Decentralized exchanges are playing an increasingly important role in supporting cryptoasset trading, particularly when it comes to tokens using the ERC-20 standard. Coin Metrics has been actively working on collecting data from major DeFi protocols. For our first release, we have added support for all major liquidity pools on **Uniswap v2, Uniswap v3,** and **Sushiswap v1.**

Resources

This notebook demonstrates basic functionality offered by the Coin Metrics Python API Client and DEX Market Data.

Coin Metrics offers a vast assortment of data for hundreds of cryptoassets. The Python API Client allows for easy access to this data using Python without needing to create your own wrappers using `requests` and other such libraries.

To understand the data that Coin Metrics offers, feel free to peruse the resources below.

- The Coin Metrics API v4 website contains the full set of endpoints and data offered by Coin Metrics.
- The Coin Metrics Knowledge Base gives detailed, conceptual explanations of the data that Coin Metrics offers.
- The API Spec contains a full list of functions.

Setup

In [1]:
```
from os import environ
import sys
import pandas as pd
import numpy as np
import seaborn as sns
import logging
from datetime import date, datetime, timedelta
from coinmetrics.api_client import CoinMetricsClient
import json
import logging
from pytz import timezone as timezone_conv
from datetime import timezone as timezone_info
import matplotlib.dates as mdates
from IPython.display import Markdown as md
import matplotlib.pyplot as plt
import plotly.express as px
%matplotlib inline
```

In [2]:
```
sns.set_theme()
sns.set_style('whitegrid')
sns.set(rc={'figure.figsize':(16,8)})
sns.set_context("notebook", font_scale=1.2, rc={"font.family": "Lato"});
```

In [3]:
```
logging.basicConfig(
    format='%(asctime)s %(levelname)-8s %(message)s',
    level=logging.INFO,
    datefmt='%Y-%m-%d %H:%M:%S'
)
```

In [4]:
```
# We recommend privately storing your API key in your local environment.
try:
    api_key = environ["CM_API_KEY"]
    logging.info("Using API key found in environment")
except KeyError:
    api_key = ""
    logging.info("API key not found. Using community client")

client = CoinMetricsClient(api_key)
```

```
2023-06-09 08:55:34 INFO     Using API key found in environment
```

DEX Market Catalog

The *catalog/markets* endpoint returns a list of available markets along with time ranges of available data. Users can pass in a list of markets, exchanges, or market types (spot, futures, options).

We can retrieve our DEX markets by fetching a list of all 'spot' markets, then filtering for the markets where the 'experimental' parameter equals *true*.

In [5]:
```
spot_markets = client.catalog_markets(
    market_type = 'spot',
).to_dataframe()
```

In [6]:
```
exp_markets = spot_markets.loc[spot_markets['experimental']==True]
```

In [7]:
```
exp_markets
```

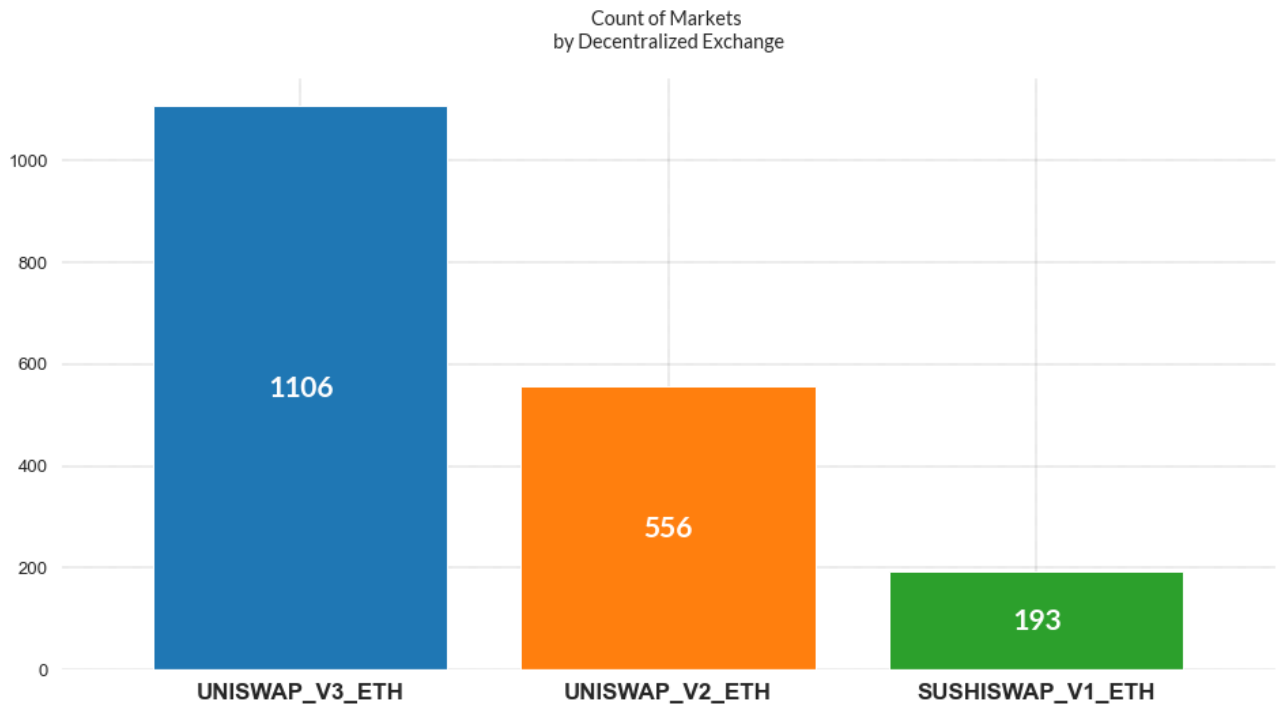
Out[7]:

