Making a Streamlit Dashboard for our Trade Executor

HangukQuant[1, 2*]

November 2, 2022

[1] [2]

Abstract

We make a dashboard to analyse trade statistics from the order executor built in previous work, and as means to diagnose the health of our trading system and positional status.

[1] ***1**: hangukquant@gmail.com, hangukquant.substack.com

[2] ***2**: DISCLAIMER: the contents of this work are not intended as investment, legal, tax or any other advice, and is for informational purposes only. It is illegal to make unauthorized copies, forward to an unauthorized user or to post this article electronically without express written consent by HangukQuant.

1 Introduction

As before, the same caveat applies - this is a result of weekly musings - there are bound to be mistakes. Please email all errors/issues to *hangukquant@gmail.com* or tweet me at *@HangukQuant*.

In the previous works, we built a trade order executor using the MetaAPI service, and after generating some log data, we wrote it onto a Mongo instance using a database service we programmed. Here we analyse the trade logs. We also conclude our engineering series before moving on to quantitative research work.

Before we sign off on the engineering series, I want to lay down some notes. First, it has both been an absolute pleasure and pain to think and write the code out by hand and think about how to bring these different components together to relay to my readers. There are some things I did not anticipate, such as the difficulty of bringing together content from different software components - each paper requires some knowledge and implementation detail from past works, and cannot be understood in isolation. This is particularly so if the reader is not able to abstract away the details and share my vision for the overall code architecture. In that sense, the complexity of the work was more challenging on the reader than it needed to be, although I suspect that without the software interactions, the individual components themselves are already somewhat sophisticated and require deeper understanding of the Python mechanics.

My intention was to elevate the reader's knowledge in both finance and programming (python/-computer science), and I suspect that readers who take the painstaking work of going through each paper and understanding the details will in fact manage this goal. However, the layered complexity will also throw many readers off and many will quit in vain. I think the fault is half-half - it is my job to simplify it and yours to stretch your mind. Perhaps in future iterations, I will do better. I am fairly certain that our content is somewhat unique in both the depth and difficulty of content out there in systematic trading. While I have no intention on pulling back, I do think we have miles to go in terms of being a better communicator. I also appeal to readers an important note once again - the purpose of us releasing code this year is not such that you can just 'run em and voila'. It is such that we can include scientific and hard explanations instead of high-level discussion and argument such that our concepts are clearer to the *UNDERSTANDING*. A significant proportion of the code we share will NOT run 'as is', since the imported libraries and services we use might

have been modified since the last time we wrote a paper on it. We will likely not update readers on these modifications. The development and engineering process is yours and yours only.

With that said, a significant proportion of the code WILL also work perfectly. Essentially, what I mean is this: you will only know if the code works perfectly or not by understanding the code. If you copy and paste some code and you are not sure if it works - it means you lack understanding. I also copy and paste from Stack Overflow all the time, but I make sure I understand what it is I am putting in my code. I have no issue with your ctrl-C ctrl-V either. The copy and paste is a convenience function conditional on you already having the conceptual knowledge. If you copy and paste blindly, it will never work anyway.

1.1 A Note of Precaution

There is absolutely no warranty or guarantee implied with this product. Use at your own risk. I provide no guarantee that it will be functional, destructive or constructive in any sense of the word. Use at your own risk. Trading is a risky operation.

1.2 Desired Behavior

First, recall how we submitted our logs to our database. Refer to previous work [1] for the executor code. The relevant parts for the different hierarchies of logging was execution window:

```
series_metadata = {"exec_level" : "window"}
series_identifier = {"type": "execution_log", **series_metadata}
```

for execution batch:

```
doc_identifier = {"type": "execution_log", "batch_stamp": batchstamp}
```

and for a single order:

```
doc_identifier = {"type": "execution_log", "order_stamp": f"{batchstamp}/{ticker}"}
```

In particular, we stored the execution window in a time series collection and the remainder of the logs in regular document collections in our NoSQL Mongo database. We need to be able to retrieve it. This is simple, since our database service handles all of these.

```
data_receiver = db_service.DbService()
async def get_window_logs(start=None, end=None):
    series_metadata = {"exec_level" : "window"}
    series_identifier = {"type": "execution_log", **series_metadata}
    res_range, df = await data_receiver.asyn_read_timeseries(
        dtype="log",
        dformat="tsexec",
        dfreq="o",
        period_start=start,
        period_end=end,
        series_metadata=series_metadata,
        series_identifier=series_identifier
    )
    return df

async def get_batch_logs(batchstamp):
    batch_doc = await data_receiver.asyn_read_docs(
        dtype="log",
        dformat="docexec",
        dfreq="o",
        doc_identifier={"type": "execution_log", "batch_stamp": batchstamp},
        metalogs=""
    )
    data = batch_doc[2]
    return data

async def get_ticker_logs(batchstamp, ticker):
    ticker_doc = await data_receiver.asyn_read_docs(
        dtype="log",
        dformat="docexec",
        dfreq="o",
        doc_identifier={"type": "execution_log", "order_stamp": f"{batchstamp}/{
    ticker}"},
        metalogs=""
    )
    return ticker_doc[2]
```

Listing 1: retrieving data

These functions retrieve the log records from our database! The code for the data receiver is from our database service code.

