Community Detection of S&P500 stocks

Hardware & Software for Big Data project

Daniele Petrillo P37000102

Aim of the project

The objective of the project is to demonstrate the utilization of PySpark and Kafka environments, as a preparation phase for a community detection analysis of financial stocks.

The work is divided into four parts:

1. **Preparation** of the dataset using **PySpark**
2. Simulation of the **streaming** of the dataset through **Kafka**
3. **E**xploratory **D**ata **A**nalysis
4. **Community Detection**

Dataset description

It consists in a time series of general information about each stock contained in the **S&P500** index.

The **Standard and Poor 500** tracks the performance of 500 large companies listed on stock exchanges in the United States.
It is considered the most famous financial benchmark in the world.

The dataset is extracted from Kaggle
(https://www.kaggle.com/datasets/andrewmvd/sp-500-stocks?select=sp500_stocks.csv)

1. Preparation of the dataset

Apache **Spark**:

- Fast and general engine for large scale data processing
- MapReduce-like
- 40x faster than Hadoop
- Fast iterative queries through in-memory data storage

Spark's fundamental data structures are **R**esilient **D**istributed **D**atasets (**RDD**s):

- Distributed collection of objects
- Automatically rebuilt upon failure
- Immutable, partitioned, logical tables
- Not materialized, keep track of changes on original data
- Only materialized once an action is executed on them

1. Preparation of the dataset

PySpark: Python API for Apache Spark

```python
# start a pyspark session
import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()
spark.conf.set("spark.sql.repl.eagerEval.enabled", True)
spark
```

SparkSession - in-memory

SparkContext

Spark UI

Version
`v3.1.1`
Master
`local[*]`
AppName
`pyspark-shell`

The dataset looks like this:

```python
# load the dataset in pyspark
df_pyspark = spark.read.csv("/content/drive/MyDrive/HWSWmoduleA/sp500_stocks.csv", header=True)
df_pyspark.show(5)
```

```
+----------+------+-----------------+-----------------+-----------------+-----------------+-----------------+-------+
|      Date|Symbol|        Adj Close|            Close|             High|              Low|             Open| Volume|
+----------+------+-----------------+-----------------+-----------------+-----------------+-----------------+-------+
|2010-01-04|   MMM|59.318885803222656| 83.0199966430664|83.44999694824219|82.66999816894531|83.08999633789062|3043700|
|2010-01-05|   MMM| 58.94734191894531|             82.5| 83.2300033569336|81.69999694824219|82.80000305175781|2847000|
|2010-01-06|   MMM|59.783294677734375|83.66999816894531| 84.5999984741211|83.51000213623047|83.87999725341797|5268500|
|2010-01-07|   MMM|59.826175689697266| 83.7300033569336|83.76000213623047|82.12000274658203|83.31999969482422|4470100|
|2010-01-08|   MMM| 60.24774932861328|84.31999969482422|84.31999969482422|83.30000305175781|83.69000244140625|3405800|
+----------+------+-----------------+-----------------+-----------------+-----------------+-----------------+-------+
only showing top 5 rows
```

1. Preparation of the dataset

```
# some stats on the dataset
df_pyspark.summary().show()
```

summary	Date	Symbol	Adj Close	Close	High	Low	Open	Volume
count	1794201	1794201	1714288	1714288	1714288	1714288	1714288	1714288
mean	null	null	95.72522509561246	102.64200552379694	103.76725430194739	101.45577584877337	102.61854800683831	5892795.611819601
stddev	null	null	195.50987514184135	195.40076270811915	197.68927856317288	193.00689930671587	195.30100508920327	1.984405429894815E7
min	2010-01-04	A	0.7	0.7	0.71	0.65	0.7	0.0
25%	null	null	29.555536	35.2	35.6	34.8	35.2	992500.0
50%	null	null	53.760475	61.9	62.54	61.24	61.896713	2140200.0
75%	null	null	102.25	111.37	112.55	110.15	111.36	4869653.0
max	2024-03-06	ZTS	7709.27	7709.27	7776.17	7651.83	7698.43	1.88099802E9

We focus on a small window of time for our analysis, from 2023-05-02 to 2023-07-31.

This corresponds to 62 effective days of stock exchange.

```
df_pyspark = df_pyspark.filter(df_pyspark.Date.isin(daterange))
df_pyspark.show(5)
```

Date	Symbol	Adj Close	Close	High	Low	Open	Volume
2023-05-02	MMM	102.98	102.98	105.7	102.66	105.51	3012000.0
2023-05-03	MMM	102.83	102.83	104.6	102.67	103.5	2056100.0
2023-05-04	MMM	101.84	101.84	102.98	100.76	102.55	2963200.0
2023-05-05	MMM	103.35	103.35	103.48	102.05	102.8	1943100.0
2023-05-08	MMM	102.34	102.34	103.93	101.65	103.56	2094000.0

1. Preparation of the dataset

To stream the dataset into a Kafka process, we limit the number of stocks to 100, this also helps us showcase basic operations in PySpark:

```
# sample 100 unique stocks from the filtered dataframe
sampled_df = df_pyspark.select("Symbol").distinct().sample(fraction=1.0).limit(100)
```

```
# make sure there are oly 100 unique stocks left
sampled_df.select("Symbol").distinct().count()
```

```
100
```

```
# join with the original data to filter the stocks
joined_df = sampled_df.join(df_pyspark, on="symbol", how="inner")
```

```
# the number of rows should now be 6200 = 62 days * 100 stocks
joined_df.count()
```

```
6200
```

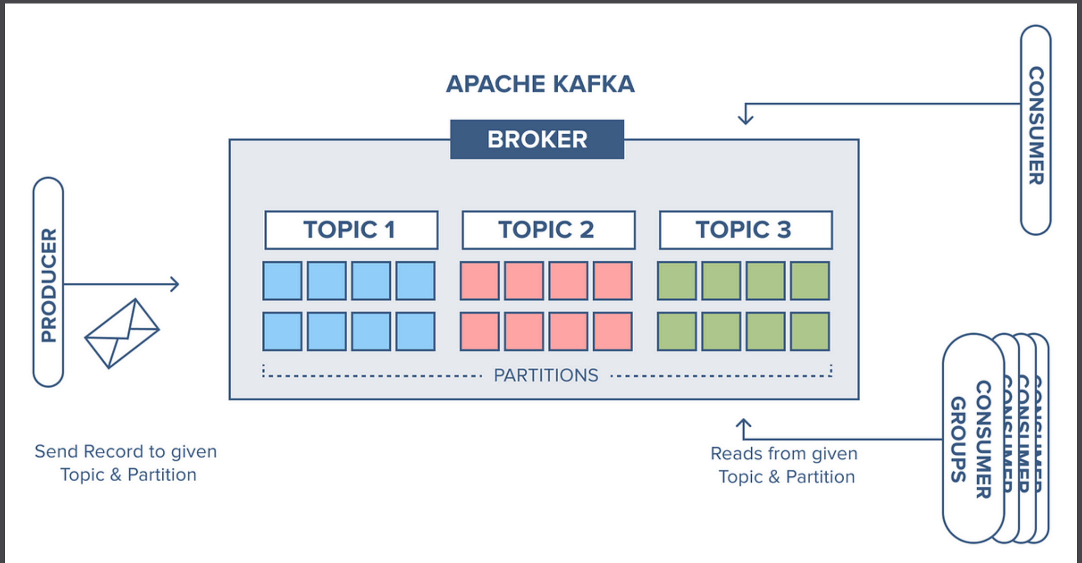
The dataset is saved in csv format in order to stream it:

```
# finally, we turn the dataframe into a csv file to transfer it through Kafka
joined_df.toPandas().to_csv("/content/drive/MyDrive/HWSWmoduleA/sp100_stocks_62days.csv")
```