	market	min_time	max_time	exchange	type	base	quote	symbol	orderbooks	quotes	...	order_price_min	or
25967	sushiswap_v1_eth-1inch-dai-spot	2020-12-25 01:08:37+00:00	2023-06-04 09:03:35+00:00	sushiswap_v1_eth	spot	1inch	dai	<NA>	NaN	NaN	...	<NA>	
25968	sushiswap_v1_eth-1inch-sushi-spot	2021-06-29 01:39:52+00:00	2023-06-02 02:03:59+00:00	sushiswap_v1_eth	spot	1inch	sushi	<NA>	NaN	NaN	...	<NA>	
25969	sushiswap_v1_eth-1inch-uni-spot	2021-07-05 04:09:59+00:00	2021-07-05 04:09:59+00:00	sushiswap_v1_eth	spot	1inch	uni	<NA>	NaN	NaN	...	<NA>	
25970	sushiswap_v1_eth-1inch-usdc-spot	2021-07-07 07:56:25+00:00	2023-05-31 12:21:23+00:00	sushiswap_v1_eth	spot	1inch	usdc	<NA>	NaN	NaN	...	<NA>	
25971	sushiswap_v1_eth-1inch-usdt_eth-spot	2022-11-19 04:56:35+00:00	2023-03-21 19:41:47+00:00	sushiswap_v1_eth	spot	1inch	usdt_eth	<NA>	NaN	NaN	...	<NA>	
...	...	...	...	...	...	...	...	...	...	...	...	...	
27861	uniswap_v3_eth-agg-yfi-cvx-spot	2023-05-31 00:00:00+00:00	2023-06-09 12:28:00+00:00	uniswap_v3_eth	spot	yfi	cvx	<NA>	NaN	NaN	...	<NA>	
27862	uniswap_v3_eth-agg-yfi-link-spot	2022-09-17 00:00:00+00:00	2023-06-09 12:28:00+00:00	uniswap_v3_eth	spot	yfi	link	<NA>	NaN	NaN	...	<NA>	
27863	uniswap_v3_eth-agg-yfi-usdc-spot	2022-07-25 00:00:00+00:00	2023-06-09 12:28:00+00:00	uniswap_v3_eth	spot	yfi	usdc	<NA>	NaN	NaN	...	<NA>	
27864	uniswap_v3_eth-agg-yfi-wbtc-spot	2021-05-11 00:00:00+00:00	2023-05-03 23:59:00+00:00	uniswap_v3_eth	spot	yfi	wbtc	<NA>	NaN	NaN	...	<NA>	
27865	uniswap_v3_eth-agg-yfi-weth-spot	2021-05-05 00:00:00+00:00	2023-06-09 12:28:00+00:00	uniswap_v3_eth	spot	yfi	weth	<NA>	NaN	NaN	...	<NA>	

1855 rows × 29 columns

In [8]:
```
# Calculate number of markets (pools) per DEX
mkt_counts = pd.DataFrame(exp_markets['exchange'].value_counts())
```

In [9]:
```
# Plot number of markets (pools) per DEX
ax1 = plt.subplot()
mkt_counts['exchange'].plot(kind='bar', width=0.8, color=sns.color_palette('tab10'))
plt.setp(ax1.get_xticklabels(), rotation=0)
ax1.set_facecolor("white")
plt.grid(color='black', linestyle='--', linewidth=0.2)
plt.title('\nCount of Markets \nby Decentralized Exchange\n', fontdict={'fontsize': 24, 'font': 'Lato'})
for i, v in enumerate(mkt_counts['exchange']):
    ax1.text(i, v/2, str(v), ha='center', va='center', color='white', font='Lato',fontweight='bold', fontsize=21)
ax1.set_xticklabels([label.get_text().upper() for label in ax1.get_xticklabels()], rotation=0, ha='center', va='top', fontsiz
plt.xticks(fontweight='bold')
plt.show()
```



In [10]:
```
# Show fields
pd.DataFrame(exp_markets.loc[exp_markets['exchange'] == 'uniswap_v3_eth'].iloc[0]).dropna()
```

Out[10]:

	26760
market	uniswap_v3_eth-1-1inch-dai-spot
min_time	2021-12-25 21:15:12+00:00
max_time	2021-12-30 00:28:29+00:00
exchange	uniswap_v3_eth
type	spot
base	1inch
quote	dai
contract_address	063332bbf9f8385e4106919b5c6ae2e6a4f72228
fee	0.01
base_address	111111111117dc0aa78b770fa6a738034120c302
quote_address	6b175474e89094c44da98b954eedeac495271d0f
experimental	True
pool_config_id	1
min_time_trades	2021-12-25T21:15:12.000000000Z
max_time_trades	2021-12-30T00:28:29.000000000Z

DEX markets include metadata for 3 different smart contracts:

- **contract_address:** The address of the liquidity pool contract. Each liquidity pool is a unique instance of a smart contract, deployed at a dedicated address. The pool contract holds both the base and the quote asset.
- **base_address:** The address of the ERC-20 token contract associated with the *base* asset.
- **quote_address:** The address of the ERC-20 token contract associated with the *quote* asset.