What we want is the ability to have the overall view of our executor logs, pick certain date ranges to zoom into. When looking at the execution logs within a date range, we want the ability to choose a particular date, and see with one look whether we have any issues in the execution performance. If there are issues with a particular order, we can check what batch of trade orders it belongs to, zoom in on the batch to check the textual logs. If there is no issue with the batch, we want to see the individual trade logs, and have a play-by-play of the position and order table of our books back in time when the execution was performed.

Let's first take a snapshot of what we will be building:

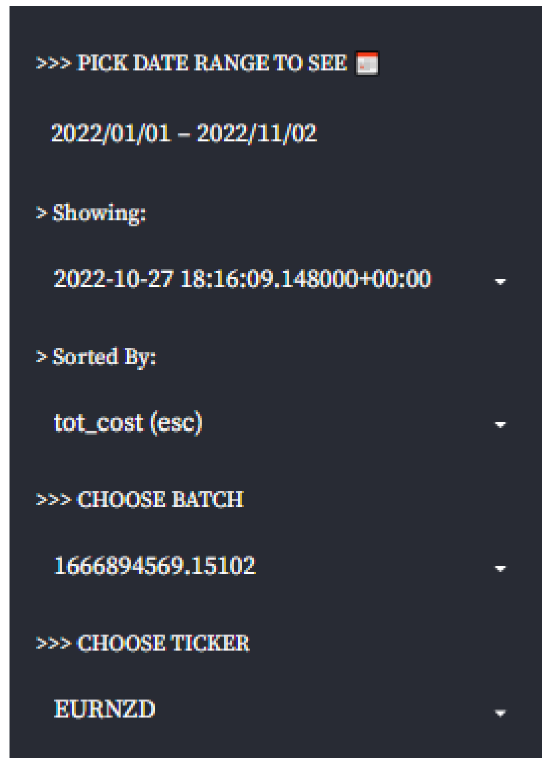

Figure 1: Hierarchical Visualization to View our Executor Logs

>>> PICK DATE RANGE TO SEE

2022/01/01 – 2022/11/02

> Showing:

2022-10-27 18:16:09.148000+00:00

> Sorted By:

tot_cost (esc)

