Analysis Report

Kaihui LUO

September 16, 2024

1 Introduction

In this report, I will give the results of the required figures of the dataset. In this process, I also explain the code I wrote. Finally, an investigation regarding the round number effect will be conducted using a hypothesis test with $\alpha = 0.05$.

1.1 Required Python Packages

For the analysis of the data, I use pandas, numpy, matplotlib.pyplot, datetime and statsmodels.api. Pandas, numpy, matplotlib.pyplot and datetime are packages of common usage for data analysis. Statsmodels.api is for the hypothesis test on the round number effect. In my code, pandas is imported as pd, numpy is imported as np, matplotlib.pyplot is imported as plt, and statsmodel.api is imported as sm. These abbreviations all follow the convention.

2 Points requiring additional operations

As already indicated in the description of the problem, the analysis should exclude the two auction periods and also take into consideration the change of the date. I will explain how I deal with these two issues in this section.

2.1 Exclusion of the auction period

This is easy to solve, I use the slice of DataFrame from pandas to select only the data with no condition code or has condition code 'XT'. The code is as follows:

```
df_selected = df[df['Condition codes'].isnull() \
| df['Condition codes'].isin(['@1', 'XT'])]
```

In the above, '@1' denotes the end of a line in the given CSV file. I include it here because there is one stock(BBHBEAT Index) in the file that does not sell on the market so data of this stock does not have condition codes.

2.2 Selection of data on the same day

I resolve this issue by groupby method in pandas. Depending on the situation, I apply it for multiple times. Here I give one example code for illustration:

```
diff_trade = df.reset_index().groupby('Date')['index'] \
.apply(lambda x:x.diff()).dropna()
```

In the above, df has an index of dtype datetime generated from the dataset.

3 Data preprocessing

In this section, I illustrate briefly how I preprocess the data.

3.1 Retrieve the data

After the attempt to directly read the CSV file, I notice that there is a mismatch in the column numbers. Therefore, I use the following code to retrieve the data:

```
df = pd.read_csv("./scandi.csv", usecols=range(12), header=None)
cc = pd.read_csv("./scandi.csv", usecols=[14], header=None)
df = df.join(cc)
```

3.2 Improve the readability of data

Then I assign a name to each column for better readability as follows:

```
df.drop([1, 9], axis=1, inplace=True)
df.columns = ['Stock identifier', 'Bid Price', 'Ask Price', 'Trade Price',
'Bid Volume', 'Ask Volume', 'Trade Volume', 'Update type', 'Date',
'Time in seconds past midnight', 'Condition codes']
```

After selecting the data in the required period as in section 2.1, I set the index of the selected dataframe to be the datetime derived from the column 'Date' and 'Time in seconds past midnight':

```
df_selected.index = pd.to_datetime(df_selected['Date']) + \
df_selected['Time in seconds past midnight'].apply(lambda x: datetime.timedelta(seconds=x))
```

3.3 Group the data

Finally, by using groupby to group the data into each stock, the preprocess is completed:

```
grouped = df_selected.groupby('Stock identifier')
```

4 Result of the analysis

In this section, I give my results on the analysis. For simplicity, I only give the results for the first 10 stocks here, as the rest can be checked in the CSV file I attached in the same folder with the name resultData.csv.

4.1 Trading time

For the statistics on the trading time, the results are as follows:

	trading_time_avg	trading_time_med	trading_max_time
ABB SS Equity	10.207768	0.0	490.0
AKA NO Equity	51.08945	0.0	1454.0
AKSO NO Equity	18.924541	0.0	807.0
ALFA SS Equity	5.528354	0.0	447.0
AMEAS FH Equity	14.927337	0.0	636.0
ASSAB SS Equity	8.393114	0.0	537.0
ATCOA SS Equity	5.8010909999999996	0.0	349.0
ATCOB SS Equity	17.703741	0.0	611.0
AZN SS Equity	17.177229	0.0	483.0
BOL SS Equity	6.489688	0.0	301.0

One thing worth noticing is that the median is all 0 above, which indicates that various trades can be completed at the same time which is a sign of high liquidity.
For the analysis of the trading time, I use the following code:

```
    for stock, df in grouped:
        df = df[df['Update type'] == 1].sort_index()
        if len(df) == 0:
            stock_not_liquid.append(stock)
        else:
            diff_trade = df.reset_index().groupby('Date')['index'] \
            .apply(lambda x:x.diff()).dropna()
            trading_max_time[stock] = np.max(diff_trade).total_seconds()
            trading_time_avg[stock] = np.mean(diff_trade).total_seconds()
            trading_time_med[stock] = diff_trade.median().total_seconds()
```

The idea is to choose first the data with trade completed and then use groupby('Date') to restrict the calculations to the same day. The if-else statement is to check if any stocks are not active in the market, in this case, this stock is BBHBEAT Index.