In [11]:
```
# DEX fields
exp_markets[['pool_config_id','contract_address','base_address','quote_address','fee']]
```

Out[11]:

	pool_config_id	contract_address	base_address	quote_
25967	<NA>	7d2103aab2612fdfb7fc0fcb3a879a36b7802ef9	111111111117dc0aa78b770fa6a738034120c302	6b175474e89094c44da98b954eedeac4
25968	<NA>	337b8d30da6f7dc740693d1d9e7a1fadca3be4ed	111111111117dc0aa78b770fa6a738034120c302	6b3595068778dd592e39a122f4f5a5cf
25969	<NA>	f7df6dbb318069c3c01c5b6ba83b1f39b3500b93	111111111117dc0aa78b770fa6a738034120c302	1f9840a85d5af5bf1d1762f925bdaddc
25970	<NA>	8793b190cfaab12c7fefc489b117efe116c0698f	111111111117dc0aa78b770fa6a738034120c302	a0b86991c6218b36c1d19d4a2e9eb0ce

	pool_config_id	contract_address	base_address	quote_
25971	<NA>	f3aef92b23746389078228709bd157a0933d8b06	111111111117dc0aa78b770fa6a738034120c302	dac17f958d2ee523a2206206994597c13
...	...	...	...	
27861	-1	<NA>	0bc529c00c6401aef6d220be8c6ea1667f6ad93e	4e3fbd56cd56c3e72c1403e103b45db9d
27862	-1	<NA>	0bc529c00c6401aef6d220be8c6ea1667f6ad93e	514910771af9ca656af840dff83e8264
27863	-1	<NA>	0bc529c00c6401aef6d220be8c6ea1667f6ad93e	a0b86991c6218b36c1d19d4a2e9eb0ce3
27864	-1	<NA>	0bc529c00c6401aef6d220be8c6ea1667f6ad93e	2260fac5e5542a773aa44fbcfedf7c193
27865	-1	<NA>	0bc529c00c6401aef6d220be8c6ea1667f6ad93e	c02aaa39b223fe8d0a0e5c4f27ead9083

1855 rows × 5 columns

Each liquidity pool is also associated with a corresponding fee percentage. With each trade, fees are distributed pro-rata to the pool's liquidity providers.

In [12]:
```
# Range of pool fees
pool_fees = pd.DataFrame(exp_markets.fee.unique().dropna())
pool_fees
```

Out[12]:

	0
0	0.3
1	0.01
2	0.05
3	1

Due to the permissionless nature of provisioning a liquidity pool, users can easily add arbitrary ERC-20 tokens to the decentralized exchange. There are over 50,000 trading pairs available on Uniswap today. To prioritize only the most relevant and liquid markets, we cover the subset of markets where both tokens are part of Coin Metrics reference rate coverage.

In [13]:
```
# Assets covered
asset_coverage = np.unique(exp_markets[['base', 'quote']].values)
print(asset_coverage)
print('\nTotal number of assets: \033[1m' + str(len(asset_coverage)) + '\033[0m\n')
```

```
['1inch' 'aave' 'ageur_eth' 'alcx' 'alpha' 'alusd' 'ampl' 'ant' 'ape'
 'api3' 'audio' 'axs_eth' 'badger' 'bal' 'band_eth' 'bat' 'bit' 'bnb_eth'
 'bnt' 'btm' 'busd' 'cbat' 'cbeth' 'ccomp' 'cdai' 'cel' 'cennz' 'ceth'
 'chz_eth' 'comp' 'cro' 'crv' 'cuni' 'cusdc' 'cusdcv3' 'cusdt' 'cvc' 'cvx'
 'cwbtc' 'czrx' 'dai' 'dar' 'dgx' 'dpi' 'drgn' 'elf' 'eng' 'enj' 'ens'
 'esd' 'ethos' 'euroc' 'eurs' 'eurt' 'fei_eth' 'fox' 'frax' 'ftm_eth'
 'ftt' 'fun' 'fwb' 'fxc' 'fxs' 'gala' 'gbpt' 'glm' 'gno' 'grt' 'gt_eth'
 'gusd' 'gyen' 'hbot' 'hbtc' 'hedg' 'ht' 'husd' 'idrt' 'imx' 'inst' 'kin1'
 'knc' 'ldo' 'lend' 'leo_eth' 'link' 'looks' 'loom' 'lpt' 'lusd' 'mana'
 'matic_eth' 'mco' 'mim' 'mkr' 'mpl' 'mtl_metal' 'myc' 'myth' 'nexo'
 'nftx' 'nmr' 'ogn' 'okb' 'omg' 'ousd' 'paid' 'pax' 'paxg' 'pay' 'perp'
 'poly' 'powr' 'ppt' 'qash' 'qnt' 'rad' 'radar' 'rai' 'ren' 'renbtc' 'rep'
 'rev_eth' 'rook' 'rsr' 'sai' 'salt' 'sand' 'seth' 'shib' 'skl' 'slp_eth'
 'snt' 'snx' 'spell' 'srm' 'srn' 'steth_lido' 'stg' 'stkaave' 'stmx'
 'storj' 'susd' 'sushi' 'swise' 'swrv' 't' 'taud_eth' 'tbtc' 'tcad_eth'
 'tgbp_eth' 'thkd_eth' 'toke' 'tusd' 'ubt' 'uma' 'uni' 'uqc' 'usdc' 'usdd'
 'usdd_eth' 'usdk' 'usdn_eth' 'usdt_eth' 'ust' 'veri' 'wbtc' 'wcelo'
 'weth' 'wluna' 'wnxm' 'wsteth' 'wust' 'wzec' 'xai' 'xaut' 'xchf'
 'xidr_eth' 'xsgd' 'xsushi' 'yfi' 'zrx']

Total number of assets: 181
```

