistical information of executed price of trades. All five clusters contain trades that has approximately same statistical parameters,

1. mean 12.3
2. standard deviation 2.8
3. min 8.45
4. 25% 10.8
5. 50% 11.5
6. 75 13.5
7. max 20

Therefore, it possible to come to conclusion that, executed price doesn't contribute to form an outlier trade since frequent rapid price changes not possible to observed in give dataset.

[0]:
```
# Calculate statistical information of clusters based on Index of trades
Cluster_Statistics_index = Trades.groupby("Cluster")['Index'].describe()
Cluster_Statistics_index
```

[0]:

Cluster	count	mean	std	min	25%	50%	75%	max
0	349.0	948.501433	530.935704	12.0	509.00	939.0	1378.00	1980.0
1	474.0	990.428270	573.933960	7.0	496.50	982.5	1484.75	1970.0
2	628.0	1040.581210	573.512601	8.0	531.75	1057.5	1542.75	1974.0
3	154.0	960.935065	572.591103	20.0	492.00	930.0	1401.50	1979.0
4	375.0	957.949333	598.225536	1.0	442.50	969.0	1501.50	1964.0

When observing above cluster statistical information of trade indexes. All five clusters contain trades spead all over full index range form 0 to 1980. Especially by looking at minimum and maximum index of each cluster, it is possible to conclude that each cluster has data points in all

over full index range. Therefore, according to above cluster statistic of executed Qty if we sort all five clusters according to count of data points within the cluster as follow,

Cluster 3 – 154 data points (7.78%)
Cluster 0 – 349 data points (17.63%)
Cluster 4 – 375 data points (18.94%)
Cluster 1 – 474 data points (23.94%)
Cluster 2 – 628 data points (31.72%)

Therefore, the cluster that has lowest count of data points (**7.78 % of all data points**) makes **outlier cluster, which is cluster 3**. Finally its feasible to concluded all the trade that enrol executed qty between 184 and 201 are outlier trades.

1.1.5 Outlier Trades

[0]:
```
# Create outlier Trdaes Dataframe

Cluster = [3]
Outlier_Trades = Trades[Trades.Cluster.isin(Cluster)]
Outlier_Trades
```

[0]:

	Trade Date	Executed Qty	...	Index	Cluster
23	01JUL2014:09:36:15	200	...	20	3
31	01JUL2014:09:44:24	199	...	28	3
90	01JUL2014:10:37:25	194	...	82	3
97	01JUL2014:10:39:49	200	...	89	3
105	01JUL2014:10:40:07	190	...	97	3
109	01JUL2014:10:41:03	200	...	101	3
129	01JUL2014:10:42:15	200	...	121	3
138	01JUL2014:10:42:33	200	...	130	3
154	01JUL2014:10:43:49	200	...	146	3
155	01JUL2014:10:43:49	200	...	147	3
166	01JUL2014:10:44:49	188	...	158	3
167	01JUL2014:10:44:49	200	...	159	3
168	01JUL2014:10:44:49	200	...	160	3
171	01JUL2014:10:45:38	188	...	163	3
174	01JUL2014:10:45:40	188	...	166	3
176	01JUL2014:10:46:33	200	...	168	3
191	01JUL2014:10:48:32	185	...	183	3
216	01JUL2014:10:58:00	188	...	208	3
217	01JUL2014:10:58:00	200	...	209	3
218	01JUL2014:10:58:00	200	...	210	3
219	01JUL2014:10:58:00	200	...	211	3
241	01JUL2014:10:59:06	200	...	233	3
245	01JUL2014:10:59:12	200	...	237	3
261	01JUL2014:11:07:02	200	...	253	3
370	01JUL2014:12:16:00	200	...	359	3
371	01JUL2014:12:16:00	200	...	360	3
372	01JUL2014:12:16:00	200	...	361	3
373	01JUL2014:12:16:00	200	...	362	3

```
374   01JUL2014:12:16:00          200  ...    363       3
375   01JUL2014:12:16:00          200  ...    364       3
...                  ...          ...  ...    ...     ...
1646  01JUL2014:16:40:38          200  ...   1629       3
1655  01JUL2014:16:41:09          200  ...   1638       3
1665  01JUL2014:16:41:46          200  ...   1648       3
1673  01JUL2014:16:46:05          195  ...   1656       3
1688  01JUL2014:16:57:00          200  ...   1671       3
1689  01JUL2014:16:57:22          185  ...   1672       3
1700  01JUL2014:17:01:00          200  ...   1683       3
1718  01JUL2014:17:01:17          200  ...   1701       3
1731  01JUL2014:17:02:22          191  ...   1714       3
1733  01JUL2014:17:03:03          184  ...   1716       3
1760  01JUL2014:17:05:58          184  ...   1743       3
1768  01JUL2014:17:07:48          200  ...   1751       3
1769  01JUL2014:17:07:54          189  ...   1752       3
1796  01JUL2014:17:10:58          200  ...   1779       3
1815  01JUL2014:17:12:02          200  ...   1798       3
1825  01JUL2014:17:12:40          200  ...   1808       3
1828  01JUL2014:17:12:47          200  ...   1811       3
1839  01JUL2014:17:14:33          200  ...   1822       3
1866  01JUL2014:17:18:02          200  ...   1849       3
1874  01JUL2014:17:19:18          186  ...   1857       3
1899  01JUL2014:17:22:55          190  ...   1882       3
1904  01JUL2014:17:23:22          200  ...   1887       3
1922  01JUL2014:17:25:02          200  ...   1904       3
1927  01JUL2014:17:25:16          200  ...   1909       3
1936  01JUL2014:17:26:11          200  ...   1918       3
1972  01JUL2014:17:28:57          190  ...   1954       3
1977  01JUL2014:17:29:50          200  ...   1959       3
1996  01JUL2014:17:35:00          191  ...   1977       3
1997  01JUL2014:17:35:00          200  ...   1978       3
1998  01JUL2014:17:35:00          200  ...   1979       3

[154 rows x 10 columns]
```

