Resources

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

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.

Notebook Setup

In [1]:
```
from os import environ
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 logging
import matplotlib.pyplot as plt
import warnings
%matplotlib inline
```

In [2]:
```
# Chart themes
sns.set_theme()
warnings.filterwarnings('ignore')
fig = plt.style.use('seaborn')
sns.set(rc={'figure.figsize':(14,9)})
sns.set_style("whitegrid",{'axes.grid' : True,'grid.linestyle': '--', 'grid.color': 'gray','axes.edgecolor': 'white','font.family':
```

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)
```

```
2023-09-19 10:42:11 INFO     Using API key found in environment
```

Calculate Estimated Validator Yield

Using ETH_CL validator metrics it is possible to estimate the yield from the protocol. In combination with historical data on priority tips, we can estimate what a validator should expect to earn. Note that maximal extractable value (MEV) is another source of revenue for validators but is currently not considered as part of this analysis.

A validator's expected annual percentage return (APR) from staking rewards accumulated on the Consensus Layer, assuming perfect performance and uptime, can be estimated with the formula below (where ValidatorActOngCnt = number of active validators):

> 2940.21 ÷ sqrt(ValidatorActOngCnt) = Staking Yield

In [5]:
```
start = '2022-01-01'
```

Retrieve Consensus Layer Metrics

In [6]:
```
consensus_metrics = client.get_asset_metrics(
    assets='eth_cl',
    metrics=['ValidatorActOngCnt'],
    start_time = start,
).to_dataframe()
```

In [7]:
```
consensus_metrics['time'] = pd.to_datetime(consensus_metrics['time'])
```

In [8]:
```
consensus_metrics
```

Out[8]:

	asset	time	ValidatorActOngCnt
0	eth_cl	2022-01-01 00:00:00+00:00	275880
1	eth_cl	2022-01-02 00:00:00+00:00	276301
2	eth_cl	2022-01-03 00:00:00+00:00	276784
3	eth_cl	2022-01-04 00:00:00+00:00	277530
4	eth_cl	2022-01-05 00:00:00+00:00	278349
...	...	...	...
621	eth_cl	2023-09-14 00:00:00+00:00	795827
622	eth_cl	2023-09-15 00:00:00+00:00	798143
623	eth_cl	2023-09-16 00:00:00+00:00	800752
624	eth_cl	2023-09-17 00:00:00+00:00	803286
625	eth_cl	2023-09-18 00:00:00+00:00	805657

626 rows × 3 columns

Rewards from staking are only one part of a validator's yield. Post-Merge, validators now also receive user transaction priority fees, or tips, that used to go to miners on the Execution Layer. Considering the historical record of fees, we can estimate the magnitude of this additional source of yield. For our analysis, we show how to estimate both staking revenues and priority tips as yields on staked ETH.

To do this, we use the results we found above and divide gross annual emission by the total number of validators to produce average validator revenue, which for this purpose only consider revenues that originate from the protocol and not from fees.

> 940.87 × sqrt(ValidatorActOngCnt) ÷ ValidatorActOngCnt = Avg. Validator Revenue

The expected annual number of blocks proposed in turn allows us to estimate the priority tip that is earned by each block proposal. Using a 14-day moving average to smooth priority tips, we then estimate what a proposer should expect to earn in tips.

> 1 ÷ ValidatorActOngCnt × 2,629,800 = Annual Num. of Proposals per Validator
>
> sma(FeePrioTotNtv ÷ BlkCnt, 14) × Ann. Num. of Proposals per Validator = Average Priority Tip per Block

Retrieve Execution Layer Metrics

In [9]:
```
execution_metrics = client.get_asset_metrics(
    assets='eth',
    metrics=['FeePrioTotNtv', 'BlkCnt'],
    start_time = start,
).to_dataframe()
```

In [10]:
```
execution_metrics['time'] = pd.to_datetime(execution_metrics['time'])
```

In [11]:
```
execution_metrics
```

Out[11]:

	asset	time	BlkCnt	FeePrioTotNtv
0	eth	2022-01-01 00:00:00+00:00	6506	747.204977
1	eth	2022-01-02 00:00:00+00:00	6495	905.938725
2	eth	2022-01-03 00:00:00+00:00	6461	898.990928
3	eth	2022-01-04 00:00:00+00:00	6494	1245.74813
4	eth	2022-01-05 00:00:00+00:00	6460	1485.351677
...	...	...	...	...

	asset	time	BlkCnt	FeePrioTotNtv
621	eth	2023-09-14 00:00:00+00:00	7120	512.693607
622	eth	2023-09-15 00:00:00+00:00	7091	493.56702
623	eth	2023-09-16 00:00:00+00:00	7080	387.820376
624	eth	2023-09-17 00:00:00+00:00	7043	323.388901
625	eth	2023-09-18 00:00:00+00:00	7112	509.694197

626 rows × 4 columns

In [12]:
```
eth_metrics = consensus_metrics.merge(execution_metrics, on='time', how='inner')
```

In [13]:
```
eth_metrics = eth_metrics[['time','ValidatorActOngCnt','BlkCnt','FeePrioTotNtv']]
```

Calculate theoretical validator yield based on Active Validator count

In [14]:
```
eth_metrics['Validator Yield'] = 100 * (
    (32 + ((940.87 * (eth_metrics['ValidatorActOngCnt'] ** (1/2))) / eth_metrics['ValidatorActOngCnt']))/32 - 1
)
eth_metrics
```

Out[14]:

	time	ValidatorActOngCnt	BlkCnt	FeePrioTotNtv	Validator Yield
0	2022-01-01 00:00:00+00:00	275880	6506	747.204977	5.597828
1	2022-01-02 00:00:00+00:00	276301	6495	905.938725	5.593561
2	2022-01-03 00:00:00+00:00	276784	6461	898.990928	5.588679
3	2022-01-04 00:00:00+00:00	277530	6494	1245.74813	5.581163
4	2022-01-05 00:00:00+00:00	278349	6460	1485.351677	5.572946
...	...	...	...	...	...
621	2023-09-14 00:00:00+00:00	795827	7120	512.693607	3.295872
622	2023-09-15 00:00:00+00:00	798143	7091	493.56702	3.291086
623	2023-09-16 00:00:00+00:00	800752	7080	387.820376	3.285721
624	2023-09-17 00:00:00+00:00	803286	7043	323.388901	3.280534
625	2023-09-18 00:00:00+00:00	805657	7112	509.694197	3.275703

626 rows × 5 columns

Calculate estimated blocks proposals per year based on Active Validator count

In [15]:
```
eth_metrics['est_block_proposals_per_yr'] = ((1/eth_metrics['ValidatorActOngCnt']) * (2629800))
```

Estimate tips per block

In [16]:
```
eth_metrics['avg_per_block_tip_2w'] = (eth_metrics['FeePrioTotNtv'] / eth_metrics['BlkCnt']).rolling(window=14).mean()
```

In [17]:
```
eth_metrics
```

Out[17]:

	time	ValidatorActOngCnt	BlkCnt	FeePrioTotNtv	Validator Yield	est_block_proposals_per_yr	avg_per_block_tip_2w
0	2022-01-01 00:00:00+00:00	275880	6506	747.204977	5.597828	9.532405	NaN
1	2022-01-02 00:00:00+00:00	276301	6495	905.938725	5.593561	9.517881	NaN
2	2022-01-03 00:00:00+00:00	276784	6461	898.990928	5.588679	9.501272	NaN
3	2022-01-04 00:00:00+00:00	277530	6494	1245.74813	5.581163	9.475732	NaN
4	2022-01-05 00:00:00+00:00	278349	6460	1485.351677	5.572946	9.447851	NaN
...	...	...	...	...	...	...	...
621	2023-09-14 00:00:00+00:00	795827	7120	512.693607	3.295872	3.304487	0.064886
622	2023-09-15 00:00:00+00:00	798143	7091	493.56702	3.291086	3.294898	0.065689
623	2023-09-16 00:00:00+00:00	800752	7080	387.820376	3.285721	3.284163	0.066248
624	2023-09-17 00:00:00+00:00	803286	7043	323.388901	3.280534	3.273803	0.065978
625	2023-09-18 00:00:00+00:00	805657	7112	509.694197	3.275703	3.264168	0.066820

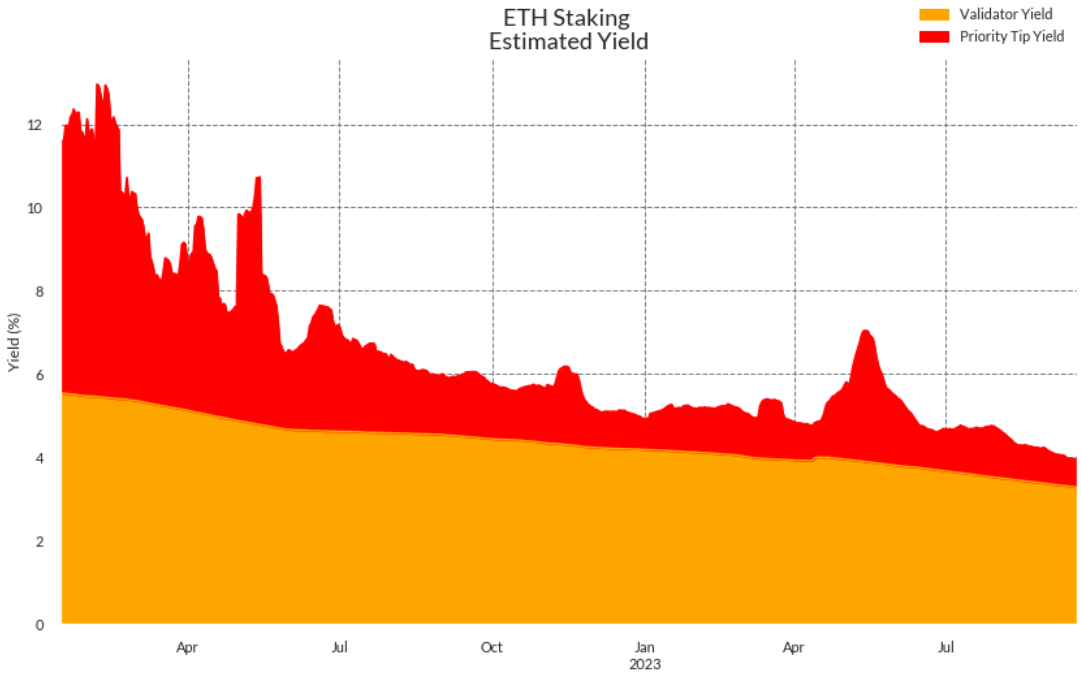
626 rows × 7 columns

Calculate priority tip yield

In [18]:
```
eth_metrics['Priority Tip Yield'] = (100 *
    ((32 + eth_metrics['avg_per_block_tip_2w'] * eth_metrics['est_block_proposals_per_yr'])/32+(-1)))
```

In [19]:
```
eth_metrics = eth_metrics.dropna().set_index('time')
```

In [35]:
```
ax = eth_metrics[['Validator Yield', 'Priority Tip Yield']].plot.area(stacked=True, figsize=(14, 8),color=['orange', 'red'])
ax.set_ylabel('Yield (%)')
ax.set_xlabel('')
ax.set_title('\nETH Staking \nEstimated Yield',fontsize=18)
plt.legend(loc='upper right', bbox_to_anchor=(1, 1.11))
plt.show()
```



Calculate ETH Supply: Staked vs. Unstaked

One of the many advantages of a blockchain-based ledger is auditability, but increasingly complex consensus architectures and supply mechanics can make it difficult to understand the full picture of asset supply. Ethereum's shift to proof-of-stake introduced a number of novel considerations in obtaining network-wide supply figures. In the following example, we combine various Supply metrics from ETH's Consensus and Execution Layers to ascertain the total amount of staked vs. unstaked supply.

In [21]:
```
start = '2022-01-01'
```

Consensus Layer Metrics

In [22]:
```
cl_supply = client.get_asset_metrics(
    assets='eth_cl',
    metrics=['SplyCur','SplyStkedNtv'],
    start_time = start,
    frequency = '1d'
).to_dataframe()

cl_supply = cl_supply.rename(columns={"SplyCur": "SplyCur_CL"})
```

Execution Layer Metrics

In [23]:
```
el_supply = client.get_asset_metrics(
    assets='eth',
    metrics=['SplyCur','SplyCLCont'],
    start_time = start,
    frequency = '1d'
).to_dataframe()

el_supply = el_supply.rename(columns={"SplyCur": "SplyCur_EL"})
```

In [24]:
```
adjusted_supply = cl_supply.merge(el_supply, on='time', how='inner')
adjusted_supply = adjusted_supply.set_index('time')
```

In [25]:
```
adjusted_supply = adjusted_supply[['SplyCur_EL','SplyCur_CL','SplyCLCont','SplyStkedNtv']]
```

Calculate the total 'adjusted' ETH supply

In [26]:
```
adjusted_supply['Total ETH Supply'] = adjusted_supply['SplyCur_EL'] + (adjusted_supply['SplyCur_CL'] - adjusted_supply['SplyCLCont'
```

In [27]:
```
adjusted_supply['Staked Supply'] = adjusted_supply['SplyStkedNtv']
```

In [28]:
```
adjusted_supply['Unstaked Supply'] = adjusted_supply['Total ETH Supply'] - adjusted_supply['Staked Supply']
```

In [29]:
```
adjusted_supply
```

Out[29]:

time	SplyCur_EL	SplyCur_CL	SplyCLCont	SplyStkedNtv	Total ETH Supply	Staked Supply	Unstaked Supply
2022-01-01 00:00:00+00:00	117660990.070996	9227820.763941	8852770.0	8828515	118036040.834937	8828515	109207525.834937
2022-01-02 00:00:00+00:00	117665304.120069	9242509.773522	8866898.0	8841891	118040915.893592	8841891	109199024.893592
2022-01-03 00:00:00+00:00	117669294.819239	9264680.81281	8887122.0	8862723	118046853.632049	8862723	109184130.632049
2022-01-04 00:00:00+00:00	117670945.089902	9286929.253893	8915506.0	8883635	118042368.343795	8883635	109158733.343795
2022-01-05 00:00:00+00:00	117671281.516053	9311811.449065	8931074.0	8907187	118052018.965118	8907187	109144831.965118
...	...	...	...	...	...	...	...
2023-09-14 00:00:00+00:00	123201307.973034	26800663.648109	29803805.264931	26645513	120198166.356212	26645513	93552653.356212
2023-09-15 00:00:00+00:00	123204942.109666	26850644.963176	29859457.264931	26686373	120196129.807912	26686373	93509756.807912
2023-09-16 00:00:00+00:00	123214608.240562	26869980.147218	29871526.264931	26711305	120213062.122849	26711305	93501757.122849
2023-09-17 00:00:00+00:00	123247156.871889	26853036.263327	29892503.264931	26720613	120207689.870285	26720613	93487076.870285
2023-09-18 00:00:00+00:00	123252665.665134	26882411.951288	29953894.264931	26744301	120181183.351491	26744301	93436882.351491

626 rows × 7 columns

Plot staked vs. unstaked supply

In [38]:
```
ax = adjusted_supply[['Staked Supply', 'Unstaked Supply']].plot.area(stacked=True, figsize=(14, 8),color=['pink', 'purple'])
ax.set_ylabel('')
ax.set_xlabel('')
ax.set_title('\nETH Staked vs. Unstaked Supply\n',fontsize=18)
ax.yaxis.set_major_formatter(lambda x, _: f'{x*1e-6}M')
plt.legend(loc='upper right', bbox_to_anchor=(1, 1.11))
plt.show()
```