The majority of DEX assets trade against **Wrapped Ether (WETH)**. Note that Ethereum's native asset ETH is not supported in Uniswap V2/V3, as the asset was created prior to the ERC-20 standard. From the Uniswap documentation:

> "Unlike Uniswap V1 pools, V2 pairs do not support ETH directly, so ETH⇄ERC-20 pairs must be emulated with WETH. The motivation behind this choice was to remove ETH-specific code in the core, resulting in a leaner codebase. End users can be kept fully ignorant of this implementation detail, however, by simply wrapping/unwrapping ETH in the periphery."
> docs.uniswap.org

In [14]:
```
# Enter an asset ticker to see available 'base' and 'quote' markets
asset = 'weth'

selected_markets = exp_markets.loc[(exp_markets['base']==asset) | (exp_markets['quote']==asset)]
print('\nTotal number of ' + '\033[1m' + asset + '\033[0m' + ' markets: \n\033[1m'
      + str(len(selected_markets)) + '\n')
```

```
Total number of weth markets:
635
```

Using the min_time parameter, we can filter for the newest DEX liquidity pool deployed in our coverage

In [15]:
```
# Check for the newest market
newest = selected_markets.loc[selected_markets['min_time'].idxmax()].dropna()
pd.DataFrame(newest)
```

Out[15]:

	26273
market	uniswap_v2_eth-cbeth-weth-spot
min_time	2023-06-08 06:20:23+00:00
max_time	2023-06-08 06:20:23+00:00
exchange	uniswap_v2_eth
type	spot
base	cbeth
quote	weth
base_address	be9895146f7af43049ca1c1ae358b0541ea49704
quote_address	c02aaa39b223fe8d0a0e5c4f27ead9083c756cc2
experimental	True
min_time_trades	2023-06-08T06:20:23.000000000Z
max_time_trades	2023-06-08T06:20:23.000000000Z

DEX Swaps Data

Swaps data is served through our existing */timeseries/market-trades* endpoint because swaps are conceptually identical to a trade. Users can see all the standard trade fields for a swap such as **time, price, and volume** but can also see defi-specific fields such as the **block height, block hash, transaction id, addresses involved** in the swap, and more.

In [16]:
```
defi_market = 'uniswap_v3_eth-2-weth-usdt_eth-spot'
start = datetime.now() - timedelta(days=3)
end = datetime.now() - timedelta(hours=1)
```

In [17]:
```
defi_trades = client.get_market_trades(
    markets=defi_market,
    start_time=start,
    end_time = end
).to_dataframe()
```

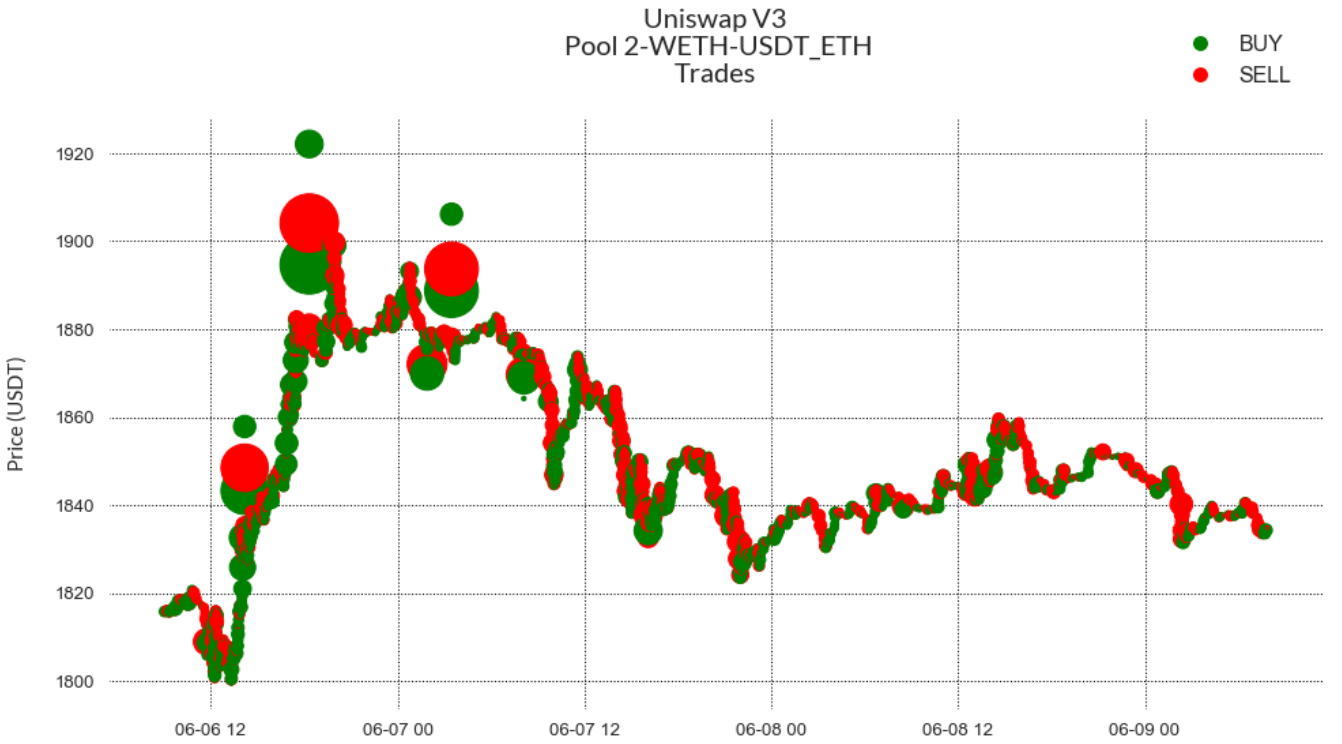
In [18]:
```
defi_trades['amount'] = defi_trades['amount'].astype(float)
defi_trades.head()
```

