Data Analysis of Binance Trade History

Xuyu Cai

Caleb Qi

Boston University

Questrom School of Business

Abstract

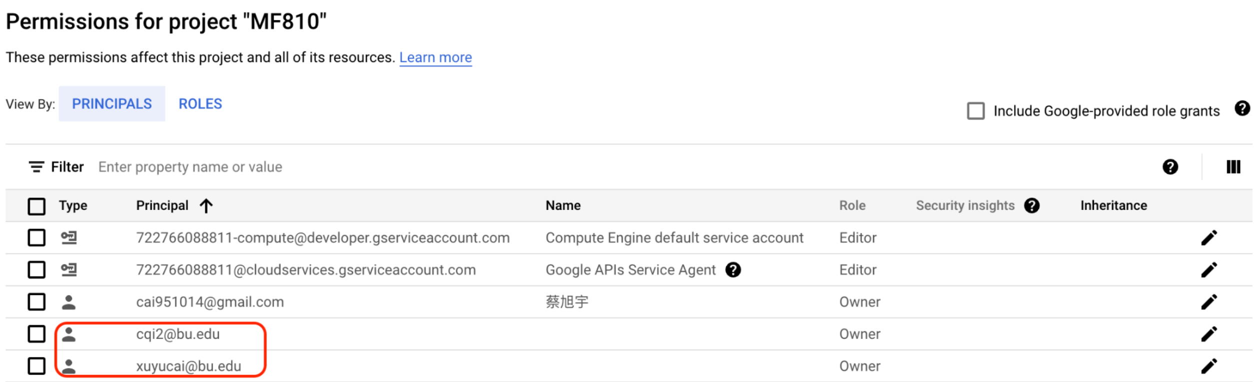
This paper proposes a data analysis methodology using Pyspark and Google Cloud. The data set we deal with comes from Binance exchange, which contains the trade history of all trading pairs of cryptos. First, we examine the structure and assess complexity of the data, highlighting the difficulties we may encounter. Then, we use pyspark for processing the data and transferring the original data into tidy data. We also apply Pyspark SQL query for data manipulation and obtain some basic statistical information of the data. Finally, we use pyspark to implement the K-means method, with the aim of exploring and classifying the market behaviors in cryptos.

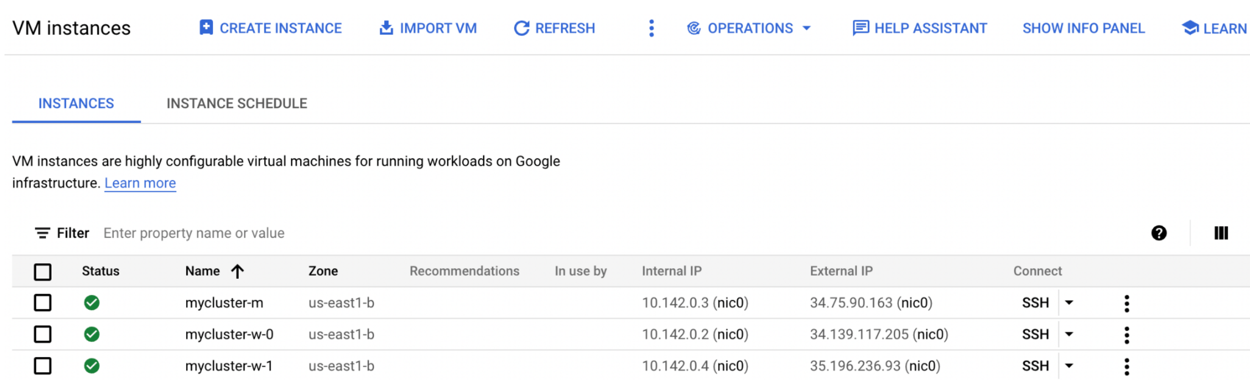
Keywords: Pyspark, Tidy data, Data manipulation, K-means

Structure and Complexity of the data

Data source and data description

The data covers the entire period during which Binance becomes the market maker of cryptos. It provides us with a collection of 1 minute candlesticks of the top 1000 cryptocurrency pairs. In order to analyze the data on Google cloud, we need to create a project and a virtual machine on Google cloud first:





We use Pyspark dataframe instead of pandas dataframe to process the data. Spark carries an easy way to use API for operation of large datasets. It not only supports ‘map reduce’, but also machine learning methodology, SQL queries, etc.