>>> CHOOSE BATCH

1666894569.15102

>>> CHOOSE TICKER

EURNZD

Execution Logs

Execution Summaries 2022-01-01 ~ 2022-11-02

datetime	batch	ticker	pre	target	target_round	post	inertia	exec_cost@mkt	exec_cost@act	slippage	comm	tot_cost (esc)	received_price	peek_ba
2022-10-27T18:16:09.1	1666894749.152346	CHX	0	3.42759	3	3	0.2	-0.17289	0.17289	-37.51783	0	-37.34494	21.03	28.87,28.97
2022-10-27T18:16:09.1	1666895769.167842	NOV	0	5.92535	6	6	0.2	-0.2878	-0.19925	-26.31432	0	-26.51356	17.88	22.52,22.65
2022-10-27T18:16:09.1	1666895049.156278	STLD	0	1.86442	2	2	0.2	-0.02048	0.02048	-25.08327	0	-25.06279	78.06	97.62,97.66
2022-10-27T18:16:09.1	1666895529.164337	HAL	0	2.32451	2	2	0.2	-0.01363	0.04088	-24.77049	0	-24.72961	29.41	36.69,36.7
2022-10-27T18:16:09.1	1666895769.167842	NCLH	0	2.26465	2	2	0.2	-0.03107	0.03107	-23.52264	0	-23.49157	13.03	16.09,16.1
2022-10-27T18:16:09.1	1666895949.170546	SLB	0	4.14235	4	4	0.2	-0.01921	0.01921	-23.45825	0	-23.43904	42.16	52.04,52.06
2022-10-27T18:16:09.1	1666896009.171131	UAL	0	1.02158	1	1	0.2	-0.02305	0.02305	-22.30054	0	-22.27748	35.47	43.37,43.39
2022-10-27T18:16:09.1	1666895709.166002	M	0	2.10631	2	2	0.2	-0.02374	0.02374	-20.64719	0	-20.62346	17.46	21.06,21.07
2022-10-27T18:16:09.1	1666895349.161145	CHWY	0	0.99299	1	1	0.2	-0.05024	0.05024	-20.67293	0	-20.62269	32.99	39.79,39.83
2022-10-27T18:16:09.1	1666895229.158714	BKR	0	1.54822	2	2	0.2	-0.01824	0.01824	-19.62025	0	-19.60201	22.91	27.4,27.41
2022-10-27T18:16:09.1	1666895469.163448	FCX	0	1.26304	1	1	0.2	-0.0152	0.0152	-19.04848	0	-19.03329	27.64	32.9,32.91
2022-10-27T18:16:09.1	1666895169.158373	ALB	0	0.58957	1	1	0.2	-0.05305	0.09548	-19.07609	0	-18.98061	237.47	282.62,282.92
2022-10-27T18:16:09.1	1666895709.166002	LNW	0	0.75115	1	1	0.2	-0.27629	0.27629	-18.02174	0	-17.74545	46	54.14,54.44
2022-10-27T18:16:09.1	1666895889.169292	ROL	0	3.54916	4	4	0.2	-0.02435	-0.36532	-17.24729	0	-17.61261	35.02	41.05,41.07
2022-10-27T18:16:09.1	1666894689.151392	APA	0	4.15647	4	4	0.2	-0.02142	0.02142	-16.92963	0	-16.90821	39.93	46.68,46.7
2022-10-27T18:16:09.1	1666894869.153671	IBKR	0	2.8926	3	3	0.2	-0.25772	-0.63486	-16.25987	0	-16.89473	68.42	79.34,79.75
2022-10-27T18:16:09.1	1666895529.164337	HES	0	1.42133	1	1	0.2	-0.03836	0.05231	-16.94673	0	-16.89442	122.59	143.31,143.42
2022-10-27T18:16:09.1	1666894689.151392	AXON	0	1.32623	1	1	0.2	-0.21827	0.23258	-16.982	0	-16.74942	119.45	139.43,140.04
2022-10-27T18:16:09.1	1666894749.152346	CSGP	0	2.8026	3	3	0.2	-0.14795	-0.90001	-15.37696	0	-16.27697	70.3	80.99,81.23
2022-10-27T18:16:09.1	1666894809.153424	EXPI	0	2.85897	3	3	0.2	-0.15175	0.15175	-16.32833	0	-16.17659	11.33	13.16,13.2
2022-10-27T18:16:09.1	1666895589.164807	HOG	0	1.07446	1	1	0.2	-0.13242	0.13242	-15.72862	0	-15.5962	35.89	41.48,41.59
2022-10-27T18:16:09.1	1666895649.1656	JWN	0	1.95411	2	2	0.2	-0.04773	0.04773	-15.6181	0	-15.57037	18.12	20.94,20.96
2022-10-27T18:16:09.1	1666895889.169292	SAIC	0	1.43886	1	1	0.2	-0.23451	-0.44508	-14.49315	0	-14.93823	91.25	104.23,104.72
2022-10-27T18:16:09.1	1666895589.164807	IGT	0	1.98272	2	2	0.2	-0.27425	0.52356	-15.19242	0	-14.66886	17.41	20,20.11
2022-10-27T18:16:09.1	1666895769.167842	MRO	0	6.22406	6	6	0.2	-0.01613	0.01613	-14.5841	0	-14.56797	27.05	30.99,31
2022-10-27T18:16:09.1	1666895589.164807	HII	0	0.98463	1	1	0.2	-0.0392	0.04704	-14.30748	0	-14.26044	223.17	255,255.2
2022-10-27T18:16:09.1	1666895889.169292	PWR	0	0.63317	1	1	0.2	-0.1806	0.20185	-14.41131	0	-14.20946	123.41	140.94,141.45
2022-10-27T18:16:09.1	1666895889.169292	RCL	0	1.35499	1	1	0.2	-0.0393	0.0393	-14.07756	0	-14.03826	44.61	50.87,50.91

1. exec_cost@act is red if execution strategy is worse than fill @market, yellow if better than market but still costly, and green if there was money earned.
2. slippage is green if positive and red if negative
3. post is red if target position was not hit despite having inertia overriden
4. tot_cost (esc) = exec_cost@act + slippage + comm
5. tot_cost (esc) is red if the executor incurred costs on position without inertia overriden, yellow if cost was incurred on inertia overriden trade, and green if there was money earned on trade (regardless of positional inertia)