Out[18]:

	market	time	coin_metrics_id	amount	price	database_time	side	blo
0	uniswap_v3_eth-2-weth-usdt_eth-spot	2023-06-06 08:56:11+00:00	300	2.122417	1815.765701	2023-06-06 08:56:26.479854+00:00	sell	e6468d18fe303e6ff4a8bfec423572d42bf6c52b9
1	uniswap_v3_eth-2-weth-usdt_eth-spot	2023-06-06 08:56:23+00:00	84	17.991000	1815.581082	2023-06-06 08:56:41.807155+00:00	sell	d2add42f375b208a5efb9bd63603a9ab3ef69962f
2	uniswap_v3_eth-2-weth-usdt_eth-spot	2023-06-06 08:56:47+00:00	81	12.554369	1815.29867	2023-06-06 08:57:01.529091+00:00	sell	c91e104c0554fadedddb27d5ae5da2f4521467c96
3	uniswap_v3_eth-2-weth-usdt_eth-spot	2023-06-06 08:57:11+00:00	180	0.408836	1815.175386	2023-06-06 08:57:26.855435+00:00	sell	7ae82b77bd74563a0cb520328500df49bd983f178
4	uniswap_v3_eth-2-weth-usdt_eth-spot	2023-06-06 08:57:23+00:00	27	0.258121	1815.173951	2023-06-06 08:57:36.564560+00:00	buy	44c7a9da111cef9dbfad80776f0768541bf72aa0

In [19]:
```
color_map = {'buy': 'green', 'sell': 'red'}
defi_price = plt.scatter(x=defi_trades['time'], y=defi_trades['price'], s=defi_trades['amount'], c=defi_trades['side'].map(co

plt.xlabel("", fontsize=15)
plt.ylabel("Price (USDT)\n", font='Lato',fontsize=15)

market_string = defi_market.split("uniswap_v3_eth-", 1)[-1].split("-spot")[0].upper()
plt.title('\nUniswap V3\n Pool ' + market_string + '\nTrades\n', font='Lato', size=20)
```

```
legend_labels = ['BUY', 'SELL']
legend_handles = [plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=color_map['buy'], markersize=12),
                  plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=color_map['sell'], markersize=12)]
legend = plt.legend(legend_handles, legend_labels, loc='lower right', fontsize=16, framealpha=0, bbox_to_anchor=(0.99, 1.02))

plt.gca().set_facecolor('white')
plt.grid(color='black', linestyle='dotted')
plt.show()
```



In [20]:
```
pd.DataFrame(defi_trades.iloc[0])
```

Out[20]:

	0
market	uniswap_v3_eth-2-weth-usdt_eth-spot
time	2023-06-06 08:56:11+00:00
coin_metrics_id	300
amount	2.122417
price	1815.765701
database_time	2023-06-06 08:56:26.479854+00:00
side	sell
block_hash	e6468d18fe303e6ff4a8bfec423572d42bf6c52b9729c0...
block_height	17420367
txid	c48a1c99ffbe3c0d5b5ef61c2e23cc6b3bd5de5e0a3cb0...
initiator	63b554b1b802f46727898d0281988722df18fab3
sender	ef1c6e67703c7bd7107eed8303fbe6ec2554bf6b
beneficiary	63b554b1b802f46727898d0281988722df18fab3

Unlike centralized exchanges, where there is an unknown buyer and a seller, each swap is associated with 3 different Ethereum addresses:

- **Initiator** is the ethereum address which submitted the transaction as a result of which the swap/liquidity action occurred
- **Sender** is the ethereum address that invoked the uniswap pool smart contract's function for swapping or adding/removing liquidity
- **Beneficiary** is the ethereum address that got credited with the output tokens (in case of a swap or liquidity removal) or with liquidity (in case of liquidity addition)