2. Streaming through Kafka

Apache **Kafka**:

- Open Source **distributed event streaming** platform
- **Publish/Subscribe** model
- Events are posted on and read from structures called **Topics**
- Enables **partitioning** of topics for memory and speed optimization
- **Replication** of topics grants high **consistency** and **availability**



Kafka's three main entities are:

- **Producer**: posts the events in the topic
- **Consumer**: reads the events from the topic
- **Broker**: a server that hosts a topic or part of it

2. Streaming through Kafka

Setup Kafka in Python:

```python
# setup the producer and consumer
import requests
from confluent_kafka import Producer
from confluent_kafka import Consumer, KafkaException

kafka_bootstrap_servers = "localhost:9092"
kafka_topic = "SMP500"
```

Configure Producer and Consumer:

```python
# Kafka producer and consumer configuration
producer_config = {
    "bootstrap.servers": kafka_bootstrap_servers,
}
consumer_config = {
    "bootstrap.servers": kafka_bootstrap_servers,
    "group.id": "your_consumer_group_id",
    "auto.offset.reset": "earliest",
}
```

Create instances and subscribe to the topic:

```python
# create instances
producer = Producer(producer_config)
consumer = Consumer(consumer_config)

# consumer subscribes to the SMP500 topic
consumer.subscribe([kafka_topic])
```

2. Streaming through Kafka

Post the event into the topic:

```
# producer posts the message to the topic
producer.produce(kafka_topic, value=dataset)
```

Read the event:

```
# consumer reads the message event
event = consumer.poll(1)
event_value = event.value()
consumer.close()
```

kafka

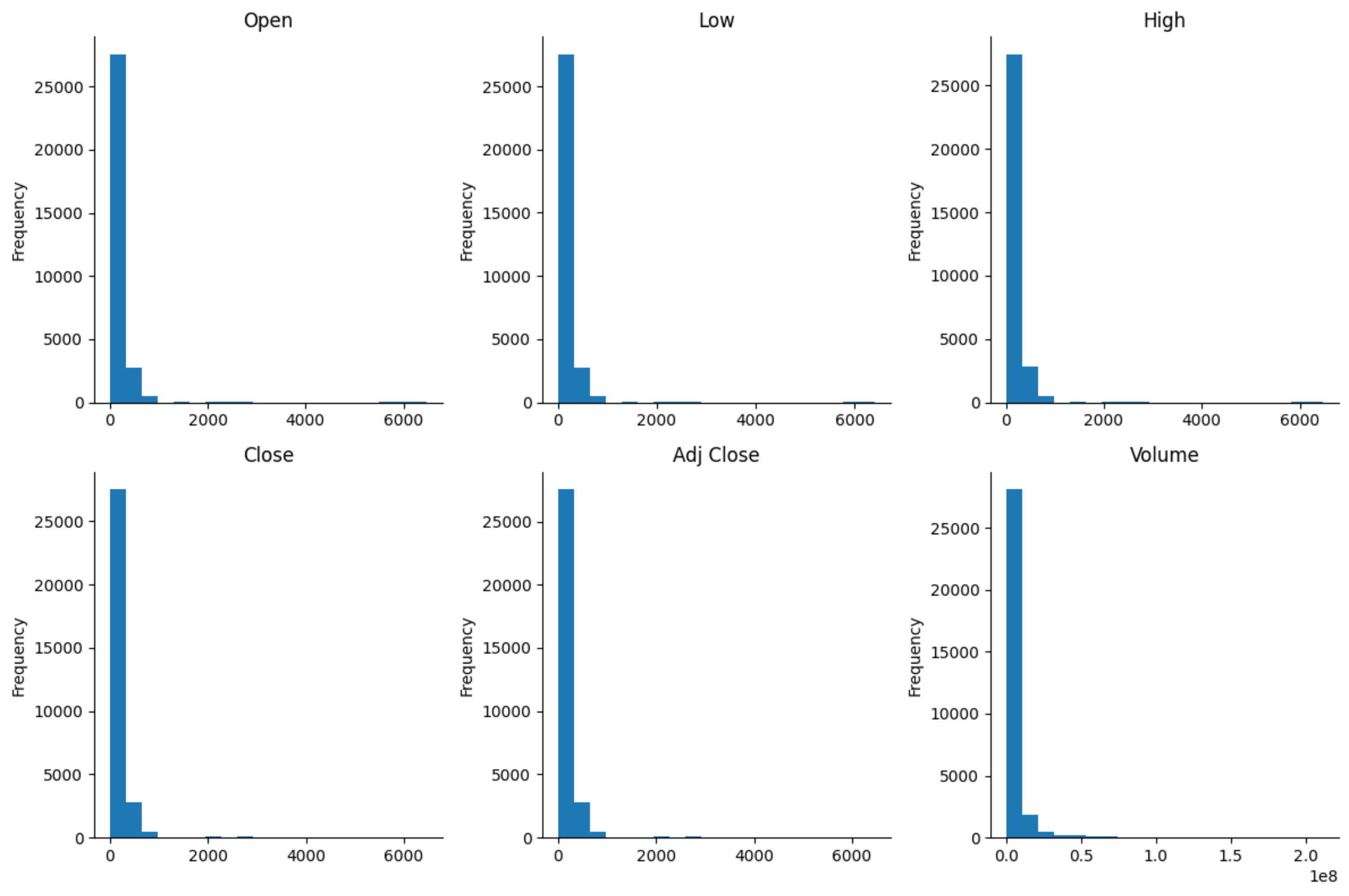
Finally, the event gets succesfully decoded and is ready for use:

```
# consumer reconstructs the dataset from the value
import json
decoded_dataset = json.loads(event_value.decode("utf-8"))
df = pd.DataFrame(decoded_dataset)
df
```

	Symbol	Date	Adj Close	Close	High	Low	Open	Volume
0	MMM	2023-05-02	102.98000	102.98	105.70	102.66	105.51	3012000.0
1	MMM	2023-05-03	102.83000	102.83	104.60	102.67	103.50	2056100.0
2	MMM	2023-05-04	101.84000	101.84	102.98	100.76	102.55	2963200.0
3	MMM	2023-05-05	103.35000	103.35	103.48	102.05	102.80	1943100.0
4	MMM	2023-05-08	102.34000	102.34	103.93	101.65	103.56	2094000.0
...	...	...	...	...	...	...	...	...
6195	ZBH	2023-07-25	139.71729	140.30	140.76	139.31	140.47	842700.0
6196	ZBH	2023-07-26	141.23099	141.82	142.26	139.46	139.77	996600.0
6197	ZBH	2023-07-27	139.27913	139.86	142.54	139.85	142.44	1441600.0
6198	ZBH	2023-07-28	138.76128	139.34	140.77	138.51	140.45	997100.0
6199	ZBH	2023-07-31	137.57622	138.15	139.31	137.68	139.31	1619500.0