See Batch Executor Logs: 1666894569.15102

See Ticker Executor Logs: EURNZD

Figure 2: Executor Log Page

datetime	batch	ticker	pre	target	target_round	post	inertia	exec_cost@mkt	exec_cost@act	slippage	comm	tot_cost (esc)	received_pric
2022-10-27T18:16:09.1	1666894869.153671	LHCG	0	1.72688	2	2	0.2	-0.25586	0.05719	-0.6392	0	-0.58201	165.05
2022-10-27T18:16:09.1	1666895709.166002	MO	0	1.88172	2	2	0.2	-0.011	0.011	-0.51957	0	-0.50857	45.23
2022-10-27T18:16:09.1	1666895709.166002	LVS	0	3.79418	4	4	0.2	-0.01413	0.01413	-0.44009	0	-0.42596	35.22
2022-10-27T18:16:09.1	1666894929.154829	MU	0	0.71442	1	1	0.2	-0.00946	0.00946	-0.29401	0	-0.28455	52.72
2022-10-27T18:16:09.1	1666895049.156278	SRPT	0	1.64947	2	2	0.2	-0.17904	-0.67138	0.65807	0	-0.01331	112.45
2022-10-27T18:16:09.1	1666896129.174324	ZEN	0	2.37573	2	2	0.2	-0.00656	0.01967	0.07208	0	0.09176	76.3
2022-10-27T18:16:09.1	1666895109.15738	WYNN	0	0.56367	1	1	0.2	-0.05338	0.05338	0.26619	0	0.31957	56.35
2022-10-27T18:16:09.1	1666895049.156278	SBUX	0	1.81192	2	2	0.2	-0.0702	-0.0936	1.04203	0	0.94843	86.37
2022-10-27T18:16:09.1	1666895889.169292	SCI	0	0.71929	1	1	0.2	-0.16694	0.16694	0.92623	0	1.09318	60.46
2022-10-27T18:16:09.1	1666894989.155757	OZK	0	3.75046	4	4	0.2	-0.14354	0.16746	1.15867	0	1.32613	42.29
2022-10-27T18:16:09.1	1666894809.153424	GH	0	0.6541	1	1	0.2	-0.30036	0.36251	0.99467	0	1.35717	48.76
2022-10-27T18:16:09.1	1666895109.15738	VRTX	0	0.71313	1	1	0.2	-0.02768	0.04153	1.44265	0	1.48418	293.21
2022-10-27T18:16:09.1	1666895769.167842	NYT	0	1.38997	1	1	0.2	-0.20776	0.20776	1.70184	0	1.90959	29.38
2022-10-27T18:16:09.1	1666895949.170546	SPR	0	1.45469	1	1	0.2	-0.12953	0.17271	1.78117	0	1.95388	23.58
2022-10-27T18:16:09.1	1666895709.166002	MAS	0	0.8143	1	1	0.2	-0.13052	0.13052	1.96204	0	2.09256	46.89
2022-10-27T18:16:09.1	1666894689.151392	BMRN	0	1.42469	1	1	0.2	-0.11721	0.12893	2.21203	0	2.34096	87.25
2022-10-27T18:16:09.1	1666895349.161145	CLF	0	2.34162	2	2	0.2	-0.03557	0.03557	2.73356	0	2.76914	14.45
2022-10-27T18:16:09.1	1666895949.170546	SNV	0	1.89001	2	2	0.2	-0.11502	0.11502	3.60926	0	3.72428	40.59
2022-10-27T18:16:09.1	1666894749.152346	CROX	0	1.30145	1	1	0.2	-0.04036	0.04036	4.1027	0	4.14306	77.51
2022-10-27T18:16:09.1	1666895289.159065	BRO	0	0.70034	1	1	0.2	-0.05271	0.05271	4.32078	0	4.37349	59.48
2022-10-27T18:16:09.1	1666895469.163448	EQT	0	1.66523	2	2	0.2	-0.03826	0.03826	4.68028	0	4.71854	41.13
2022-10-27T18:16:09.1	1666894629.149228	NZDUSD	0	-0.10763	-0.11	-0.11	0.2	-0.00684	0.00684	4.79913	0	4.80597	0.55781
2022-10-27T18:16:09.1	1666894689.151392	AZPN	0	0.84778	1	1	0.2	-0.15622	0.19961	4.7924	0	4.99201	242.05
2022-10-27T18:16:09.1	1666894989.155757	PNFP	0	1.39463	1	1	0.2	-0.1357	0.20972	5.1153	0	5.32502	85.43
2022-10-27T18:16:09.1	1666894869.153671	IONS	0	4.0907	4	4	0.2	-0.06879	-0.41275	5.93184	0	5.51909	46.36
2022-10-27T18:16:09.1	1666895229.158714	AWI	0	0.53113	1	1	0.2	-0.34093	-0.3543	5.91824	0	5.56394	79.5
2022-10-27T18:16:09.1	1666895949.170546	SKX	0	1.08265	1	1	0.2	-0.0153	0.04591	5.83573	0	5.88164	34.7
2022-10-27T18:16:09.1	1666895169.158373	ALL	0	1.56502	2	2	0.2	-0.18673	0.13802	6.9994	0	7.13742	132.44