In [21]:
```
print('\nTotal number of ' + '\033[1m' + defi_market + '\033[0m' + ' trades (' + str(start.date()) + ' to ' + str(end.date())
      + str(len(defi_trades)) + '\033[0m\n')

print('\nTotal number of ' + '\033[1m' + defi_market + '\033[0m' + ' buys (' + str(start.date()) + ' to ' + str(end.date())+
```

```
      + str(len(defi_trades.loc[(defi_trades['side']=='buy')])) + '\033[0m\n')

print('\nTotal number of ' + '\033[1m' + defi_market + '\033[0m' + ' unique buyers (' + str(start.date()) + ' to ' + str(end.
      + str(len((defi_trades.loc[(defi_trades['side']=='buy')]).beneficiary.unique())) + '\n')
```

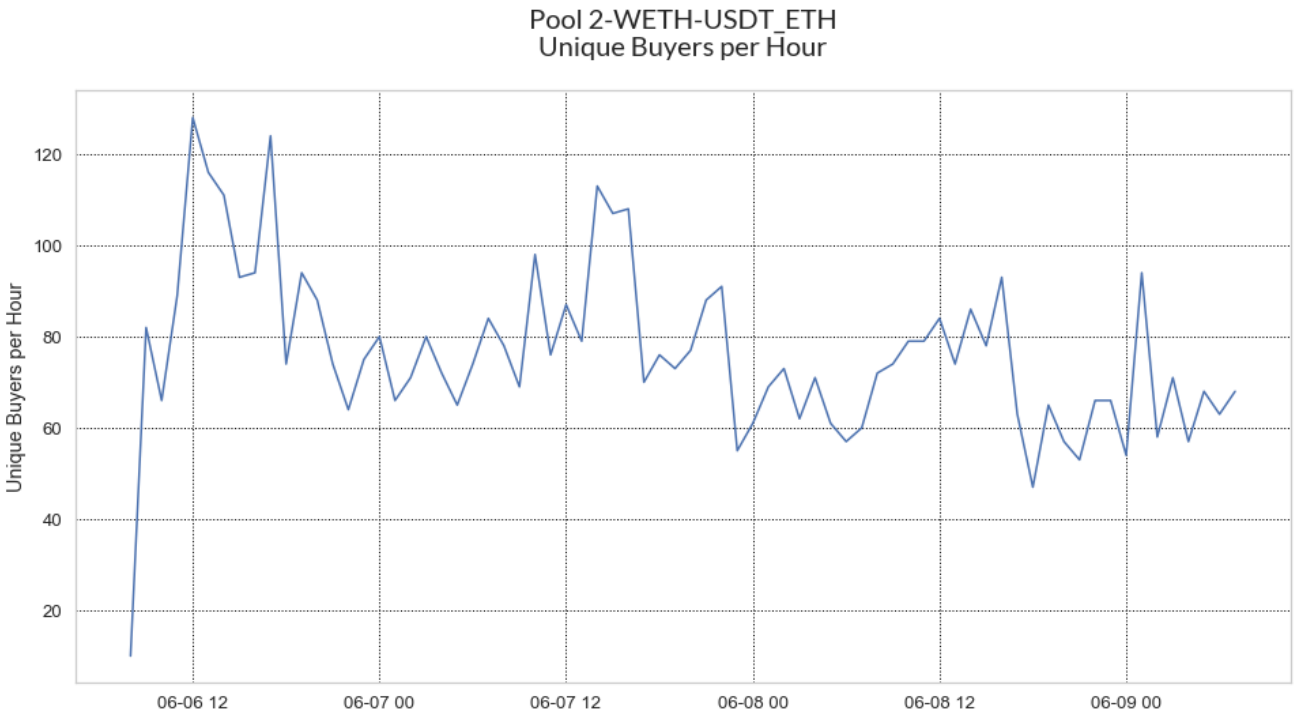
```
Total number of uniswap_v3_eth-2-weth-usdt_eth-spot trades (2023-06-06 to 2023-06-09):
12063


Total number of uniswap_v3_eth-2-weth-usdt_eth-spot buys (2023-06-06 to 2023-06-09):
6289


Total number of uniswap_v3_eth-2-weth-usdt_eth-spot unique buyers (2023-06-06 to 2023-06-09):
417
```

In [22]:

```
defi_buyers = pd.DataFrame(defi_trades)
defi_buyers['time'] = pd.to_datetime(defi_buyers['time'])
defi_buyers.set_index('time', inplace=True)
```

In [23]:

```
hourly_unique_buyers = defi_buyers['beneficiary'].resample('H').nunique()
sns.set_style('whitegrid')
fig, ax = plt.subplots()
sns.lineplot(data=hourly_unique_buyers, ax=ax)
ax.set_xlabel('')
ax.set_ylabel('Unique Buyers per Hour')
ax.set_title('\nPool ' + str(market_string) + '\nUnique Buyers per Hour\n', font='Lato', size=20)
plt.grid(color='black', linestyle='dotted')
plt.show()
```



In [24]:

```
# Unique buyer addresses
((defi_trades.loc[(defi_trades['side']=='buy')]).beneficiary.unique())
```