4.2 Tick changing time

I consider tick-changing time as the change in the trade price compared to the price of the last completed trade.
The data for the first ten stocks are as follows:

	tick_time_avg	tick_time_med	tick_max_time
ABB SS Equity	59.599409	32.0	780.0
AKA NO Equity	125.004813	30.0	1454.0
AKSO NO Equity	45.880017	9.0	807.0
ALFA SS Equity	26.948176	3.0	796.0
AMEAS FH Equity	52.868271	10.0	1119.0
ASSAB SS Equity	80.713524	27.0	984.0
ATCOA SS Equity	29.836699	13.0	590.0
ATCOB SS Equity	62.758942	33.0	747.0
AZN SS Equity	90.030483	54.0	910.0
BOL SS Equity	34.590285	17.0	579.0

From the above data, one obvious change is that the median is no longer 0, which is reasonable as from the last section the majority of one trade is completed together with some other trade at the same time, and trade happening at the same time should have the same trade price in general.
For the code to analyze the data, I write:

```
for stock, df in grouped:
    df = df[df['Update type'] == 1].sort_index().reset_index()
    if len(df) > 0:
        tick_diff = df.groupby('Date')['Trade Price'].diff().fillna(0)
        idx = tick_diff != 0
        time_diff = df[['index', 'Date']][idx].groupby('Date')['index'].diff().dropna()
        tick_time_avg[stock] = np.mean(time_diff).total_seconds()
        tick_time_med[stock] = time_diff.median().total_seconds()
        tick_max_time[stock] = np.max(time_diff).total_seconds()
```

The idea is to first select the index with the nonzero difference in trade price, and use this index to reselect the data to compute the changing time of the trade price. In this step, we need to apply groupby('Date') again to restrict the computation of the data to the same day.

4.3 Bid-ask spread

The data for the bid-ask spread is as follows:

	spread_bid_ask_avg	spread_bid_ask_med
ABB SS Equity	0.08730468962072582	0.10000000000002274
AKA NO Equity	0.0029621136113897883	0.060000000000002274
AKSO NO Equity	0.06313254364789761	0.14999999999999858

ALFA SS Equity	0.07566027982631504	0.10000000000002274
AMEAS FH Equity	0.022436073687104742	0.030000000000001137
ASSAB SS Equity	0.1669915226575396	0.5
ATCOA SS Equity	0.0285409919507043	0.19999999999998863
ATCOB SS Equity	0.11753028505697484	0.20000000000004547
AZN SS Equity	0.6650177470577247	0.5
BOL SS Equity	0.06727795000142214	0.19999999999998863

It is rather intuitive to analyze the bid-ask spread as this does not involve data through different days. I skip the code snippet for simplicity.

4.4 Round number effect

To analyze this effect, I investigate the frequency of the last digit and the data is as follows:

	last_digit_vol_trade_freq	last_digit_prx_trade_freq
ABB SS Equity	0.3670423575143599	0.12255013735517986
AKA NO Equity	0.21542210118962304	0.017438249486407 72
AKSO NO Equity	0.19377233855423942	0.01066636725986639
ALFA SS Equity	0.19374870769866978	0.13311737542215177
AMEAS FH Equity	0.20357760921250545	0.00959096863931165
ASSAB SS Equity	0.1413356710529997	0.4896527926734871
ATCOA SS Equity	0.21142126149175478	0.1137036219578662
ATCOB SS Equity	0.2272547539451454	0.11275567058485209
AZN SS Equity	0.3019316271249767	0.5080851858770783
BOL SS Equity	0.25602229743181365	0.12176559255993857

The code is as follows:

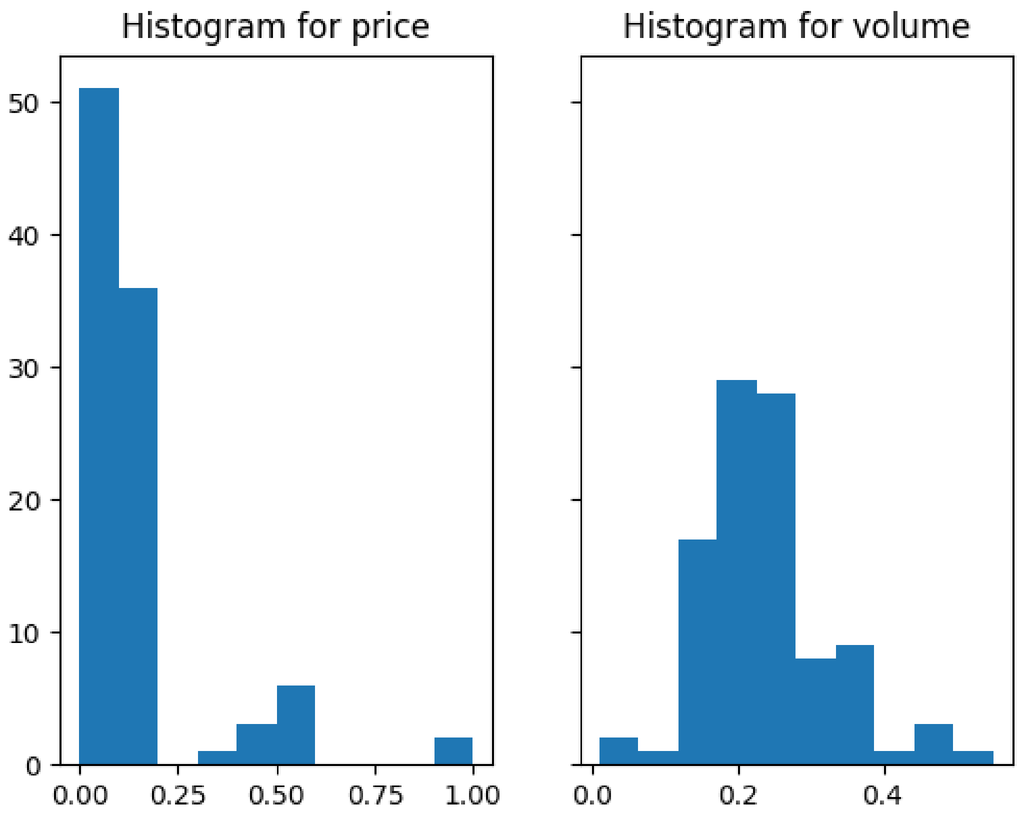
```
for stock, df in grouped:
    if stock not in stock_not_liquid:
        vol_trade_num = df['Trade Volume'].map(lambda x:str(x)[-1])
        prx_trade_num = df['Trade Price'].map(lambda x:str(x)[-1])
        if (vol_trade_num == '0').any():
            digit_vol_trade_freq[stock] = vol_trade_num.value_counts().loc['0'] / len(df)
        else:
            digit_vol_trade_freq[stock] = 0


        if (prx_trade_num == '0').any():
            digit_prx_trade_freq[stock] = prx_trade_num.value_counts().loc['0'] / len(df)
        else:
            digit_prx_trade_freq[stock] = 0
```

The idea is to first find the last digit, here I first shift the value to a string and then take the last digit. Then by using *value_counts* from pandas, I can deduce the frequency of '0'.

5 Test on the round number effect

First of all, to gain an understanding of the data, I visualize the data of the frequency for volume and price:



As can be seen from the graph, the trade volume should have the effect but the trade price remains unclear.
To investigate quantitatively, I use the hypothesis test with an alpha equal to 0.05.
The result is as follows:

	vol_test_res	prx_test_res
ABB SS Equity	True	True
AKA NO Equity	True	False
AKSO NO Equity	True	False
ALFA SS Equity	True	True
AMEAS FH Equity	True	False
ASSAB SS Equity	True	True
ATCOA SS Equity	True	True
ATCOB SS Equity	True	True
AZN SS Equity	True	True
BOL SS Equity	True	True

As can be seen from the first ten stocks, the effect for volume is much more obvious than that of price.
The code for analysis is as follows:

```
for stock, df in grouped:
    if stock not in stock_not_liquid:
        vol_trade_num = df['Trade Volume'].map(lambda x: str(x)[-1])
        prx_trade_num = df['Trade Price'].map(lambda x: str(x)[-1])

        success_vol = (vol_trade_num == '0').sum()

        success_prx = (prx_trade_num == '0').sum()

        _, p_score_vol = sm.stats.proportions_ztest(success_vol, len(df), 0.1, \
        alternative='larger')
        _, p_score_prx = sm.stats.proportions_ztest(success_prx, len(df), 0.1, \
        alternative='larger')

        prx_test_res[stock] = p_score_prx < 0.05  # True if we reject H0

        vol_test_res[stock] = p_score_vol < 0.05  # True if we reject H0
```

The snippet above applies the function *stats.proportions_ztest* from statsmodel.api to compute the z-score and then compare it to our alpha. The null hypothesis is that the frequency is less than or equal to 0.1 and the alternative hypothesis is that the frequency is larger than 0.1. Therefore, I assign True to the stock when the null hypothesis is rejected.