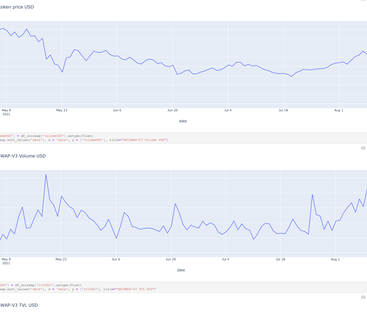
```
import requests

import plotly.express as px
pio.templates.default="plotly"

import time
import datetime
import pandas as pd
```

Information about UNI (DEX) and UNISWAP V3

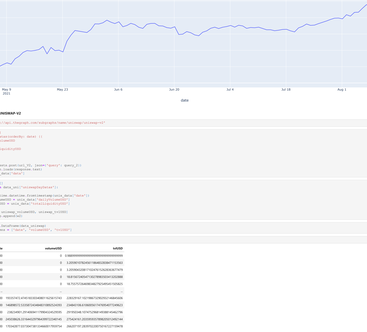
```
url_V2 = "https://api.thegraph.com/subgraphs/name/uniswap/uniswap-v2"

query = f"""
    {{
        tokenDayDatas(
            first: 1000,
            skip: 0,
            token: "0x1f9840a85d5af5bf1d1762f925bdaddc4201f984",
            date_lt: {date},
            orderBy:date,
            orderDirection: asc
            ) {{
                id
                date
                totalLiquidityUSD
                priceUSD
                volumeUSD
            }}
        }}
    """
response = requests.post(url_V2, json={'query': query})
json_data = json.loads(response.text)
data_uni = pd.DataFrame(['data'])
```

```
data_uniswap = []
for date in pd.date_range(start='2020-09-01',end='2021-08-31'):

    date = datetime.datetime.fromtimestamp(int(date.date_time))
    time_subset = eval(str(data_frame['tokenDayDatas'].
    total_liquidity = eval(str(time_subset['totalLiquidityUSD'].
    token_volumeUSD = eval(str(time_subset['volumeUSD'].
    token_priceUSD = eval(str(time_subset['priceUSD'].

    if (len(time_subset['date'])==1 and time_subset['date'].
    token_volumeUSD, token_priceUSD=
    else:
    token_volumeUSD, token_priceUSD=

    if (len(time_subset['date'])==1 and time_subset['date'].
    token_volumeUSD, token_priceUSD=
    else:
    token_volumeUSD, token_priceUSD=

    d = {date, token_volumeUSD, token_priceUSD}
    data_uniswap.append(d)
```

```
import plotly.express as px
pd.DataFrame(data_uniswap).astype(float)
px.line(pd.DataFrame(data_uniswap).astype(float),
    x='date',y='token_volumeUSD',title='UNI token volume (DEX (DEX)')
```




```
pd.DataFrame(data_uniswap).astype(float)
px.line(pd.DataFrame(data_uniswap).astype(float),
    x='date',y='token_priceUSD',title='UNI token price USD')
```




```
pd.DataFrame(data_uniswap).astype(float)
px.line(pd.DataFrame(data_uniswap).astype(float),
    x='date',y=['token_volumeUSD'],title='UNISWAP-V3 Volume USD')
```

```
pd.DataFrame(data_uniswap).astype(float)
px.line(pd.DataFrame(data_uniswap).astype(float),
    x='date',y=['totalLiquidUSD'],title='UNISWAP-V3 TVL USD')
```

Information about UNISWAP V2

```
url_V2 = "https://api.thegraph.com/subgraphs/name/uniswap/uniswap-v2"

query_2 = f"""
    {{
        pairHourDataDays(
            first: 1000,
            skip: 0,
            pair: "0xd3d2e2692501a5c9ca623199d38826e8ba8bd549",
            date_lt: {date},
            orderBy:date,
            orderDirection: asc
            ) {{
                id
                date
                totalLiquidUSD
            }}
        }}
    """
response = requests.post(url_V2, json={'query': query_2})
json_data = json.loads(response.text)
data_uni = pd.DataFrame(['data'])
```

```
data_uniswap = []
for date in pd.date_range(start='2020-09-01',end='2021-08-31'):

    date = datetime.datetime.fromtimestamp(int(date.date_time))
    uniswap_volumeUSD = eval(str(data['pairHourDayDatas'].
    uniswap_totalLiquidUSD = eval(str(data['pairHourDayDatas'].

    d = {date, uniswap_volumeUSD, uniswap_totalLiquidUSD}
    data_uniswap.append(d)
```

```
pd.DataFrame(data_uniswap).astype(float)
px.line(pd.DataFrame(data_uniswap).astype(float),
    x='date',y=['uniswap_volumeUSD'],title='UNISWAP-V2 Volume USD')
```

```
pd.DataFrame(data_uniswap).astype(float)
px.line(pd.DataFrame(data_uniswap).astype(float),
    x='date',y=['uniswap_totalLiquidUSD'],title='UNISWAP-V2 TVL USD')
```

Information about UNI (DEX) from Binance

```
BASE = "https://api.binance.com"
endpoint = "/fapi/v1/klines"
response = requests.get(BASE + endpoint)
```

```
import pandas as pd
import time, sys
import datetime
from dateutil.parser import datetime
from binance.client import Client

    def __init__(self, api_key: str = "", secret_key: str = "",
        self._api_key = api_key
        self._secret_key = secret_key
        self._session = self._init_session()  # also sets all the base endpoints and ping a connectivity checkpoint
    def _init_request(self, method: str, endpoint: str, payload: dict = None, sign: bool = False, api_url: str = None:
        headers = {}
        if sign:
            timestamp = kwargs.pop('timestamp', self._get_ts())
            payload = {
                'timestamp': timestamp,
                'recvWindow': kwargs.pop('recvWindow', 5000),
                **payload
            }
        if payload:
            query = urlencode(payload, True)
            payload = {}
        if method == "POST":
            response = request.get(api_url + endpoint, params, headers=headers)
        elif method == "DELETE":
            response = request.post(api_url + endpoint, params, headers=headers)
        if response.status_code != 200:
            return response.text()
        data = response.text(parse_float=str)
        sys.stdout.write("[%s] %s %s\n" % (method, endpoint, '[response_status_code] - [response_reason]'
        return ResponseObject(resp)
    def get_klines(self, limit=None, interval: str, start_time: datetime = None, end_time: datetime = None, **kwargs: Di
        """
        :
        :
        :
```