Out[24]:

```
<StringArray>
['493f461aead031cee2027f1b95370a692611acb9',
 '1111111254eeb25477b68fb85ed929f73a960582',
 'ef1c6e67703c7bd7107eed8303fbe6ec2554bf6b',
 '1136b25047e142fa3018184793aec68fbb173ce4',
 '3b3ae790df4f312e745d270119c6052904fb6790',
 'e592427a0aece92de3edee1f18e0157c05861564',
 'cc0fcd959871e69f84fe9464c629f327941b655b',
 '7054b0f980a7eb5b3a6b3446f3c947d80162775c',
 'a69babef1ca67a37ffaf7a485dfff3382056e78c',
 'f6c3595cbd6858b47e93c83312cef89750cea3a4',
 ...
 'af997affb94c5ca556b28b024e162aa3164f4a43',
 '75f4fa23c6a2727ba507362e1f52946c810073c0',
 '196d674ab02e4813c06440b77362432cb1f7b7e2',
 '7c2543938a27554f82be178a2ac02a72c8ad1925',
 '30cb2c51fc4f031fa5f326d334e1f5da00e19ab5',
```

```
 '540b9658044acb24a7f32341c6564818176b0f1e',
 '80a0102a1e601c55fd3f136128bb2d222a879ff3',
 '30e08ef9a9625a6d3dc25f7f0bf4da2d80b5db60',
 '5003fdfc747a0acf06266c54fe7285abbca14f25',
 'c09bf2b1bc8725903c509e8caeef9190857215a8']
Length: 417, dtype: string
```

In [25]:

```
# Calculate approximate USD volume (NOTE: quote asset must be a stablecoin)
defi_trades['DEX Volume (USD)'] = (defi_trades['amount'])*(defi_trades['price'])

# Use only the amount field if base asset is a stablecoin
#defi_trades['DeFi Volume (USD)'] = (defi_trades['amount'])
```

In [26]:

```
# Largest trade
largest = defi_trades.loc[defi_trades['amount'].idxmax()]
trade_size_usd = '${:,.2f}'.format(largest['DEX Volume (USD)'])
pd.DataFrame(largest)
```

Out[26]:

	2130
market	uniswap_v3_eth-2-weth-usdt_eth-spot
time	2023-06-06 18:19:23+00:00
coin_metrics_id	19
amount	1841.707537
price	1894.630054
database_time	2023-06-06 18:19:35.333509+00:00
side	buy
block_hash	f67317f16b1ceb639af6e07cb373b738f2685ed6c01157...
block_height	17423134
txid	ee5c21048778055c1faf0dfa8d4be37ccc7dc8225c614d...
initiator	654fae4aa229d104cabead47e56703f58b174be4
sender	00000000032962b51589768828ad878876299e14
beneficiary	00000000032962b51589768828ad878876299e14
DEX Volume (USD)	3489354.449985

In [27]:

```
print('\nLARGEST TRADE: \n\n' + trade_size_usd + ' by address 0x' + largest['beneficiary'] + '\n')
```

```
LARGEST TRADE:

$3,489,354.45 by address 0x00000000032962b51589768828ad878876299e14
```

In [28]:

```
# ATLAS links for blockchain metadata
md('<br>**ATLAS** by Coin Metrics <br> **Blockchain Search Engine** <br><br>**BUYER ADDRESS:** https://atlas.coinmetrics.io/a
```

Out[28]:

ATLAS by Coin Metrics
Blockchain Search Engine

BUYER ADDRESS: https://atlas.coinmetrics.io/address-details?asset=weth&address=00000000032962b51589768828ad878876299e14

TRANSACTION INFO: https://atlas.coinmetrics.io/transaction-details?asset=weth&tx_hash=ee5c21048778055c1faf0dfa8d4be37ccc7dc8225c614d47e6dd87558e0b41bc

BlOCK INFO: https://atlas.coinmetrics.io/block-details?asset=weth&block_hash=f67317f16b1ceb639af6e07cb373b738f2685ed6c01157bc44bc0863c2d4cdf3

For more info on **ATLAS** visit: https://coinmetrics.io/atlas/

In [29]:

```
defi_trades['Datetime'] = pd.to_datetime(defi_trades.time.dt.strftime('%m/%d/%y %H:00'))
defi_vol = defi_trades.groupby('Datetime')['DEX Volume (USD)'].sum()
```

In [30]:

```
pd.DataFrame(defi_vol)
```

Out[30]:

Datetime	DEX Volume (USD)
2023-06-06 08:00:00	65201.716977
2023-06-06 09:00:00	2084367.429207
2023-06-06 10:00:00	1672934.234875
2023-06-06 11:00:00	3281393.623737
2023-06-06 12:00:00	9065312.89202
...	...
2023-06-09 03:00:00	1098728.127931
2023-06-09 04:00:00	1154651.21676
2023-06-09 05:00:00	594494.732207
2023-06-09 06:00:00	1142769.345904
2023-06-09 07:00:00	1417499.881004

72 rows × 1 columns

Retrieving volume for a centralized exchange via the market-candles endpoint

In [31]:
```
cex_market = 'coinbase-eth-usd-spot'
```

In [32]:
```
cex_vol = client.get_market_candles(
    markets=cex_market,
    frequency='1h',
    start_time=start,
    end_time=end- timedelta(hours=1)
).to_dataframe()

cex_vol = cex_vol.rename(columns={"candle_usd_volume": "CEX Volume (USD)"})
cex_vol['Datetime'] = pd.to_datetime(cex_vol.time.dt.strftime('%m/%d/%y %H:00'))
cex_vol = cex_vol.groupby('Datetime')['CEX Volume (USD)'].sum()
```