The following figure shows the trading pairs it contains:

Complexity

The first and most obvious complexity is the **size of our dataset**. The fact that our dataset consists of 1 minute candlesticks means that for each pair we will have massive amount of datapoints. This makes every line of code more time consuming, which we will need to leverage google cloud and pyspark to mediate. As a result we chose the pairs that contain the stablecoin USDT (259 total pairs).

A second complexity is that our data **does not contain a pair ID**. Our dataset came in the form of 1000 separate .parquet files, one for each pair. This requires the additional step of taking into account the pair identity for each datapoint in a way that follows tidy data requirements. This creates difficulties when grouping by pairs, which we ran into multiple times in our processing.

Another complexity comes from the extremely **different means and variances** in the features for each pair. This could be due to the nature of the pairs, but also due to extreme outliers, which we would need to take care of in a reasonable way. Taking this into consideration will mean we might have to scale the data for all the pairs somehow so that they are comparable.

As an example, these are the summary statistics for the volume of five pairs.

pair	count	mean	std	min	%25	%50	%75	max
AAVEDOWN-USDT.par...	500702	915186.4493402537	5635454.308337428	0.0	60.88999938964844	22073.734375	167287.67578125	5.23223328E8
1INCH-USDT.parquet	635183	9296.258355739477	22943.61749030991	0.0	1688.094970703125	3948.89990234375	9091.21533203125	2156054.0
AAVEUP-USDT.parquet	502157	1048.035569555473	7904.621297131593	0.0	2.0299999713897705	65.13999938964844	780.1900024414062	1869921.5
AAVE-USDT.parquet	738631	158.17117874175608	320.4647179258576	0.0	26.16925048828125	67.01239776611328	165.4029998779297	31506.877
ACM-USDT.parquet	548503	302.66560987569676	1852.5812180102423	0.0	0.0	20.863000869750977	156.5	466814.66

Finally, below is the schema of one example pair in our dataset (ACM-USDT.parquet) after adding the pair identifying column.

```
root
 |-- open: float (nullable = false)
 |-- high: float (nullable = false)
 |-- low: float (nullable = false)
 |-- close: float (nullable = false)
 |-- volume: float (nullable = false)
 |-- quote_asset_volume: float (nullable = false)
 |-- number_of_trades: integer (nullable = true)
 |-- taker_buy_base_asset_volume: float (nullable = false)
 |-- taker_buy_quote_asset_volume: float (nullable = false)
 |-- open_time: timestamp (nullable = true)
 |-- pair: string (nullable = false)
```

Data Processing Technologies

Google Cloud

One part of google cloud we used was storage**.** Due to the abnormally large sized data, we used google cloud to store our 28 GB of data in the cloud storage section of google cloud. This brings the data closer to the processing, so that when we use dataproc to run a jupyter notebook, we can deliver the data efficiently and faster. The cost of storing data was very minimal, even with the standard option. We chose the South Carolina (us-east1) storage region, which cost $0.02 per GB per month for the standard storage option.

Another area of google cloud we used was computation. We used dataproc to create a cluster to run on Google Cloud. The reasoning is that our laptops simply do not have enough RAM to cache gigabytes of data. We used e2-highmem-4 machines for both our master and worker machines. We created 4 worker machines, so a total of 5 machines, each with 4 CPUs and 32 GB of memory. The creation of many clusters in dataproc allows us to process data faster and more efficiently than on our computers. In total, we used around $10 of our $50 allowance in our school accounts.

Spark

Our use of spark was very crucial for our analysis. Traditional pandas dataframe does not scale as well as spark does. This is due to the fact that pyspark is capable of parallel and in memory processing with the concept of map reduce. As an example, if we want to compute the mean volume for each pair, pyspark is capable of splitting the volume data into chunks for each pair with the map function, and reducing each chunk of those data simultaneously by taking the mean. This is much faster than iterating over the chunks with a for loop.

Data manipulation

Pre-processing

We found that there are many nulls in our data. Moreover, some rows cannot provide effective information. For example, if no trade happens in one minutes, the price will not change. Hence, we need to delete these rows and also the rows which don't have enough trading volumes.