1.2 part 2 :- Identifing outlier traders based on sum of Executed Qty using Hierarchical Clustering

Form above part 1 it's possible to conclude that outlier trades form due to executed quantity of trades (the trade that enrol executed qty between 184 and 201) Therefore its possible to identify the outlier traders using hierarchical clustering based on the sum of executed qty for each buyer. ### Data preprocessing with pandas

[0]:

```
# Create a new DataFrame contain sum of Executed Qty for each traders who␣
 ↪brought stock ES0158252033
Traders = dataset[dataset.Stock=="ES0158252033"].groupby("Buy Broker␣
 ↪ID")['Executed Qty'].sum().reset_index()
Traders
```

[0]:

	Buy Broker ID	Executed Qty
0	A11174628	2900
1	A11288376	6588
2	A125250	1445
3	A128271	2278
4	A163878	527
5	A170820	211
6	A2007006	4928
7	A203841	103
8	A550725	511
9	A8605026	2834
10	A8982441	4068
11	A95703	243
12	B11293128	1036
13	B1139295	390
14	B124251	4526
15	B128778	27694
16	B133386	634
17	B144453	1443
18	B169653	655
19	B204480	312
20	B228150	371
21	B405882786	728
22	B429816540	10907
23	B433077165	6570
24	B8734110	10080
25	C11084986	8904
26	C129286	481
27	C133903	525
28	C156520	14446
29	C1840321	1291
30	C424759231	10118
31	C439398190	1924
32	C8329321	15612
33	C8390656	332
34	C9324721	18160
35	C9943570	1454

[0]:
```
# Add indexing column to Traders Dataframe
Traders['Index'] = range(1, len(Traders) + 1)
Traders
```

[0]:

	Buy Broker ID	Executed Qty	Index
0	A11174628	2900	1
1	A11288376	6588	2
2	A125250	1445	3
3	A128271	2278	4
4	A163878	527	5
5	A170820	211	6
6	A2007006	4928	7
7	A203841	103	8
8	A550725	511	9
9	A8605026	2834	10
10	A8982441	4068	11
11	A95703	243	12
12	B11293128	1036	13
13	B1139295	390	14
14	B124251	4526	15
15	B128778	27694	16
16	B133386	634	17
17	B144453	1443	18
18	B169653	655	19
19	B204480	312	20
20	B228150	371	21
21	B405882786	728	22
22	B429816540	10907	23
23	B433077165	6570	24
24	B8734110	10080	25
25	C11084986	8904	26
26	C129286	481	27
27	C133903	525	28
28	C156520	14446	29
29	C1840321	1291	30
30	C424759231	10118	31
31	C439398190	1924	32
32	C8329321	15612	33
33	C8390656	332	34
34	C9324721	18160	35
35	C9943570	1454	36

[0]:
```
# Create NumPy array with sum of Executed Qty of Traders
Y = Traders.iloc[:, [2, 1]].values
Y
```