In [33]:
```
vol_comp = pd.merge(defi_vol, cex_vol,on="Datetime", how="left").dropna()
vol_comp["DEX Volume (USD)"] = vol_comp['DEX Volume (USD)'].astype(int)
vol_comp["CEX Volume (USD)"] = vol_comp['CEX Volume (USD)'].astype(int)
```

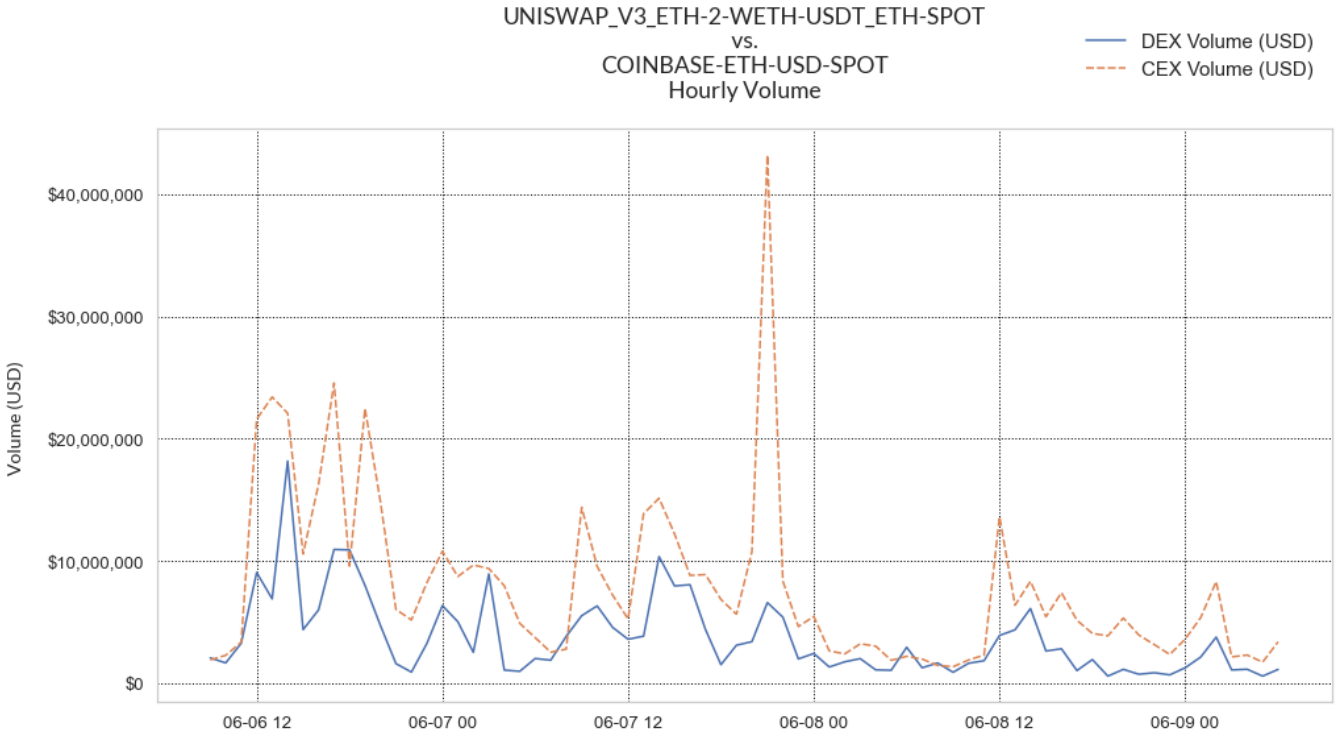
In [34]:
```
vol_comp
```

Out[34]:

Datetime	DEX Volume (USD)	CEX Volume (USD)
2023-06-06 09:00:00	2084367	1914510
2023-06-06 10:00:00	1672934	2297436
2023-06-06 11:00:00	3281393	3367881
2023-06-06 12:00:00	9065312	21614036
2023-06-06 13:00:00	6912261	23423713
...	...	...
2023-06-09 02:00:00	3783448	8330682
2023-06-09 03:00:00	1098728	2167268
2023-06-09 04:00:00	1154651	2319560
2023-06-09 05:00:00	594494	1737127
2023-06-09 06:00:00	1142769	3399740

70 rows × 2 columns

In [35]:
```
sns.set_style('whitegrid')
vc = sns.lineplot(data=vol_comp)
vc.yaxis.set_major_formatter('${x:,.0f}')
vc.set_title(str('\n' + defi_market.upper() + '\n vs. \n' + cex_market.upper() + '\n Hourly Volume \n'),font='Lato',fontsize=
vc.set_ylabel("Volume (USD) \n", font='Lato',fontsize = 14)
plt.grid(color='black', linestyle='dotted')
legend = vc.legend(fontsize=15, framealpha=0, bbox_to_anchor=(1.0, 1.2))
vc.set_xlabel("");
```


UNISWAP_V3_ETH-2-WETH-USDT_ETH-SPOT
vs.
COINBASE-ETH-USD-SPOT
Hourly Volume
DEX Volume (USD)
CEX Volume (USD)
Volume (USD)
$40,000,000
$30,000,000
$20,000,000
$10,000,000
$0
06-06 12
06-07 00
06-07 12
06-08 00
06-08 12
06-09 00