6200 rows × 8 columns

3. Exploratory Data Analysis

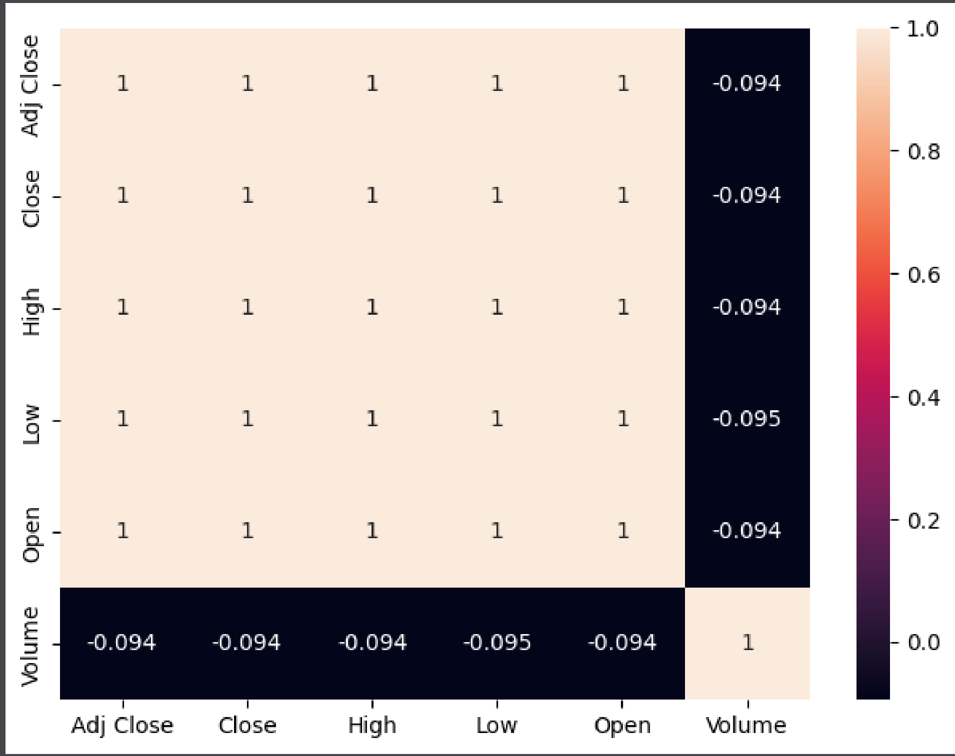


Distribution of variables value across the dataset:

The distributions are quite similar, this is because the variables have high dependancy from each other.

3. Exploratory Data Analysis

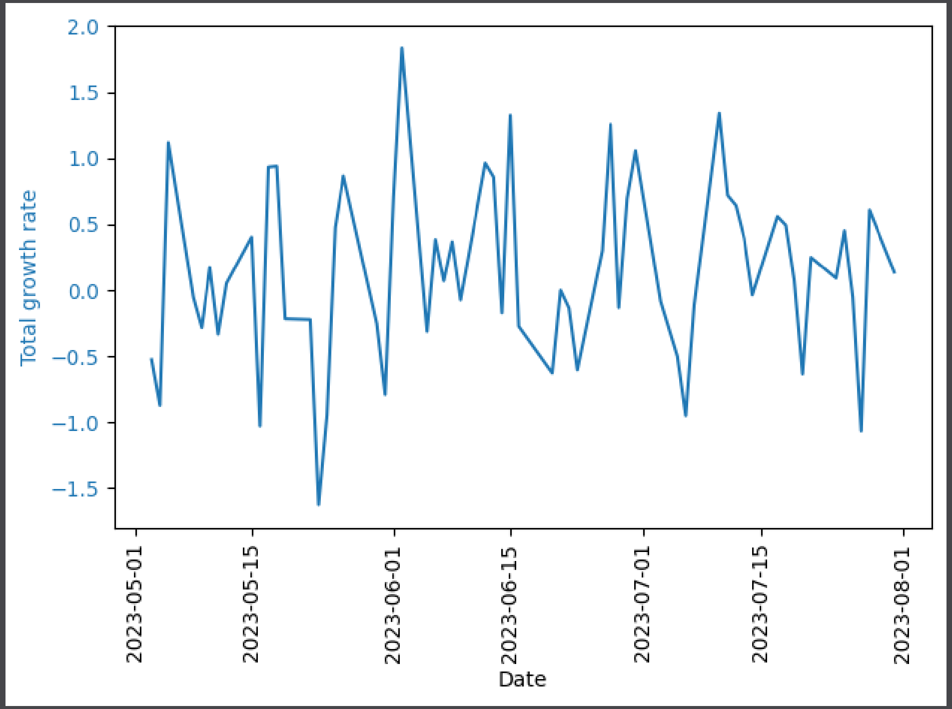
As expected, all variables except Volume are highly correlated:



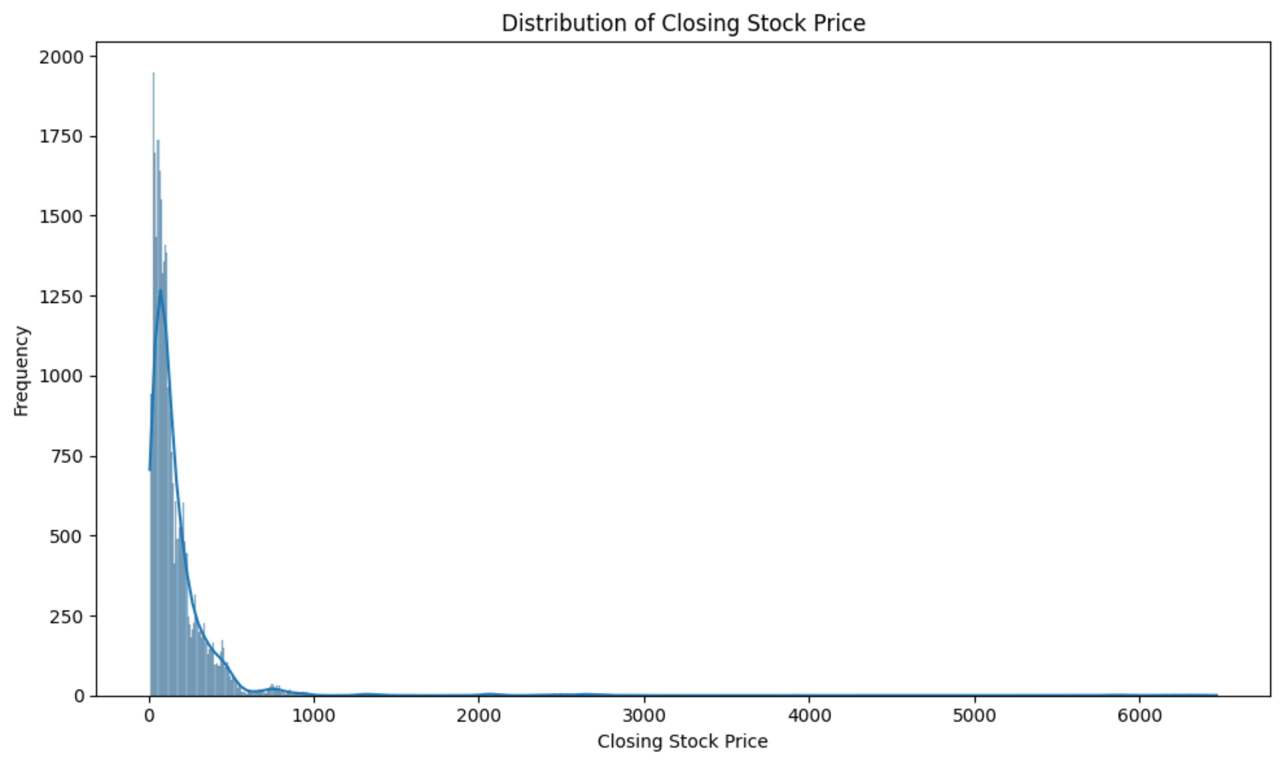
Total volume of the index vs total open value:

3. Exploratory Data Analysis

Evolution of the grow rate of the index:



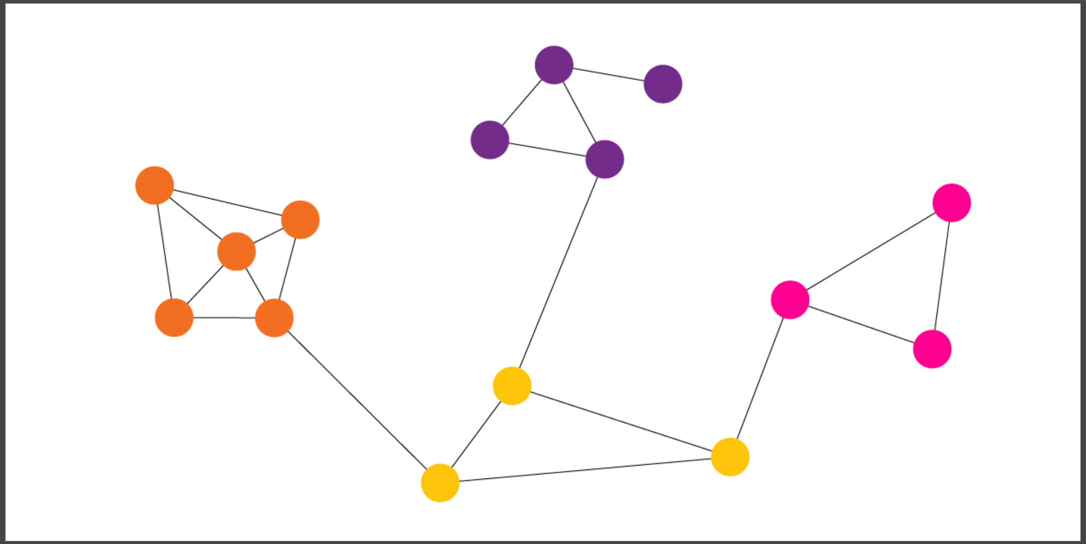
Closer look at the distribution of closing price:



4. Community Detection

Subfield of Graph Theory that focuses in finding communities in a network graph.

Communities in a network are the dense groups of the vertices, which are tightly coupled to each other inside the group and loosely coupled to the rest of the vertices in the network.



Many algorithms have been proposed in the literature to this day, each one differing in structure, complexity and suitable use cases.

4. Community Detection

To make use of community detection algorithms to find families of similar stocks in the S&P500 index, we need to transform the dataset into a **graph**.

We construct a **Cross Correlation Matrix** between all the stocks.
Each entry contains the **correlation** of two corresponding stocks given by the formula:

$$\rho_{mn} = \frac{\sum_t [(p_m(t) - \overline{p_m})(p_n(t) - \overline{p_n})]}{\sqrt{\sum_t (p_m(t) - \overline{p_m})^2} \sqrt{\sum_t (p_n(t) - \overline{p_n})^2}}$$

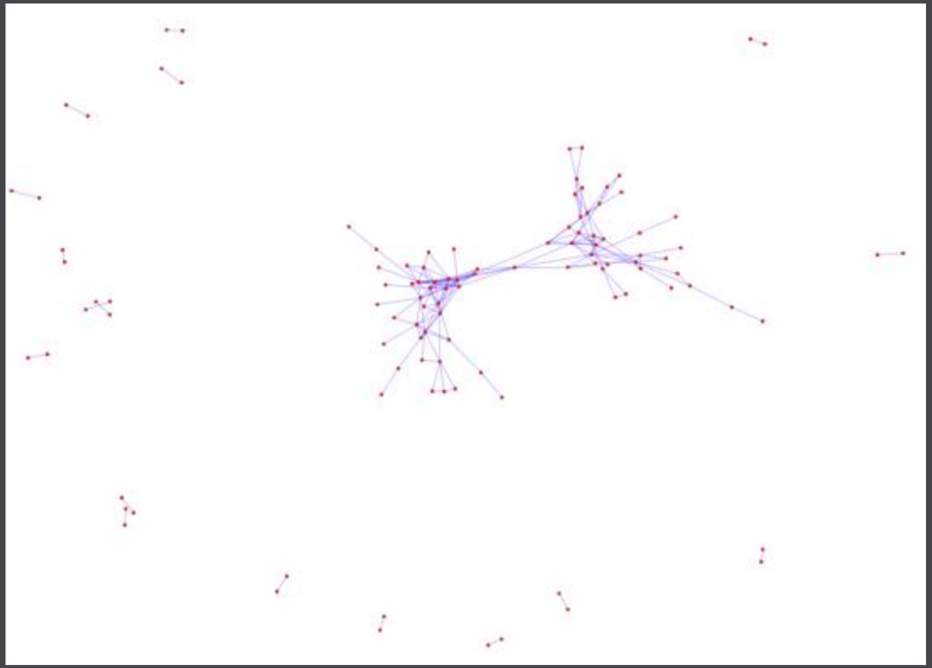
	MMM	AOS	ABT	ABBV	ACN	ADBE	AMD	AES	AFL	A	...
MMM	1.000000	0.606726	0.596777	0.527495	0.276243	0.398541	-0.105007	0.560766	0.701678	0.405469	...
AOS	0.606726	1.000000	0.511341	-0.002029	0.505706	0.720898	0.056813	0.494367	0.901640	0.049813	...
ABT	0.596777	0.511341	1.000000	0.661504	-0.248528	0.025558	-0.677162	0.775966	0.425377	0.730267	...
ABBV	0.527495	-0.002029	0.661504	1.000000	-0.505428	-0.416701	-0.658047	0.641973	0.041680	0.858748	...
ACN	0.276243	0.505706	-0.248528	-0.505428	1.000000	0.909772	0.810656	-0.286594	0.641939	-0.624848	...
...	...	...	...	...	...	...	...	...	...	...	...
YUM	0.520911	0.425945	0.662196	0.405129	-0.092536	0.079004	-0.422886	0.650328	0.428364	0.492998	...
ZBRA	0.481796	0.866469	0.294468	-0.140644	0.642616	0.773498	0.291203	0.309016	0.852654	-0.139335	...
ZBH	0.401950	0.723799	0.493251	-0.103132	0.295257	0.563174	-0.125229	0.469763	0.665035	0.013273	...
ZION	0.581287	0.610230	0.107888	-0.016668	0.768128	0.763655	0.532874	-0.027957	0.755739	-0.222174	...
ZTS	0.654115	0.280577	0.858482	0.803387	-0.341266	-0.174140	-0.609051	0.688654	0.272993	0.781339	...