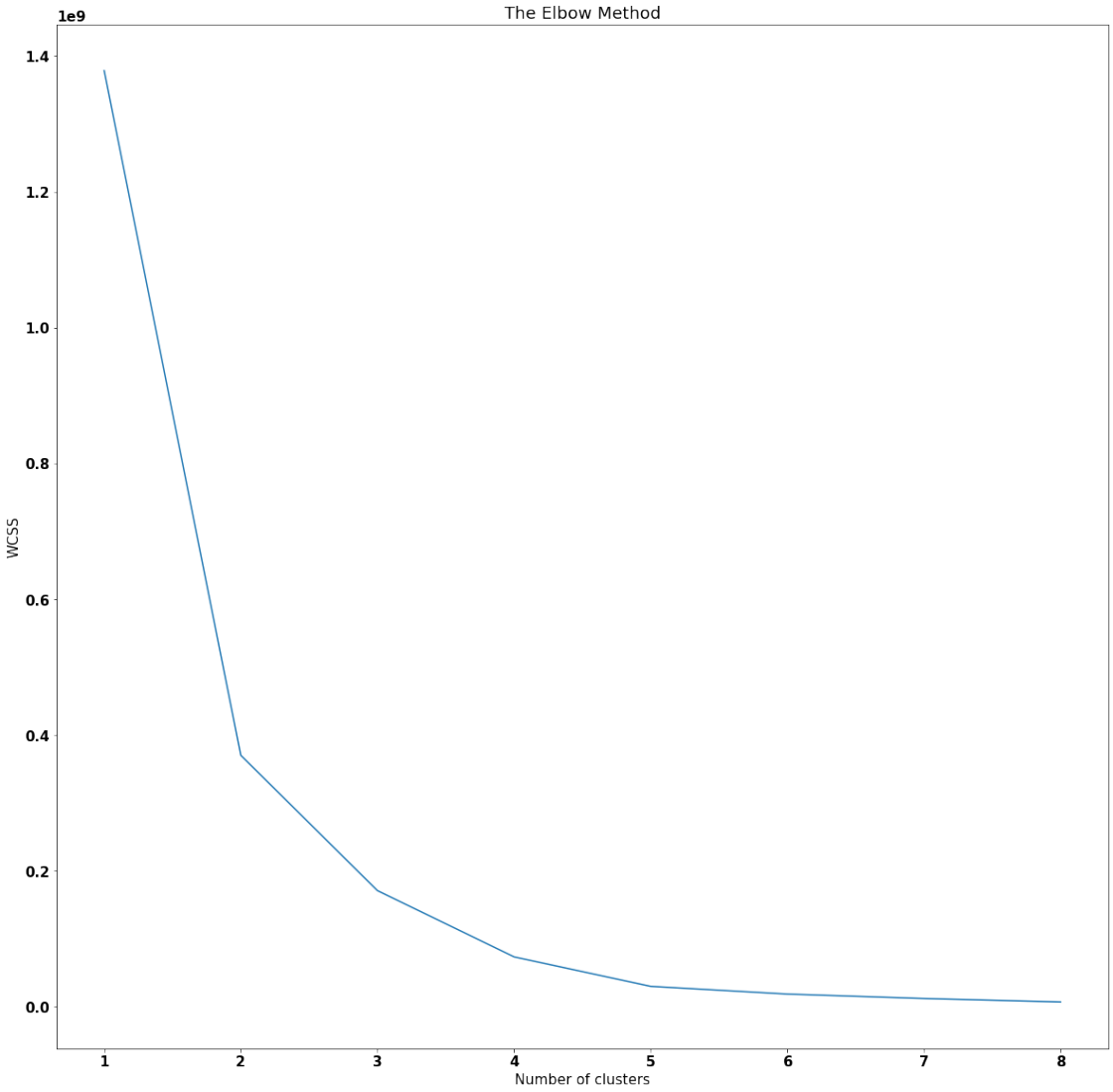
[0]:
```
array([[    1,  2900],
       [    2,  6588],
       [    3,  1445],
       [    4,  2278],
       [    5,   527],
       [    6,   211],
```

```
[    7,  4928],
[    8,   103],
[    9,   511],
[   10,  2834],
[   11,  4068],
[   12,   243],
[   13,  1036],
[   14,   390],
[   15,  4526],
[   16, 27694],
[   17,   634],
[   18,  1443],
[   19,   655],
[   20,   312],
[   21,   371],
[   22,   728],
[   23, 10907],
[   24,  6570],
[   25, 10080],
[   26,  8904],
[   27,   481],
[   28,   525],
[   29, 14446],
[   30,  1291],
[   31, 10118],
[   32,  1924],
[   33, 15612],
[   34,   332],
[   35, 18160],
[   36,  1454]])
```

1.2.1 Using the dendrogram to find the optimal number of clusters suitable to identify outliners

[0]:
```
# change the graph siXe of a matplotlib plot
plt.rcParams['figure.figsize'] = (20, 20)

from sklearn.cluster import KMeans
wcss = []
for i in range(1, 9):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(Y)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 9), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
```

```
plt.show()
```



1.2.2 Apply Hierarchical Clustering

[0]:
```
# Fitting Hierarchical Clustering to the array X
hc = AgglomerativeClustering(n_clusters = 3, affinity = 'euclidean', linkage =␣
 ↪'ward')
# Create Array with clusters
Y_hc = hc.fit_predict(Y)
Y_hc
```

[0]:
```
array([2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 0, 2, 2, 2, 2, 2, 2,
       1, 1, 1, 1, 2, 2, 0, 2, 1, 2, 0, 2, 0, 2])
```

[0]:
```
# Add a X_hc array as Cluster column to a Trades dataframe
Traders['Cluster'] = Y_hc
Traders
```

[0]:

	Buy Broker ID	Executed Qty	Index	Cluster
0	A11174628	2900	1	2
1	A11288376	6588	2	1
2	A125250	1445	3	2
3	A128271	2278	4	2
4	A163878	527	5	2
5	A170820	211	6	2
6	A2007006	4928	7	1
7	A203841	103	8	2
8	A550725	511	9	2
9	A8605026	2834	10	2
10	A8982441	4068	11	1
11	A95703	243	12	2
12	B11293128	1036	13	2
13	B1139295	390	14	2
14	B124251	4526	15	1
15	B128778	27694	16	0
16	B133386	634	17	2
17	B144453	1443	18	2
18	B169653	655	19	2
19	B204480	312	20	2
20	B228150	371	21	2
21	B405882786	728	22	2
22	B429816540	10907	23	1
23	B433077165	6570	24	1
24	B8734110	10080	25	1
25	C11084986	8904	26	1
26	C129286	481	27	2
27	C133903	525	28	2
28	C156520	14446	29	0
29	C1840321	1291	30	2
30	C424759231	10118	31	1
31	C439398190	1924	32	2
32	C8329321	15612	33	0
33	C8390656	332	34	2
34	C9324721	18160	35	0
35	C9943570	1454	36	2