1. exec_cost@act is red if execution strategy is worse than fill @market, yellow if better than market but still costly, and green if there was money earned.
2. slippage is green if positive and red if negative
3. post is red if target position was not hit despite having inertia overriden
4. tot_cost (esc) = exec_cost@act + slippage + comm
5. tot_cost (esc) is red if the executor incurred costs on position without inertia overriden, yellow if cost was incurred on inertia overriden trade, and green if there was money earned on trade (regardless of positional inertia)

Figure 3: Executor Table

See Batch Executor Logs: 1666894569.15102

TICKERS

EURNZD, EURNOK, EURAUD, AUDCAD, GBPJPY, AUDUSD, GBPUSD, NZDJPY, AUDCHF, EURCHF, EURTRY, GBPAUD, CHFJPY, CADJPY, NZDCAD, AUDNZD, NZDCHF, GBPCHF, USDJPY, USDNOK, GBPCAD, EURJPY, EURGBP, EURSEK, USDMXN, USDCAD, USDCHF, USDSEK, USDSGD, EURCAD

Batch Logs

datetime
2022-10-27T18:16:09.151000
2022-10-27T18:16:09.151000
2022-10-27T18:36:10.707000
2022-10-27T18:36:11.639000
2022-10-27T18:36:11.639000

Figure 4: Executor Batch

See Ticker Executor Logs: EURNOK

Order Specs


```
{
  "ticker" : "EURNOK"
  "pre" : 0
  "post" : 0.08
  "target" : 0.08
  "target_round" : 0.08
  "inertia" : 0.2
  "is_overriden" : true
}
```

```
{
  "comm" : 0
  "peek_ba" : [
    0 : 10.23247
    1 : 10.2356
  ]
  "received_price" : 14.0926
  "slippage" : 27.380078906660238
  "market_spread_penalty" : -0.015292111078388427
  "execution_spread_penalty" : 0.016953234965468320
}
```


Tickets and Fills

[['2012854024', 10.2323], ['2012854017', 10.2323], ['2012854018', 10.2323], ['2012854025', 10.2323], ['2012854026', 10.2323], ['2012854021', 10.2323], ['2012854023', 10.2323], ['2012854019', 10.2323]]

Figure 5: Executor Order, 1

Order Playback

Order Logs

datetime	message
2022-10-27T18:16:09.411000	make limit @10.23247 from 0.0 -> 0.08
2022-10-27T18:16:09.411000	make limit_order @10.23247 0.08 contracts
2022-10-27T18:16:09.412000	submmited limit_order @10.23247
2022-10-27T18:16:09.412000	submmited limit_order @10.23247
2022-10-27T18:16:09.412000	submmited limit_order @10.23247
2022-10-27T18:16:09.412000	submmited limit_order @10.23247
2022-10-27T18:16:09.412000	submmited limit_order @10.23247
2022-10-27T18:16:09.412000	submmited limit_order @10.23247
2022-10-27T18:16:09.412000	submmited limit_order @10.23247
2022-10-27T18:16:09.412000	submmited limit_order @10.23247
2022-10-27T18:16:10.672000	made limit_orders split 8.0 tickets: ['2012854017', '2012854025', '2012854019', '2012854023', '2012854021', '2012854018', '2012854026', '2012854024']
2022-10-27T18:16:10.672000	taking position and order snapshots
2022-10-27T18:21:10.673000	taking position and order snapshots
2022-10-27T18:21:10.679000	finished execution with position filled 0.08 contracts
2022-10-27T18:36:10.707000	received force order, forcing...
2022-10-27T18:36:11.166000	killed posids []
2022-10-27T18:36:11.166000	killed ordids []
2022-10-27T18:36:10.707000	taking position and order snapshots
2022-10-27T18:36:11.181000	posdiff: 0.0 overriden False

