

Capstone Project 3

Creating a real Volume Profile for Cryptocurrencies

By Burak Ugar















Introduction

- In this project, I will show you how I have created an instant and accurate volume profile for cryptocurrencies as a first step.
- The next step is to show that this is automatically written to a database.
- The last step is how I read it from the database and publish it on a small website.

(All this is done on a Raspberry Pi 4 and works automatically.)

First Part

As you can see below, the volume on each exchange for each pair of bitcoin is different from each other.

# ▲	Source	Pairs	Price	+2% Depth	-2% Depth	Volume	Volume %
1	 Binance	BTC/USDT	\$19,716.82	\$27,401,988.19	\$31,202,792.82	\$5,578,782,315	18.50%
2	 Binance	BTC/BUSD	\$19,711.05	\$8,470,423.50	\$9,325,232.14	\$2,374,968,702	7.88%
3	 Coinbase Exchange	BTC/USD	\$19,715.59	\$13,352,616.28	\$14,125,761.61	\$485,807,471	1.61%
4	 FTX	BTC/USD	\$19,717.00	\$20,466,670.11	\$27,301,065.37	\$371,539,041	1.23%
5	 Gate.io	BTC/USDT	\$19,715.80	\$3,213,002.30	\$5,014,586.02	\$288,764,842	0.96%
6	 Gate.io	BTC/USD	\$19,713.60	\$116,655.07	\$124,736.07	\$208,026,481	0.69%
7	 Binance	BTC/USDC	\$19,716.40	\$2,630,556.92	\$2,080,727.76	\$170,529,981	0.57%
8	 Binance.US	BTC/USD	\$19,710.29	\$11,395,536.22	\$4,445,755.27	\$168,242,100	0.56%
9	 Binance	ETH/BTC	\$19,722.91	\$2,639,875.68	\$3,352,926.57	\$165,170,023	0.55%
10	 Bitfinex	BTC/USD	\$19,726.53	\$12,985,119.66	\$8,273,253.58	\$136,634,900	0.45%
11	 KuCoin	BTC/USDT	\$19,716.35	\$12,267,303.32	\$8,193,957.83	\$127,924,870	0.42%
12	 Huobi Global	BTC/USDT	\$19,715.40	\$8,133,038.55	\$12,117,266.83	\$104,346,719	0.35%
13	 Kraken	XBT/USD	\$19,715.00	\$24,645,863.56	\$35,022,413.39	\$72,310,844	0.24%
14	 Bitfinex	BTC/GBP	\$19,737.77	\$14,765,295.07	\$12,099,775.73	\$65,213,258	0.22%

Libraries

I imported the ccxt library to pull price information from exchanges instantly.

Also imported, other required libraries for editing, analysis and visualization.

```
[60] import psychopg2
      from psychopg2.extensions import register_adapter, AsIs
      psychopg2.extensions.register_adapter(np.int64, psychopg2._psychopg.AsIs)
```

```
[2] import ccxt, config, time, sys
      import websocket, json
      import pandas as pd
      %matplotlib inline
      import matplotlib
      import matplotlib.pyplot as plt
      from pandas.plotting import register_matplotlib_converters
      import numpy as np
      import datetime
      import seaborn as sns
      from sklearn.model_selection import train_test_split
      from scipy import stats, signal
      import plotly.express as px
      import plotly.graph_objects as go
```

```
[3] binance = ccxt.binance({
      'enableRateLimit': True,
      'apiKey': config.API_KEY,
      'secret': config.SECRET_KEY
    })
```

```
[4] coinbase = ccxt.coinbasepro({
      'enableRateLimit': True,
      'apiKey': 'x',
      'secret': 'x'
    })
```

Exchanges, Pairs and APIs

- I entered api keys for my accounts on 5 different exchanges.
 - I pull the information of a total of 11 btc pairs from 5 different exchanges.
(This can be increased depending on the request.)
- Binance BTC USDT
 - Binance BTC BUSD
 - Binance BTC USDC
 - Coinbase BTC USD
 - Coinbase BTC USDT
 - FTX BTC USD
 - FTX BTC USDT
 - Huobi BTC USDT
 - Huobi BTC USDC
 - Gate io BTC USDT
 - Gate io BTC USD

