

thegraph_data_access

March 28, 2021

1 TheGraph data access

courtesy of @markusbkoch submitted by @mzargham

```
[1]: import pandas as pd
import json
import requests
import matplotlib.pyplot as plt
url = 'https://api.thegraph.com/subgraphs/name/balancer-labs/balancer'
query = ''
query {{
    pools(first: 1000, skip:{}) {{
        id
        liquidity
    }}
}}'''
n = 0
pools = []
while True:
    print(f'request {n+1}')
    v= query.format(n*1000)
    print(v)
    r = requests.post(url, json = {'query':v})
    p = json.loads(r.content)['data']['pools']
    print(f'results {len(p)}')
    pools.extend(p)
    print(f'total {len(pools)}')
    n += 1
    if len(p) < 1000:
        break
subgraph_tv1 = pd.DataFrame(pools)
```

request 1

```
query {
  pools(first: 1000, skip:0) {
    id
    liquidity
  }
}
```

```

    }
}
results 1000
total 1000
request 2

```

```

query {
  pools(first: 1000, skip:1000) {
    id
    liquidity
  }
}
results 1000
total 2000
request 3

```

```

query {
  pools(first: 1000, skip:2000) {
    id
    liquidity
  }
}
results 1000
total 3000
request 4

```

```

query {
  pools(first: 1000, skip:3000) {
    id
    liquidity
  }
}
results 1
total 3001

```

```
[2]: subgraph_tv1.head()
```

```

[2]:                                     id \
0  0x002ad19fb25c6206d6d19e524f363ea846afe4a5
1  0x002d3737e074fb4521036f2c41beba05d221ba69
2  0x003a70265a3662342010823bea15dc84c6f7ed54
3  0x004e74ff81239c8f2ec0e2815defb970f3754d86
4  0x0077732357ac0f29e26ea629b79ab3b266ddb796

                                     liquidity
0  3192867.479620907831246560378953618
1  67851.211822

```

```

2  1661954.460770052569839235795743499
3  1426.781480757952119413024482275002
4  0.8653140420464888814426818591183125

```

Dealing with pagination here is a pain and the query string above does not actually run in the explorer as written. In order to make it easier to move back and forth between the [explorer](#) and the python environment we should build a function to run the same query we use in the explorer, for example:

```

{pools(first:1000){
    id
    liquidity
  }
}

```

```

[3]: def query_theGraph(raw_query, field_name, url, verbose=False, hardcap=5000):

    query_parts =raw_query.split(',')
    paginator = ", skip:{}"
    #this expects the raw query to gave a `first:1000` term
    n = 0
    records = []
    while True:
        print(f'request {n+1}')
        skipper = paginator.format(n*1000)
        query = 'query '+query_parts[0]+skipper+')'+query_parts[1]

        if verbose:
            print(query)

        r = requests.post(url, json = {'query':query})

        try:
            d = json.loads(r.content)['data'][field_name]
        except:
            #print(r.content)
            errors = json.loads(r.content)['errors']
            #print(errors)
            for e in errors:
                print(e['message'])

        print(f'results {len(d)}')
        records.extend(d)
        print(f'total {len(records)}')

        if n*1000>hardcap:
            break

```

```

        n += 1
        if len(d) < 1000:
            break

    return pd.DataFrame(records)

```

```

[4]: raw_query = '''{pools(first:1000){
      id
      liquidity
    }
  }
'''
field_name = 'pools'

subgraph_tv12 = query_theGraph(raw_query, field_name, url, True)

```

```

request 1
query {pools(first:1000, skip:0){
  id
  liquidity
}
}

```

```

results 1000
total 1000
request 2
query {pools(first:1000, skip:1000){
  id
  liquidity
}
}

```

```

results 1000
total 2000
request 3
query {pools(first:1000, skip:2000){
  id
  liquidity
}
}

```

```

results 1000
total 3000

```

```
request 4
query {pools(first:1000, skip:3000){
  id
  liquidity
}}
}
```

```
results 1
total 3001
```

```
[5]: subgraph_tv12
```

```
[5]:
```