Figure 6: Executor Order, 2

Position Snapshots

2022-10-27 18:16:09.158000

ticket	type

2022-10-27 18:16:10.672000

ticket	type

2022-10-27 18:21:10.673000

ticket	type	volume	tp	sl
2012854017	_BUY	0.01	0	0
2012854018	_BUY	0.01	0	0

2022-10-27 18:36:10.707000

ticket	type	volume	tp	sl
2012854017	_BUY	0.01	0	0
2012854018	_BUY	0.01	0	0

Order Snapshots

2022-10-27 18:16:09.158000

ticket

2022-10-27 18:16:10.672000

ticket	posid	left_Δ	filled_Δ	total_Δ
2012854017	2012854017	0.01	0	0.01
2012854018	2012854018	0.01	0	0.01

2022-10-27 18:21:10.673000

ticket

2022-10-27 18:36:10.707000

ticket

Figure 7: Executor Order, 3

2 Implementation

Let's go ahead and write the code out. To use streamlit's features, we can simply do

```
python3 -m pip install streamlit
python3 -m streamlit run myexecutorpage.py
```

Streamlit allows us to use markdowns, automatically formats dataframes, dictionaries and so on. We will not go into their documentation, but I encourage interested readers to read up on their documentation.

There is also a book https://www.amazon.com/Getting-Started-Streamlit-Data-Science/dp/180056550X: Getting Started with Streamlit for Data Science: Create and deploy Streamlit web applications from scratch in Python which details writing and deploying streamlit apps step-by-step. We will just integrate it and use the code, so that interested readers can reference.

First, a config *toml* file for streamlit to see our dark blue theme. This goes in the path '.streamlit/config.toml' from our root directory.

```
[theme]

# Accepted values (serif | sans serif | monospace)
font = "serif"
primaryColor = "#ffc8c8" #red
backgroundColor = "#282b34" #dark blue
secondaryBackgroundColor = "#282b34"
textColor = "#FFFFFF"
```

We begin by importing some libraries, instantiating some theme constants and our data service.

```
1 import pytz
2 import asyncio
3 import pandas as pd
4 import numpy as np
5 import streamlit as st
6
```

```
7 import plotly.express as px
8 import plotly.graph_objects as go
9
10 from datetime import datetime
11 from data_service.db import db_service
12
13 if __name__ == "__main__":
14     st.set_page_config(
15         layout="wide"
16     )
17
18 elevated_blue = "#454a59"
19 deep_blue = "#282b34"
20 white_red = "#e97770"
21 black_red = "#ffc8c8"
22 white_green = "#2a8e34"
23 black_green = "#c8ffcd"
24 white_yellow = "#FFB055"
25 st.markdown("### Execution Logs")
26
27 st.write("")
28 st.write("")
29 st.write("")
30
31 data_receiver = db_service.DbService()
```

As seen, we have primarily three sections in our page. We can just write them in linear fashion, which makes our code really easy to compartmentalize.

```
1 async def main():
2     with st.spinner('Getting Execution Report...'):
3         df = await display_window_logs()
4
5     st.write("")
6     st.write("")
7
8     with st.spinner("Getting Batch Reports..."):
9         batchstamp, batch_tickers = await display_batch_logs(df)
```

```
10
11     st.write("")
12     st.write("")
13
14     with st.spinner("Getting Order Reports..."):
15         await display_order_logs(batchstamp, batch_tickers)
16
17 if __name__ == "__main__":
18     asyncio.run(main())
```

We will be working with dataframes often, so let's just create a utility function to format a dataframe nicely into plotly tables with arbitrary dataframes.

```
1  def make_fig(df, title, height, color_df=None):
2      if color_df is None:
3          color_df = pd.DataFrame().reindex_like(df).fillna(deep_blue)
4      fig = go.Figure(
5          data=[
6              go.Table(
7                  header=dict(
8                      values=df.columns,
9                      font=dict(size=12, color = 'white'),
10                     fill_color = '#454a59',
11                     line_color = '#454a59',
12                     align = ['left','center'],
13                     height=30
14                 ),
15                 cells=dict(
16                     values = [df[K].tolist() for K in df.columns],
17                     font=dict(size=12),
18                     align = ['left','center'],
19                     fill_color = [color_df[K].tolist() for K in color_df.columns],
20                     line_color = [color_df[K].tolist() for K in color_df.columns],
21                     height=20)
22             )
23         ]
24     )
25     fig.update_layout(title_text=title, title_font_color = 'white', title_x=0,
```

```
    margin= dict(l=0,r=10,b=20,t=30), height=height)
    return fig
```

