Guaranteed VWAP Project

Algorithmic Trading & Quantitative Strategies

Ju Hyung Kang

jk8448

0. Preprocess

Dates

- 20070703 is removed since the trading data is not full.

Tickers

- To secure liquidity, if there is a 30-minute bucket with less than 100 trades, the ticker is removed.

- For comparability and coherency, the tickers that have data for full days (64 days excluding 20070703) are used.

- In total, 327 tickers are used.

Trading Data

- The trading data is merged into 1-min data for simplicity, coherency, and continuity (.

- A VWAP is used for a one-minute bucket price.

- If there is no trade for a one-minute bucket, the missing price is back-filled. If there still are missing prices, the missing prices are forward-filled.

1. Volume Model

The purpose of the volume model is to estimate the total daily volume and the proportion of the volumes for each 30-minute bucket.

Estimating Total Daily Volume

To estimate total daily volume, a linear regression is used putting i-day (for i=1,2,3,4,5) lagging total daily volume values as X and the total daily volume as y.

	R-squared	0.876	**F-statistics**	2.721e+04
	Coefficient	**Standard error**	**t**	**P>|t|**
Lag 1	0.5468	0.007	76.053	0.000
Lag 2	0.0770	0.008	9.663	0.000
Lag 3	-0.0074	0.008	-0.919	0.358
Lag 4	0.2824	0.008	35.346	0.000
Lag 5	0.0676	0.007	9.359	0.000

Table 1. Linear Regression Summary (1, 2, 3, 4, 5 lag)

Since 3-day lagging total daily volume showcases a large p-value, it is omitted.

	R-squared	0.876	**F-statistics**	3.401e+04
	Coefficient	**Standard error**	**t**	**P>|t|**
Lag 1	0.5463	0.007	76.178	0.000
Lag 2	0.0739	0.007	10.232	0.000
Lag 4	0.2793	0.007	38.638	0.000
Lag 5	0.0670	0.007	9.315	0.000

Table 2. Linear Regression Summary (1, 2, 4, 5 lag)

It is observable that the R-squared value does not decrease even though the 3-day lagging total daily volume is omitted. Also, F-statistics increased. Thus, i-day (for i=1, 2, 4, 5) lagging total daily volume values are used as features to estimate total daily volume.

Estimating Proportion of Volumes for each 30-minute Bucket

Since it is well known that daily trading volumes plot U-shaped pattern, historical mean is used to estimate proportion of volumes for each 30-minute bucket.

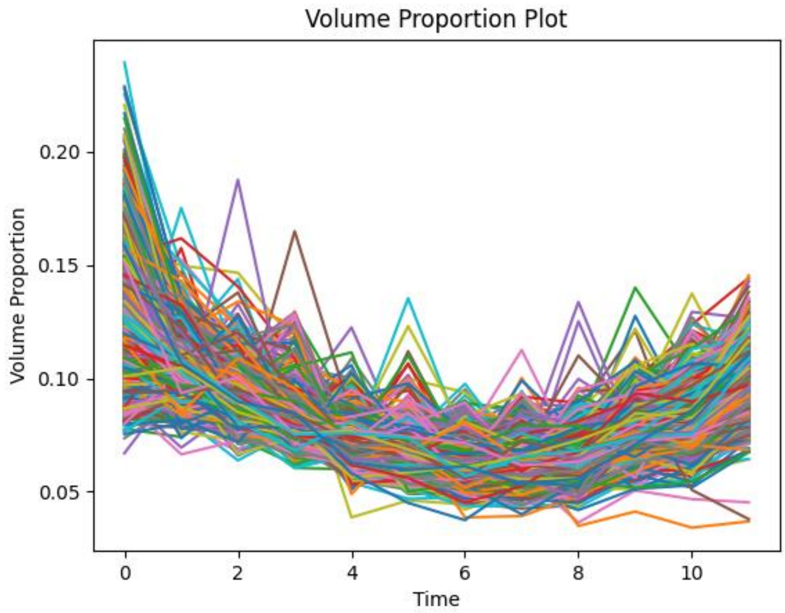


Figure 1. Volume Proportion Plot

2. Trading Model

2.1 Model Details

σ Calculation

- Considering the assumption given in the lecture note, the price volatility (in dollar unit) for 30 minute is used for σ.

h Calculation

The following equation is used to calculate h:

$$h = \sigma\eta \; sgn(X) \left|\frac{X}{VT}\right|^{\beta}$$

where X=(The proportion for the 30-mins bucket) * (The required total volumes to be traded) and VT=(The total volumes traded for the 30-mins bucket which includes X). It is assumed that h increases linearly throughout 30 minutes by continuous trading.

g Calculation

g decays linearly following the assumption given in the lecture note.