1.2.3 Visualising the clusters

[0]:
```
# change the graph size of a matplotlib plot
plt.rcParams['figure.figsize'] = (20, 10)
# Visualising the clusters
plt.scatter(Y[Y_hc == 0, 0], Y[Y_hc == 0, 1], s = 30, c = 'red', label =␣
 ↪'Cluster 0')
plt.scatter(Y[Y_hc == 1, 0], Y[Y_hc == 1, 1], s = 30, c = 'orange', label =␣
 ↪'Cluster 1')
plt.scatter(Y[Y_hc == 2, 0], Y[Y_hc == 2, 1], s = 30, c = 'green', label =␣
 ↪'Cluster 2')
plt.title('Clusters of traders')
plt.xlabel('Trader Index')
plt.ylabel('Sum of Executed Qty')
plt.legend()
plt.show()
```

1.2.4 Calculating statistical information of clusters

[0]:
```
# Calculating statistical information of clusters
Cluster_Statistics = Traders.groupby("Cluster")['Executed Qty'].describe()
Cluster_Statistics
```

[0]:

Cluster	count	mean	std	...	50%	75%	max
				...			
0	4.0	18978.000000	6014.060193	...	16886.0	20543.5	27694.0
1	9.0	7409.888889	2644.599140	...	6588.0	10080.0	10907.0
2	23.0	983.826087	828.834212	...	634.0	1444.0	2900.0

```
[3 rows x 8 columns]
```

1.2.5 Outlier Traders

[0]:
```
# Create outlier Trdaes Dataframe
Cluster = [0, 1]
Outlier_Traders= Traders[Traders.Cluster.isin(Cluster)]
Outlier_Traders
```

[0]:

	Buy Broker ID	Executed Qty	Index	Cluster
1	A11288376	6588	2	1
6	A2007006	4928	7	1
10	A8982441	4068	11	1
14	B124251	4526	15	1
15	B128778	27694	16	0
22	B429816540	10907	23	1
23	B433077165	6570	24	1
24	B8734110	10080	25	1
25	C11084986	8904	26	1
28	C156520	14446	29	0
30	C424759231	10118	31	1
32	C8329321	15612	33	0
34	C9324721	18160	35	0

2 Q2

2.1 Identifing collusive trader group using Apriori Algorithm

Association rule learning is a rule-based machine learning method for discovering complex relationships between variables in large dataset. **Apriori** is an algorithm that use for frequent item set mining and association rule learning within databsets. It proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database.

Stock market manipulations as an organized **collusive trader groups** in stock market can be consider as one of major formats of market abuse, a Stock market manipulations can be extremely damaging to the proper functioning and integrity of capital markets.

Market manipulation refers to artificially inflating or deflating the price of a stocks or otherwise influencing the behavior of the market for personal gain. Two common types of stock manipulation are pump and dump and poop and scoop. The pump and dump is the most frequently used manipulation to inflate a microcap stock by artificially buying and then sell out, leaving later followers to hold the loss. Manipulation is variously called price manipulation, stock manipulation, and market manipulation. However, in this approach let's focus more about **price manipulation** and **run Apriori to identify frequent traders within the segments which shows considerable amount of stock price changes**.

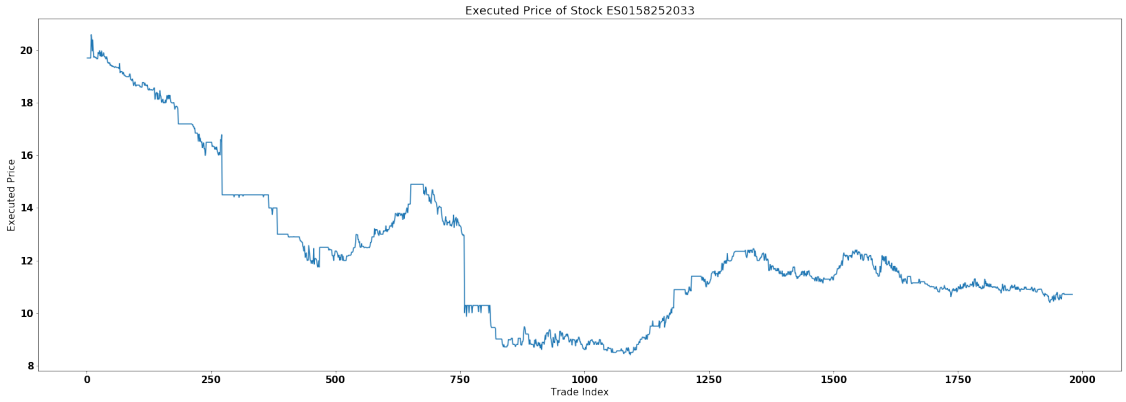
2.2 Data preprocessing with pandas

[0]:
```
# Create NumPy array with Execute Price of Trades
Z = Trades.iloc[:, [ 8, 2]].values
Z
```

[0]:
```
array([[1.000e+00, 1.971e+01],
       [2.000e+00, 1.971e+01],
       [3.000e+00, 1.971e+01],
       ...,
       [1.978e+03, 1.071e+01],
       [1.979e+03, 1.071e+01],
       [1.980e+03, 1.071e+01]])
```

2.2.1 Visualising the variation of executed price within the dataset

[0]:
```
# change the graph size of a matplotlib plot
plt.rcParams['figure.figsize'] = (30, 10)
# Plotting the graph of Executed Price of Stock ES0158252033
plt.plot(Z[:,0], Z[:,1])
plt.title('Executed Price of Stock ES0158252033')
plt.xlabel('Trade Index')
plt.ylabel('Executed Price')
plt.show()
```



By observing above price variation within the dataset its clear the requirement of some technique to identify the segment within dataset that has considerable amount of price variations. Hierarchical clustering with larger of clusters **(200 clusters)** can used to sample dataset into smaller segments. Then the standard deviation of the data points within each cluster can used as parameter of price variance present in the cluster. Then Apriori algorithm can run to identify the frequent traders who trade in these segments.

