Though decentralized finance has garnered significant attention in recent years, centralized exchanges remain a crucial cornerstone of the crypto asset industry. These trading venues serve as some of the deepest pools of liquidity for the top crypto assets, and facilitate the onboarding and offboarding of millions of users by sitting at the intersection of digital assets and fiat currencies. Coin Metrics collects a wide variety of data from a universe of 39+ top crypto exchanges, allowing our users to easily assess exchange dominance, asset trading volumes, and even the composition of trading patterns in specific markets.

Resources

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

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 (https://docs.coinmetrics.io/api/v4) website contains the full set of endpoints and data offered by Coin Metrics.
- The Coin Metrics Knowledge Base (https://docs.coinmetrics.io/info) gives detailed, conceptual explanations of the data that Coin Metrics offers.
- The API Spec (https://coinmetrics.github.io/api-client-python/site/api_client.html) contains a full list of functions.

Notebook Setup

In [1]:

```
from os import environ
import sys
import pandas as pd
import numpy as np
import logging
from datetime import date, datetime, timedelta
from coinmetrics.api_client import CoinMetricsClient
import json
import logging
import matplotlib.pyplot as plt
import plotly.express as px
%matplotlib inline
```

In [2]:

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

In [3]:

```
# 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)
```

```
2023-02-10 15:28:53 INFO     Using API key found in environment
```

Daily Exchange Volumes

Coin Metrics creates aggregated metrics to allow users to easily compare the total trading volume of dominant centralized exchanges. By leveraging the *timeseries/exchange-metrics* endpoint, we can retrieve the daily reported spot trading volume for a single trading venue or a list of exchanges.

In [4]:

```
end = pd.to_datetime(datetime.now())
start = end - timedelta(days=30)
```

In [5]:

```
exchange_list = ['binance','binance.us','coinbase','bitbank',
                 'bitfinex','bitflyer','bitstamp','bittrex',
                 'cex.io','coinbase','gate.io','gemini',
                 'itbit','kraken','liquid','poloniex',
                  'therocktrading','upbit']
```

In [6]:

```
volumes = client.get_exchange_metrics(
    exchanges=exchange_list,
    metrics='volume_reported_spot_usd_1d',
    frequency='1d',
    start_time=start
).to_dataframe()
```

In [7]:

```
pivot_volumes = volumes.pivot(index="time",
                columns="exchange",
                values="volume_reported_spot_usd_1d")
```

In [8]:

```
pivot_volumes.head()
```

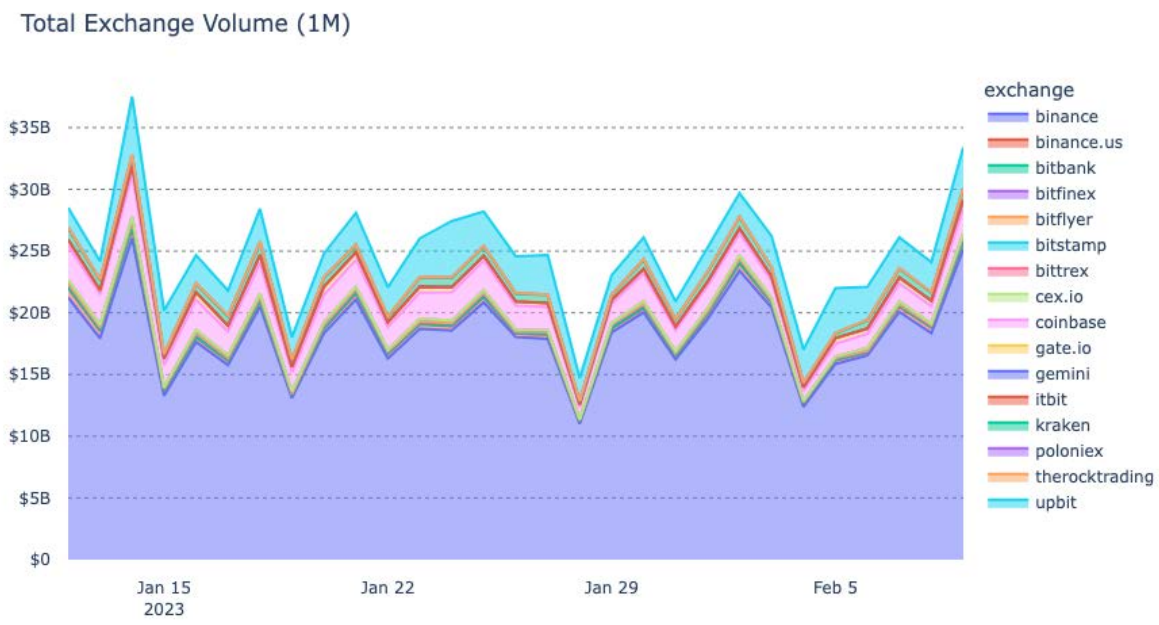
Out[8]:

exchange time	binance	binance.us	bitbank	bitfinex	bitflyer	bitstamp	bittrex	cex.io
2023-01-12 00:00:00+00:00	21236534489.906601	814679379.415409	28718915.645166	265889217.089283	61316766.427784	239870503.421748	12514693.358403	5317702.76128
2023-01-13 00:00:00+00:00	17926255822.7864	629735775.846911	21777771.637721	174593538.61614	57998248.02181	198808073.798907	12887766.155145	3139208.58750
2023-01-14 00:00:00+00:00	26056190600.464901	875309485.284003	43801905.875432	410945764.106318	100227748.08508	272571865.191488	14070712.307648	8042452.53231
2023-01-15 00:00:00+00:00	13277432837.0872	402508709.220971	19084326.541365	136335609.324348	39460654.374623	74550442.642391	10605110.233031	3351120.15825
2023-01-16 00:00:00+00:00	17645051439.9072	387287105.192892	29936816.06588	222228880.883837	81149891.66134	234910312.366957	11725220.983083	3822742.11698

In [9]:

```
fig = px.area(
        pivot_volumes,
        width=900,
        height=500)
fig.update_yaxes(title="",matches=None, showticklabels=True, visible=True,
                 showgrid=True,gridcolor='gray', griddash='dot')
fig.update_layout(plot_bgcolor="white", margin=dict(pad=12), title='Total Exchange Volume (1M)'
)
fig.for_each_yaxis(lambda yaxis: yaxis.update(tickprefix="$"))

fig.update_xaxes(title="")
fig.show()
```



In [10]:

```
for i in range(pivot_volumes.shape[0]):
    row_sum = pivot_volumes.iloc[i, 0:].sum()
    pivot_volumes.iloc[i, 0:] = pivot_volumes.iloc[i, 0:].div(row_sum)*100
```

In [29]:

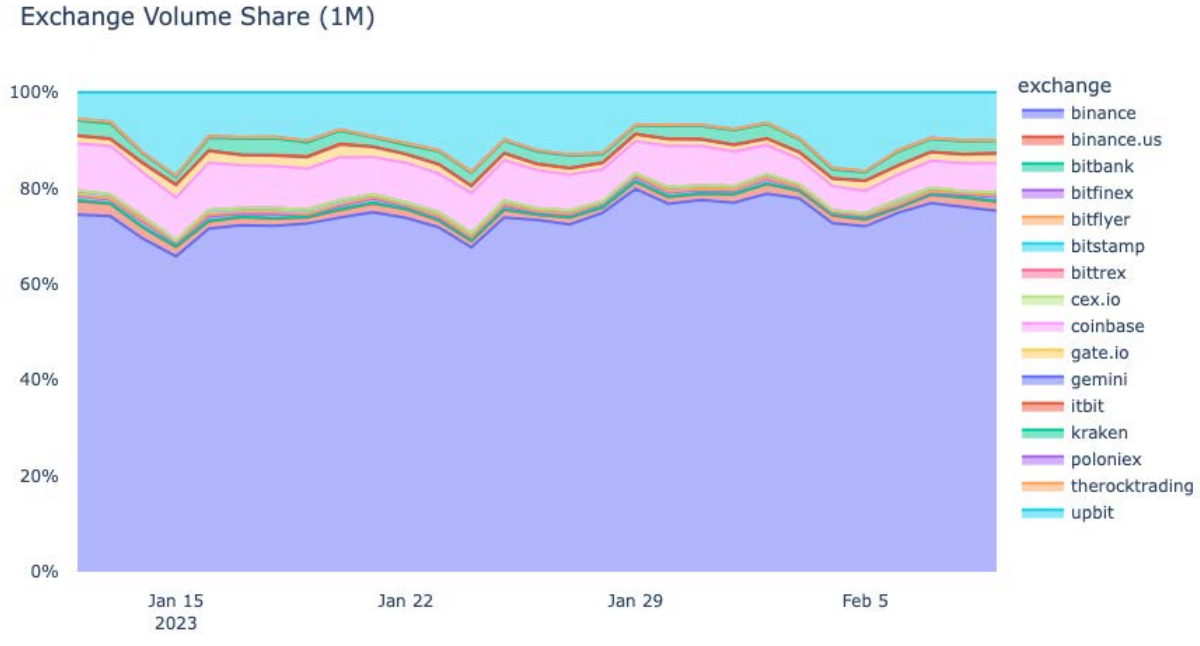
```
pivot_volumes.tail()
```

Out[29]:

exchange	binance	binance.us	bitbank	bitfinex	bitflyer	bitstamp	bittrex	cex.io	coinbase	gate.io	gemini	itbit	kraken	poloniex
time														
2023-02-05 00:00:00+00:00	72.060764	1.440396	0.111467	0.52245	0.187981	0.378153	0.031303	0.009075	4.738541	2.018461	0.069263	0.031456	1.721333	0.222224
2023-02-06 00:00:00+00:00	74.883121	1.015791	0.097187	0.40428	0.225542	0.927406	0.048592	0.010182	5.229216	1.795256	0.106492	0.031361	2.870897	0.231423
2023-02-07 00:00:00+00:00	76.912458	1.74829	0.082392	0.397121	0.250853	0.622946	0.075718	0.011557	5.64972	1.725948	0.092661	0.031279	2.673934	0.189196
2023-02-08 00:00:00+00:00	76.158592	1.915785	0.100864	0.485004	0.218992	0.467932	0.049804	0.016232	5.818571	1.72839	0.09752	0.04028	2.644992	0.210854
2023-02-09 00:00:00+00:00	75.289368	1.892096	0.139691	0.890756	0.327031	0.554054	0.051811	0.014551	6.073632	1.855294	0.217574	0.052564	2.558216	0.17407

In [12]:

```
fig = px.area(
        pivot_volumes,
        width=900,
        height=500)
fig.update_layout(
    paper_bgcolor="white",
    title='Exchange Volume Share (1M)',
    plot_bgcolor = 'white',
    margin=dict(pad=12)
)
fig.update_yaxes(title="",matches=None, showticklabels=True, visible=True,showgrid=True,ticksuffix='%')
fig.update_xaxes(title="")
fig.show()
```



Asset Share

In addition to retrieving aggregated trading volumes for all markets on an exchange, we can also utilize the *timeseries/market-candles* to obtain the USD trading volume of individual markets on a particular exchange.

In [13]:

```
# The wildcard parameter (*) enables users to retrieve all markets matching the designated format
exchange = 'coinbase'
markets=f"{exchange}-*-spot"
```

In [14]:

```
df = client.get_market_candles(
    markets=markets,
    start_time=start,
    frequency='1d'
).to_dataframe()
```

In [15]:

```
df
```

Out[15]:

	market	time	price_open	price_close	price_high	price_low	vwap	volume	candle_usd_volume	candle_trades_count
0	coinbase-1inch-btc-spot	2023-01-12 00:00:00+00:00	0.000024	0.000024	0.000025	0.000024	0.000024	15234.67	6661.80338	181
1	coinbase-1inch-btc-spot	2023-01-13 00:00:00+00:00	0.000024	0.000023	0.000024	0.000023	0.000024	7704.66	3511.925201	73
2	coinbase-1inch-btc-spot	2023-01-14 00:00:00+00:00	0.000023	0.000023	0.000024	0.000022	0.000023	48327.41	23082.759963	249
3	coinbase-1inch-btc-spot	2023-01-15 00:00:00+00:00	0.000023	0.000024	0.000024	0.000023	0.000024	16343.14	8043.252944	169
4	coinbase-1inch-btc-spot	2023-01-16 00:00:00+00:00	0.000024	0.000023	0.000024	0.000023	0.000023	19790.49	9782.750546	156
...	...	...	...	...	...	...	...	...	...	...
15885	coinbase-zrx-usd-spot	2023-02-05 00:00:00+00:00	0.259239	0.249039	0.263523	0.241888	0.251682	4844581.46949	1219292.214365	5444
15886	coinbase-zrx-usd-spot	2023-02-06 00:00:00+00:00	0.249038	0.245029	0.257681	0.240951	0.249274	3621276.49094	902691.314734	4766
15887	coinbase-zrx-usd-spot	2023-02-07 00:00:00+00:00	0.245077	0.261059	0.262272	0.243382	0.253134	4746441.56465	1201487.575623	5910
15888	coinbase-zrx-usd-spot	2023-02-08 00:00:00+00:00	0.261031	0.255517	0.261493	0.243463	0.25628	5057372.71359	1296105.834138	5978
15889	coinbase-zrx-usd-spot	2023-02-09 00:00:00+00:00	0.255826	0.229094	0.259653	0.221806	0.245045	7288318.33875	1785969.519361	7353

15890 rows × 10 columns

In [16]:

```
df["candle_usd_volume"] = df.candle_usd_volume.astype(float)
df["time"] = pd.to_datetime(df.time)
```

In [17]:

```
df.sort_values(["market","time"],inplace=True)

# Create Addt. Cols
df['exchange'] = df.market.apply(lambda x: x.split("-")[0])
df['exchange-base'] = df.market.apply(lambda x: x.split("-")[0]+"-"+x.split("-")[1])
df['market_type'] = df.market.apply(lambda x: x.split("-")[-1])
df['base'] = df.market.apply(lambda x: x.split("-")[1])
df['quote'] = df.market.apply(lambda x: x.split("-")[2])

# Get volume by base asset by day

# Get top 10 assets by volume
total_vol_by_base = df.groupby('base',as_index=False).candle_usd_volume.sum()
total_vol_by_base.sort_values(by="candle_usd_volume",inplace=True)
base_top_list = total_vol_by_base.tail(10).base.tolist()
df["base2"] = np.where(df.base.isin(base_top_list),df.base,f"{len(total_vol_by_base)-10} others")

# Get sum by base asset by day
df_vol_by_base = df.groupby(["time","base2"],as_index=False).candle_usd_volume.sum()
df_vol_by_base['total_vol'] = df_vol_by_base.groupby("time").candle_usd_volume.transform(sum)
df_vol_by_base.columns=["time","base_asset","vol","total_vol"]
df_vol_by_base["vol_pct"]=(df_vol_by_base.vol/df_vol_by_base.total_vol)*100
df_vol_by_base.sort_values(["base_asset","time"],inplace=True)
```

In [18]:

```
df_vol_by_base
```

Out[18]:

	time	base_asset	vol	total_vol	vol_pct
0	2023-01-12 00:00:00+00:00	227 others	3.024039e+08	2.816129e+09	10.738284
11	2023-01-13 00:00:00+00:00	227 others	2.769317e+08	2.456469e+09	11.273568
22	2023-01-14 00:00:00+00:00	227 others	5.333341e+08	3.470438e+09	15.367918
33	2023-01-15 00:00:00+00:00	227 others	3.796145e+08	1.797935e+09	21.113917
44	2023-01-16 00:00:00+00:00	227 others	3.790459e+08	2.441579e+09	15.524622
...	...	...	...	...	...
274	2023-02-05 00:00:00+00:00	usdt	7.260107e+07	1.058640e+09	6.857957
285	2023-02-06 00:00:00+00:00	usdt	1.126150e+08	1.184402e+09	9.508177
296	2023-02-07 00:00:00+00:00	usdt	1.496484e+08	1.499942e+09	9.976947
307	2023-02-08 00:00:00+00:00	usdt	1.298067e+08	1.427891e+09	9.090800
318	2023-02-09 00:00:00+00:00	usdt	1.290931e+08	2.065112e+09	6.251145

319 rows × 5 columns

In [19]:

```
num_assets = len(total_vol_by_base)
top_assets  = df_vol_by_base[~df_vol_by_base.base_asset.isin(["btc","eth",f"{num_assets-10} others"])].groupby("base_asset").vo
l.sum().sort_values(ascending=False).index.tolist()
top_assets  = ["btc","eth",f"{num_assets-10} others"] + top_assets

# Pivot back to assets in columns
df_vol_pivot = df_vol_by_base.pivot(index='time',
                            columns="base_asset",
                            values="vol_pct")
```

In [20]:

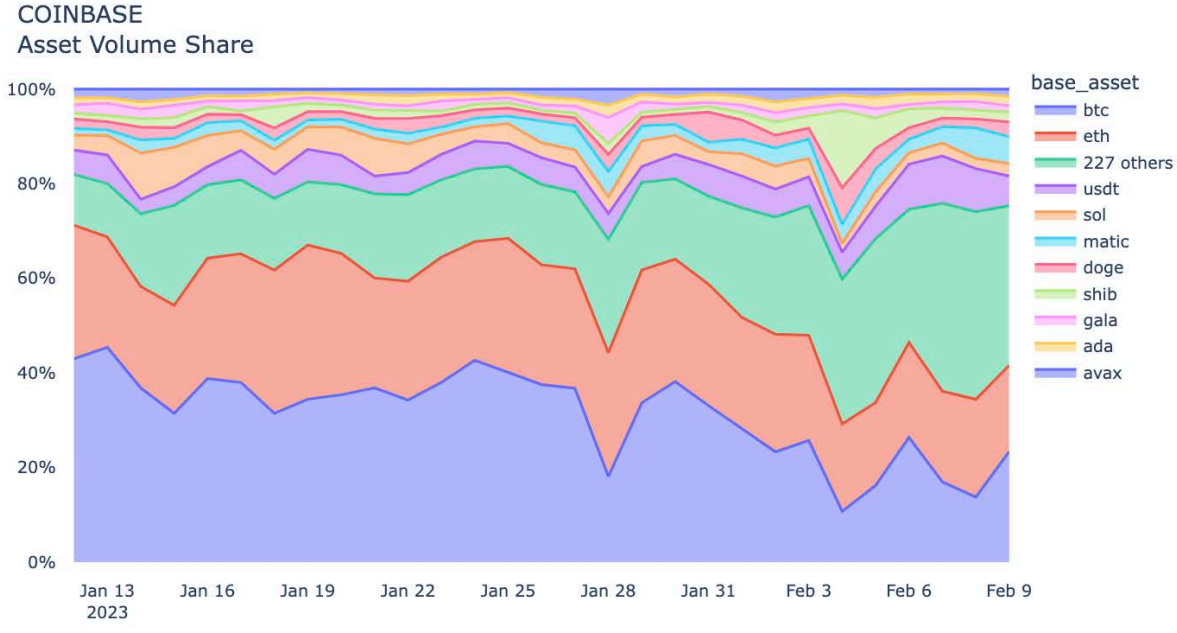
```
df_vol_pivot = df_vol_pivot[top_assets]
df_vol_pivot
```

Out[20]:

base_asset time	btc	eth	227 others	usdt	sol	matic	doge	shib	gala	ada	avax
2023-01-12 00:00:00+00:00	43.002704	28.217149	10.738284	5.150693	3.145261	1.449141	1.987004	1.238759	1.778773	1.458887	1.833345
2023-01-13 00:00:00+00:00	45.404984	23.350422	11.273568	6.036524	4.147833	1.204779	1.696790	1.347472	2.570647	1.184985	1.781996
2023-01-14 00:00:00+00:00	36.815908	21.459899	15.367918	3.083051	9.735007	2.804529	2.692829	1.749845	2.024259	1.573024	2.693731
2023-01-15 00:00:00+00:00	31.469197	22.824562	21.113917	3.949436	8.340945	1.917863	2.239986	2.144392	2.511546	1.299254	2.188901
2023-01-16 00:00:00+00:00	38.807304	25.453402	15.524622	3.821370	6.609530	2.700137	1.789007	1.595139	1.157596	1.170972	1.370922
2023-01-17 00:00:00+00:00	37.983520	27.206254	15.638299	6.240999	4.123963	2.138901	1.250758	0.819545	2.127328	0.981910	1.488525
2023-01-18 00:00:00+00:00	31.451696	30.300827	15.152288	5.080720	5.299129	1.917765	2.600150	4.494695	1.256860	1.365420	1.080450
2023-01-19 00:00:00+00:00	34.446320	32.591316	13.348447	6.878660	4.773283	1.453781	1.731894	1.809630	1.168813	0.984800	0.813055
2023-01-20 00:00:00+00:00	35.364057	29.948610	14.498315	6.221399	5.976357	1.637947	1.591652	1.377192	1.075448	1.330618	0.978405
2023-01-21 00:00:00+00:00	36.838630	23.236153	17.811115	3.754462	7.983104	1.953909	2.260308	1.742959	1.235503	1.898185	1.285672
2023-01-22 00:00:00+00:00	34.279661	25.091421	18.339283	4.632006	6.104783	2.205409	3.140634	1.707684	0.961586	2.146970	1.390563
2023-01-23 00:00:00+00:00	37.999648	26.479898	16.359033	5.359929	4.283335	1.497903	2.398307	0.983943	2.119249	1.507851	1.010905
2023-01-24 00:00:00+00:00	42.685127	25.068268	15.394954	5.862188	3.016694	1.832087	1.758856	1.152452	1.151263	1.001127	1.076984
2023-01-25 00:00:00+00:00	40.019375	28.430702	15.209171	4.880357	4.168386	1.601250	1.690216	1.116870	1.065190	1.088373	0.730108
2023-01-26 00:00:00+00:00	37.543342	25.312934	16.979653	5.628812	3.191940	4.589538	1.480125	0.819121	1.054052	1.700290	1.700192
2023-01-27 00:00:00+00:00	36.728176	25.273393	16.321196	5.212332	3.667146	5.091014	1.635560	1.012366	1.481184	1.477693	2.099942
2023-01-28 00:00:00+00:00	18.112376	26.173246	23.948825	5.474287	3.531600	5.359897	3.608320	2.299846	5.483709	2.620253	3.387640
2023-01-29 00:00:00+00:00	33.673637	28.077256	18.522623	3.346395	5.425037	3.222460	1.775022	1.100398	2.141555	1.643778	1.071839
2023-01-30 00:00:00+00:00	38.158785	25.904348	16.973749	5.187226	4.001058	2.320736	2.100799	1.074927	1.108184	1.527682	1.642505
2023-01-31 00:00:00+00:00	33.078171	25.702576	18.593066	6.672646	2.731453	2.038556	6.318253	1.216976	0.846729	1.793531	1.008042
2023-02-01 00:00:00+00:00	28.256213	23.450214	23.189140	6.696134	4.756348	3.077337	4.024163	1.572574	1.601258	1.860400	1.516220
2023-02-02 00:00:00+00:00	23.318684	24.853651	24.768282	5.917427	4.888009	3.810593	2.693003	2.830827	1.919697	2.290563	2.709264
2023-02-03 00:00:00+00:00	25.708560	22.270788	27.390271	6.101638	3.899661	4.095965	2.262352	2.606265	1.745353	1.914197	2.004948
2023-02-04 00:00:00+00:00	10.743401	18.498478	30.612665	5.648731	1.888724	4.067819	7.675197	16.306143	1.421254	1.838224	1.299363
2023-02-05 00:00:00+00:00	16.203371	17.555564	34.624873	6.857957	3.081349	4.844216	4.292267	6.467579	1.938292	2.396621	1.737912
2023-02-06 00:00:00+00:00	26.365577	20.071088	28.183257	9.508177	2.397627	2.907889	2.381283	4.017383	0.915588	2.245686	1.006445
2023-02-07 00:00:00+00:00	16.972316	19.171137	39.732473	9.976947	2.731202	3.489965	1.807612	2.132732	1.295804	1.614768	1.075044
2023-02-08 00:00:00+00:00	13.746535	20.672862	39.674158	9.090800	2.188947	6.424302	1.901669	1.891125	1.780507	1.657435	0.971659
2023-02-09 00:00:00+00:00	23.344196	18.273066	33.753204	6.251145	2.635787	5.687889	3.109561	2.094447	1.348125	1.962724	1.539855

In [21]:

```
fig = px.area(
        df_vol_pivot,
        width=900,
        height=500)
fig.update_layout(
    paper_bgcolor="white",
    title=str(exchange.upper() + '<br>Asset Volume Share'),
    plot_bgcolor = 'white',
    margin=dict(pad=12),
    colorway = ['#ff9900']
)
fig.update_yaxes(title="",matches=None, showticklabels=True, visible=True,showgrid=True,ticksuffix='%')
fig.update_xaxes(title="")
fig.show()
```



Retrieve Trade Sizes

Trade size is a useful metric for understanding the composition of an exchange's trading patterns. Is the platform more popular for retail traders, or large institutions? Which assets are driven by whale trades vs. small buys? These patterns are especially relevant for venues like Coinbase, where fee revenue is determined by trade size.

In [22]:

```
def get_trade_size_stats(start,end,market):
    """
    For a given date and market, get stats on number of trades and dist of trade sizes
        Returns a df with:
             Number of trades
             Volume (USD/Native)
             Number of trades by size groupings
             Volume derived from trades of various size groupings
    """

    #Call api
    df_trades = client.get_market_trades(markets=market,
                                         start_time=start,
                                         end_time=end).to_dataframe()
    #Prep data
    df_trades["amount_usd"] = df_trades.amount*df_trades.price
    df_trades["amount_usd_groups"] = pd.cut(df_trades["amount_usd"],bins=[0,1e3,1e4,1e5,1e6,1e7,1e100])
    print(df_trades.time.min())
    print(df_trades.time.max())

    #Get stats by group
    sum_count_by_size = df_trades.groupby("amount_usd_groups").agg({"amount_usd":['count',sum]})

    #Collect into a df
    df_day = pd.DataFrame()

    df_day.loc[start,"NumTrades"] = len(df_trades)
    df_day.loc[start,"VolUSD"] = df_trades.amount_usd.sum()
    df_day.loc[start,"VolNTV"] = df_trades.amount.sum()

    df_day.loc[start,"AvgSizeUSD"] = df_trades.amount_usd.mean()
    df_day.loc[start,"MedSizeUSD"] = df_trades.amount_usd.median()
    df_day.loc[start,"MaxSizeUSD"] = df_trades.amount_usd.max()

    df_day.loc[start,"NumTrades_0-1K"]      = sum_count_by_size.iloc[0,0]
    df_day.loc[start,"NumTrades_1K-10K"]    = sum_count_by_size.iloc[1,0]
    df_day.loc[start,"NumTrades_10K-100K"]  = sum_count_by_size.iloc[2,0]
    df_day.loc[start,"NumTrades_100K-1M"]   = sum_count_by_size.iloc[3,0]
    df_day.loc[start,"NumTrades_1M-10M"]    = sum_count_by_size.iloc[4,0]
    df_day.loc[start,"NumTrades_10M-Over"]  = sum_count_by_size.iloc[5,0]

    df_day.loc[start,"VolUSD_Trades_0-1K"]     = sum_count_by_size.iloc[0,1]
    df_day.loc[start,"VolUSD_Trades_1K-10K"]   = sum_count_by_size.iloc[1,1]
    df_day.loc[start,"VolUSD_Trades_10K-100K"] = sum_count_by_size.iloc[2,1]
    df_day.loc[start,"VolUSD_Trades_100K-1M"]  = sum_count_by_size.iloc[3,1]
    df_day.loc[start,"VolUSD_Trades_1M-10M"]   = sum_count_by_size.iloc[4,1]
    df_day.loc[start,"VolUSD_Trades_10M-Over"] = sum_count_by_size.iloc[5,1]

    return df_day
```

In [23]:

```
start = '2023-01-01'
end = '2023-01-02'
```

In [24]:

```
df = get_trade_size_stats(start, end,'coinbase-btc-usd-spot')
```

```
2023-01-01 00:00:00.012981+00:00
2023-01-02 23:59:59.858294+00:00
```

In [25]:

```
df.transpose()
```

Out[25]:

	2023-01-01
NumTrades	5.953310e+05
VolUSD	4.029298e+08
VolNTV	2.422920e+04
AvgSizeUSD	6.768165e+02
MedSizeUSD	1.455323e+02
MaxSizeUSD	2.352945e+05
NumTrades_0-1K	5.099630e+05
NumTrades_1K-10K	8.009200e+04
NumTrades_10K-100K	5.249000e+03
NumTrades_100K-1M	2.700000e+01
NumTrades_1M-10M	0.000000e+00
NumTrades_10M-Over	0.000000e+00
VolUSD_Trades_0-1K	9.278940e+07
VolUSD_Trades_1K-10K	2.110799e+08
VolUSD_Trades_10K-100K	9.528703e+07
VolUSD_Trades_100K-1M	3.773535e+06
VolUSD_Trades_1M-10M	0.000000e+00
VolUSD_Trades_10M-Over	0.000000e+00

In [56]:

```
trade_sizes = df[['VolUSD_Trades_0-1K',
                  'VolUSD_Trades_1K-10K',
                  'VolUSD_Trades_10K-100K',
                  'VolUSD_Trades_100K-1M',
                  'VolUSD_Trades_1M-10M',
                  'VolUSD_Trades_10M-Over']]
```

In [57]:

```
trade_sizes = trade_sizes.rename(columns=lambda x: x.split('_')[2])
#trade_sizes.rename(columns=trade_sizes.columns.str.split('_')[:][2])
trade_sizes
```

Out[57]:

	0-1K	1K-10K	10K-100K	100K-1M	1M-10M	10M-Over
2023-01-01	9.278940e+07	2.110799e+08	9.528703e+07	3.773535e+06	0.0	0.0

In [58]:

```
fig = px.bar(trade_sizes.transpose(),
       title=str('Coinbase Volume by Trade Size<br>(' + start + ' to ' + end + ')'),
        width=800,
        height=600)
fig.update_layout(plot_bgcolor="white",
                  showlegend=False)
fig.update_yaxes(title="",matches=None, showticklabels=True, tickprefix='$',
                 showgrid=True,gridcolor='gray', griddash='dot')
fig.update_xaxes(title="",showgrid=True)
fig.show()
```