VWAP Calculation

The market VWAP and the model VWAP are calculate as follows:

$$VWAP_{market} = \frac{\sum_t (p_t + h_t + g_t) s_t^{market}}{\sum_t s_t^{market}}$$

$$VWAP_{model} = \frac{\sum_t (p_t + h_t + g_t) s_t^{model}}{\sum_t s_t^{model}}$$

Required Total Volumes to be Traded

To incorporate the cost of market impact into the model, it is assumed that 5% of total trading volume is requested to be traded.

2.2 Volume Proportion Adjustment

At the start of each 30-mins bucket, the remaining volume proportions are front/back-weighted if we under/over-bought. There are mainly two ways to adjust the proportion: fixed kappa and stochastic kappa.

Fixed Kappa

Under this approach, we apply the same kappa for a stock regardless of how big the volume errors are. Using κ, the cumulative volume can be calculated as follows:

$$\int_{t_0}^{t_k} x_s ds = 1 - \frac{\sinh(\kappa(1 - t_k))}{\sinh(\kappa)}$$

where x_t is a trading rate.

Stochastic Kappa

To keep the U-shape of the proportion, the following approach is adapted. 5 percentage points (or a minimum proportion value) are deducted from each proportion, e.g., [5%, 5%, …, 5%]. The uniform proportion is front/back-weighted with κ, resulting a weighted proportion array, e.g., [6.5%, 6.3%, …, 4.5%]. Then, the weighted proportion is added back to the model volume proportions.

κ should be reactive to the volume proportion error up to the adjustment point, weighting more when the error is large. Thus, $\kappa_t = f(e_t)$ where e_t is the volume proportion error at time t.

Assuming a linear relationship between e_t and κ_t, β_t is introduced: $\kappa_t = \beta e_t$. There is no constant term since $\kappa_t > 0$.

Using κ_t, the cumulative volume can be calculated as follows:

$$\int_{t_0}^{t_k} x_s ds = 1 - \frac{\sinh(\kappa_{t_0}(1 - t_k))}{\sinh(\kappa_{t_0})}$$

where x_t is a trading rate.

3. Fit

3.1 Training/Test Set Separation

The first 70% of dates are used as a training set, while the remaining 30% of dates are used as a test set.

3.2 Model Fit

The minimize function under scipy.optimize module is used to optimize the model parameter. The fitted parameters for the two models are as follows:

	GOOG	GS	AAPL	EEM	MA	RIMM	LVS	OIH	AMZN	BSC
Fixed κ	1.3165	0.8186	1.0098	0.8739	0.5189	1.1774	0.0000	0.7596	1.2750	0.6191
β(Stoch. κ)	3.2131	1.7718	2.2138	1.8579	0.0106	2.6321	0.0096	1.5535	2.7962	1.2435

Table 3. Fitted Parameters

3.2 Model Robustness

	Train			**Test**		
	No Adj.	Fixed κ	Stochastic κ	No Adj.	Fixed κ	Stochastic κ
GOOG	723.6779	0.3942	0.4118	724.3908	0.3144	0.3197
GS	201.4191	0.2832	0.2825	183.2226	0.1410	0.1364
AAPL	163.0215	0.2205	0.2170	170.0962	0.1898	0.1765
EEM	150.5326	0.1597	0.1635	151.9568	0.1273	0.1284
MA	156.8070	0.7847	0.8036	135.9485	0.3915	0.4297
RIMM	277.5802	0.3110	0.3184	116.5649	0.1377	0.1355
LVS	92.1196	0.3819	0.3819	114.2892	0.2292	0.2292
OIH	109.6432	0.1727	0.1813	113.3925	0.1724	0.1837
AMZN	98.5047	0.0968	0.1020	111.0577	0.0883	0.0856
BSC	125.7177	0.3887	0.3847	108.4034	0.2778	0.2791

Table 4. E+LV Values for Train and Test sets

It is observable that the E+LV values for the test set is similar to the values for the train set, which showcases the robustness of the models.

3.3 Optimal Model

Let $P = E + LV$. Then, define the improvement metric as $-\frac{P_{Stochastic\,\kappa} - P_{Fixed\,\kappa}}{P_{Fixed\,\kappa}}$.