| | id \ | liquidity |
|------|---|--------------------------------------|
| 0 | 0x002ad19fb25c6206d6d19e524f363ea846afe4a5 | 3192867.479620907831246560378953618 |
| 1 | 0x002d3737e074fb4521036f2c41beba05d221ba69 | 67851.211822 |
| 2 | 0x003a70265a3662342010823bea15dc84c6f7ed54 | 1661954.460770052569839235795743499 |
| 3 | 0x004e74ff81239c8f2ec0e2815defb970f3754d86 | 1426.781480757952119413024482275002 |
| 4 | 0x0077732357ac0f29e26ea629b79ab3b266ddb796 | 0.8653140420464888814426818591183125 |
| ... | ... | ... |
| 2996 | 0xffe8c31fb0ab62c99fc6e8c724d0f1949dbaa44f | 2288.873674457686799301414804642943 |
| 2997 | 0xffff293e1f6c174867f23351c1510833c8087fecb | 0 |
| 2998 | 0xffff29c8bce4fbe8702e9fa16e0e6c551f364f420 | 0 |
| 2999 | 0xffff2a5f81d14729408201341df42af29f3b30458 | 5509027.9325697205516180566661843 |
| 3000 | 0xffff82910d352abe04d00d542f0ded0bfc8516f78 | 0 |

```
[3001 rows x 2 columns]
```

```
[6]: subgraph_tv12.head()
```

```
[6]:
```

| | id \ |
|---|--|
| 0 | 0x002ad19fb25c6206d6d19e524f363ea846afe4a5 |
| 1 | 0x002d3737e074fb4521036f2c41beba05d221ba69 |
| 2 | 0x003a70265a3662342010823bea15dc84c6f7ed54 |

```

3  0x004e74ff81239c8f2ec0e2815defb970f3754d86
4  0x0077732357ac0f29e26ea629b79ab3b266ddb796

```

```

                                liquidity
0    3192867.479620907831246560378953618
1                                67851.211822
2    1661954.460770052569839235795743499
3    1426.781480757952119413024482275002
4    0.8653140420464888814426818591183125

```

```
[7]: subgraph_tv12.columns = ['id', 'liquidity2']
```

```
[8]: checker = subgraph_tv1.merge(subgraph_tv12)
```

```
[9]: checker['matches'] = checker.liquidity==checker.liquidity2
```

```
[10]: checker.matches.describe()
```

```

[10]: count      3001
      unique        1
      top         True
      freq       3001
      Name: matches, dtype: object

```

Now that have checked the data we can proceed with some exploratory analysis.

```
[11]: subgraph_tv1.liquidity= subgraph_tv1.liquidity.apply(float)
```

```
[12]: subgraph_tv1.sort_values('liquidity', inplace=True)
```

```
[13]: subgraph_tv1.liquidity
```

```

[13]: 1500    0.000000e+00
      1739    0.000000e+00
      1737    0.000000e+00
      1734    0.000000e+00
      1733    0.000000e+00
      ...
      1091    6.912330e+07
      1642    7.784344e+07
      1071    2.407509e+08
      370     3.320281e+08
      2315    3.777230e+08
      Name: liquidity, Length: 3001, dtype: float64

```

```

[14]: plt_df=subgraph_tv1[subgraph_tv1.liquidity>1].copy().sort_values('liquidity',
↪ascending=False)