4. Community Detection

Each node in the graph represents a stock, while each edge the correlation between stocks.

An edge between two stocks is created only if the correlation is greater (in absolute value) than a given threshold (typically 0.97), in order to link together only highly connected stocks.

The graph can be either weighted (edge weight = cross correlation) or unweighted (1/0 connection).

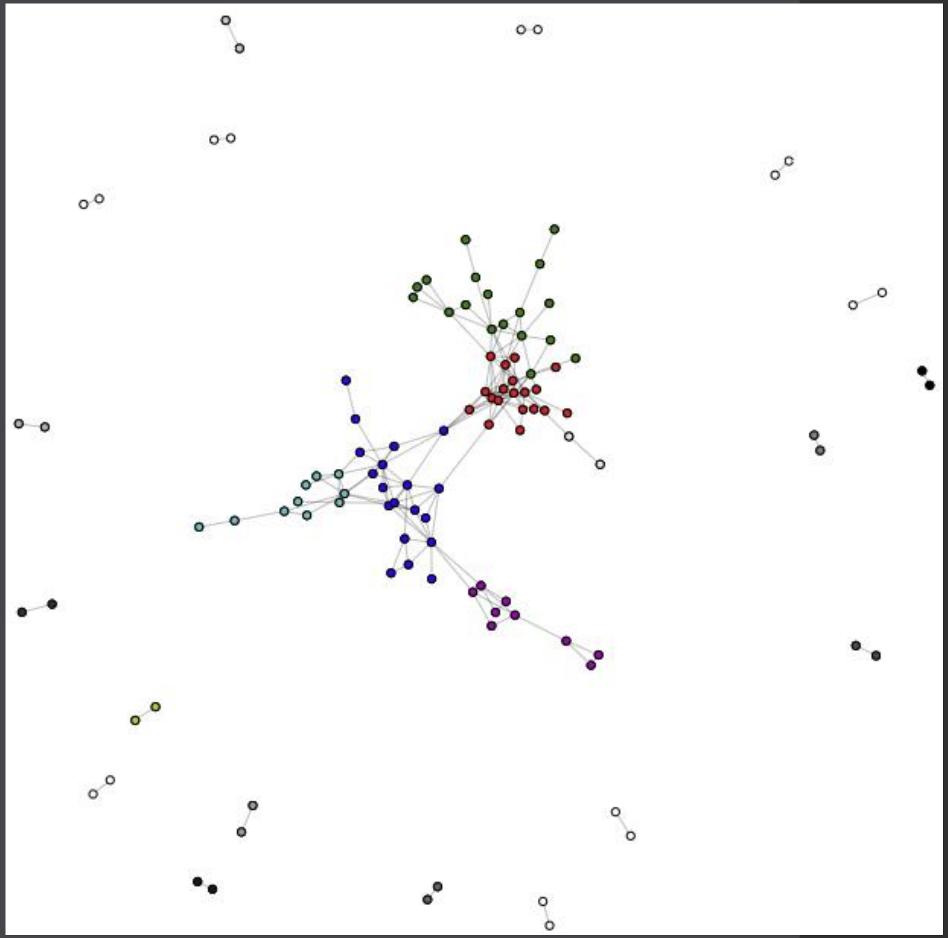


4. Community Detection

Once created the graph, we can run different algorithms to find some clusters.

Lovuain (unweighted)

	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6	Cluster 7	Cluster 8	Cluster 9	Cluster 10	...
Fraction over median degree	0.368421	0.473684	0.333333	0.400000	0.444444	0.0	0.0	0.0	0.0	0.0	...
Conductance	0.145161	0.152174	0.206897	0.225806	0.076923	0.0	0.0	0.0	0.0	0.0	...
Cut ratio	0.010078	0.007839	0.007018	0.006796	0.002137	0.0	0.0	0.0	0.0	0.0	...
Normalized cut ratio	0.208991	0.197926	0.242611	0.245636	0.082670	0.0	0.0	0.0	0.0	0.0	...



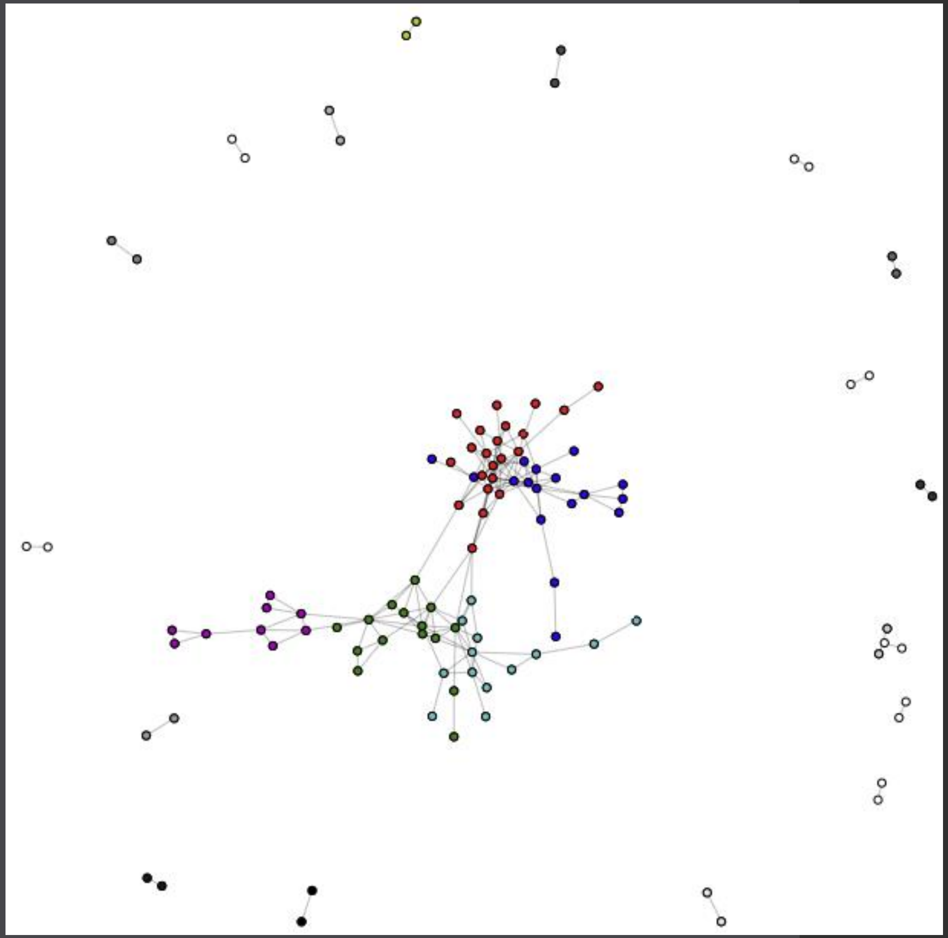
Note: a lot of the small clusters usually consist in pairs of correlated and isolated stocks, so their benchmark doesn't have significance

4. Community Detection

Results of the algorithms

Lovuain (weighted)

	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6	Cluster 7	Cluster 8	Cluster 9	Cluster 10	...
Fraction over median degree	0.409091	0.470588	0.466667	0.307692	0.444444	0.0	0.0	0.0	0.0	0.0	...
Conductance	0.159664	0.211268	0.173333	0.219512	0.076923	0.0	0.0	0.0	0.0	0.0	...
Cut ratio	0.009596	0.009288	0.008935	0.006993	0.002157	0.0	0.0	0.0	0.0	0.0	...
Normalized cut ratio	0.225866	0.257139	0.214086	0.245599	0.082703	0.0	0.0	0.0	0.0	0.0	...