After we finish the data processing, the data has the structure we desired. Then we used spark dataframe and pyspark SQL queries to get more insights of our data and prepare for the K-means analysis. First, we check the structure using the ‘schema’ command and the ‘description’ command to get the shape of the data and obtain basic statistics of the whole data (including all crypto pairs).

```
root
 |-- open: float (nullable = true)
 |-- high: float (nullable = true)
 |-- low: float (nullable = true)
 |-- close: float (nullable = true)
 |-- volume: float (nullable = true)
 |-- number_of_trades: integer (nullable = true)
 |-- open_time: timestamp (nullable = true)
 |-- pair: string (nullable = false)
```

```
+-------+------------------+------------------+------------------+------------------+
|summary|              open|              high|               low|             close|
+-------+------------------+------------------+------------------+------------------+
|  count|         259224519|         259224519|         259224519|         259224519|
|   mean|320.37616322724466| 320.6724286094436| 320.0752100316714| 320.3760722789879|
| stddev| 3068.485539876243|3071.3221821332572|3065.6224207709392|3068.4870095457914|
|    min|           1.86E-6|           1.86E-6|           1.85E-6|           1.85E-6|
|    max|           93784.0|           95000.0|          92307.08|          93935.04|
+-------+------------------+------------------+------------------+------------------+
```

```
+--------------------+-----------------+------------------+
|              volume| number_of_trades|              pair|
+--------------------+-----------------+------------------+
|           259224519|        259224519|         259224519|
|2.7470807176305085E7|55.02120888880886|              null|
|1.5838630786646495E9|262.3537753796993|              null|
|                 0.0|                0|1INCH-USDT.parquet|
|        1.5729488E12|            65486|  ZRX-USDT.parquet|
+--------------------+-----------------+------------------+
```

We can see that the differences between historical low and high price are significantly high in most crypto pairs, indicating that the crypto market is very volatile. Then we want to check the performance of cryptos with the largest market values (BTC, ETH, DOGE, XRP, etc.). We examine the pairs which incorporate the stable coin (USDT as an example).

summary	open	high	low	close	volume	number_of_trades	pair
count	2394622	2394622	2394622	2394622	2394622	2394622	2394622
mean	18419.6737294236	18432.9880994136	18406.19630208668	18419.672507131527	34.93366998975672	538.774393620371	null
stddev	17570.046161613813	17582.33781994292	17557.80397081218	17570.05961606099	57.295502140951	810.9079146585232	null
min	2830.0	2830.0	2817.0	2817.0	0.0	0	BTC-USDT.parquet
max	69000.0	69000.0	68786.7	69000.0	3564.1394	55181	BTC-USDT.parquet

summary	open	high	low	close	volume	number_of_trades	pair
count	2394621	2394621	2394621	2394621	2394621	2394621	2394621
mean	974.7873966803892	975.6615935779329	973.9063215722817	974.7859881150247	377.6375321825056	325.8132731651481	null
stddev	1203.7137921305136	1204.6799139654906	1202.7480350280835	1203.7148805676763	697.050615079679	632.5783834511692	null
min	82.02	82.08	81.79	82.03	0.0	0	ETH-USDT.parquet
max	4865.22	4868.0	4861.38	4865.22	35632.598	40469	ETH-USDT.parquet

summary	open	high	low	close	volume	number_of_trades	pair
count	1408924	1408924	1408924	1408924	1408924	1408924	1408924
mean	0.08784499983656371	0.08798172362716916	0.08770742227854617	0.08784523330530292	1584175.7195584392	285.01152936567195	null
stddev	0.1234813436703236	0.12371859466219677	0.1232427116800121	0.12348149099952002	7190049.797550385	1168.3455206593367	null
min	0.0011454	0.0011488	0.0011345	0.0011488	0.0	0	DOGE-USDT.parquet
max	0.73768	0.73995	0.73413	0.73805	6.1841984E8	64287	DOGE-USDT.parquet

summary	open	high	low	close	volume	number_of_trades	pair
count	2022660	2022660	2022660	2022660	2022660	2022660	2022660
mean	0.4874010090864216	0.4879019611325577	0.4868921127369384	0.4873993385950855	248707.20259080132	212.1194788051378	null
stddev	0.31761871879550063	0.31810074723717024	0.31713709866487216	0.3176182561997884	650436.2584885011	543.9787205020785	null
min	0.10573	0.10684	0.10129	0.10589	0.0	0	XRP-USDT.parquet
max	1.96471	1.96689	1.95835	1.96471	6.3661784E7	40535	XRP-USDT.parquet

Depolarizing/Outlier Processing

As previously mentioned, we noticed that for each pair there were many outliers. We believe that these outliers will not contribute useful information for clustering the pairs, because an outlier for one pair might end up in the middle of the data for another pair. Consequently, we depolarized the data by removing outliers greater or less than 3 standard deviations for volume. Since we want to “clump” our data together, we also recentered the data on their means for each pair.