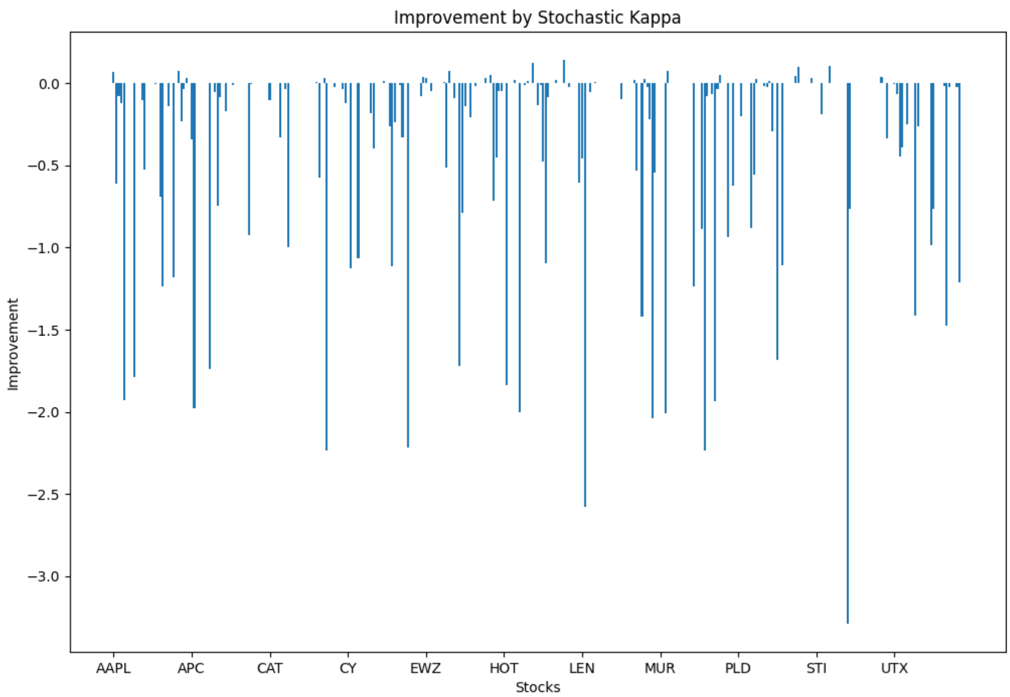


Figure 2. Improvement by Stochastic Kappa model

For the test set data, only 31.8% of stocks result in the lower E+LV value under the stochastic kappa model than under the fixed kappa model. Also, based on the plot, it is noticeable that the improvement is marginal, while the deterioration is huge.

Thus, the fixed kappa model is optimal over the stochastic kappa model.