```

```
[15]: subgraph_tv1.describe()
```

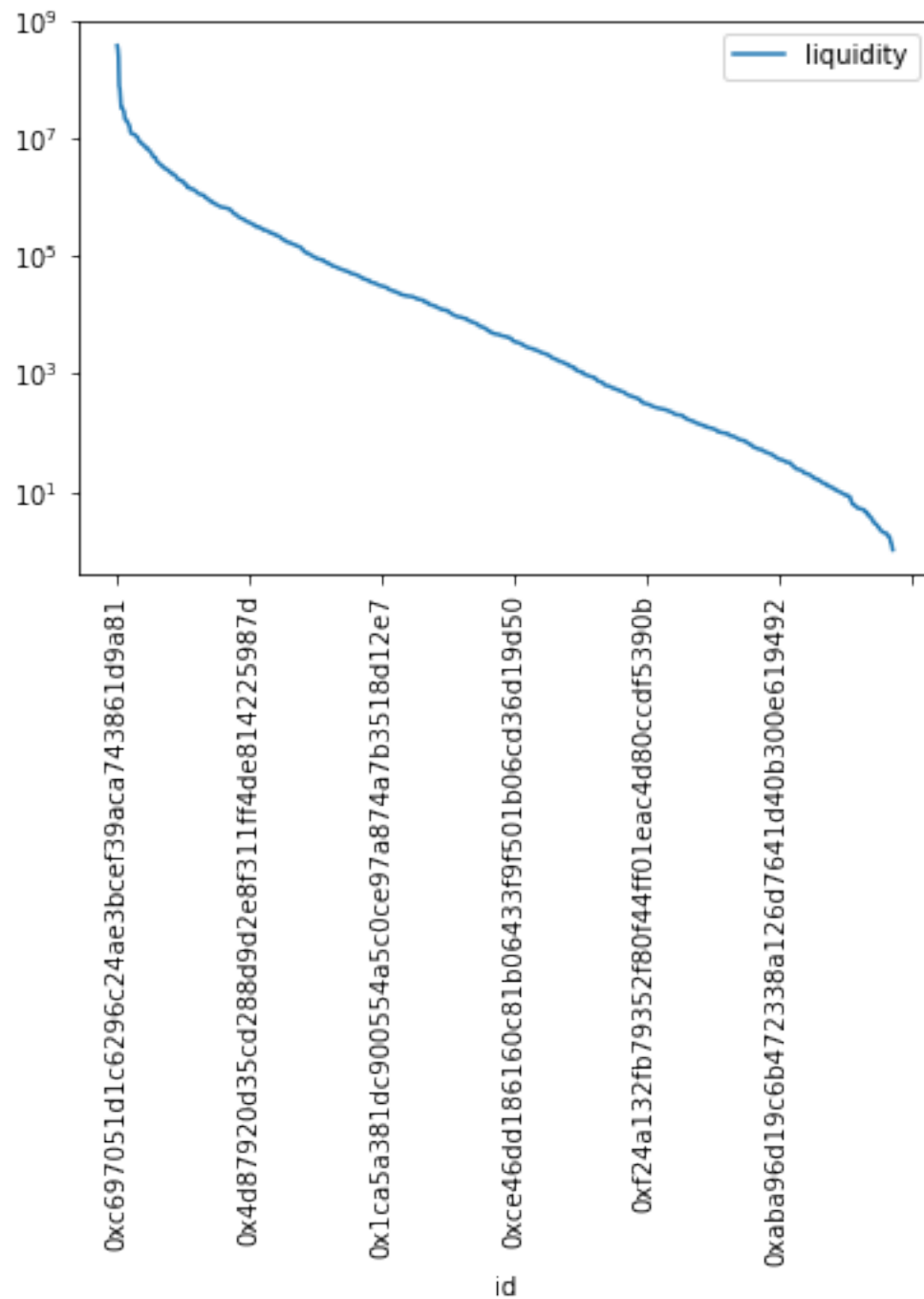
```
[15]:          liquidity
count    3.001000e+03
mean     6.780660e+05
std      1.055964e+07
min      0.000000e+00
25%      0.000000e+00
50%      0.000000e+00
75%      5.737744e+02
max      3.777230e+08
```

```
[16]: plt_df.tail()
```

```
[16]:          id liquidity
344    0x1d261ec7ab834fedb01602c5b7ffc6fc68362bbf    1.577654
999    0x53f160490d7e48ba2c31be4790f3d87a2f4dc662    1.371422
1868   0x9e4a4b53e19410ae519be74f92659e5b0ef9489b    1.330313
2380   0xcb8ec8236aff8e112517f4e9a9ffb413a237e6b7    1.153105
1312   0x6d42692518c8b09c883e7c1e69c97518107f2185    1.030083
```

```
[17]: plt_df.plot(x='id', y='liquidity', logy=True)
plt.xticks(rotation=90)
```

```
[17]: (array([-200.,   0.,  200.,  400.,  600.,  800., 1000., 1200., 1400.]),
 [Text(-200.0, 0, '0x5df340aee23aee3feedeead6890fa1aaa94c19ed'),
  Text(0.0, 0, '0xc697051d1c6296c24ae3bcef39aca743861d9a81'),
  Text(200.0, 0, '0x4d87920d35cd288d9d2e8f311ff4de814225987d'),
  Text(400.0, 0, '0x1ca5a381dc900554a5c0ce97a874a7b3518d12e7'),
  Text(600.0, 0, '0xce46dd186160c81b06433f9f501b06cd36d19d50'),
  Text(800.0, 0, '0xf24a132fb79352f80f44ff01eac4d80ccdf5390b'),
  Text(1000.0, 0, '0xaba96d19c6b472338a126d7641d40b300e619492'),
  Text(1200.0, 0, ''),
  Text(1400.0, 0, '')])
```