Due to the logic statements involved in depolarizing/recentering the data, (i.e. if they were more than 3 standard deviations away from the mean), we could not use the groupby function. Instead, we created a user defined function that inputs and outputs a series, performing our depolarizing and recentering over the series. We then called the function over each pair with the window package and with functions like when().otherwise().

After depolarizing/recentering the data, we saved these data into new dataframes that shared the open_time column with the main dataframe. This allows us to join the new data onto the main dataframe.

open	high	low	close	volume	number_of_trades	pair	feature2	feature3	open_time	depolar	d_return	normalized	n_return
6.51	6.6	6.472	6.6	1098.58	23	AAVEDOWN-USDT.par...	0.01382483449869108	0.29687633877847525	2020-11-26 13:58:00	2598509.433460359	5.011963437991066...	0.6969636778549227	0.04528278608162539
6.51	6.6	6.472	6.6	1098.58	23	AAVEDOWN-USDT.par...	0.01382483449869108	0.29687633877847525	2020-11-26 13:58:00	2598509.433460359	0.013443542326877464	0.6969636778549227	2.5464055511076897
6.51	6.6	6.472	6.6	1098.58	23	AAVEDOWN-USDT.par...	0.01382483449869108	0.29687633877847525	2020-11-26 13:58:00	176.181925951993	5.011963437991066...	-0.42120745326340325	0.04528278608162539
6.51	6.6	6.472	6.6	1098.58	23	AAVEDOWN-USDT.par...	0.01382483449869108	0.29687633877847525	2020-11-26 13:58:00	176.181925951993	0.013443542326877464	-0.42120745326340325	2.5464055511076897
7.1	7.1	7.086	7.1	813.2	26	AAVEDOWN-USDT.par...	0.0	1.0	2020-11-26 19:55:00	2598224.0535165113	-0.00536929072770...	0.696840866966665	-0.954959832971643
7.1	7.1	7.086	7.1	813.2	26	AAVEDOWN-USDT.par...	0.0	1.0	2020-11-26 19:55:00	2598224.0535165113	-3.81292171813615...	0.696840866966665	-0.02661451524308...
7.1	7.1	7.086	7.1	813.2	26	AAVEDOWN-USDT.par...	0.0	1.0	2020-11-26 19:55:00	356.79893034652423	-0.00536929072770...	-0.4211297262359006	-0.954959832971643
7.1	7.1	7.086	7.1	813.2	26	AAVEDOWN-USDT.par...	0.0	1.0	2020-11-26 19:55:00	356.79893034652423	-3.81292171813615...	-0.4211297262359006	-0.02661451524308...
5.291	5.291	5.291	5.291	67.98	22	AAVEDOWN-USDT.par...	0.0	null	2020-11-27 04:26:00	2597478.8335076612	-3.81292171813615...	0.6965201677262054	-0.02661451524308...
6.031	6.076	6.031	6.072	51.97	34	AAVEDOWN-USDT.par...	0.006798190754466899	0.0	2020-11-27 11:36:00	2597462.823505525	-8.30908780845981...	0.696513277954777	-0.11029526797212663

only showing top 10 rows

In the above table, depolar is the depolarized volume, normalized is the normalized and depolarized volume. d_return is the depolarized return, and n_return is the normalized and depolarized return.