4. Guaranteed VWAP Contract Price Formula

4.1 Volume Errors vs VWAP Price

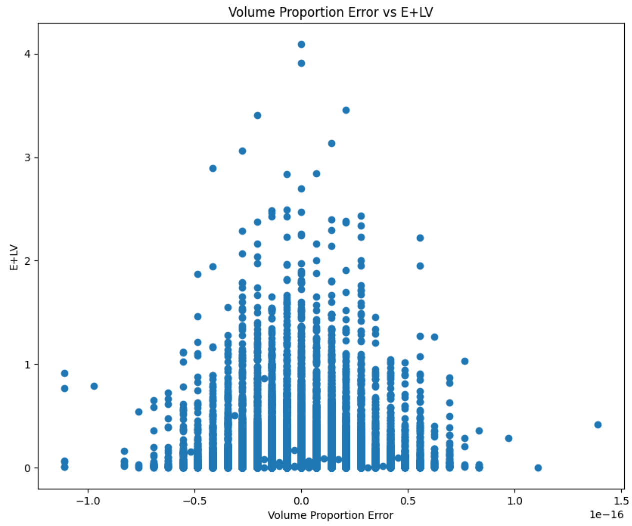


Figure 3. Volume Proportion Error vs E+LV

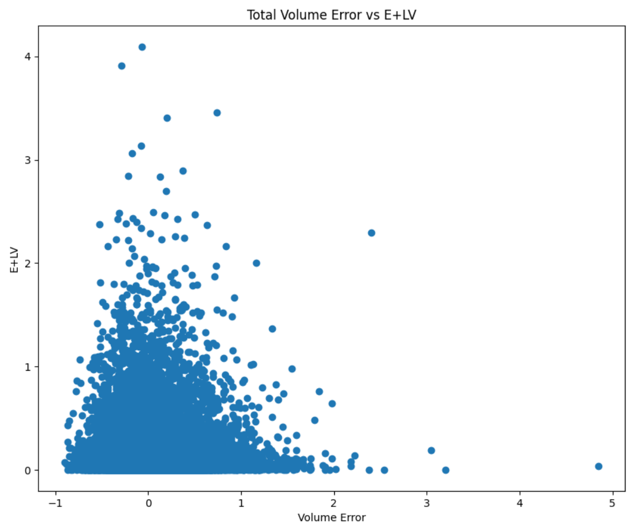


Figure 4. Total Volume Error vs E+LV

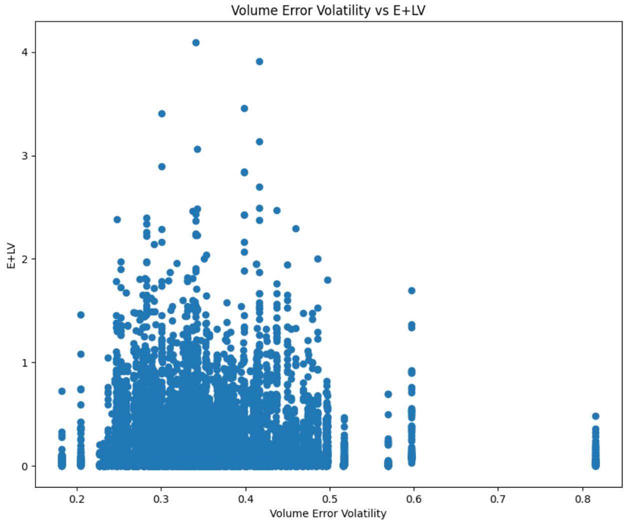


Figure 5. Volume Error Volatility vs E+LV

Based on the plots of the errors of the volume model, it is observable that the factors are heteroskedastic, showcasing high variance around 0. The volatility of the error also shows heteroskedastic variance.

It is suspicious that there is any pattern between the errors (or the volatility of the error) and the cost.

4.2 Liquidity vs VWAP Price

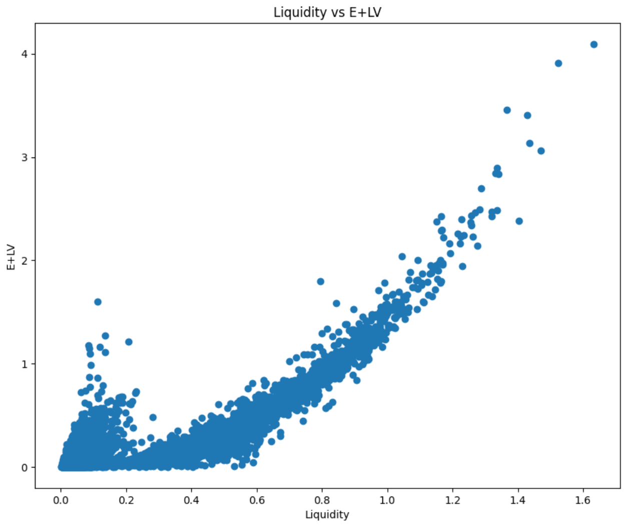


Figure 5. Liquidity vs E+LV

Liquidity proxy ($\sqrt{\frac{\sigma^2}{V}}$) showcases a relatively strong non-linear relationship with the cost.

Thus, it is possible to formulate a formula for the VWAP price using liquidity as a input variable. Put the base functional form as follows:

$$y = \alpha x^{\beta} + \frac{\gamma}{x}$$

Then, the graph will look as follows with $\alpha = \beta = \gamma = 1$:

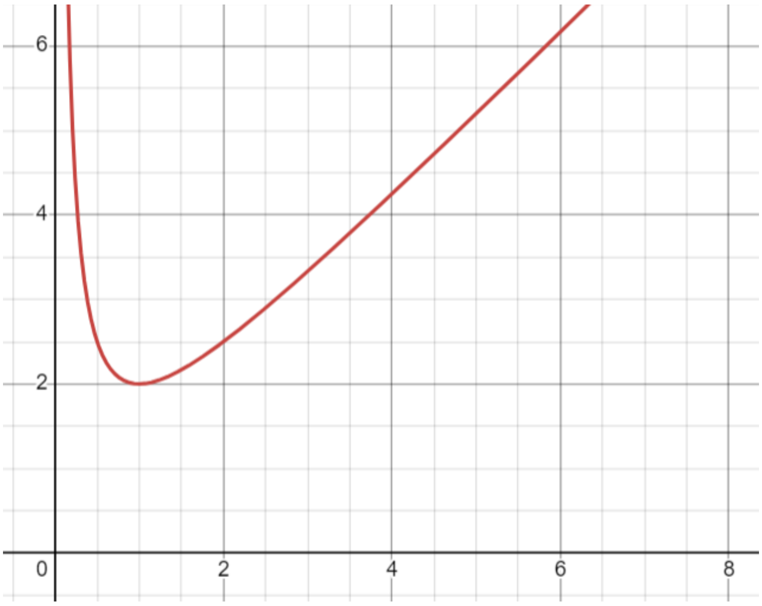


Figure 6. Basic Functional Form

By fitting the functional form with scipy.curvefit,

$$\alpha = 1.441813656654553,$$

$$\beta = 2.1426912307742385,$$

$$\gamma = 0.000455432887944772$$

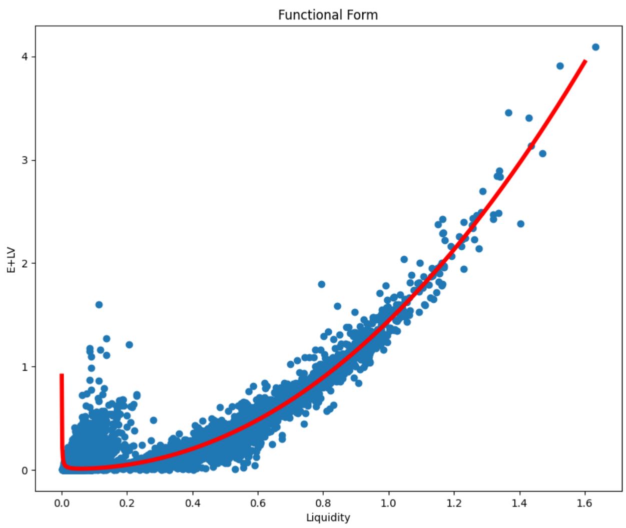


Figure 7. Fitted Functional Form

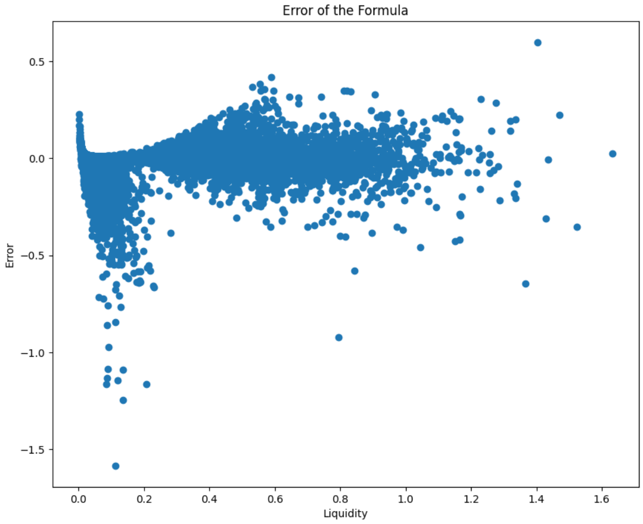


Figure 8. Error of the Formula

The residual plot suggests that the errors behave better when the liquidity is high. Thus, the formula is more likely to work better for liquid stocks than for illiquid stocks.

4.3 Guaranteed VWAP Contract Price Formula

The formula can be given as follows:

$$(VWAP\ Contract\ Price) = \alpha(Liquidity)^{\beta} + \frac{\gamma}{(Liquidity)}$$

$$\alpha = 1.441813656654553,$$

$$\beta = 2.1426912307742385,$$

$$\gamma = 0.000455432887944772$$

5. Unit tests

Deviation Class Test

Test_Deviation.py

```python
import unittest
from tradingModel.Deviation import *


class Test_Deviation(unittest.TestCase):

    def testDeviation(self):
        dev = Deviation( ticker: 'AAPL',  date: '20070626')
        dev.calc_deviation(1)
        self.assertEqual(dev.ticker,  second: 'AAPL')
        self.assertEqual(dev.date,  second: '20070626')
        self.assertAlmostEqual(dev.deviation,  second: 0.6877512116998048)


if __name__ == "__main__":
    unittest.main()
```

Run | Python tests in Test_Deviation.py

Test Results 64 ms

Tests passed: 1 of 1 test – 64 ms

```
C:\Users\kangj\PycharmProjects\GuaranteedVWAP\venv\Sc
Testing started at 12:11 PM ...
Launching unittests with arguments python -m unittest



Ran 1 test in 0.065s

OK

Process finished with exit code 0
```

MergebyTime Test

Test_MergebyTime.py ×

```
import unittest
from vwapUtils.MergebyTime import *


class Test_MergebyTime(unittest.TestCase):

    def testMergebyTime(self):
        reader = TAQTradesReader(f'../data/trades/20070620/AAPL_trades.bi
        merged_reader = MergebyTime(reader)
        merged_reader.merge_by_one_min()
        p = merged_reader.p_1m
        s = merged_reader.s_1m

        self.assertAlmostEqual(p[0], second: 123.94309319, places: 5)
        self.assertAlmostEqual(p[1], second: 123.94309319, places: 5)
        self.assertAlmostEqual(p[2], second: 124.409122, places: 5)
        self.assertAlmostEqual(p[3], second: 124.27455858, places: 5)
        self.assertAlmostEqual(p[4], second: 123.92541723, places: 5)

        self.assertEqual(s[1], second: 211371)
        self.assertEqual(s[2], second: 285524)
        self.assertEqual(s[3], second: 163188)
        self.assertEqual(s[4], second: 224038)
        self.assertEqual(s[5], second: 233271)


if __name__ == "__main__":
    unittest.main()
```

