1. UNIV3 TWAP data cleaning

Cleaning UNIV3 data

In [1]:
```
import ast
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
```

In [2]:
```
# UNIV3 swap data, MKR-ETH pool
# source:
#   https://thegraph.com/hosted-service/subgraph/uniswap/uniswap-v3
#   https://github.com/Uniswap/v3-core/blob/main/contracts/UniswapV3Pool.sol
with open("./data-univ3-mkr-eth.py") as f:
    uni = ast.literal_eval(f.read())
```

In [3]:
```
df_uni = pd.DataFrame(uni)
```

In [4]:
```
df_uni = df_uni.astype({'sqrtPriceX96': 'float', 'amountUSD': 'float', 'amou
df_uni = df_uni.rename(columns={'id': 'txid'})
```

In [5]:
```
# goal: add datetime to make dataframe a time series

df_uni['dt'] = pd.to_datetime(df_uni['timestamp'], unit='s')
df_uni = df_uni.sort_values(by='dt')

# a block may have multiple swaps
# let's use the execution price of the last tx of the block, as the executio
df_uni = df_uni.groupby(by='dt').last()
```

In [6]:
```
# goal: derive MKR-USD spot after each swap ie. the execution price of the s

# from: UNIV3, MKR-ETH, 0.3% pool: https://info.uniswap.org/#/pools/0xe8c6c9
# note: this column is the price observation after each swap, not a quote gi
# derivation: sqrtPriceX96 = sqrt(price) * 2 ** 96, https://docs.uniswap.org
# derivation: https://github.com/Uniswap/v3-core/blob/main/contracts/Uniswap
df_uni['umkr-eth'] = df_uni.apply(lambda r: r['sqrtPriceX96'] ** 2 / (2 ** (

# ETH-USD effectively is from USDC-ETH 0.3% pool, $100M+ TVL: https://github
df_uni['ueth-usd'] = df_uni.apply(lambda r: abs(r['amountUSD'] / r['amount1'

# finally, we get the spot mkr-usd after the swap
df_uni['umkr-usd'] = df_uni.apply(lambda r: r['umkr-eth'] * r['ueth-usd'], a
```

In [7]:
```
# make dataframe a timeseries with 1-second resolution: start, start+1s, sta
df_srange = pd.DataFrame(pd.date_range(df_uni.index.min(), df_uni.index.max(
df_uni = df_srange.join(df_uni)
```

In [8]:
```
# derive the accumulated price column

df_uni['umkr-usd-shift'] = df_uni['umkr-usd'].shift(1)
df_uni['umkr-usd-spot'] = df_uni['umkr-usd-shift'].fillna(method='ffill')
df_uni
```

Out[8]:

dt	txid	timestamp	amount0
2022-08-01 07:17:51	0x0782b63ab0ea2eb031c4539bc0fbf49c0bc44cf6a93b...	1659338271	4.284759831749187092
2022-08-01 07:17:52	NaN	NaN	NaN
2022-08-01 07:17:53	NaN	NaN	NaN
2022-08-01 07:17:54	NaN	NaN	NaN
2022-08-01 07:17:55	NaN	NaN	NaN
...	...	...	...
2022-11-17 19:50:31	NaN	NaN	NaN
2022-11-17 19:50:32	NaN	NaN	NaN
2022-11-17 19:50:33	NaN	NaN	NaN
2022-11-17 19:50:34	NaN	NaN	NaN
2022-11-17 19:50:35	0xe52c780227272a4220783a9882737f7ad820b7b22c8e...	1668714635	7.456844435909475845

9376365 rows × 11 columns

In [9]:
```
df_uni['umkr-usd-spot-obsv'] = df_uni.where(df_uni['umkr-usd'].notna())['umk
df_uni['umkr-usd-interpolated'] = df_uni['umkr-usd-spot-obsv'].interpolate(m
```

```
start  = pd.Timestamp('2022-11-17 17:45:32').to_pydatetime()
end    = pd.Timestamp('2022-11-17 18:03:50').to_pydatetime()
mask   = (start < df_uni.index) & (df_uni.index < end)

# `umkr-usd-spot` is the MKR quote before swap
# `umkr-usd-interpolated` is the interpolation between quotes ie. draw a str

# Eg. for timestamps a) 2022-11-17 17:45:35 b) 2022-11-17 18:03:47
# `umkr-usd-interpolated` == `umkr-usd-spot` ie. when there are swaps observ
# the values in between a) and b) are inferred by applying a constant amount

# note `umkr-usd-spot` changes only **after the swap** as is implemented in

df_uni[mask][['umkr-usd', 'umkr-usd-spot', 'umkr-usd-interpolated']]
```

Out[9]:

dt	umkr-usd	umkr-usd-spot	umkr-usd-interpolated
2022-11-17 17:45:33	NaN	663.320663	663.320713
2022-11-17 17:45:34	NaN	663.320663	663.320688
2022-11-17 17:45:35	659.278154	663.320663	663.320663
2022-11-17 17:45:36	NaN	659.278154	663.316961
2022-11-17 17:45:37	NaN	659.278154	663.313259
...	...	...	...
2022-11-17 18:03:45	NaN	659.278154	659.285557
2022-11-17 18:03:46	NaN	659.278154	659.281855
2022-11-17 18:03:47	661.490661	659.278154	659.278154
2022-11-17 18:03:48	NaN	661.490661	659.278635
2022-11-17 18:03:49	NaN	661.490661	659.279116

1097 rows × 3 columns

In [10]:
```
# for calculating twap using accumulated (and interpolated) prices
df_uni['umkr-usd-accum'] = df_uni['umkr-usd-interpolated'].cumsum()
```

TWAP calculation (based on accumulated prices)

https://uniswapv3book.com/docs/milestone_5/price-oracle/#how-uniswap-price-oracle-works

In [11]:
```
def twap(df_uni, minutes):
    def f(row):
        # (a2 - a1) / 3600
        # row.name is the timestamp, ie. the index, of the row
        a1 = df_uni.loc[row.name - timedelta(minutes=minutes)]['umkr-usd-acc
        a2 = df_uni.loc[row.name]['umkr-usd-accum']
        return (a2 - a1) / (minutes * 60)
    return f
```

In [12]:
```
# filter down to the timestamps where there are swaps, for efficiency sake (
df_uni_swap_observed = df_uni[df_uni['umkr-usd'].notna()]

# **at last** calculate the 1 hour TWAP
idx = df_uni_swap_observed.index
df_filterd = df_uni_swap_observed[idx > idx.min() + timedelta(minutes=60)]

df_uni['umkr-usd-1htwap'] = df_filterd.apply(twap(df_uni, 60), axis=1)
df_uni['umkr-usd-1htwap'] = df_uni['umkr-usd-1htwap'].interpolate(method='ti

df_uni['umkr-usd-1mtwap'] = df_filterd.apply(twap(df_uni, 1), axis=1)
df_uni['umkr-usd-1mtwap'] = df_uni['umkr-usd-1mtwap'].interpolate(method='ti

# rows before 1st swap observation will be NaN, let's filter them out
df_uni = df_uni[df_uni['umkr-usd-1htwap'].notna()]
```

In [13]:
```
df_uni[['umkr-usd-1htwap', 'umkr-usd-1mtwap']]
```

Out[13]:

dt	umkr-usd-1htwap	umkr-usd-1mtwap
2022-08-01 08:48:50	1100.903984	1100.568007
2022-08-01 08:48:51	1100.904483	1100.574451
2022-08-01 08:48:52	1100.904982	1100.580895
2022-08-01 08:48:53	1100.905481	1100.587339
2022-08-01 08:48:54	1100.905981	1100.593783
...	...	...
2022-11-17 19:50:31	661.070771	661.143400
2022-11-17 19:50:32	661.070886	661.144507
2022-11-17 19:50:33	661.071002	661.145614
2022-11-17 19:50:34	661.071118	661.146721
2022-11-17 19:50:35	661.071233	661.147829

9370906 rows × 2 columns

In [14]:
```
# df_uni[df_uni.index.duplicated()]
```

That's it!

2. Chainlink oracle update data cleaning

In [15]:
```
# source: https://thegraph.com/hosted-service/subgraph/openpredict/chainlink
with open("./data-cl-mkr-usd.py") as f:
    cl = ast.literal_eval(f.read())
```

In [16]:
```
df_cl = pd.DataFrame(cl)
df_cl = df_cl.astype({'price': 'float'})
df_cl['cdt'] = pd.to_datetime(df_cl['timestamp'], unit='s') # datetime
df_cl['cmkr-usd'] = df_cl['price'] / 1e8
df_cl = df_cl.drop(['timestamp', 'price'], axis=1)
df_cl = df_cl.set_index('cdt')
df_cl = df_cl.sort_index()
df_cl
```

Out[16]:

cdt	cmkr-usd
2022-07-13 00:55:51	831.099155
2022-07-13 01:55:52	835.288180
2022-07-13 02:08:30	840.866118
2022-07-13 02:55:44	840.477612
2022-07-13 03:55:57	836.213225
...	...
2022-11-17 12:30:35	664.493542
2022-11-17 13:21:47	657.017308
2022-11-17 14:22:23	654.740000
2022-11-17 15:08:59	662.610000
2022-11-17 16:09:23	659.380737

4000 rows × 1 columns

In [17]:
```
# df_cl[df_cl.index.duplicated()]
```

3. Analysis

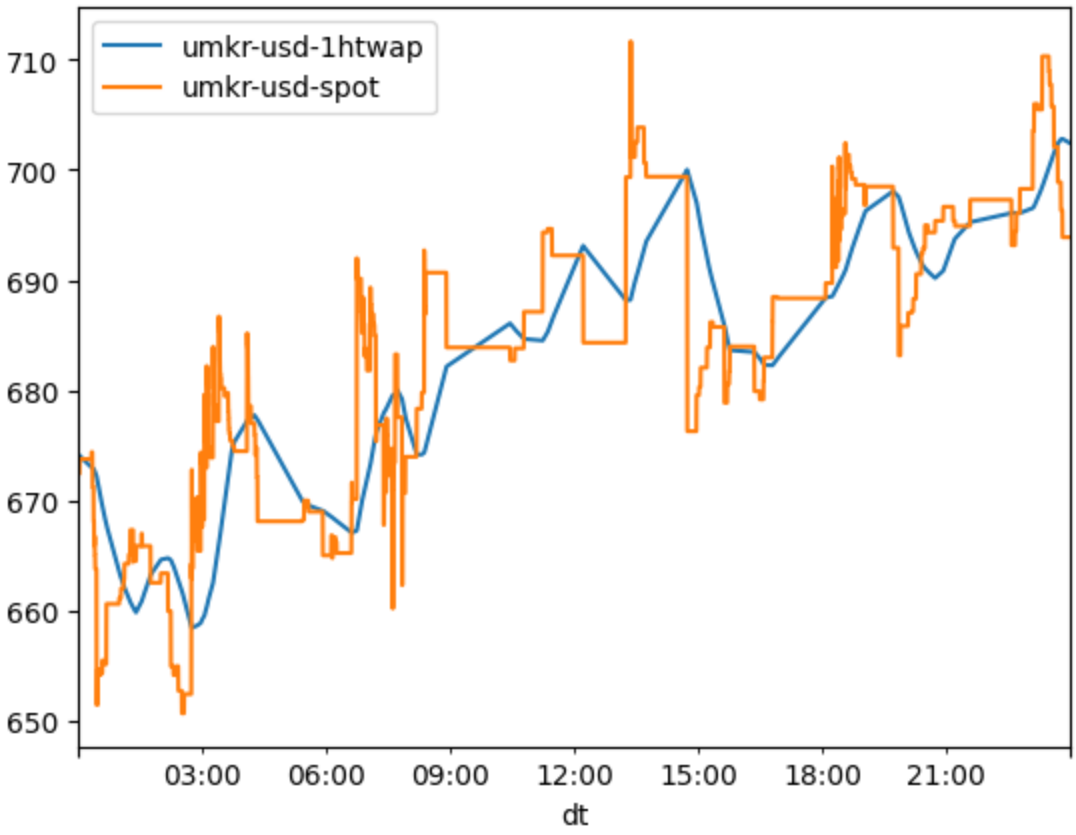
In [18]:
```
start  = pd.Timestamp('2022-08-15 00:00:00').to_pydatetime()
end    = pd.Timestamp('2022-11-15 00:00:00').to_pydatetime()
mask   = (start < df_uni.index) & (df_uni.index < end)
df_uni = df_uni[mask]
```

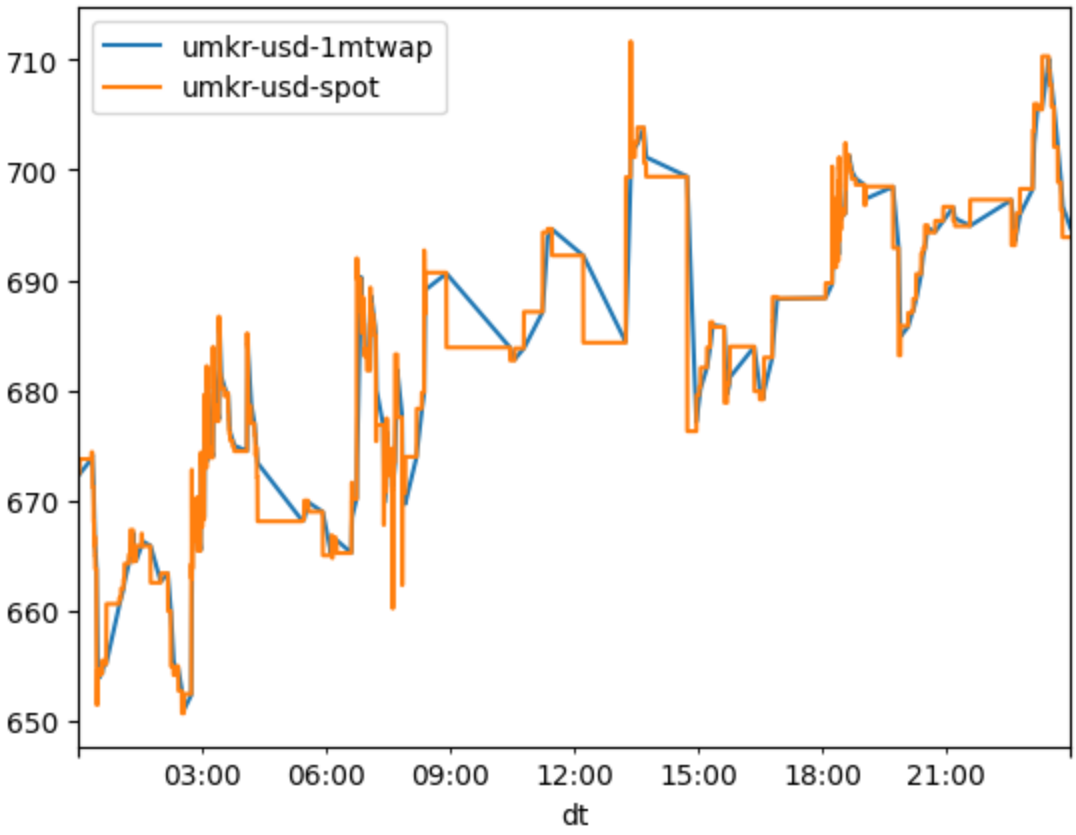
Peek: 1 day data

In [19]:
```
start  = pd.Timestamp('2022-11-14 00:00:00').to_pydatetime()
end    = pd.Timestamp('2022-11-15 00:00:00').to_pydatetime()
mask   = (start < df_uni.index) & (df_uni.index < end)

df_uni[mask][['umkr-usd-1htwap', 'umkr-usd-spot']].plot()
df_uni[mask][['umkr-usd-1mtwap', 'umkr-usd-spot']].plot()
```

Out[19]:
```
<AxesSubplot: xlabel='dt'>
```





Difference between TWAP and spot price

In [20]:
```
df_joined = df_uni.join(df_cl, how='left')
```

In [21]:
```
df_joined['cmkr-usd'] = df_joined['cmkr-usd'].fillna(method='ffill')
```

In [63]:
```
df_joined['mkr-usd-devi-1h'] = df_joined['umkr-usd-1htwap'] - df_joined['umk
df_joined['mkr-usd-devi-1hfrac'] = df_joined['mkr-usd-devi-1h'] / df_joined[
df_joined['mkr-usd-devi-1m'] = df_joined['umkr-usd-1mtwap'] - df_joined['umk
df_joined['mkr-usd-devi-1mfrac'] = df_joined['mkr-usd-devi-1m'] / df_joined[
```

In [64]:
```
pd.set_option('display.float_format', lambda x: '%.4f' % x)
```

In [65]:
```
mask_1h = df_joined['mkr-usd-devi-1hfrac'] >= 0
df_joined[mask_1h]['mkr-usd-devi-1hfrac'].describe()
```

Out[65]:
```
count   3934676.0000
mean          0.0048
std           0.0058
min           0.0000
25%           0.0013
50%           0.0030
75%           0.0062
max           0.1054
Name: mkr-usd-devi-1hfrac, dtype: float64
```

In [66]:
```
mask_1m = df_joined['mkr-usd-devi-1mfrac'] >= 0
df_joined[mask_1m]['mkr-usd-devi-1mfrac'].describe()
```

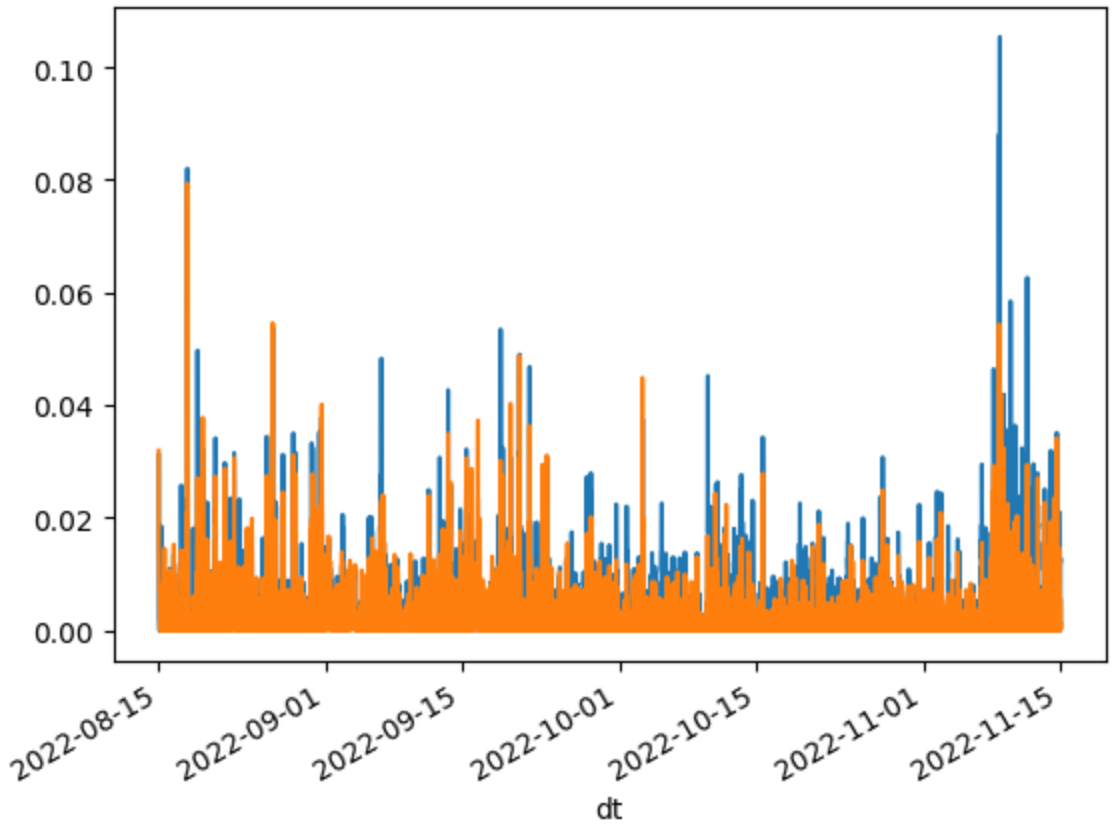
Out[66]:
```
count   3908499.0000
mean          0.0019
std           0.0029
min           0.0000
25%           0.0004
50%           0.0010
75%           0.0022
max           0.0794
Name: mkr-usd-devi-1mfrac, dtype: float64
```

Pattern of over-valuation of MKR

Over-valuation over time

In [67]:
```
df_joined[mask_1h]['mkr-usd-devi-1hfrac'].plot()
df_joined[mask_1m]['mkr-usd-devi-1mfrac'].plot()
```

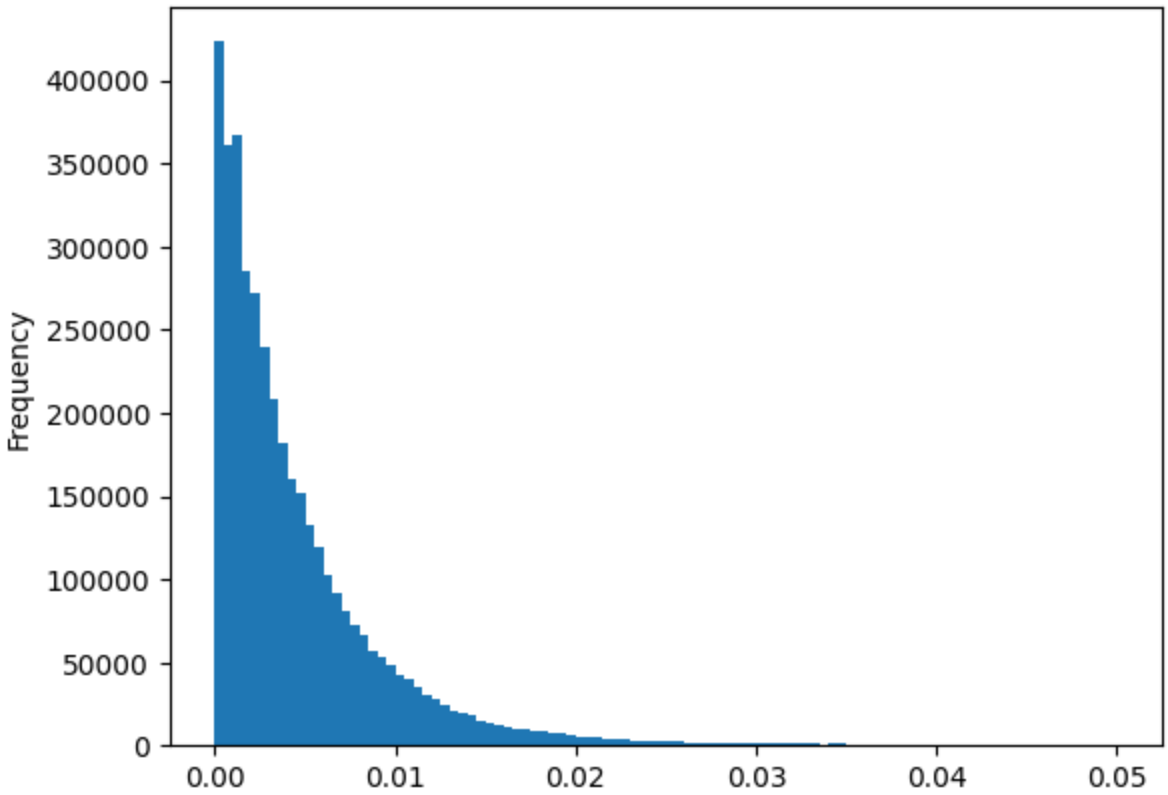
Out[67]:
```
<AxesSubplot: xlabel='dt'>
```



Degree of over-valuation

In [72]:
```
df_joined[mask_1h]['mkr-usd-devi-1hfrac'].plot.hist(bins=100, range=(0, 0.05
```

Out[72]:
```
<AxesSubplot: ylabel='Frequency'>
```



4. Simulation

https://mathisonian.github.io/kde/

In [69]:
```
import scipy
```

In [70]:
```
lot = 30000

# mean of 50 trials, each time simulate a $3m open market program, $30,000 e
# find the total over payment comparing 1h-TWAP to spot price
def print_stats(df):
    kernel = scipy.stats.gaussian_kde(df)
    trials = np.asarray([
        np.sum(kernel.resample(100)[0]) * lot
        for i in range(50)]
    )
    print(pd.Series(trials).describe())

print('---1 hour TWAP---')
print_stats(df_joined[mask]['mkr-usd-devi-1hfrac'])

print('---1 minute TWAP---')
print_stats(df_joined[mask]['mkr-usd-devi-1mfrac'])
```

```
---1 hour TWAP---
count      50.0000
mean     8941.3001
std      2156.6957
min      3892.8330
25%      7433.8469
50%      8841.7192
75%     10514.1229
max     15499.7326
dtype: float64
---1 minute TWAP---
count     50.0000
mean    5683.1317
std     1218.0798
min     3718.2578
25%     4806.5711
50%     5471.2300
75%     6267.4156
max     9370.5347
dtype: float64
```

In [71]:
```
print(f'Using 3 months data, over 2022-08 through 2022-11, from 1 hour TWAP
```

```
Using 3 months data, over 2022-08 through 2022-11, from 1 hour TWAP to 1 mi
nute TWAP, there is a 35% reduction of over-paying.
```