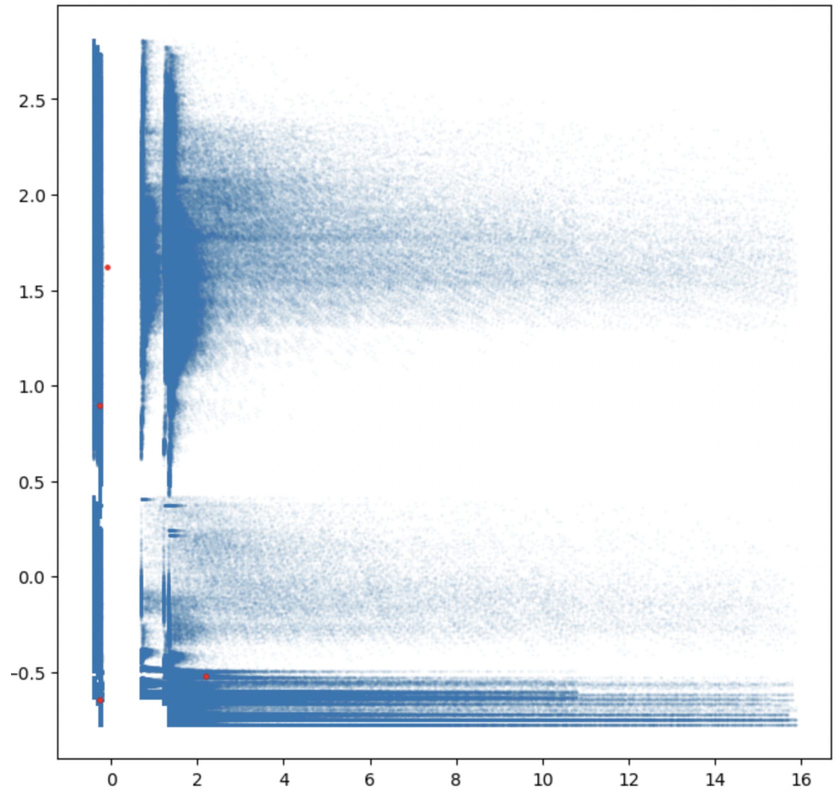
K-means Algorithms

We decided to use K-Means to tackle our cluster problem because it is easy to explain how spark optimizes every step of the K-Means algorithm with K-Means.

- In our code, the closest point function is an action, because each point in the array is summarized into one point (the closest point) after taking in data.
- The reduce by key function is a transformation because it transforms the closest point by reducing it.
- The new point map is also a transformation because it creates a new point from the closest point.
- When we update tempdist, we also perform an action, because it summarizes statistics for all the points by summing their distance.
- Lastly, when we remap the centers, we do another transformation because each point in the RDD is mapped to a new point.

Pyspark is able to perform these actions and transformations of our dataset much better than sklearn due to the usage of mapreduce built in pyspark. Consequently, our Spark ML version of K-Means should run faster than the sklearn version of K-Means.

In order to provide an example, we only use the crypto pairs that contain USDT(one of the stable coins in the crypto market) because it can reflect the relative value of cryptos. We choose two features to train the model. The first one is the return in one minute. The other one is the trading volume in one minutes. Both two features are depolarized and standardized. We also choose K = 3 or 4 to describe the market states. Theoretically, we can divide the market states in four categories: bull, bear, mild bull, mild bear. In the first two states, the price of cryptos are either increasing or decreasing significantly. We also call the last two states a static market, which indicates that the market has no dramatic fluctuations. In our analysis, we expected the one-minute candlesticks in our data can be exactly fitted into these four categories.



The figure above shows the results of K-means clustering. The X-axis represents the volume after depolarization and standardization. The Y-axis represents the returns after depolarization and standardization. The red spots highlight the centers for each cluster. We can easily determine the red spots on top left and bottom right. These two clusters mean that the market is in a static state (mild decreasing or mild increasing). The red spot on top right is the center of the cluster which represents the bull market. Both trading volume and returns are high. Last, the spot on bottom right represents the state in which the market goes through bear periods.

Reference

1. Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. In *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10)*.

2. Wagstaff, K., Cardie, C., Rogers, S., & Schrödl, S. (2001). Constrained k-means clustering with background knowledge. In *Icml* (Vol. 1, pp. 577-584).

3. Silge, J., & Robinson, D. (2016). tidytext: Text mining and analysis using tidy data principles in R. *Journal of Open Source Software*, *1*(3), 37.

4. Fil, M., & Kristoufek, L. (2020). Pairs trading in cryptocurrency markets. *IEEE Access*, *8*, 172644-172651.

5. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to algorithms*. MIT press.