Now, we just need to implement the plotting, sorting and filtering functionality we provided to the user.

```
async def display_window_logs():

    date_picked = st.sidebar.date_input(">>> PICK DATE RANGE TO SEE ", [datetime
    (2022, 1, 1), datetime.now(pytz.utc)])
    pick_start = date_picked[0] if len(date_picked) >= 1 else None
    pick_end = date_picked[1] if len(date_picked) == 2 else None
    pick_start = datetime(pick_start.year, pick_start.month, pick_start.day) if
    pick_start else pick_start
    pick_end = datetime(pick_end.year, pick_end.month, pick_end.day + 1) if
    pick_end else pick_end
    pick_start = pytz.utc.localize(pick_start) if pick_start else None
    pick_end = pytz.utc.localize(pick_end) if pick_end else None

    df = await get_window_logs(start=pick_start, end=pick_end)

    if df.empty:
        st.write("No execution data to see.")
        st.stop()

    df = df.iloc[::-1].reset_index(drop=True)
    executed_groups = df.groupby("datetime")
    group_list = list(executed_groups.groups.keys())
    selected_opt = st.sidebar.selectbox("> Showing: ", ["all"] + group_list)
    if selected_opt != "all":
        df = executed_groups.get_group(selected_opt)

    df["tot_cost (esc)"] =
        df["execution_spread_penalty"] + df["slippage"] + df["comm"]

    display_df = df[["datetime", "batch", "ticker", "pre", "target", "target_round
    ", "post", "inertia", \
        "market_spread_penalty", "execution_spread_penalty", "slippage", "comm", "
```

```
tot_cost (esc)", "received_price", "peek_ba"]]
29
30  display_df[["target", "market_spread_penalty", "execution_spread_penalty", "
slippage", "tot_cost (esc)"]] \
31      = display_df[["target", "market_spread_penalty", "execution_spread_penalty
", "slippage", "tot_cost (esc)"]].round(decimals=5)
32
33  display_df = display_df.rename(columns={"market_spread_penalty":"exec_cost@mkt
", "execution_spread_penalty": "exec_cost@act"})
34
35  sort_by = st.sidebar.selectbox("> Sorted By: ", ["-"] + list(display_df.
columns))
36  if sort_by != "-":
37      display_df = display_df.sort_values(by=sort_by)
38      df = df.loc[display_df.index]
39
40  color_df = pd.DataFrame().reindex_like(display_df).fillna(deep_blue)
41  exec_diff = display_df["exec_cost@act"] - display_df["exec_cost@mkt"]
42  color_df["exec_cost@act"] = exec_diff.apply(
43      lambda diff: white_red if diff < 0 else (white_yellow if diff > 0 else
deep_blue)
44  )
45  color_df["exec_cost@act"] = np.where(
46      display_df["exec_cost@act"] > 0, white_green, color_df["exec_cost@act"]
47  )
48  color_df["slippage"] = display_df["slippage"].apply(
49      lambda slip: white_green if slip > 0 else (white_red if slip < 0 else
deep_blue)
50  )
51
52  color_df["post"] = np.where(
53      np.logical_and(display_df["target_round"] != display_df["post"], df["
is_overriden"]),
54      white_red,
55      deep_blue
56  )
57  color_df["tot_cost (esc)"] = display_df["tot_cost (esc)"].apply(lambda cost:
```

```
        white_green if cost > 0 else deep_blue)
58     color_df["tot_cost (esc)"] = np.where(
59         np.logical_and(~df["is_overriden"], display_df["tot_cost (esc)"] < 0),
60         white_red,
61         color_df["tot_cost (esc)"]
62     )
63     color_df["tot_cost (esc)"] = np.where(
64         np.logical_and(df["is_overriden"], display_df["tot_cost (esc)"] < 0),
65         white_yellow,
66         color_df["tot_cost (esc)"]
67     )
68 
69     display_df = display_df.fillna("-").reset_index(drop=True)
70     fig = make_fig(df=display_df, title=f"Execution Summaries {pick_start.date()}
    ~ {pick_end.replace(day=pick_end.day - 1).date()}", height=640, color_df=
    color_df)
71     st.plotly_chart(fig, use_container_width=True)
72 
73     st.caption(
74         r'''
75         1. exec_cost@act is red if execution strategy is worse than fill @market,
    yellow if better than market but still costly, and green if there was money
    earned.
76         2. slippage is green if positive and red if negative
77         3. post is red if target position was not hit despite having inertia
    overriden
78         4. tot_cost (esc) = exec_cost@act + slippage + comm
79         5. tot_cost (esc) is red if the executor incurred costs on position
    without inertia overriden, yellow if cost was incurred on inertia overriden
    trade, and green if there was money earned on trade (regardless of positional
    inertia)
80         '''
81     )
82     return df
```