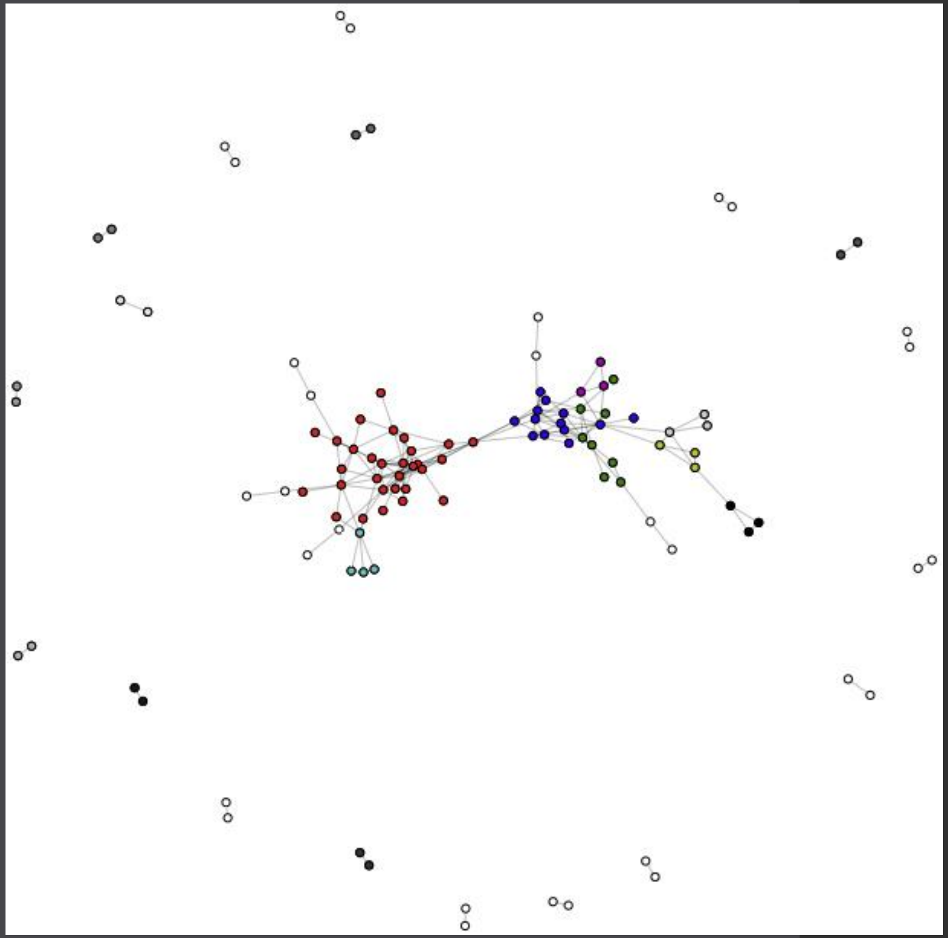


4. Community Detection

Results of the algorithms

Label Propagation (unweighted)

	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6	Cluster 7	Cluster 8	Cluster 9	Cluster 10	...
Fraction over median degree	0.433333	0.384615	0.375000	0.500000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	...
Conductance	0.058824	0.232877	0.285714	0.230769	0.333333	0.400000	0.142857	0.333333	0.0	0.0	...
Cut ratio	0.004016	0.013077	0.009524	0.006881	0.009091	0.012121	0.003030	0.009091	0.0	0.0	...
Normalized cut ratio	0.104278	0.284236	0.308061	0.239034	0.341508	0.410870	0.145597	0.341508	0.0	0.0	...

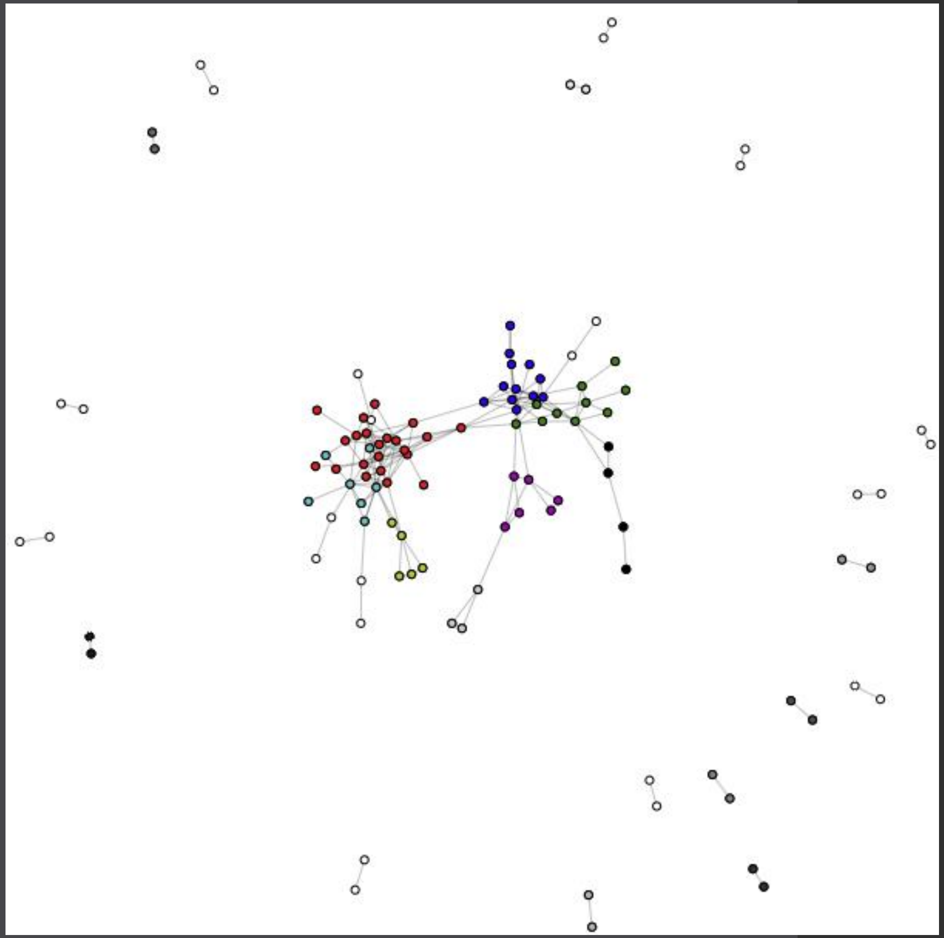


4. Community Detection

Results of the algorithms

Infomap (unweighted)

	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6	Cluster 7	Cluster 8	Cluster 9	Cluster 10	...
Fraction over median degree	0.500000	0.500000	0.300000	0.285714	0.500000	0.400000	0.500000	0.000000	0.0	0.0	...
Conductance	0.107914	0.193548	0.302326	0.448276	0.157895	0.200000	0.250000	0.142857	0.0	0.0	...
Cut ratio	0.007493	0.009901	0.012621	0.017520	0.004673	0.005556	0.004587	0.003030	0.0	0.0	...
Normalized cut ratio	0.165385	0.229693	0.339153	0.483698	0.166298	0.208310	0.255464	0.145597	0.0	0.0	...