2.2.2 Applying Hierarchical Clustering to identify segment where considerable amount of price variations are present within the dataset.

[0]:
```
# Fitting Hierarchical Clustering to the array Z
hc = AgglomerativeClustering(n_clusters = 200, affinity = 'euclidean', linkage␣
 ↪= 'ward')
# Create Array with clusters
Z_hc = hc.fit_predict(Z)
Z_hc
```

[0]:
```
array([174, 174, 174, ..., 144, 144, 144])
```

[0]:
```
# Add a Z_hc array as Cluster column to a Trades dataframe
Trades['Cluster'] = Z_hc
Trades
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

[0]:

	Trade Date	Executed Qty	...	Index	Cluster
3	01JUL2014:09:00:00	1	...	1	174
4	01JUL2014:09:00:00	1	...	2	174
5	01JUL2014:09:00:00	2	...	3	174
6	01JUL2014:09:00:00	6	...	4	174
7	01JUL2014:09:00:00	12	...	5	174
8	01JUL2014:09:00:00	18	...	6	174
9	01JUL2014:09:00:00	44	...	7	174
10	01JUL2014:09:00:00	75	...	8	174
11	01JUL2014:09:00:03	14	...	9	93
12	01JUL2014:09:00:03	64	...	10	93
13	01JUL2014:09:00:03	70	...	11	93
14	01JUL2014:09:00:03	136	...	12	93
15	01JUL2014:09:04:25	60	...	13	93
17	01JUL2014:09:14:29	4	...	14	93
18	01JUL2014:09:14:29	16	...	15	93
19	01JUL2014:09:29:05	9	...	16	93
20	01JUL2014:09:32:38	3	...	17	193
21	01JUL2014:09:32:38	14	...	18	193
22	01JUL2014:09:32:38	133	...	19	193
23	01JUL2014:09:36:15	200	...	20	193
24	01JUL2014:09:43:08	13	...	21	193
25	01JUL2014:09:43:08	29	...	22	193

```
26    01JUL2014:09:44:24           55  ...     23     193
27    01JUL2014:09:44:24           74  ...     24     153
28    01JUL2014:09:44:24           98  ...     25     153
29    01JUL2014:09:44:24          100  ...     26     153
30    01JUL2014:09:44:24          133  ...     27     153
31    01JUL2014:09:44:24          199  ...     28     153
32    01JUL2014:09:49:47           10  ...     29     153
38    01JUL2014:10:03:00           11  ...     30     153
...                  ...          ...  ...    ...     ...
1969  01JUL2014:17:28:57           82  ...   1951      89
1970  01JUL2014:17:28:57           91  ...   1952      89
1971  01JUL2014:17:28:57          100  ...   1953      89
1972  01JUL2014:17:28:57          190  ...   1954      89
1973  01JUL2014:17:29:19          152  ...   1955      89
1974  01JUL2014:17:29:32           83  ...   1956      89
1975  01JUL2014:17:29:32          100  ...   1957      89
1976  01JUL2014:17:29:32          133  ...   1958      89
1977  01JUL2014:17:29:50          200  ...   1959      89
1978  01JUL2014:17:29:52           32  ...   1960      34
1979  01JUL2014:17:29:52           46  ...   1961      34
1980  01JUL2014:17:29:52           63  ...   1962      34
1981  01JUL2014:17:29:56          100  ...   1963      34
1983  01JUL2014:17:35:00           10  ...   1964      34
1984  01JUL2014:17:35:00           27  ...   1965      34
1985  01JUL2014:17:35:00           30  ...   1966      34
1986  01JUL2014:17:35:00           46  ...   1967      34
1987  01JUL2014:17:35:00           50  ...   1968      34
1988  01JUL2014:17:35:00           51  ...   1969      34
1989  01JUL2014:17:35:00           60  ...   1970      34
1990  01JUL2014:17:35:00          100  ...   1971      34
1991  01JUL2014:17:35:00          100  ...   1972     144
1992  01JUL2014:17:35:00          100  ...   1973     144
1993  01JUL2014:17:35:00          105  ...   1974     144
1994  01JUL2014:17:35:00          140  ...   1975     144
1995  01JUL2014:17:35:00          150  ...   1976     144
1996  01JUL2014:17:35:00          191  ...   1977     144
1997  01JUL2014:17:35:00          200  ...   1978     144
1998  01JUL2014:17:35:00          200  ...   1979     144
1999  01JUL2014:17:35:10          179  ...   1980     144

[1980 rows x 10 columns]
```

2.2.3 Identify the clusters which has considerable amount of price variations

[0]:
```
# compute a summary statistic for each clusters using groupby aggregation
Cluster_Statistics_Price = Trades.groupby("Cluster")['Executed Price'].
 ↪describe()
# Add a index clounm to Cluster Statistics Price dataframe
Cluster_Statistics_Price['Index'] = range(0, len(Cluster_Statistics_Price)+0)
# Sort all clusters according to ascending order by standard deviation
Cluster_Statistics_Price = Cluster_Statistics_Price.sort_values(by ='std',␣
 ↪ascending=False).reset_index()
Cluster_Statistics_Price
```

[0]:

	Cluster	count	mean	std	...	50%	75%	max	Index
0	0	14.0	11.612143	1.482577e+00	...	11.620	12.9950	13.26	0
1	2	14.0	15.297857	9.765889e-01	...	14.500	16.1150	16.79	2
2	93	8.0	20.078750	3.409414e-01	...	19.985	20.4000	20.60	93
3	38	13.0	9.609231	3.193342e-01	...	9.450	9.5100	10.29	38
4	5	14.0	14.189286	2.870128e-01	...	14.000	14.5000	14.50	5
5	66	10.0	12.092000	2.258712e-01	...	12.000	12.1525	12.57	66
6	8	14.0	11.942143	2.104639e-01	...	12.005	12.0550	12.25	8
7	31	12.0	12.258333	2.078388e-01	...	12.255	12.3950	12.55	31
8	65	12.0	11.970833	2.017405e-01	...	11.995	12.0550	12.46	65
9	27	13.0	16.379231	1.938841e-01	...	16.470	16.5200	16.66	27
10	3	14.0	12.191429	1.825401e-01	...	12.155	12.3825	12.45	3
11	108	10.0	12.091000	1.575119e-01	...	12.165	12.2000	12.29	108
12	194	7.0	18.382857	1.566008e-01	...	18.380	18.4850	18.57	194
13	172	7.0	11.294286	1.559762e-01	...	11.270	11.4000	11.51	172
14	52	12.0	10.031667	1.551441e-01	...	10.000	10.1925	10.20	52
15	109	12.0	11.926667	1.456854e-01	...	11.890	11.9850	12.27	109
16	28	12.0	13.466667	1.421480e-01	...	13.450	13.5500	13.73	28
17	35	10.0	14.383000	1.392879e-01	...	14.450	14.5000	14.51	35
18	39	11.0	9.252727	1.374839e-01	...	9.200	9.3300	9.48	39
19	47	12.0	13.044167	1.372760e-01	...	13.065	13.1800	13.21	47
20	123	9.0	8.971111	1.366057e-01	...	8.960	9.0000	9.21	123
21	158	9.0	8.892222	1.343296e-01	...	8.890	9.0000	9.07	158
22	173	7.0	14.527143	1.337553e-01	...	14.510	14.6000	14.69	173
23	81	8.0	12.867500	1.316652e-01	...	12.890	12.9825	12.99	81
24	147	8.0	14.058750	1.297732e-01	...	14.140	14.1500	14.15	147
25	77	9.0	8.902222	1.281384e-01	...	9.000	9.0100	9.01	77
26	49	13.0	14.057692	1.269615e-01	...	14.030	14.1000	14.25	49
27	167	8.0	11.158750	1.268787e-01	...	11.145	11.2925	11.30	167
28	95	9.0	12.438889	1.257422e-01	...	12.480	12.4900	12.69	95
29	16	13.0	18.146154	1.257337e-01	...	18.150	18.2800	18.29	16
..	...	...	...	...	...	...	...	...	...
170	105	8.0	8.502500	7.071068e-03	...	8.500	8.5000	8.52	105
171	149	9.0	12.901111	3.333333e-03	...	12.900	12.9000	12.91	149
172	174	8.0	19.710000	3.798011e-15	...	19.710	19.7100	19.71	174
173	179	8.0	11.400000	1.899005e-15	...	11.400	11.4000	11.40	179

174	181	8.0	14.900000	1.899005e-15	...	14.900	14.9000	14.90	181
175	90	8.0	11.400000	1.899005e-15	...	11.400	11.4000	11.40	90
176	94	8.0	14.900000	1.899005e-15	...	14.900	14.9000	14.90	94
177	170	8.0	12.900000	1.899005e-15	...	12.900	12.9000	12.90	170
178	151	9.0	11.150000	1.884111e-15	...	11.150	11.1500	11.15	151
179	133	10.0	14.900000	1.872445e-15	...	14.900	14.9000	14.90	133
180	70	12.0	14.500000	0.000000e+00	...	14.500	14.5000	14.50	70
181	144	9.0	10.710000	0.000000e+00	...	10.710	10.7100	10.71	144
182	143	8.0	10.890000	0.000000e+00	...	10.890	10.8900	10.89	143
183	63	12.0	14.500000	0.000000e+00	...	14.500	14.5000	14.50	63
184	159	8.0	17.200000	0.000000e+00	...	17.200	17.2000	17.20	159
185	152	8.0	14.500000	0.000000e+00	...	14.500	14.5000	14.50	152
186	166	8.0	16.500000	0.000000e+00	...	16.500	16.5000	16.50	166
187	129	9.0	14.000000	0.000000e+00	...	14.000	14.0000	14.00	129
188	163	8.0	14.500000	0.000000e+00	...	14.500	14.5000	14.50	163
189	104	8.0	12.500000	0.000000e+00	...	12.500	12.5000	12.50	104
190	182	8.0	12.500000	0.000000e+00	...	12.500	12.5000	12.50	182
191	180	8.0	10.890000	0.000000e+00	...	10.890	10.8900	10.89	180
192	178	8.0	17.200000	0.000000e+00	...	17.200	17.2000	17.20	178
193	177	8.0	14.500000	0.000000e+00	...	14.500	14.5000	14.50	177
194	176	8.0	8.560000	0.000000e+00	...	8.560	8.5600	8.56	176
195	98	8.0	13.000000	0.000000e+00	...	13.000	13.0000	13.00	98
196	99	8.0	17.200000	0.000000e+00	...	17.200	17.2000	17.20	99
197	169	8.0	11.290000	0.000000e+00	...	11.290	11.2900	11.29	169
198	165	8.0	9.010000	0.000000e+00	...	9.010	9.0100	9.01	165
199	183	8.0	13.000000	0.000000e+00	...	13.000	13.0000	13.00	183

[200 rows x 10 columns]

Lets select all the clusters which has standard deviation more than 0.1 to input in to Apriori algorithm. From above summary statistic for each clusters using groupby aggregation we can identify 52 clusters which has standard deivation more than 0.1.

[0]:
```
# Drop all clusters has standard deviation less than 0.1
T_Clusters = Cluster_Statistics_Price.iloc[0:52, [9]]
T_Clusters
```

2.2.4 Create a list of lists to input to Apriori algorithm

Apriori library I am going to use requires input dataset to be in the form of a list of lists. Therefore as a final step of the Data preprocessing process lets create transaction list of list.

[0]:
```
# Create a list of lists ()
transactions = (
     [Trades.loc[Trades.Cluster==0, :]['Buy Broker ID'].unique().tolist() +␣
 ↪Trades.loc[Trades.Cluster==0, :]['Sell Broker ID'].unique().tolist()] +
     [Trades.loc[Trades.Cluster==2, :]['Buy Broker ID'].unique().tolist() +␣
 ↪Trades.loc[Trades.Cluster==2, :]['Sell Broker ID'].unique().tolist()] +
```