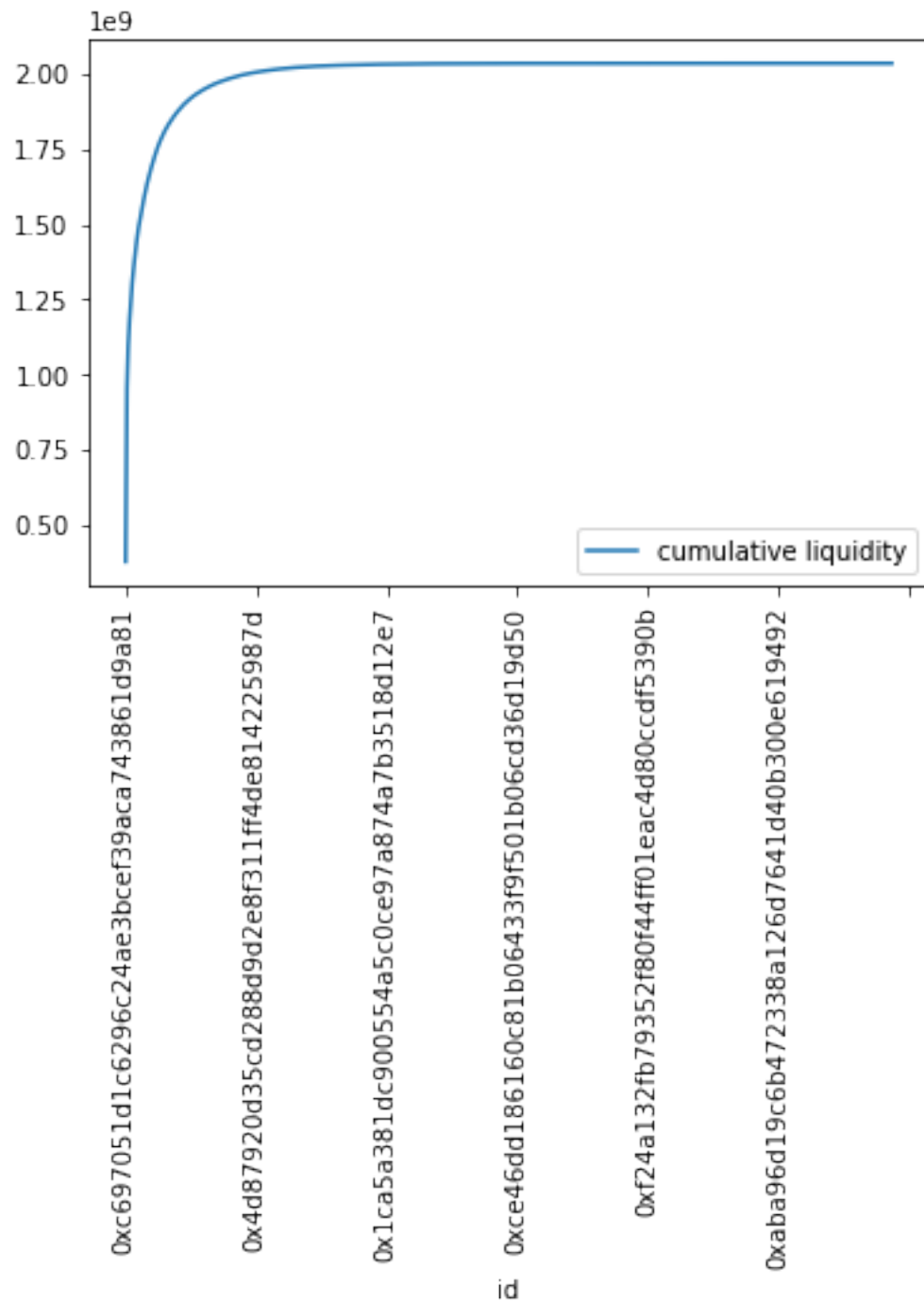


```
[18]: plt_df['cumulative liquidity'] = plt_df.liquidity.cumsum()
```

```
[19]: plt_df.plot(x='id', y='cumulative liquidity', logy=False)
plt.xticks(rotation=90)
```



```
[19]: (array([-200.,    0.,  200.,  400.,  600.,  800., 1000., 1200., 1400.]),
      [Text(-200.0, 0, '0x5df340aee23aee3feedeead6890fa1aaa94c19ed'),
       Text(0.0, 0, '0xc697051d1c6296c24ae3bcef39aca743861d9a81'),
       Text(200.0, 0, '0x4d87920d35cd288d9d2e8f311ff4de814225987d'),
       Text(400.0, 0, '0x1ca5a381dc900554a5c0ce97a874a7b3518d12e7'),
       Text(600.0, 0, '0xce46dd186160c81b06433f9f501b06cd36d19d50'),
       Text(800.0, 0, '0xf24a132fb79352f80f44ff01eac4d80ccdf5390b'),
       Text(1000.0, 0, '0xaba96d19c6b472338a126d7641d40b300e619492'),
       Text(1200.0, 0, ''),
       Text(1400.0, 0, '')])
```



[]:

[]: