In [1]:
```
%reload_ext watermark
%watermark -a "borrito"
%watermark
```

```
Author: borrito

Last updated: 2021-12-21T22:44:29.433878+01:00

Python implementation: CPython
Python version       : 3.8.0
IPython version      : 7.28.0

Compiler    : MSC v.1916 64 bit (AMD64)
OS          : Windows
Release     : 10
Machine     : AMD64
Processor   : AMD64 Family 23 Model 96 Stepping 1, AuthenticAMD
CPU cores   : 16
Architecture: 64bit
```

In [2]:
```
"""set random seeds"""
import random
import numpy as np

random.seed(42)
np.random.seed(42)
```

Index

Problem Statement

Hey the members of the YFI Data Analysis team, this is borrito3179. I recently joined the chat and was scrolling over the previous chats to spy on stuff that I may be able to work on, then noticed the pinned message and thought I could also try contributing to the problem. But I am a real noob in these concepts, so I wanted to check whether my current understanding is correct before I actually dive into the data and models. Please feel free to correct anything that I am wrong about, and of course, you can always ask questions about this notebook.

So this is my current understanding of the problem -

(figure from Compare Automated Market Maker returns)

we want to create a forecast of this graph for each pool, that has some additional caveats:

- it must vary according to the difference in TVL we may introduce through investing into this pool, cf. Link from Wot_Is Goin_On
- some uncertainty measure of the forecast must be given, e.g., the variance
- for now we only consider the simplest version of the Constant Function Market Makers, $xy = k$

The first part requires us to formulate the fee income using the TVL and trading volumes. So I took the formula from this post:

$$\text{fee income} \approx \text{constant} \times \sqrt{1 + \text{price change}} \times \frac{\text{trading volume}}{\text{TVL}} \tag{1}$$

The relationship is not an equality, since there may be time delays in minting/burning the pool tokens and there may also be goveranance tokens/bounties that may change the net income. Indeed, this value may differ from the actual increase in value of the pool tokens, which is the way the fee income in the graphs at Compare Automated Market Maker returns were calculated:

$$\text{fee income} = \frac{\sqrt{x_1 y_1}}{p_1} \Big/ \frac{\sqrt{x_0 y_0}}{p_0} - 1.0 \tag{2}$$

where x_t, y_t, p_t stands for the reserves for token x, y and the pool token p at time t.

Thus, I first tried to validate (1) by comparing it to (2), and to see if their ratio remains constant for the most of the time.

In [3]:
```
"""load UNI/WETH data from uniswapv2, data from https://github.com/reednaaxeelaa/AMMroi"""
import os
import numpy as np
import pandas as pd

path = '../data/uniswapv2/roi/uni&weth.csv'
df = pd.read_csv(os.path.join(path))
df.head(5)
```

Out[3]:

	block	timestamp	reserve0	reserve1	total supply	trade volume	price	sINV
0	10877000	1600310338	1.609515e+05	1103.622803	8699.289593	7.293238e+06	145.839251	1.532054
1	10878000	1600323285	9.099534e+05	5921.362151	44763.490921	2.331044e+07	153.672986	1.639821
2	10879000	1600336910	1.257959e+06	9697.752834	65620.409676	4.037519e+07	129.716586	1.683178
3	10880000	1600349788	2.224438e+06	15595.313813	109628.351932	5.111244e+07	142.635014	1.698965
4	10881000	1600362916	2.948266e+06	27786.343197	167417.230750	6.168956e+07	106.104851	1.709617

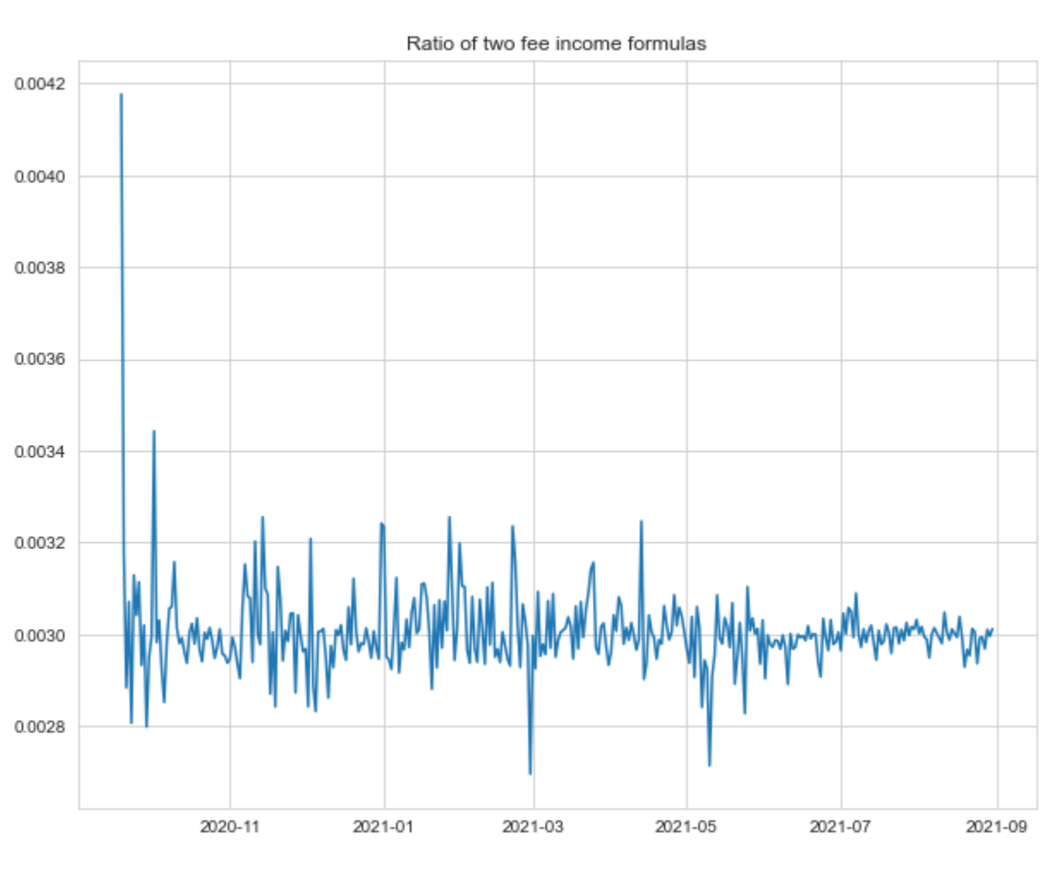
In [4]:
```
"""convert to daily data"""
df.index = pd.to_datetime(df.timestamp, unit='s')
df = df.groupby(pd.Grouper(freq='1D', closed='right', label='right')).last()
df.head(5)
```

Out[4]:

	block	timestamp	reserve0	reserve1	total supply	trade volume	price	sINV
timestamp								
2020-09-18	10882000	1600375976	3.425707e+06	31574.584301	191466.028545	7.176193e+07	108.495724	1.717719
2020-09-19	10889000	1600469993	3.153338e+06	62165.565679	252183.967176	1.205525e+08	50.724833	1.755669
2020-09-20	10895000	1600549386	4.439527e+06	68569.097626	311777.773974	1.402334e+08	64.745305	1.769650
2020-09-21	10902000	1600641298	4.980687e+06	64944.592952	319580.918355	1.581727e+08	76.691323	1.779653
2020-09-22	10908000	1600719844	5.289162e+06	69934.583528	340544.935481	1.704119e+08	75.630129	1.785932

In [5]:
```
"""compare two fee income formulas"""
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_style('whitegrid')

# eq(1)
price_change = df.price / df.price.shift() - 1.0
tvl = df.reserve0 + df.reserve1 * df.price  # TVL measured in UNI
vol = df['trade volume'].diff()             # volume measured in UNI
fee1 = np.sqrt(1 + price_change) * vol / tvl

# eq(2)
sqrt_xy = df.sINV
fee2 = sqrt_xy / sqrt_xy.shift() - 1.0

plt.figure(figsize=(10, 8))
plt.title("Ratio of two fee income formulas")
plt.plot(fee2 / fee1)
```

Out[5]:
```
[<matplotlib.lines.Line2D at 0x1dc7b92b490>]
```



The above graph seemed to remain constant near the fee rate (0.3%), so I decided to work with (1) at the moment.

Then now, we have 4 variables to either assume or estimate from the data:

- price change
- trading volume
- TVL
- the proportional coefficient (constant)

I thought the former 2 (price change, trading volume) are the variables that we should try to extrapolate from data, and the latter 2 (TVL, proportional coefficient) are the values that should be given, or manually set by the strategy. For instance, if a strategy aims to invest a portfolio worth of $1 million, the TVL would have to change accordingly.

To this end, now we need to use some cool statistics to forecast the first two variables, the price and volume. If I could know in more detail what the governing evaluation metric for the portfolio allocations/strategies is, I may be able to model it directly, but for now I had no idea so I went on to estimate the mean and variance of the expected fee income for a fixed horizon. For the following I have set the horizon to be equal to 10 days.

Back to Top

Minimal Working Example

Ideally, one could model the joint distribution of the dynamic process of price and volume, but my statistical powers are too weak to just pop that out of thin air. So before I take some more time and study the data, I wanted to first check if this workflow is on the right track using a simpler setting.

That is, I have modelled the price and volume separately using univariate AR-GARCH processes (even without proper statistical tests and PACF plots, my apologies) and tried to simulate scenarios from the two processes. Let me assume for the moment that this modelling is super adequate and explains the dynamics correctly.

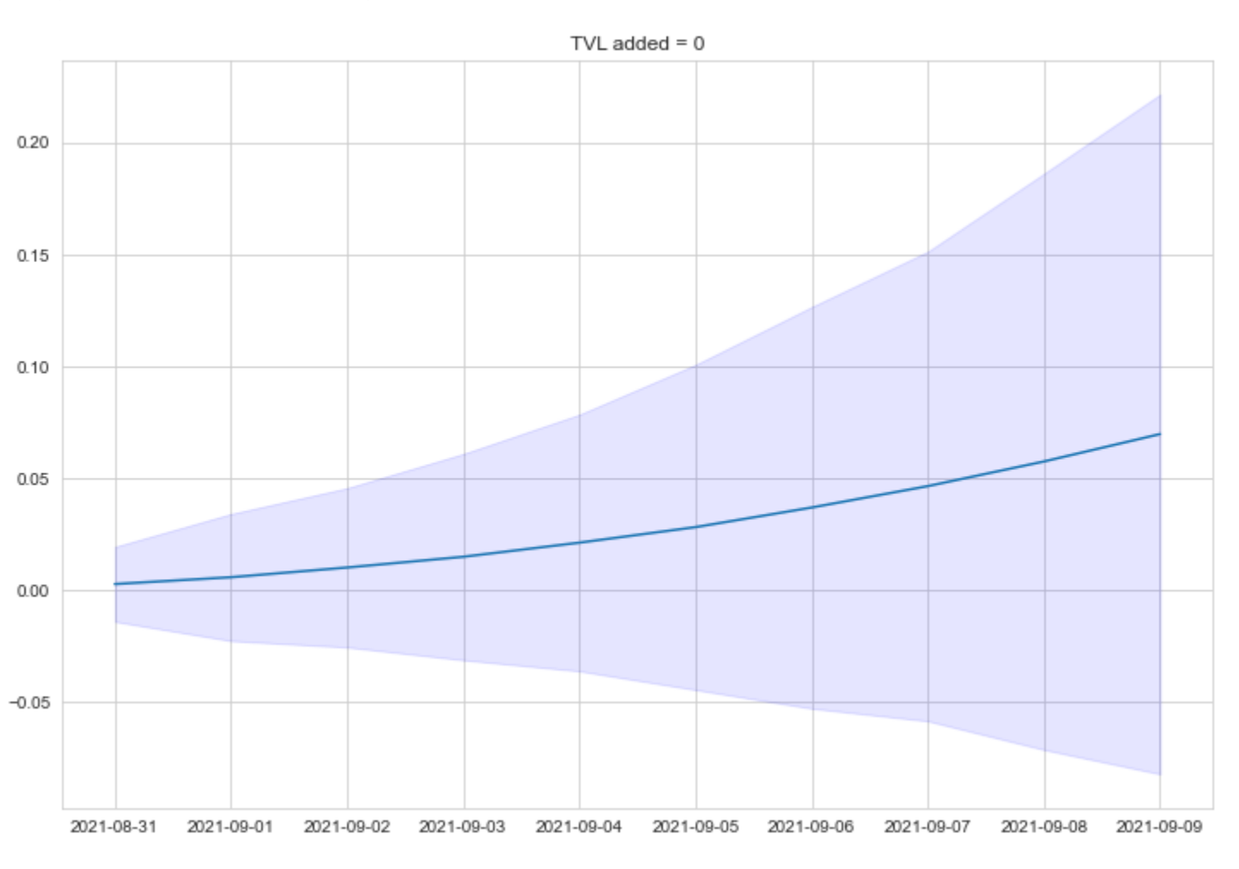
The formula for the impermanent loss used in the following code is:

$$\text{impermanent loss} = 2\sqrt{\frac{\text{price}_0}{\text{price}_1}} \Big/ \left(1 + \frac{\text{price}_0}{\text{price}_1}\right)$$

In [6]:
```
from arch.univariate import ARX, GARCH, SkewStudent
from arch.univariate.base import DataScaleWarning
import warnings
warnings.simplefilter("ignore", category=DataScaleWarning)  # did not even scale properly

horizon = 10
num_samples = 10000
fee_rate = 0.003

def simulate_returns(tvl_added=0.0):
    # create simulations for price change
    diffs = np.log(df.price).diff().dropna()  # log-diff
    arx = ARX(diffs, lags=[1, 2, 3])  # arbitrary
    arx.volatility = GARCH()
    arx.distribution = SkewStudent()
    res = arx.fit(disp="off")

    forecasts = res.forecast(horizon=horizon, method='simulation', simulations=num_samples, reindex=False)
    _price_changes = np.exp(forecasts.simulations.values[-1]) - 1.0  # (num_samples, horizon)

    # create simulations for volume
    vols = np.log(df['trade volume'].diff().dropna())  # just log
    arx = ARX(vols, lags=[1, 2, 3, 4])  # arbitrary
    arx.volatility = GARCH()
    arx.distribution = SkewStudent()
    res = arx.fit(disp="off")

    forecasts = res.forecast(horizon=horizon, method='simulation', simulations=num_samples, reindex=False)
    _volumes = np.exp(forecasts.simulations.values[-1])              # (num_samples, horizon)

    # estimated trading fee
    TVL = df.reserve0[-1] + df.reserve1[-1] * df.price[-1] + tvl_added
    fee = np.sqrt(_price_changes + 1.0) * _volumes / TVL * fee_rate

    # impermanent loss
    inv_price_changes = 1 / (1 + _price_changes)
    il = 2 * np.sqrt(inv_price_changes) / (1 + inv_price_changes) - 1.0

    # net return
    return (np.cumprod((fee + 1.0) * (il + 1.0), axis=1) - 1.0) * 100  # return in %
```

In [7]:
```
from datetime import timedelta

returns = simulate_returns(0.0)  # assume that there is no TVL added
means = returns.mean(axis=0)
stds = returns.std(axis=0)

plt.figure(figsize=(12, 8))
plt.title("TVL added = 0")
xs = pd.date_range(df.index[-1] + timedelta(days=1), periods=horizon)
plt.plot(xs, means)
plt.fill_between(xs, means - 1.0 * stds, means + 1.0 * stds, color='b', alpha=.1)
```

Out[7]:
```
<matplotlib.collections.PolyCollection at 0x1dc7d4bc2e0>
```

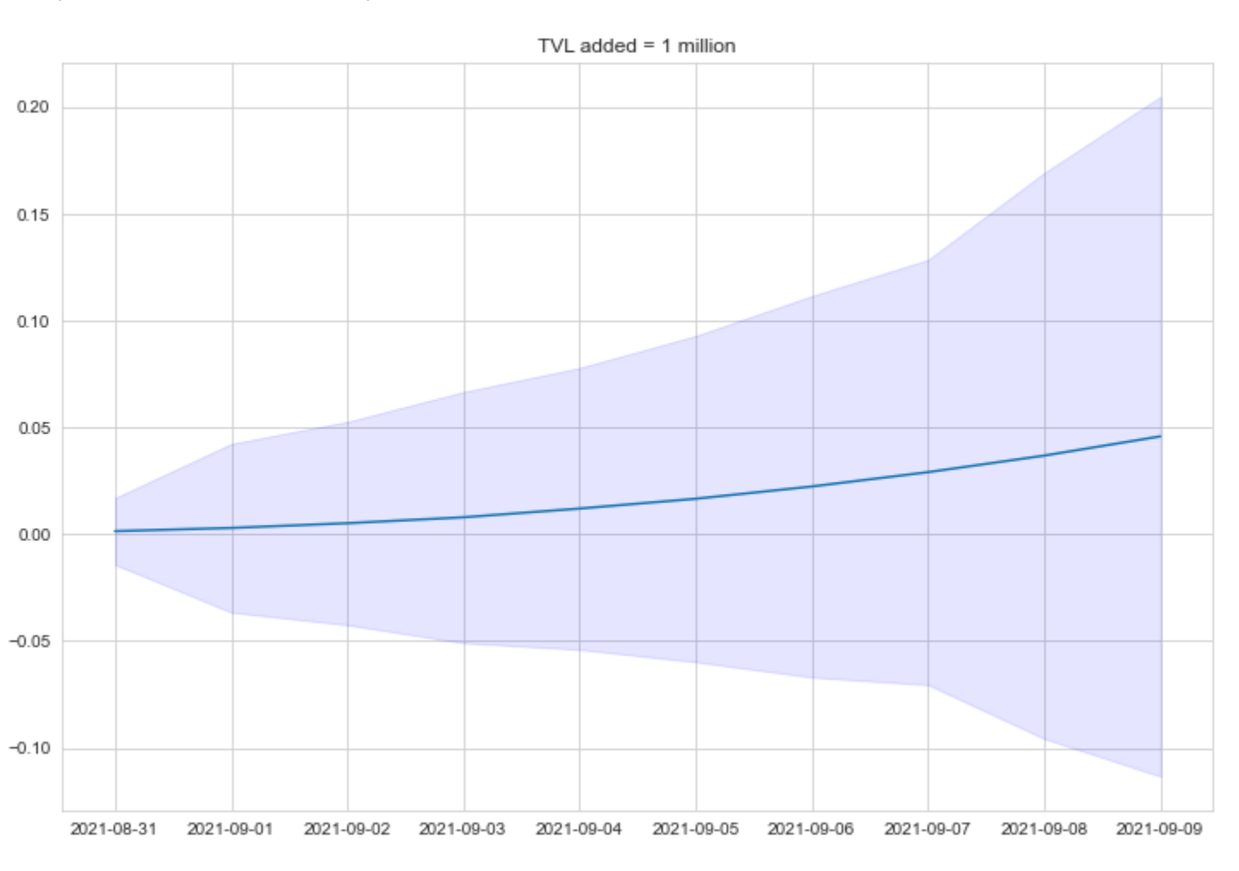


Then from the fitted processes, we can create a graph like the above using stochastic simulations. However, as you can see from the graph, the current model has a low precision in estimating the future confidence intervals, definitely has a lot of room for improvement.

Now since we can change the TVL by changing the input argument to the function, we may create multiple plots varying the value for TVL:

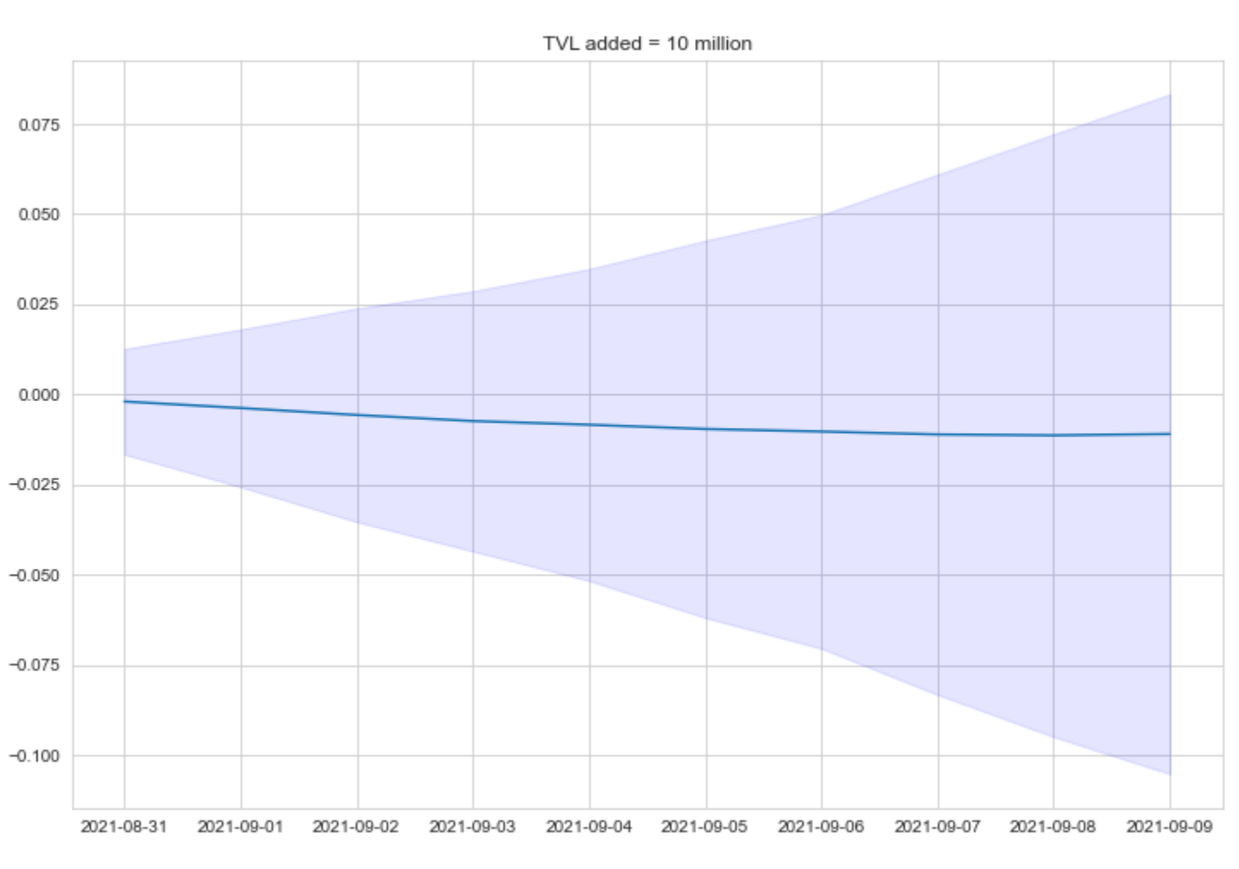
In [8]:
```
returns = simulate_returns(1e6)  # 1 million
means = returns.mean(axis=0)
stds = returns.std(axis=0)

plt.figure(figsize=(12, 8))
plt.title("TVL added = 1 million")
xs = pd.date_range(df.index[-1] + timedelta(days=1), periods=horizon)
plt.plot(xs, means)
plt.fill_between(xs, means - 1.0 * stds, means + 1.0 * stds, color='b', alpha=.1)
```

Out[8]:
```
<matplotlib.collections.PolyCollection at 0x1dc7d4eddf0>
```



In [9]:
```
returns = simulate_returns(1e7)  # 10 million
means = returns.mean(axis=0)
stds = returns.std(axis=0)

plt.figure(figsize=(12, 8))
plt.title("TVL added = 10 million")
xs = pd.date_range(df.index[-1] + timedelta(days=1), periods=horizon)
plt.plot(xs, means)
plt.fill_between(xs, means - 1.0 * stds, means + 1.0 * stds, color='b', alpha=.1)
```

Out[9]:
```
<matplotlib.collections.PolyCollection at 0x1dc7d873760>
```



We can see that the expected yield drops as the size of the investment increases, as stated in Link from Wot_Is Goin_On.

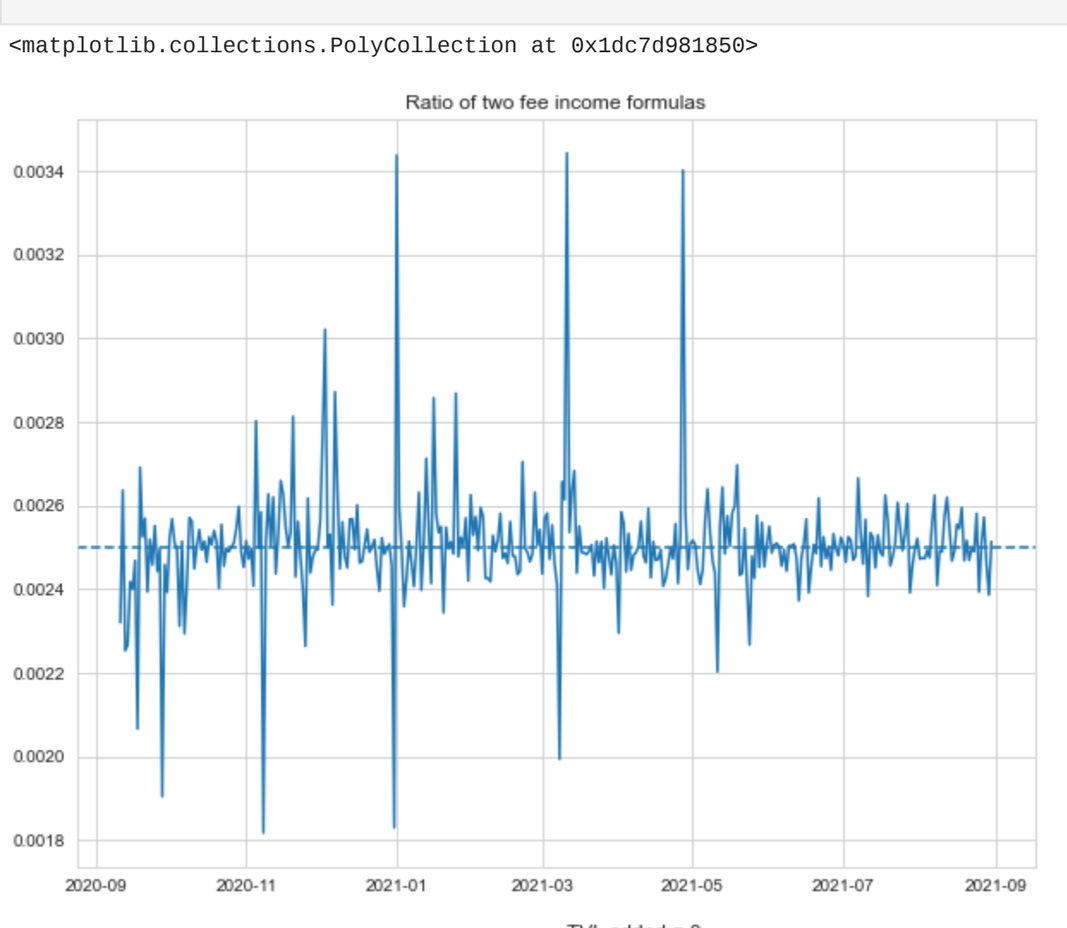
Back to Top

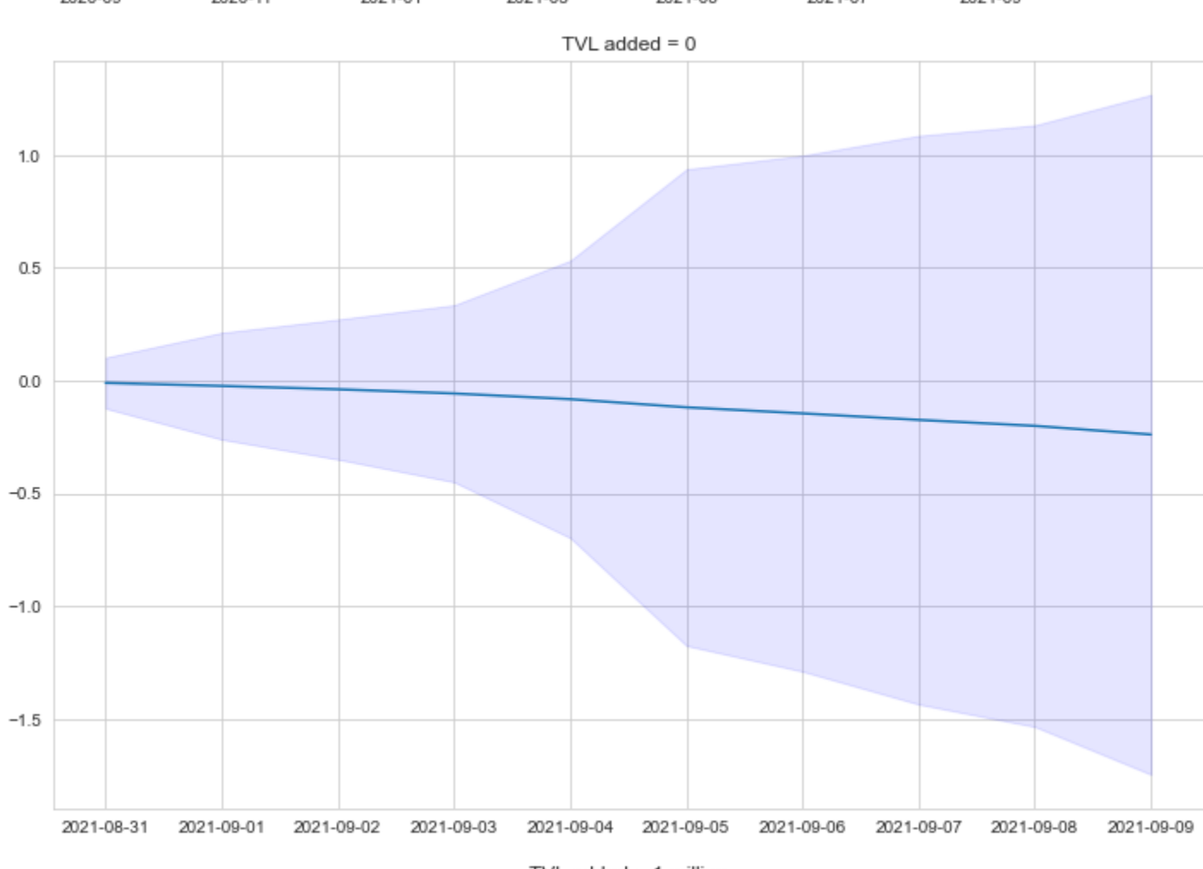
Some More Outputs

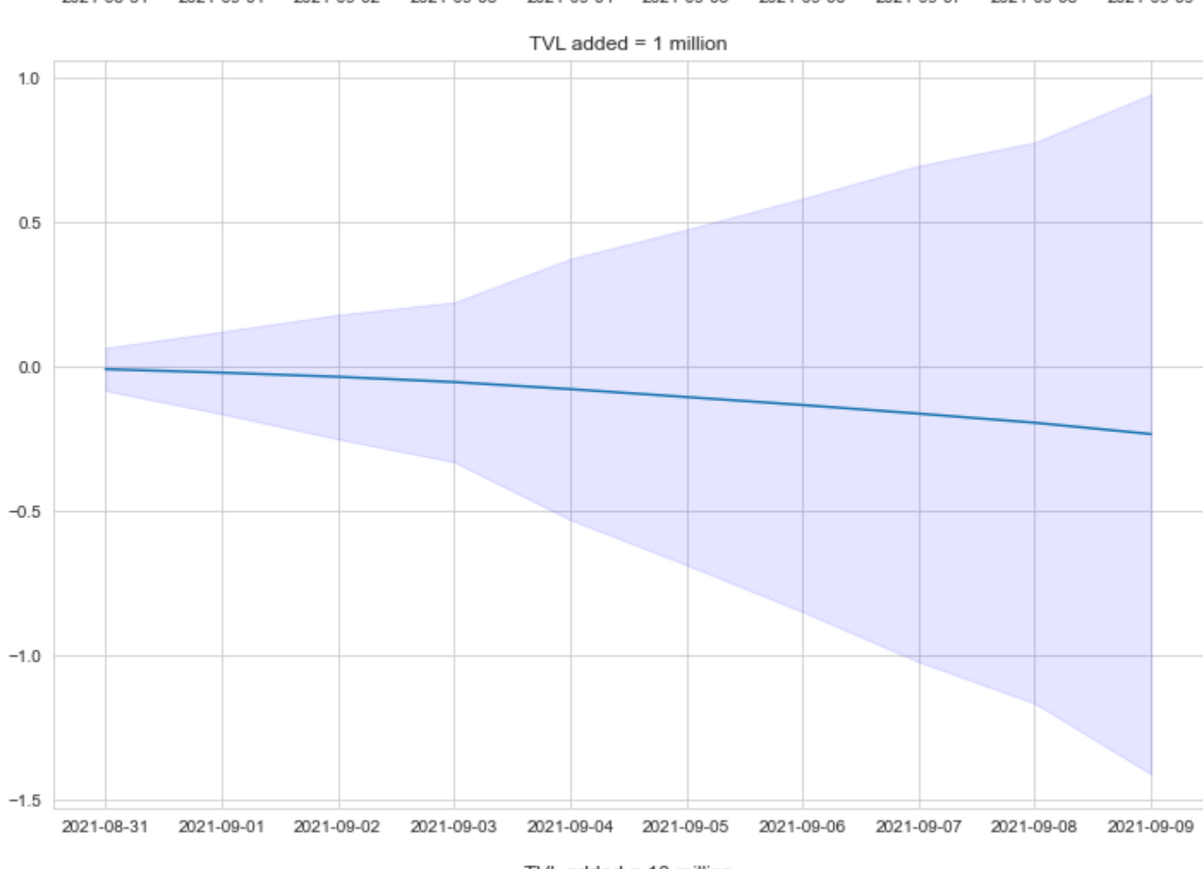
This was it for this notebook, but just for those who may be curious I include some extra plots from a different exchange. This time I have just estimated the proportional coefficient by the median of the ratios (2) / (1).

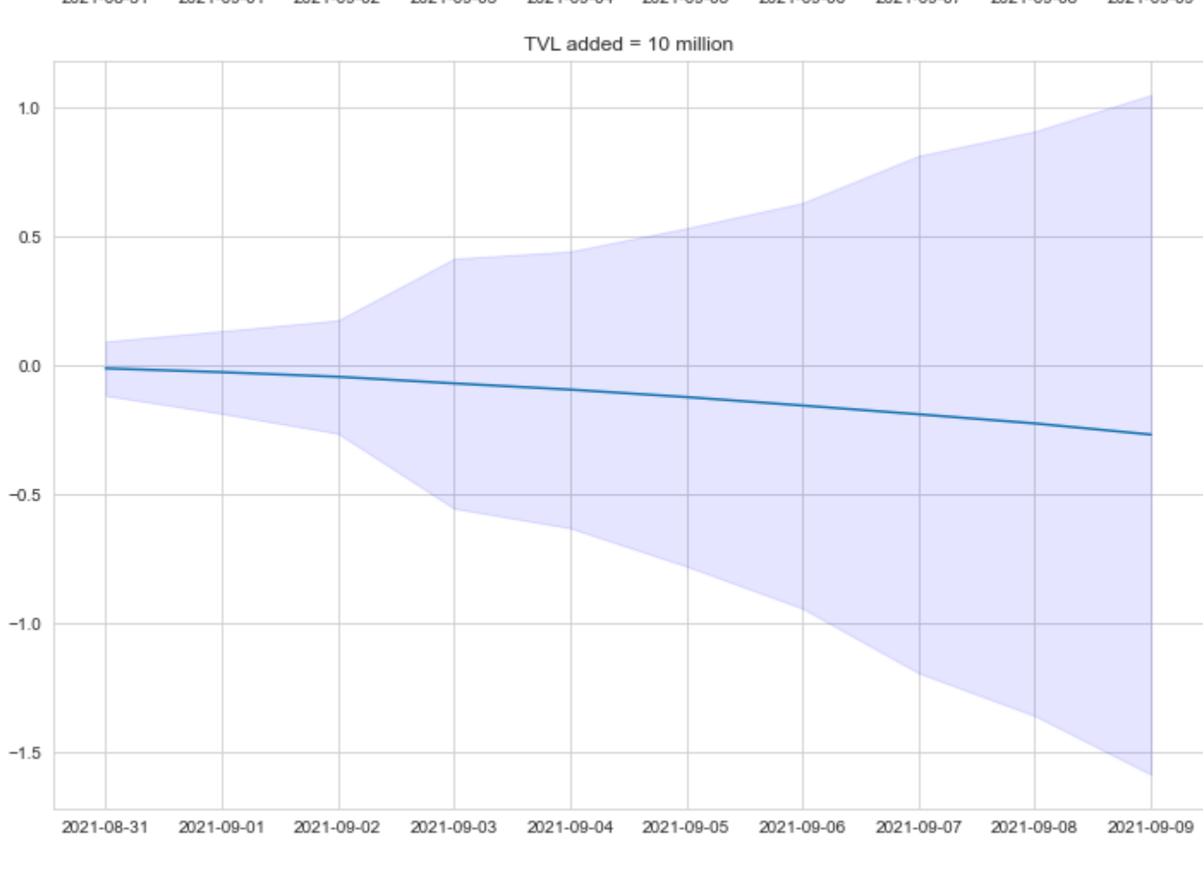
In [10]:
```
"""load SUSHI/WETH data from sushiswap, data from https://github.com/reednaaxeelaa/AMMroi"""
path = '../data/sushiswap/roi/sushi&weth.csv'
df = pd.read_csv(os.path.join(path))
df.index = pd.to_datetime(df.timestamp, unit='s')
df = df.groupby(pd.Grouper(freq='1D', closed='right', label='right')).last()

"""compare two fee income formulas"""
# eq(1)
price_change = df.price / df.price.shift() - 1.0
tvl = df.reserve0 + df.reserve1 * df.price  # TVL measured in UNI
vol = df['trade volume'].diff()             # volume measured in UNI
fee1 = np.sqrt(1 + price_change) * vol / tvl

# eq(2)
sqrt_xy = df.sINV
fee2 = sqrt_xy / sqrt_xy.shift() - 1.0

plt.figure(figsize=(10, 8))
plt.title("Ratio of two fee income formulas")
plt.plot(fee2 / fee1)

fee_rate = np.nanmedian(fee2 / fee1)
plt.axhline(fee_rate, linestyle='--')

"""create plots"""
# 0 million
returns = simulate_returns(0.0)
means = returns.mean(axis=0)
stds = returns.std(axis=0)

plt.figure(figsize=(12, 8))
plt.title("TVL added = 0")
xs = pd.date_range(df.index[-1] + timedelta(days=1), periods=horizon)
plt.plot(xs, means)
plt.fill_between(xs, means - 1.0 * stds, means + 1.0 * stds, color='b', alpha=.1)

# 1 million
returns = simulate_returns(1e6)
means = returns.mean(axis=0)
stds = returns.std(axis=0)

plt.figure(figsize=(12, 8))
plt.title("TVL added = 1 million")
xs = pd.date_range(df.index[-1] + timedelta(days=1), periods=horizon)
plt.plot(xs, means)
plt.fill_between(xs, means - 1.0 * stds, means + 1.0 * stds, color='b', alpha=.1)

# 10 million
returns = simulate_returns(1e7)
means = returns.mean(axis=0)
stds = returns.std(axis=0)

plt.figure(figsize=(12, 8))
plt.title("TVL added = 10 million")
xs = pd.date_range(df.index[-1] + timedelta(days=1), periods=horizon)
plt.plot(xs, means)
plt.fill_between(xs, means - 1.0 * stds, means + 1.0 * stds, color='b', alpha=.1)
```

Out[10]:
```
<matplotlib.collections.PolyCollection at 0x1dc7d981850>
```









Back to Top