Run   Python tests in Test_MergebyTime.py ×

Test Results 133 ms

Tests passed: 1 of 1 test – 133 ms

```
C:\Users\kangj\PycharmProjects\GuaranteedVWAP\ve
Testing started at 5:42 PM ...
Launching unittests with arguments python -m uni



Ran 1 test in 0.133s

OK

Process finished with exit code 0
```

Minimizer Class Test

Test_Minimizer.py ×

```python
import unittest
from tradingModel.Minimizer import *


class Test_Minimizer(unittest.TestCase):

    def testMinimizer(self):
        minimizer = Minimizer( timeout: 300,  maxfun: 20,  maxiter: 40)
        res = minimizer.minimize('AAPL')
        self.assertEqual(res[0],  second: 'AAPL')
        self.assertAlmostEqual(res[1],  second: 1.009781257189332)

if __name__ == "__main__":
    unittest.main()

```

Run   Python tests in Test_Minimizer.py ×

Test Results 1 min 18 sec

Tests passed: 1 of 1 test – 1 min 18 sec

```
C:\Users\kangj\PycharmProjects\GuaranteedVWAP\venv\Scripts\python.exe "C:/Progra
Testing started at 12:01 PM ...
Launching unittests with arguments python -m unittest C:\Users\kangj\PycharmProj

Working on AAPL


Ran 1 test in 77.646s

OK
AAPL Finished. Beta: 1.009781257189332. Elapsed: 77.641 sec

Process finished with exit code 0
```

TradingStatsAnalysis Class Test

Test_TradingStatsAnalysis.py ×

```python
import unittest

import numpy as np

from tradingModel.TradingStatsAnalysis import *


class Test_TradingStatsAnalysis(unittest.TestCase):

    def testTradingStatsAnalysis(self):
        tsa = TradingStatsAnalysis()
        tsa.train_test()
        tsa_train_dates = tsa.train_dates
        tsa_test_dates = tsa.test_dates

        dates = np.load('../data/dates.npy')
        holdout = int(len(dates) * 0.3)

        train_dates = dates[:-holdout]
        test_dates = dates[-holdout:]

        self.assertTrue(all(np.equal(train_dates, tsa_train_dates)))
        self.assertTrue(all(np.equal(test_dates,  tsa_test_dates)))

        tsa.fixed_kappa(['20070620'])
        tsa.no_adjustment(['20070620'])
        tsa.optimized_beta(['20070620'])

        df = pd.DataFrame(columns=['fixed_kappa', 'no_adj', 'opt_beta'])
        df.loc['AAPL', 'fixed_kappa'] = 0.119897
        df.loc['AAPL', 'no_adj'] = 152.169667
        df.loc['AAPL', 'opt_beta'] = 0.113496

        self.assertAlmostEqual(tsa.stats_df.loc['AAPL', 'fixed_kappa'],
                               df.loc['AAPL', 'fixed_kappa'])
        self.assertAlmostEqual(tsa.stats_df.loc['AAPL', 'no_adj'],
                               df.loc['AAPL', 'no_adj'])
        self.assertAlmostEqual(tsa.stats_df.loc['AAPL', 'opt_beta'],
                               df.loc['AAPL', 'opt_beta'])

if __name__ == "__main__":
    unittest.main()
```

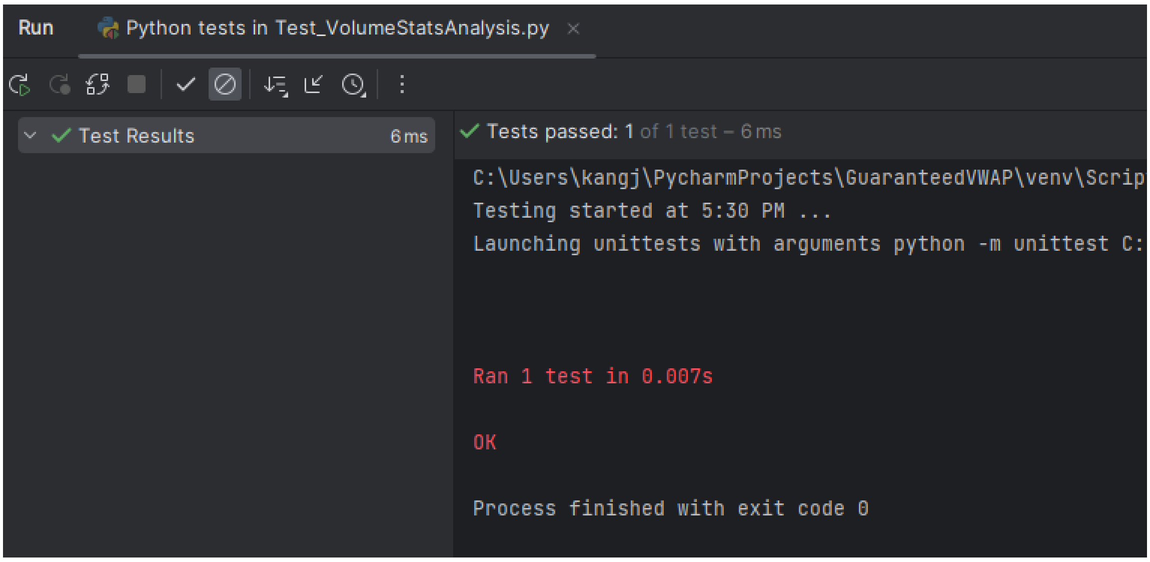
Run   Python tests in Test_TradingStatsAnalysis.py ×

Test Results 1 min 6 sec

Tests passed: 1 of 1 test – 1 min 6 sec

```
C:\Users\kangj\PycharmProjects\GuaranteedVWAP\venv\Scr
Testing started at 5:27 PM ...
Launching unittests with arguments python -m unittest

100%|██████████| 327/327 [00:25<00:00, 13.02it/s]
100%|██████████| 327/327 [00:20<00:00, 16.13it/s]
100%|██████████| 327/327 [00:20<00:00, 16.08it/s]


Ran 1 test in 65.803s

OK

Process finished with exit code 0
```

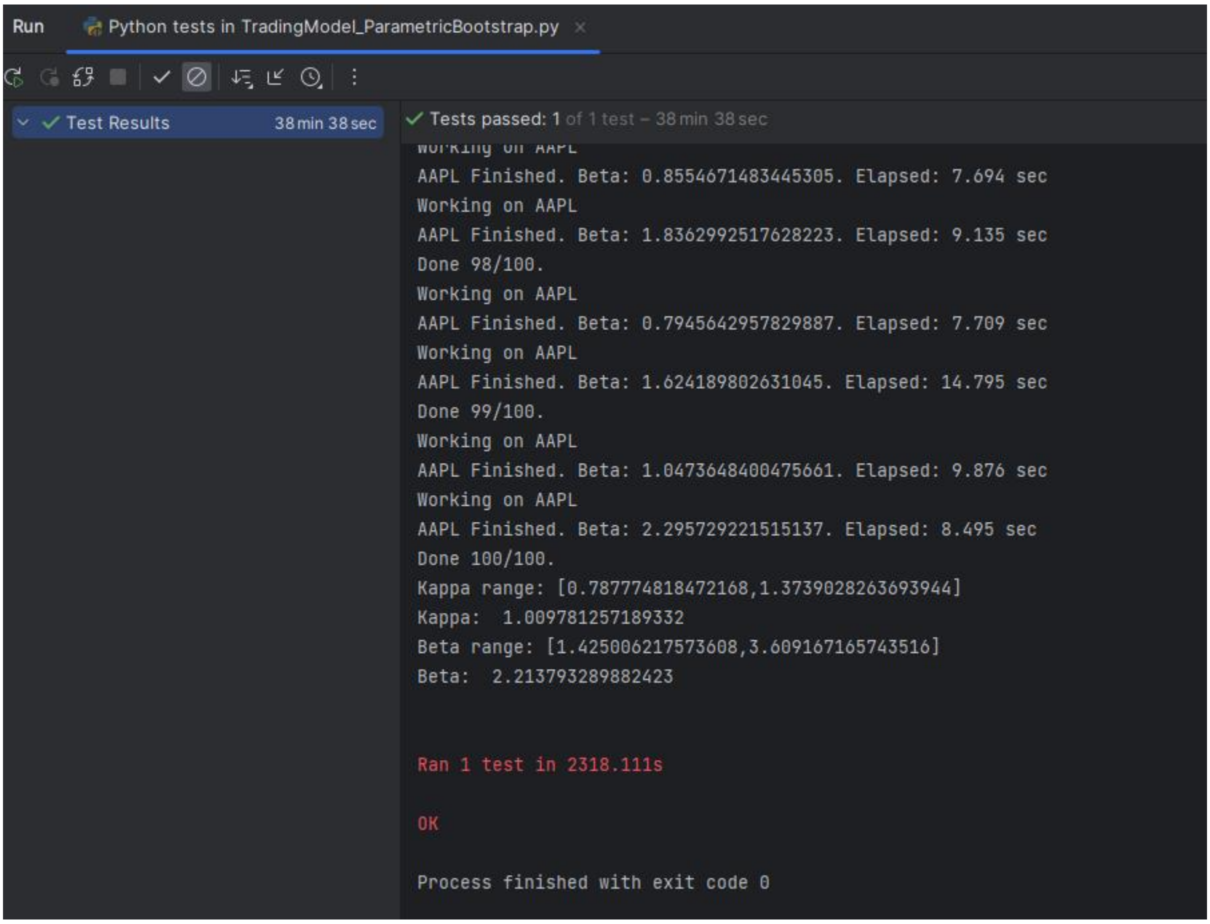
VolumeStatsAnalysis Test



Parametric Bootstrap for the Trading Model

TradingModel_ParametricBootstrap.py

```python
from tradingModel.TradingStatsAnalysis import *
from tradingModel.Minimizer import *

class TradingModel_ParametricBootstrap(unittest.TestCase):

    def test_TradingModel_ParametricBootstrap(self):
        tsa = TradingStatsAnalysis()
        tsa.train_test()
        dates = tsa.train_dates

        kappa_df = pd.read_csv('../data/fixed_kappa.csv')
        kappa_df.set_index(kappa_df.columns[0], inplace=True)

        kappa = kappa_df.loc['AAPL', 'beta']

        beta_df = pd.read_csv('../data/beta.csv')
        beta_df.set_index(beta_df.columns[0], inplace=True)

        beta = beta_df.loc['AAPL', 'beta']

        n_sample = 100

        kappa_arr = np.zeros(n_sample)
        beta_arr = np.zeros(n_sample)

        for i in range(n_sample):
            ridx = np.random.choice(len(dates), size=6, replace=False)
            simul_dates = dates[ridx]

            minimizer = Minimizer(timeout: 300, maxfun: 20, maxiter: 40)
            ticker, kappa_simul = minimizer.minimize(ticker='AAPL',
                                                     arb_dates=simul_dates,
                                                     mode='fixed_kappa')

            kappa_arr[i] = kappa_simul

            minimizer1 = Minimizer(timeout: 300, maxfun: 20, maxiter: 40)
            ticker, beta_simul = minimizer1.minimize(ticker='AAPL',
                                                     arb_dates=simul_dates)

            beta_arr[i] = beta_simul

            print(f"Done {i+1}/{n_sample}.")

        def is_within_range(value, min_value, max_value):
            return min_value <= value <= max_value

        print(f"Kappa range: [{np.mean(kappa_arr) - np.std(kappa_arr)},"
              f"{np.mean(kappa_arr) + np.std(kappa_arr)}]")
        print(f"Kappa: ", kappa)

        self.assertTrue(is_within_range(
            kappa,
            np.mean(kappa_arr) - np.std(kappa_arr),
            np.mean(kappa_arr) + np.std(kappa_arr)))

        print(f"Beta range: [{np.mean(beta_arr) - np.std(beta_arr)},"
              f"{np.mean(beta_arr) + np.std(beta_arr)}]")
        print(f"Beta: ", beta)

        self.assertTrue(is_within_range(
            beta,
            np.mean(beta_arr) - np.std(beta_arr),
            np.mean(beta_arr) + np.std(beta_arr)))


if __name__ == "__main__":
    unittest.main()
```


Run
Python tests in TradingModel_ParametricBootstrap.py
Test Results 38 min 38 sec
Tests passed: 1 of 1 test – 38 min 38 sec
AAPL Finished. Beta: 0.8554671483445305. Elapsed: 7.694 sec
Working on AAPL
AAPL Finished. Beta: 1.8362992517628223. Elapsed: 9.135 sec
Done 98/100.
Working on AAPL
AAPL Finished. Beta: 0.7945642957829887. Elapsed: 7.709 sec
Working on AAPL
AAPL Finished. Beta: 1.624189802631045. Elapsed: 14.795 sec
Done 99/100.
Working on AAPL
AAPL Finished. Beta: 1.0473648400475661. Elapsed: 9.876 sec
Working on AAPL
AAPL Finished. Beta: 2.295729221515137. Elapsed: 8.495 sec
Done 100/100.
Kappa range: [0.787774818472168,1.3739028263693944]
Kappa: 1.009781257189332
Beta range: [1.425006217573608,3.609167165743516]
Beta: 2.213793289882423
Ran 1 test in 2318.111s
OK
Process finished with exit code 0