```
    [Trades.loc[Trades.Cluster==93, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==93, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==38, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==38, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==5, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==5, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==66, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==66, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==8, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==8, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==31, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==31, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==65, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==65, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==27, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==27, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==3, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==3, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==108, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==108, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==194, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==194, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==172, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==172, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==52, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==52, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==109, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==109, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==28, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==28, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==35, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==35, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==39, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==39, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==47, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==47, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==123, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==123, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==158, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==158, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==173, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==173, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==81, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==81, :]['Sell Broker ID'].unique().tolist()] +
```

```
    [Trades.loc[Trades.Cluster==147, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==147, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==77, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==77, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==49, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==49, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==167, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==167, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==95, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==95, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==16, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==16, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==134, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==134, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==6, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==6, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==13, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==13, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==60, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==60, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==146, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==146, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==64, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==64, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==191, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==191, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==132, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==132, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==155, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==155, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==83, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==83, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==50, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==50, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==128, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==128, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==40, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==40, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==186, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==186, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==137, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==137, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==17, :]['Buy Broker ID'].unique().tolist() +␣
↪Trades.loc[Trades.Cluster==17, :]['Sell Broker ID'].unique().tolist()] +
```

```
    [Trades.loc[Trades.Cluster==29, :]['Buy Broker ID'].unique().tolist() +␣
 ↪Trades.loc[Trades.Cluster==29, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==97, :]['Buy Broker ID'].unique().tolist() +␣
 ↪Trades.loc[Trades.Cluster==97, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==91, :]['Buy Broker ID'].unique().tolist() +␣
 ↪Trades.loc[Trades.Cluster==91, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==68, :]['Buy Broker ID'].unique().tolist() +␣
 ↪Trades.loc[Trades.Cluster==68, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==85, :]['Buy Broker ID'].unique().tolist() +␣
 ↪Trades.loc[Trades.Cluster==85, :]['Sell Broker ID'].unique().tolist()] +
    [Trades.loc[Trades.Cluster==10, :]['Buy Broker ID'].unique().tolist() +␣
 ↪Trades.loc[Trades.Cluster==10, :]['Sell Broker ID'].unique().tolist()])
transactions
```

2.3 Training Apriori on the dataset to identify potential collusive trader groups (Model forming)

[0]:
```
rules_test = apriori(transactions, min_support = 0.1,min_confidence = 0.3,␣
 ↪min_lift = 1.0001)
# Create a list with all potential collusive trader groups
results_test = list(rules_test)
# Visualising the number of potential collusive trader groups
print(len(results_test))
```

```
717
```

2.4 Visualising the potential collusive trader groups

[0]:
```
  the_rules = []
for result in results_test:
    the_rules.append({'Collusive Trader Groups Test': ','.join(result.items),
                      'Support':result.support,
                      'Confidence':result.ordered_statistics[0].confidence,
                      'Lift':result.ordered_statistics[0].lift})
# Create a dataframe with all potential collusive trader groups
collusive_trader_groups_test = (pd.DataFrame(the_rules, columns = ['Collusive␣
 ↪Trader Groups Test', 'Support', 'Confidence', 'Lift'])).sort_values(by␣
 ↪='Support', ascending=False)
# Sort all potential collusive trader groups according to ascending order by␣
 ↪support
collusive_trader_groups_test = collusive_trader_groups_test.sort_values(by␣
 ↪='Support', ascending=False)
collusive_trader_groups_test
```

[0]:

	Collusive Trader Groups Test	...	Lift
52	B128778,B429816540	...	1.011442

56	B128778,C8329321	...	1.008226
57	B128778,C9324721	...	1.033540
72	B429816540,C8329321	...	1.035562
73	B429816540,C9324721	...	1.016541
86	C8329321,C9324721	...	1.003861
70	B429816540,C424759231	...	1.026316
252	B128778,C9324721,B429816540	...	1.048739
271	B128778,C8329321,C9324721	...	1.035498
68	B433077165,B429816540	...	1.258947
14	A11288376,C8329321	...	1.010135
54	B128778,C156520	...	1.036232
15	A11288376,C9324721	...	1.021429
83	C9324721,C424759231	...	1.021429
250	B128778,C424759231,B429816540	...	1.003676
6	A8605026,A11288376	...	1.100806
13	A11288376,C424759231	...	1.066406
267	B128778,C9324721,C424759231	...	1.100529
247	B433077165,B128778,B429816540	...	1.223529
297	B429816540,C8329321,C9324721	...	1.061224
132	B128778,A11288376,C8329321	...	1.003861
133	B128778,A11288376,C9324721	...	1.061224
115	A8605026,B128778,A11288376	...	1.022774
283	B433077165,B429816540,C8329321	...	1.160987
76	B433077165,C8329321	...	1.068108
69	B429816540,C156520	...	1.083333
516	B128778,C8329321,C9324721,B429816540	...	1.114286
78	B8734110,C8329321	...	1.099882
197	A8605026,B128778,C9324721	...	1.017391
130	B128778,A11288376,C424759231	...	1.044643
..	...	...	...
640	B128778,C439398190,C9324721,B429816540,A8605026	...	1.846154
697	B128778,C439398190,C9324721,B429816540,A860502...	...	1.114286
217	A8982441,B128778,B206403	...	2.228571
637	B128778,C439398190,B429816540,A8605026,C424759231	...	2.000000
186	C11084986,A2007006,C424759231	...	2.888889
667	B128778,C156520,B8734110,C8329321,B429816540	...	1.054054
185	B429816540,C9324721,A2007006	...	1.172932
669	B128778,B8734110,C8329321,B429816540,C424759231	...	1.054054
184	B429816540,C8329321,A2007006	...	1.026316
672	B128778,C156520,C439398190,B429816540,C424759231	...	2.000000
179	A8605026,C11084986,A2007006	...	2.166667
686	C156520,B433077165,C9324721,B429816540,C424759231	...	1.273469
675	B128778,C156520,C439398190,C9324721,B429816540	...	1.273469
170	A128271,B429816540,C156520	...	1.300000
176	A128271,C8329321,C9324721	...	1.114286
175	A128271,C156520,C9324721	...	1.300000
2	A11174628,C11084986	...	1.238095

```
679      B128778,B8734110,C8329321,B433077165,C9324721  ...  1.405405
680       B128778,C156520,C8329321,B433077165,C9324721  ...  1.114286
172                         A128271,B429816540,C8329321  ...  1.026316
657     B128778,C8329321,B433077165,B206403,B429816540  ...  1.405405
157                           A125250,B128778,C424759231  ...  1.130435
692  B128778,C156520,C8329321,B429816540,A8605026,A...  ...  1.625000
643      B128778,B8734110,C9324721,A8605026,C424759231  ...  1.114286
699  B128778,C156520,C8329321,A8605026,A11288376,C4...  ...  1.054054
227                           A8982441,C8329321,C9324721  ...  1.054054
683      B128778,B8734110,C8329321,C9324721,C424759231  ...  1.130435
223                       B433077165,A8982441,B429816540  ...  1.248000
698  B128778,C8329321,C9324721,B429816540,A8605026,...  ...  1.114286
716  B128778,C156520,C9324721,B429816540,A8605026,A...  ...  1.273469

[717 rows x 4 columns]
```

2.4.1 Training Apriori on the transaction list to filter best collusive trader groups (Model Tuning)

As a step of model tuning in order to filter out strongest rules form above 717 potential rules, lets set the parameters of Apriori algorithm as follow,

Minimum Support = 0.15
Minimum Confidence = 0.75
Minimum Lift = 2
Minimum length = 2

[81]:
```
rules = apriori(transactions, min_support = 0.15, min_confidence = 0.75,␣
 ↪min_lift = 2, min_length = 2)
# Create a list with all collusive trader groups
results = list(rules)
# Visualising the number of collusive trader groups
print(len(results))
```

```
2
```

2.5 Visualising the final collusive trader groups

[82]:
```
the_rules = []
for result in results:
    the_rules.append({'Collusive Trader Groups': ','.join(result.items),
                      'Support':result.support,
                      'Confidence':result.ordered_statistics[0].confidence,
                      'Lift':result.ordered_statistics[0].lift})
# Create a dataframe with all collusive trader groups
collusive_trader_groups = pd.DataFrame(the_rules, columns = ['Collusive Trader␣
 ↪Groups', 'Support', 'Confidence', 'Lift'])
collusive_trader_groups
```

[82]:

	Collusive Trader Groups	Support	Confidence	Lift
0	C11084986,A2007006	0.192308	0.769231	2.222222
1	C11084986,A11288376,A2007006	0.153846	1.000000	2.888889

As a conclusion, we can identify **C11084986** and **A2007006** traders appear together within 10 clusters out of selected 52 clusters. In addition, **C11084986**, **A11288376** and **A2007006** traders appear together within 8 clusters out of selected 52 clusters.