Fetching Data

As you see, I pulled all price informations from each exchange/pair and creating DF for each one.

```
[4] symbol = 'BTCUSD'
    timeframe = '1m'
    limit = 300

bars = binance.fetch_ohlcv(symbol, timeframe=timeframe, limit=limit) # binance btcusdt
df = pd.DataFrame(bars, columns=['timestamp', 'open', 'high', 'low', 'close', 'volume'])
df['timestamp'] = pd.to_datetime(df['timestamp'], unit='ms')

[9]

symbol2 = 'BTCBUSD' # binance BTC BUSD

[10]

bars2 = binance.fetch_ohlcv(symbol2, timeframe=timeframe, limit=limit) # binance BTC BUSD
df2 = pd.DataFrame(bars2, columns=['timestamp', 'open', 'high', 'low', 'close', 'volume'])
df2['timestamp'] = pd.to_datetime(df2['timestamp'], unit='ms')

[11]

symbol3 = 'BTCUSDC' # binance BTC USDC

[12]

bars3 = binance.fetch_ohlcv(symbol3, timeframe=timeframe, limit=limit) # binance BTC BUSD
df3 = pd.DataFrame(bars3, columns=['timestamp', 'open', 'high', 'low', 'close', 'volume'])
df3['timestamp'] = pd.to_datetime(df3['timestamp'], unit='ms')

[13]

symbol4 = 'BTC/USD' # coinbase BTC USD

[14]

bars4 = coinbase.fetch_ohlcv(symbol4, timeframe=timeframe, limit=limit) # coinbase BTC USD
df4 = pd.DataFrame(bars4, columns=['timestamp', 'open', 'high', 'low', 'close', 'volume'])
df4['timestamp'] = pd.to_datetime(df4['timestamp'], unit='ms')

[15]

symbol5 = 'BTC/USDT' # coinbase BTC USDT

[16]
```

Data Preprocessing

On some exchanges the volume is given in currency, so I converted them to btc. Also converted timestamp, checked data, setting index and made other adjustments.

```
[18] symbol6 = 'BTC/USD' # ftx BTC USD

bars6 = ftx.fetch_ohlcv(symbol6, timeframe=timeframe, limit=limit) # ftx BTC USD
df6 = pd.DataFrame(bars6, columns=['timestamp', 'open', 'high', 'low', 'close', 'volume'])
df6['timestamp'] = pd.to_datetime(df6['timestamp'], unit='ms')

[19]

# convert df6 volume
df6['volume'] = df6['volume'] / df6['close']

[20]
```

New DF with sum of all Volume Data

I created a new dataframe.

I have collected all volume data for the new dataframe.








(For other datas I used the data from binance exchange which has the highest volume.)

```
[36] # create a new dataframe with timestamp high low close volume  
df_fresh = pd.DataFrame(columns=['timestamp', 'open', 'high', 'low', 'close', 'volume'])
```

```
[37] # volume = sum of volumes from all exchanges  
df_fresh['volume'] = df['volume'] + df2['volume'] + df3['volume'] + df4['volume'] + df5['volume'] + df6['volume'] + df7['volume'] + df8['volume'] + df9['volume'] + df10['volume'] + df11['volume']  
# other prices equal to binance btcusdt price  
df_fresh['close'] = df['close']  
df_fresh['open'] = df['open']  
df_fresh['high'] = df['high']  
df_fresh['low'] = df['low']  
df_fresh['timestamp'] = df['timestamp']
```


Creating a DB and setting up variables

After making the necessary adjustments on the Raspberry Pi (I installed postgresql on the docker), I created the database and tables with python.

<input type="checkbox"/>	Name	State 	Quick Actions	Stack	Image
<input type="checkbox"/>	btcpostrges	running 	    	-	postgres

creating db

```
# conn = psycopg2.connect(
#     database="btcp1m", user='x', password='x', host='192.168.1.65', port= '5432'
# )
# conn.autocommit = True
# cursor = conn.cursor()
# sql = '''CREATE DATABASE btcp1m''';
# cursor.execute(sql)
# print("Database created successfully!")
# conn.close()
```

[68]

setting up db variables

```
# conn = psycopg2.connect(
#     database="btcp1m", user='x', password='x', host='192.168.1.65', port= '5432'
# )
# conn.autocommit = True
# cur = conn.cursor()
# cur.execute('''CREATE TABLE prices
# (
#     Timestamp TIMESTAMP WITHOUT TIME ZONE NOT NULL,
#     Open FLOAT NOT NULL,
#     High FLOAT NOT NULL,
#     Low FLOAT NOT NULL,
#     Close FLOAT NOT NULL,
#     Volume BIGINT NOT NULL
# );''')
# print("Table created successfully")
# conn.close()
```

[69]

Data insertion

I insert the data into postgresql,
then delete duplicates.

```
# PSYCOG2 adapter data insertion
conn = psychog2.connect(
    database="btclmin", user='x', password='x', host='192.168.1.65', port= '5432'
)
conn.autocommit = True
cur = conn.cursor()
query = """INSERT INTO prices (Timestamp, Open, High, Low, Close, Volume)
        VALUES (%s, %s, %s, %s, %s, %s)"""
cur = conn.cursor()
cur.executemany(query, records)
conn.close()
print("Data insert successfully!")
```

[70]

... Data insert successfully!

```
# delete duplicates from postgresql table
conn = psychog2.connect(
    database="btclmin", user='x', password='x', host='192.168.1.65', port= '5432'
)
conn.autocommit = True
cur = conn.cursor()
cur.execute('''delete from prices
        where exists (select 1
                        from prices t2
                        where t2.Timestamp = prices.Timestamp and
                              t2.ctid < prices.ctid
                        );''')
conn.close()
print("Duplicates deleted successfully!")
```

[71]

... Duplicates deleted successfully!

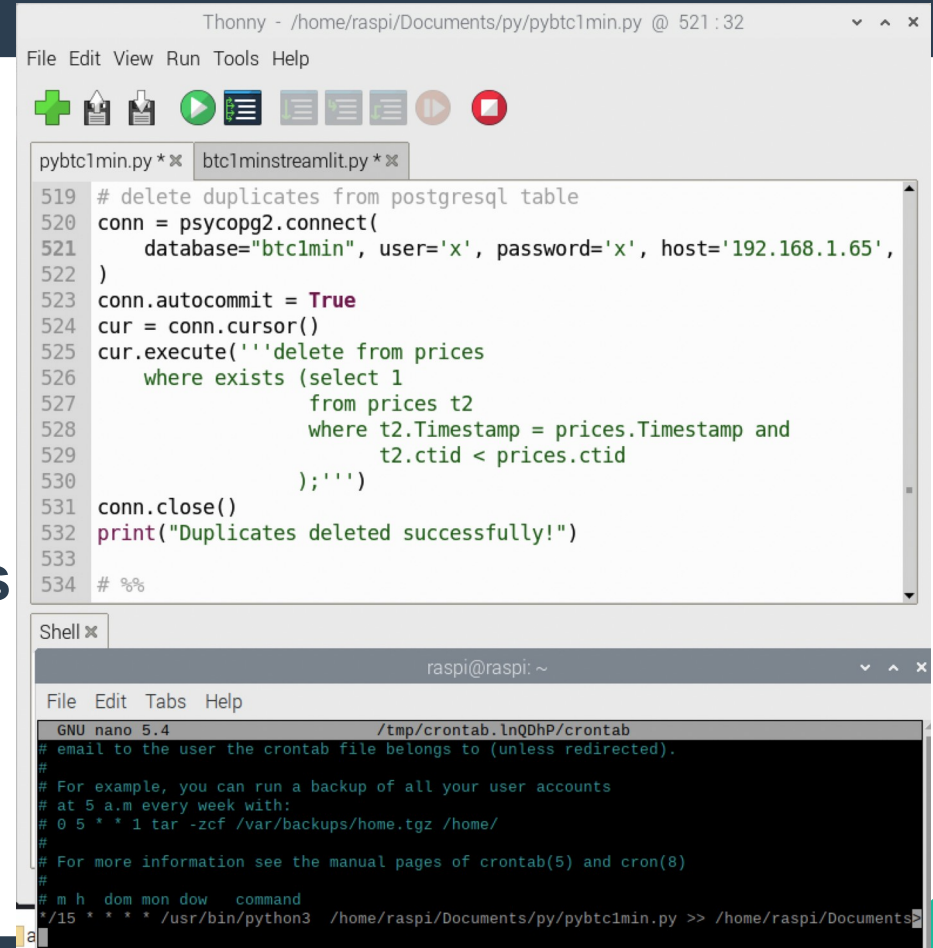
```
# read data from postgres database and convert to dataframe
conn = psychog2.connect(
    database="btclmin", user='x', password='x', host='192.168.1.65', port= '5432'
)
conn.autocommit = True
cur = conn.cursor()
cur.execute("SELECT * FROM prices")
rows = cur.fetchall()
df_fresh = pd.DataFrame(rows)
df_fresh.columns = [x[0] for x in cur.description]
print("Data read successfully!")
```

[104]

... Data read successfully!

Crontab

I converted the jupyter (using vs code) work I did for the first part into a .py script and entered the necessary crontab information for the Raspberry to do this process every 15 minutes. (Data insertion, followed by deleting duplicates.)



The screenshot shows a Thonny IDE window titled "Thonny - /home/raspi/Documents/py/pybtc1min.py @ 521:32". The main editor displays a Python script named "pybtc1min.py" with the following code:

```
519 # delete duplicates from postgresql table
520 conn = psycopg2.connect(
521     database="btc1min", user='x', password='x', host='192.168.1.65',
522 )
523 conn.autocommit = True
524 cur = conn.cursor()
525 cur.execute('''delete from prices
526     where exists (select 1
527         from prices t2
528         where t2.Timestamp = prices.Timestamp and
529             t2.ctid < prices.ctid
530     );''')
531 conn.close()
532 print("Duplicates deleted successfully!")
533
534 # %%
```

Below the editor is a terminal window titled "Shell" with the prompt "raspi@raspi: ~". It shows the output of a command to view the crontab file:

```
GNU nano 5.4 /tmp/crontab.lnQdHP/crontab
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow   command
*/15 * * * * /usr/bin/python3 /home/raspi/Documents/py/pybtc1min.py >> /home/raspi/Documents/
```

Reading data from postgresql

At the very beginning of this section, I prepared the other file, imported the streamlit library and made the necessary adjustments for DF.

```
pybtc1min.py  btc1minstreamlit.py *  
35 conn.autocommit = True  
36 cur = conn.cursor()  
37 cur.execute("SELECT * FROM prices")  
38 rows = cur.fetchall()  
39 df_fresh = pd.DataFrame(rows)  
40 df_fresh.columns = [x[0] for x in cur.description]  
41 #print("Data read successfully!")  
42  
43 # make a copy of df_fresh with timestamp as index  
44 daily = df_fresh.copy()  
45 daily.set_index('timestamp', inplace=True)  
46 #daily.shape  
47 # change index name to Date  
48 daily.index.name = 'Date'  
49 # set Date as datetimeindex  
50 #daily.index = pd.to_datetime(daily.index)  
51  
52 # convert Date to datetime format and set as index  
53 df_fresh.set_index('timestamp', inplace=True)  
54 #df_fresh.tail()  
55  
56 # reset index df_fresh  
57 df_fresh.reset_index(inplace=True)  
58 #df_fresh.tail()  
59  
60 # sort by timestamp  
61 df_fresh.sort_values(by=['timestamp'], inplace=True, ascending=True)  
62
```

Sidebar

I created time periods and put them in the sidebar.

```
64 # put radio button in sidebar
65 st.sidebar.markdown('---')
66 st.sidebar.markdown('## Select a time period')
67 period = st.sidebar.radio('Select a time period', ('Last 1h', 'Last 2h', 'L
68 if period == 'Last 1h':
69     # use last 60 rows of df_fresh
70     df_fresh = df_fresh.tail(60)
71 elif period == 'Last 2h':
72     # use last 120 rows of df_fresh
73     df_fresh = df_fresh.tail(120)
74 elif period == 'Last 4h':
75     # use last 240 rows of df_fresh
76     df_fresh = df_fresh.tail(240)
77 elif period == 'Last 12h':
78     # use last 720 rows of df_fresh
79     df_fresh = df_fresh.tail(720)
80 else:
81     # use all rows of df_fresh
82     df_fresh = df_fresh
```

Seaborn

I created a close, volume chart with Seaborn scatterplot.

```
pybtc1min.py  btc1minstreamlit.py *  
100 # scatter plot of Close and Volume with seaborn  
101 plt.figure(figsize=(16,12), dpi=200)  
102 sns.scatterplot(x='timestamp', y='close', data=df_fresh, size='volume', size  
103 # add close as a line plot with seaborn  
104 sns.lineplot(x='timestamp', y='close', data=df_fresh, color='white', alpha=0.3)  
105 a = round((df_fresh['close'].max() + 200.0), -2)  
106 b = round((df_fresh['close'].min() - 200.0), -2)  
107 plt.yticks(np.arange(b, a, 100))  
108 # show y axis at right  
109 plt.gca().yaxis.tick_right()  
110  
111 # round a and b to nearest 100  
112 # x axis ticks and labels with matplotlib  
113 plt.xticks(rotation=45)  
114 plt.gca().xaxis.set_major_locator(matdates.DayLocator(interval=1))  
115 plt.gca().xaxis.set_major_formatter(matdates.DateFormatter('%Y-%m-%d-%H:%M'))  
116 plt.gca().xaxis.set_minor_locator(matdates.HourLocator(interval=1))  
117 plt.gca().xaxis.set_minor_formatter(matdates.DateFormatter('%H:%M'))  
118 plt.xlim(df_fresh['timestamp'].min(), (df_fresh['timestamp'].max() + pd.Timedelta(days=1)))  
119 # add grid for x axis  
120 plt.grid(axis='x', which='major', color='white', alpha=0.3)  
121 plt.grid(axis='x', which='minor', color='white', alpha=0.1)  
122  
123 #plt.ylim(b, a)  
124 plt.savefig('btc1min.png')  
125 st.title('BTC 1 Minute Chart')  
126 st.image('btc1min.png')
```

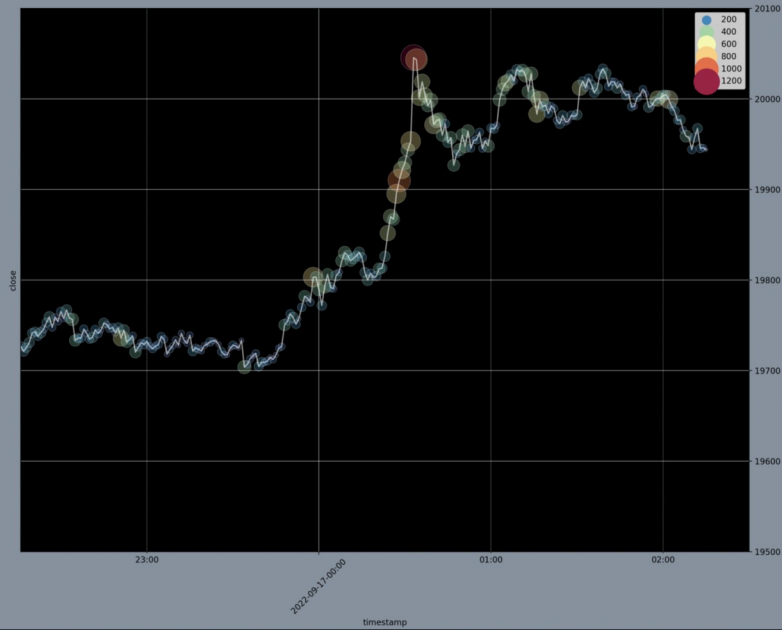
Seaborn, scatter chart

Select a time period

Select a time period

- ☐ Last 1h
- ☐ Last 2h
- ☒ Last 4h
- ☐ Last 12h
- ☐ ALL

BTC 1 Minute Chart



First Timestamp: 2022-09-16 22:16:00 (UTC)

Last Timestamp: 2022-09-17 02:15:00 (UTC)

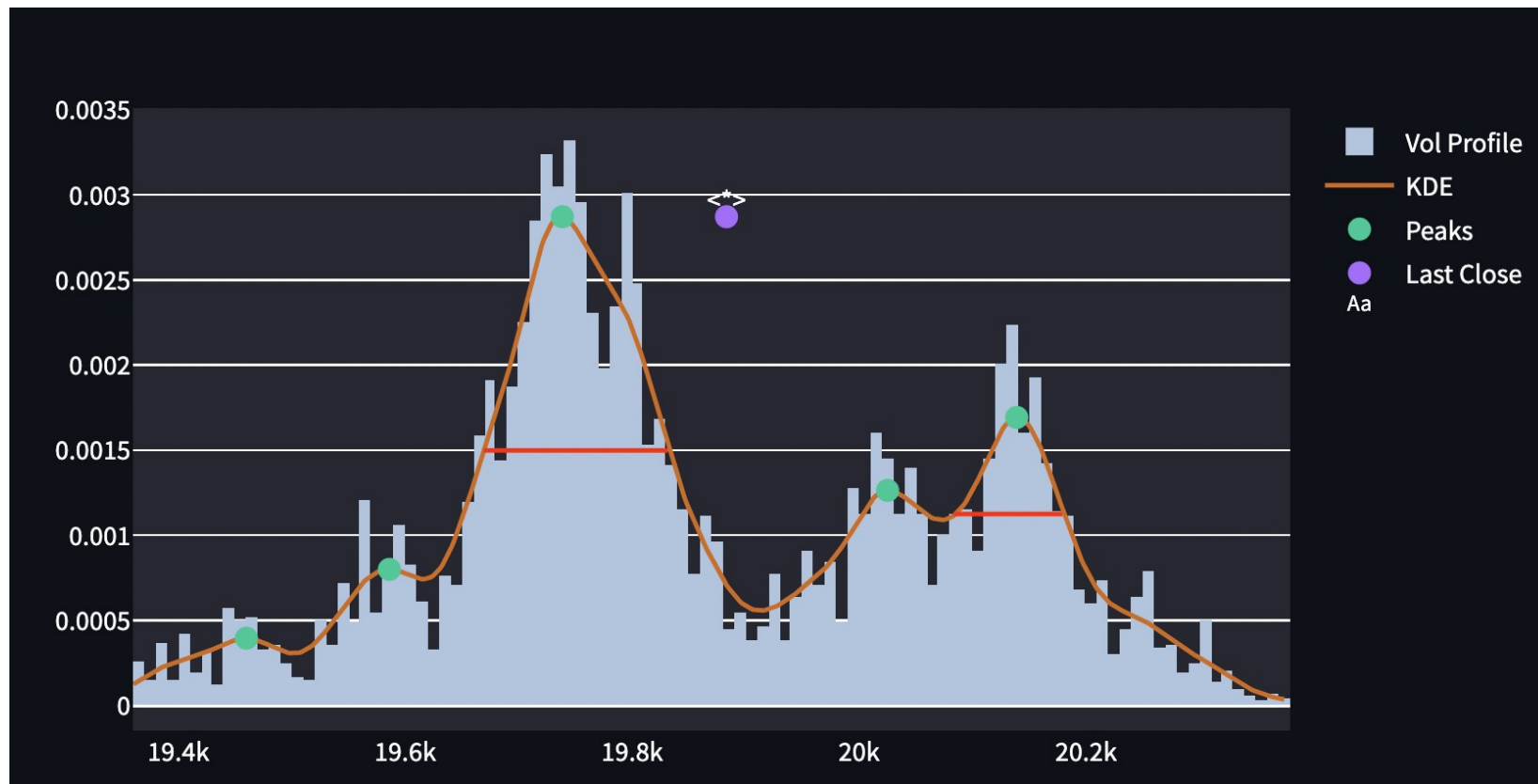
Volume Profile

Volume profile,
some of coding
part.

```
Thonny - /home/raspi/Documents/py/btc1minstreamlit.py @ 152:17
File Edit View Run Tools Help
+ [Icons]
pybtc1min.py btc1minstreamlit.py *
152 kde_factor = 0.1
153 num_samples = 500
154 kde = stats.gaussian_kde(df_fresh['close'], weights=df_fresh['volume'], bw_method=kde_factor)
155 xr = np.linspace(df_fresh['close'].min(), df_fresh['close'].max(), num_samples)
156 kdy = kde(xr)
157 ticks_per_sample = (xr.max() - xr.min()) / num_samples
158
159 def get_dist_plot(c, v, kx, ky):
160     fig = go.Figure()
161     fig.add_trace(go.Histogram(name='Vol Profile', x=c, y=v, nbinsx=150,
162                               histfunc='sum', histnorm='probability density',
163                               marker_color='#B0C4DE'))
164     fig.add_trace(go.Scatter(name='KDE', x=kx, y=ky, mode='lines', marker_color='red'))
165     return fig
166
167 #get_dist_plot(df_fresh['close'], df_fresh['volume'], xr, kdy).show()
168
169 peaks, _ = signal.find_peaks(kdy)
170 ppx = xr[peaks]
171 pky = kdy[peaks]
172
173 pk_marker_args=dict(size=10)
174 # fig = get_dist_plot(close, volume, xr, kdy)
175 # fig.add_trace(go.Scatter(name="Peaks", x=ppx, y=pky, mode='markers', marker=pk_marker_args))
176
177 min_prom = kdy.max() * 0.3
```

```
Shell
Python 3.9.2 (/usr/bin/python3)
>>>
```


Volume Profile



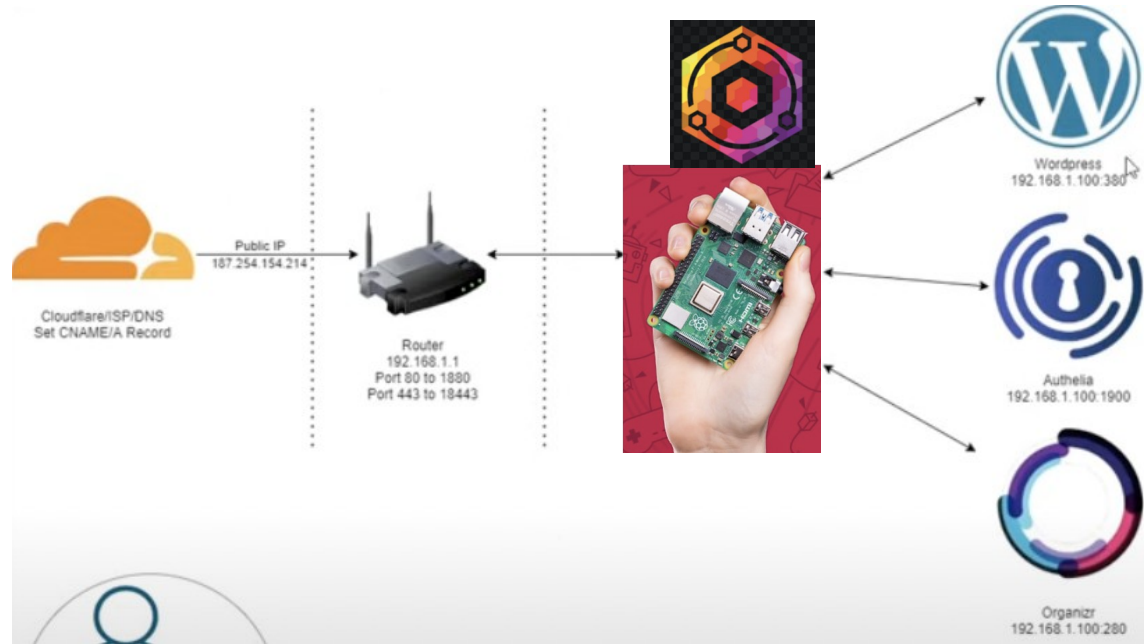
Volume Profile

- There are several strategies volume profile and auction market theory (like mean reversion strategy etc.)
- Pretty much information about in the books and web. I don't want to elaborate further here as this is not our subject.

Self hosting from Raspberry Pi 4

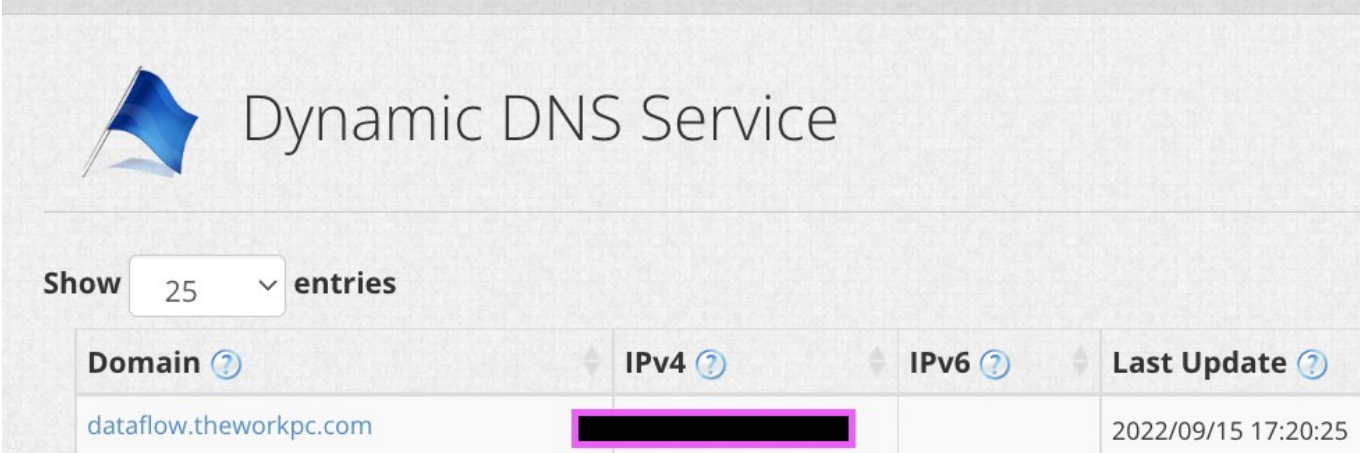
Postgresql
and nginx on
docker, 2
python files,
and nginx is
ready for
proxy host.

All on
Raspberry Pi 4



Dynamic DNS

I made the necessary arrangements to forward the domain to my ip.



The image shows a screenshot of a 'Dynamic DNS Service' web interface. At the top, there is a blue flag icon and the text 'Dynamic DNS Service'. Below this, there is a 'Show' button, a dropdown menu set to '25', and the text 'entries'. A table with four columns is displayed below: 'Domain', 'IPv4', 'IPv6', and 'Last Update'. The first row of the table shows the domain 'dataflow.theworkpc.com', an IPv4 address that has been redacted with a black box (the box has a pink border), an empty IPv6 field, and a 'Last Update' timestamp of '2022/09/15 17:20:25'. Each column header has a small blue question mark icon next to it.

Domain ?	IPv4 ?	IPv6 ?	Last Update ?
dataflow.theworkpc.com	[REDACTED]		2022/09/15 17:20:25

Nginx Proxy Manager

Settings for hosts and nginx.

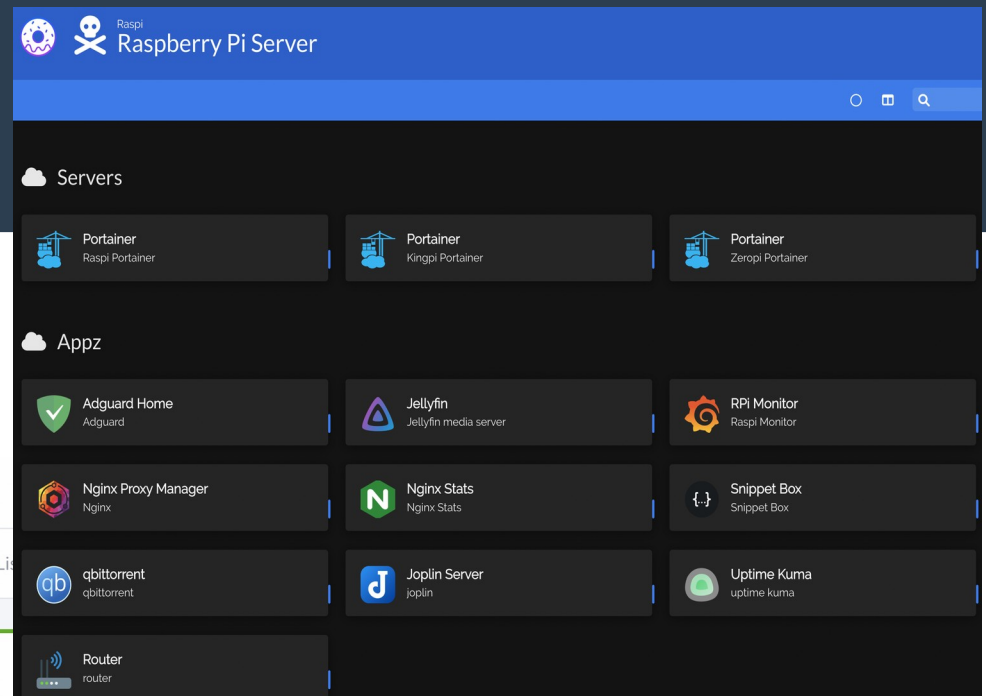
- NGINX is open source software for web serving, reverse proxying, caching, load balancing, media streaming, and more.
- I am using lot on my Raspberry Pi Servers.


 Nginx Proxy Manager

[Dashboard](#) [Hosts](#) [Access List](#)

Proxy Hosts

SOURCE	DESTINATION	SSL	ACCESS	STATUS
 dataflow.theworkpc.com Created: 16th September 2022	http://192.168.1.134:8501	HTTP only	Public	● Online
 [REDACTED]	http://192.168.1.134:8096	HTTP only	Public	● Online



**local** up 2022-09-16 22:04:58

5 stacks 19 containers - 17 2 / 5 0 15 volumes 26 images

4 8.2 GB - No tags

Things I plan to do in the future

- I am thinking of adding various analyses.
- Adding inputs where users can make their own choices.
- A structure where they can choose the desired timeframe.
(like 1h, 4h, daily, weekly candlesticks)
- Adding other pairs and allow the user to select the one they want in the sidebar.
- I did the part up to 10 years in the daily timeframe but I didn't cover it here.
- Creating a live chart with datastream (rest api)

**You can check it out, it is
live!
dataflow.theworkpc.com**