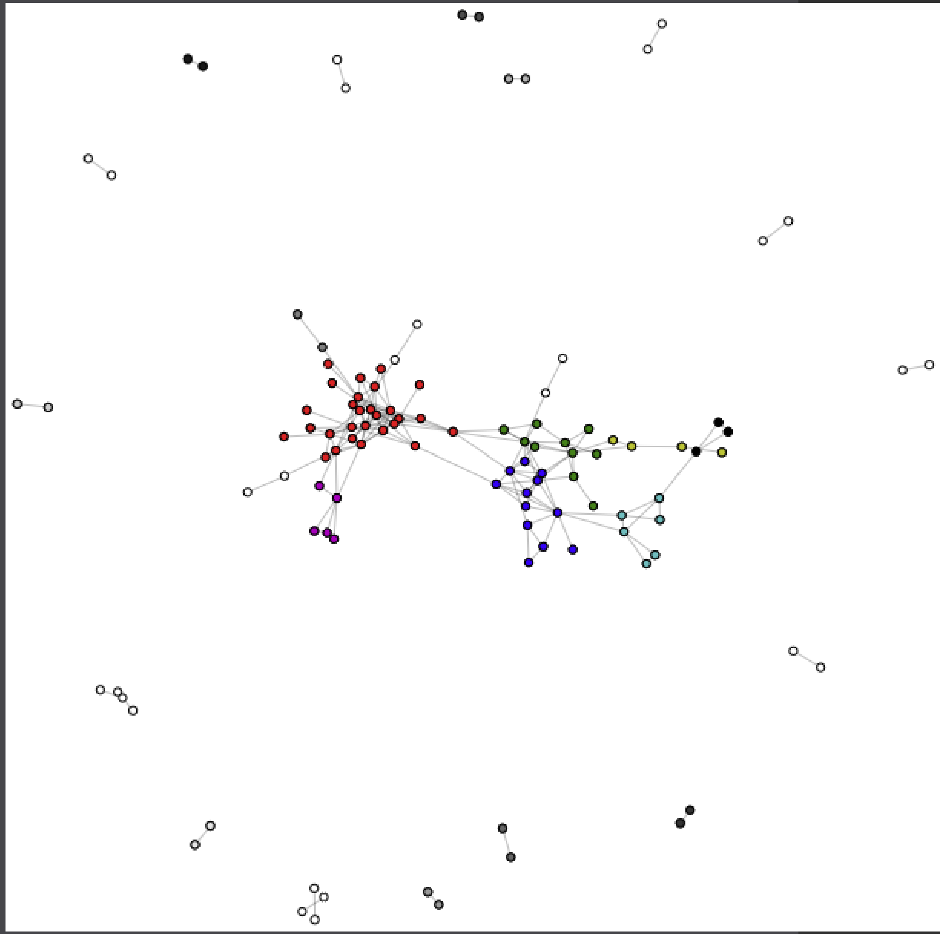


4. Community Detection

Results of the algorithms

Infomap (weighted)

	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6	Cluster 7	Cluster 8	Cluster 9	Cluster 10	...
Fraction over median degree	0.464286	0.500000	0.300000	0.500000	0.400000	0.500000	0.000000	0.0	0.0	0.0	...
Conductance	0.060241	0.193548	0.302326	0.157895	0.200000	0.250000	0.142857	0.0	0.0	0.0	...
Cut ratio	0.004252	0.010000	0.012745	0.004717	0.005607	0.004630	0.003058	0.0	0.0	0.0	...
Normalized cut ratio	0.105286	0.229912	0.339363	0.166345	0.208357	0.255495	0.145612	0.0	0.0	0.0	...

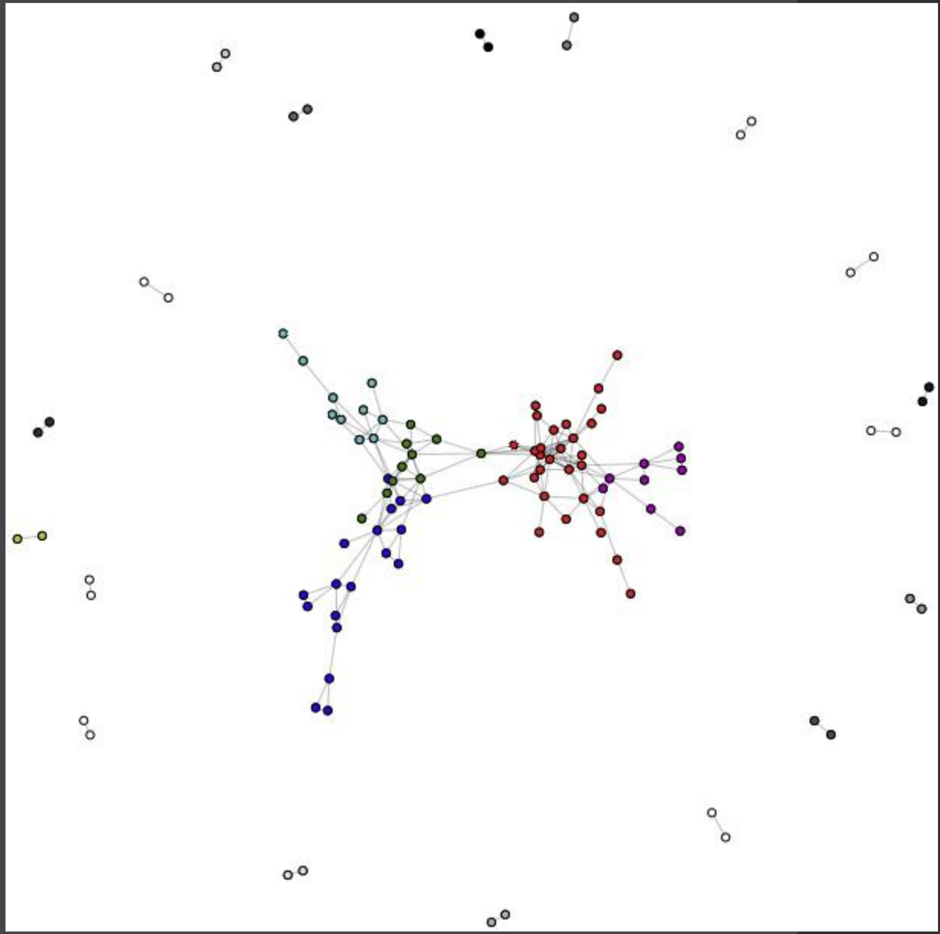


4. Community Detection

Results of the algorithms

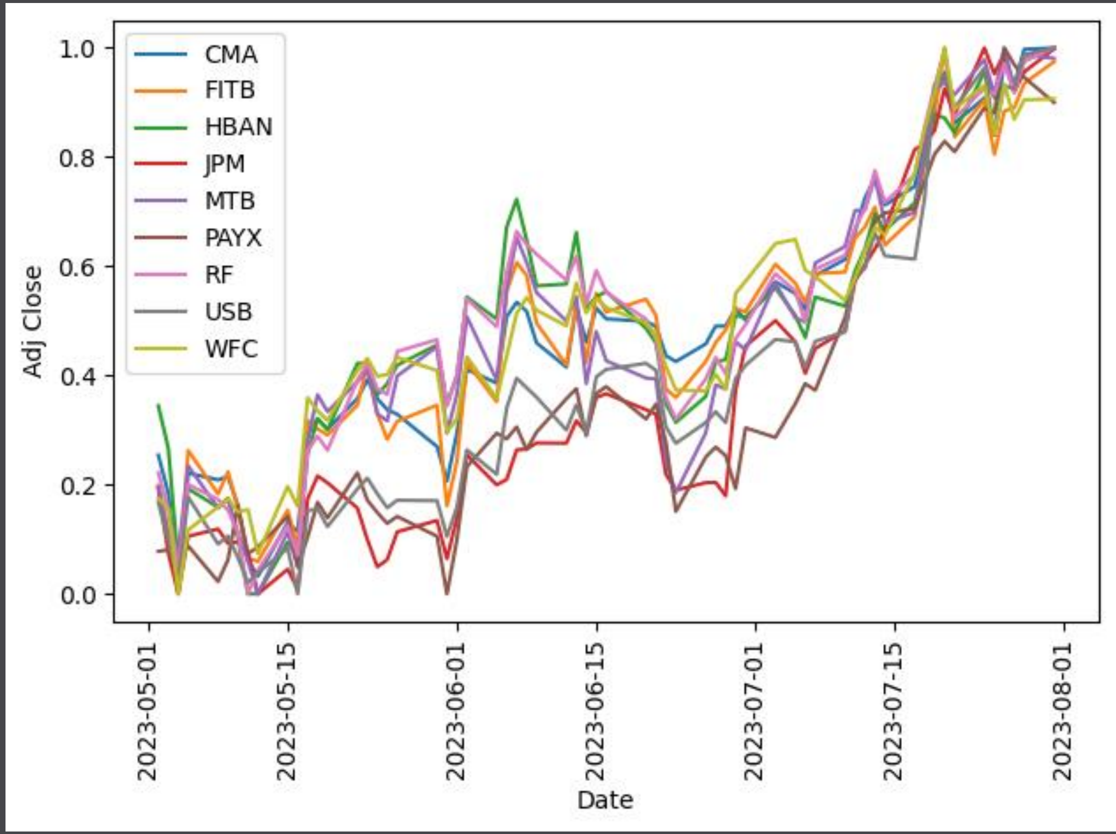
AGDL (weighted)

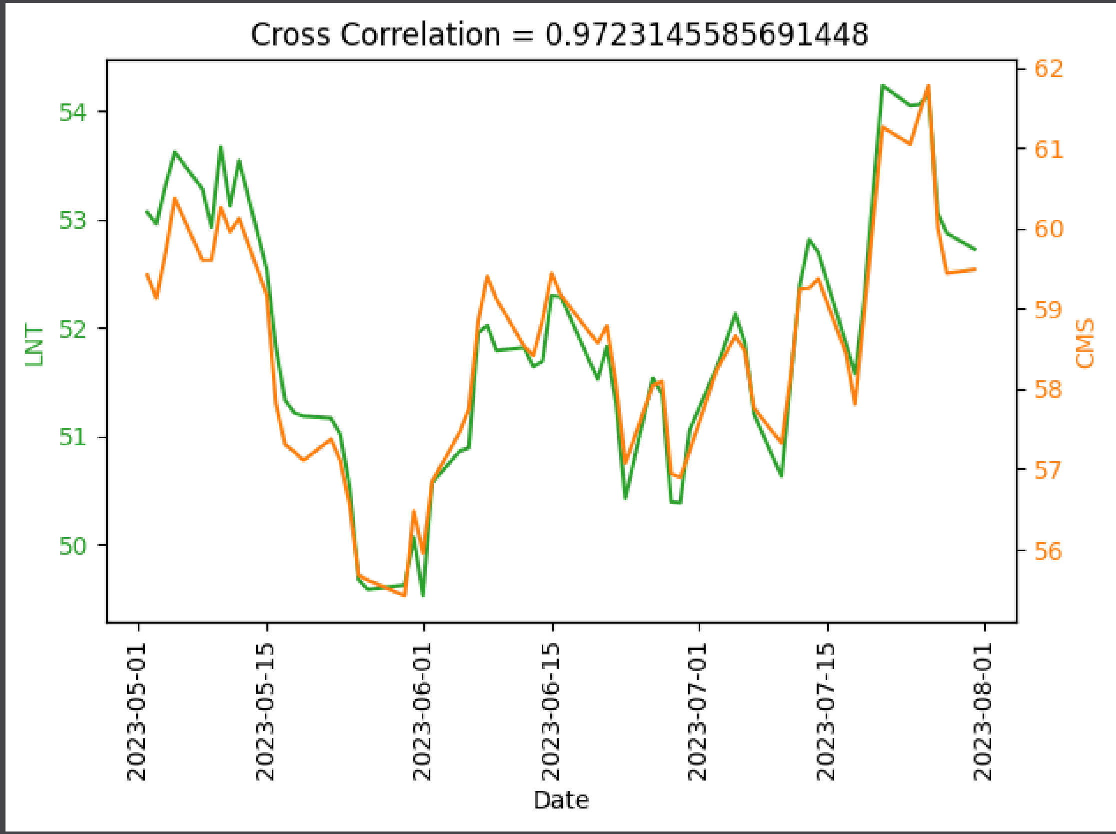
	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6	Cluster 7	Cluster 8	Cluster 9	Cluster 10	...
Fraction over median degree	0.448276	0.333333	0.300000	0.400000	0.333333	0.0	0.0	0.0	0.0	0.0	...
Conductance	0.084967	0.159420	0.346939	0.225806	0.266667	0.0	0.0	0.0	0.0	0.0	...
Cut ratio	0.005401	0.006501	0.016667	0.006863	0.008630	0.0	0.0	0.0	0.0	0.0	...
Normalized cut ratio	0.138909	0.193688	0.395097	0.245749	0.289266	0.0	0.0	0.0	0.0	0.0	...



4. Community Detection

As an example, let's visualize some of the families generated by the Louvain algorithm:





Conclusions

- The algorithms seem to output valuable stock families
- The benchmarks may be better but are justified by the dimension of the dataset
- Cross correlation looks reasonably effective as indicator of relations between stocks

Future improvements:
- A bigger dataset and/or extended period of time shall be exploited
- Further tuning of hyperparameters could give better algorithm performance
- A natural evolution of the work would be to correlate stocks at given lagged periods, in order to predict the rise or fall of a stock price in the immediate future