Now, the batch logs.

```
1 async def display_batch_logs(df):
```

```
    batch_groupdf = df.groupby("batch")
    group_lists = list(batch_groupdf.groups.keys())
    batch_selected = st.sidebar.selectbox(">>> CHOOSE BATCH", group_lists, key=2)

    with st.expander(f"See Batch Executor Logs: {batch_selected}"):
        data = await get_batch_logs(batchstamp=batch_selected)
        batch_tickers = data["batch_tickers"]
        c1, c2 = st.columns(2)
        with c1:
            st.write("# TICKERS")
            st.write(f"##### {batch_tickers}".replace("'", "").replace("[", "").
replace("]", ""))
        with c2:
            message_df = pd.DataFrame(data["batch_logs"])
            fig = make_fig(message_df, "Batch Logs", height=200)
            st.plotly_chart(fig, use_container_width=True)
        st.write('')

    return batch_selected, batch_tickers
```

and the order logs...

```
async def display_order_logs(batchstamp, tickers):
    ticker_selected = st.sidebar.selectbox(">>> CHOOSE TICKER", tickers, key=3)
    with st.expander(f"See Ticker Executor Logs: {ticker_selected}"):
        data = await get_ticker_logs(batchstamp, ticker_selected)
        pos_snaps = data["pos_snaps"]
        ord_snaps = data["ord_snaps"]
        show_keys_1 = ['ticker', 'pre', 'post', 'target', 'target_round', 'inertia
', 'is_overriden']
        show_keys_2 = ['comm', 'slippage', 'market_spread_penalty', '
execution_spread_penalty', 'peek_ba', 'received_price']
        show_dict1 = {k : v for k,v in data.items() if k in show_keys_1}
        show_dict2 = {k : v for k,v in data.items() if k in show_keys_2}
        st.markdown("## Order Specs")
        c1, c2 = st.columns(2)
        with c1:
            c1.write(show_dict1)
```

```python
        with c2:
            c2.write(show_dict2)
        st.write('')
        st.write('')
        st.write('')
        st.write("## Tickets and Fills")
        if data["tickets_and_fills"]:
            st.write(str(data["tickets_and_fills"]))
        else:
            st.write("no tickets involved.")
        st.write('')
        st.write('')
        st.write('')
        st.markdown("## Order Playback")
        st.write('')
        st.write('')

        st.markdown("##### Order Logs")
        logs = data["order_logs"]
        log_df = pd.DataFrame(logs)
        fig = make_fig(log_df, "", height=480)
        st.plotly_chart(fig, use_container_width=True)

        c1, c2 = st.columns((1.6, 2.4))

        with c1:
            st.markdown("##### Position Snapshots")
            for pos_snap in pos_snaps:
                pos_time = pos_snap[0]
                pos_df = pd.DataFrame.from_dict(pos_snap[1], orient="index").reset_index().rename(columns={"index": "ticket"})
                pos_df = pos_df.drop(columns="meta") if not pos_df.empty else pos_df
                pos_df["type"] = pos_df["type"].str.replace("POSITION_TYPE", "") if not pos_df.empty else pos_df
                fig = make_fig(pos_df, str(pos_time), height=480 / len(pos_snaps))
                st.plotly_chart(fig, use_container_width=True)
```

```
49        with c2:
50            st.markdown("##### Order Snapshots")
51            for ord_snap in ord_snaps:
52                ord_time = ord_snap[0]
53                ord_df = pd.DataFrame.from_dict(ord_snap[1], orient="index").
    reset_index().rename(columns={"index": "ticket"})
54
55                if not ord_df.empty:
56                    ord_df = ord_df[["ticket", "posid", "left_volume", "
    filled_volume", "total_volume", "expiration", "price_at", "open_at", "
    order_type"]]
57                    ord_df.columns = [col.replace("volume", "v") for col in ord_df
    .columns]
58                    ord_df["expiration"] = ord_df["expiration"].str.replace("
    ORDER_TIME", "")
59                    ord_df["order_type"] = ord_df["order_type"].str.replace("
    ORDER_TYPE", "")
60
61                fig = make_fig(ord_df, str(ord_time), 480 / len(ord_snaps))
62                st.plotly_chart(fig, use_container_width=True)
```

3 Concluding Remarks

That wasn't that hard..., it was just a lot of formatting and calling the write API functions from streamlit. It is surprisingly powerful, and it took ~ 300 lines of code to make the executor page! I hope this was a simple and easy introduction to streamlit, and that this will inspire you to research further into the technology and come up with your own beautiful dashboards. But remember. beauty $!=$ money in trading. Happy trading and see you on the other side.

References

[1] HangukQuant. A Trade Order Executor for Reducing Transaction Costs. `https://hangukquant.substack.com/p/a-trade-order-executor-for-reducing`.