thegraph_data_access

March 29, 2021

1 TheGraph data access

courtesty 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_tvl = 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 4
total 3004
```

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

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

                              liquidity
0   3192867.479620907831246560378953618
1                         855355.145728
```

```
2    1584512.42902548719982460687576808
3   1426.78148075795211941302448227500 2
4  0.865314042046488881442681859118312 5
```

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]:
```python
def query_theGraph(raw_query, field_name, url, verbose=False, hardcap=5000):

    query_parts =raw_query.split(')')
    paginator = ", skip:{}"
    #this expectes 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_tvl2 = 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 4
total 3004
```

[5]:
```
subgraph_tvl2
```

[5]:
```
                                                  id  \
0     0x002ad19fb25c6206d6d19e524f363ea846afe4a5
1     0x002d3737e074fb4521036f2c41beba05d221ba69
2     0x003a70265a3662342010823bea15dc84c6f7ed54
3     0x004e74ff81239c8f2ec0e2815defb970f3754d86
4     0x0077732357ac0f29e26ea629b79ab3b266ddb796
…                                               …
2999  0xffe8c31fb0ab62c99fc6e8c724d0f1949dbaa44f
3000  0xfff293e1f6c174867f23351c1510833c8087fecb
3001  0xfff29c8bce4fbe8702e9fa16e0e6c551f364f420
3002  0xfff2a5f81d14729408201341df42af29f3b30458
3003  0xfff82910d352abe04d00d542f0ded0bfc8516f78

                                liquidity
0      3192867.47962090783124656037895 3618
1                           855355.145728
2       1584512.42902548719982460687576808
3      1426.78148075795211941302448227 5002
4     0.86531404204648888144268185911831 25
…                                        …
2999   2288.87367445768679930141480464 2943
3000                                      0
3001                                      0
3002   4940376.29359940025711533691235 7738
3003                                      0

[3004 rows x 2 columns]
```

[6]:
```
subgraph_tvl2.head()
```

[6]:
```
                                                id  \
0  0x002ad19fb25c6206d6d19e524f363ea846afe4a5
1  0x002d3737e074fb4521036f2c41beba05d221ba69
2  0x003a70265a3662342010823bea15dc84c6f7ed54
```

```
3  0x004e74ff81239c8f2ec0e2815defb970f3754d86
4  0x0077732357ac0f29e26ea629b79ab3b266ddb796

                                liquidity
0   3192867.47962090783124656037895 3618
1                          855355.145728
2    1584512.42902548719982460687576808
3   1426.78148075795211941302448227 5002
4  0.86531404204648888144268185911831 25
```

[7]:
```
subgraph_tvl2.columns = ['id','liquidity2']
```

[8]:
```
checker = subgraph_tvl.merge(subgraph_tvl2)
```

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

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

[10]:
```
count     3004
unique       1
top       True
freq      3004
Name: matches, dtype: object
```

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

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

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

[13]:
```
subgraph_tvl.liquidity
```

[13]:
```
1501    0.000000e+00
1736    0.000000e+00
1735    0.000000e+00
1734    0.000000e+00
1733    0.000000e+00
           …
1092    6.912330e+07
1644    8.250209e+07
1072    2.566254e+08
370     3.523437e+08
2317    4.004134e+08
Name: liquidity, Length: 3004, dtype: float64
```

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

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

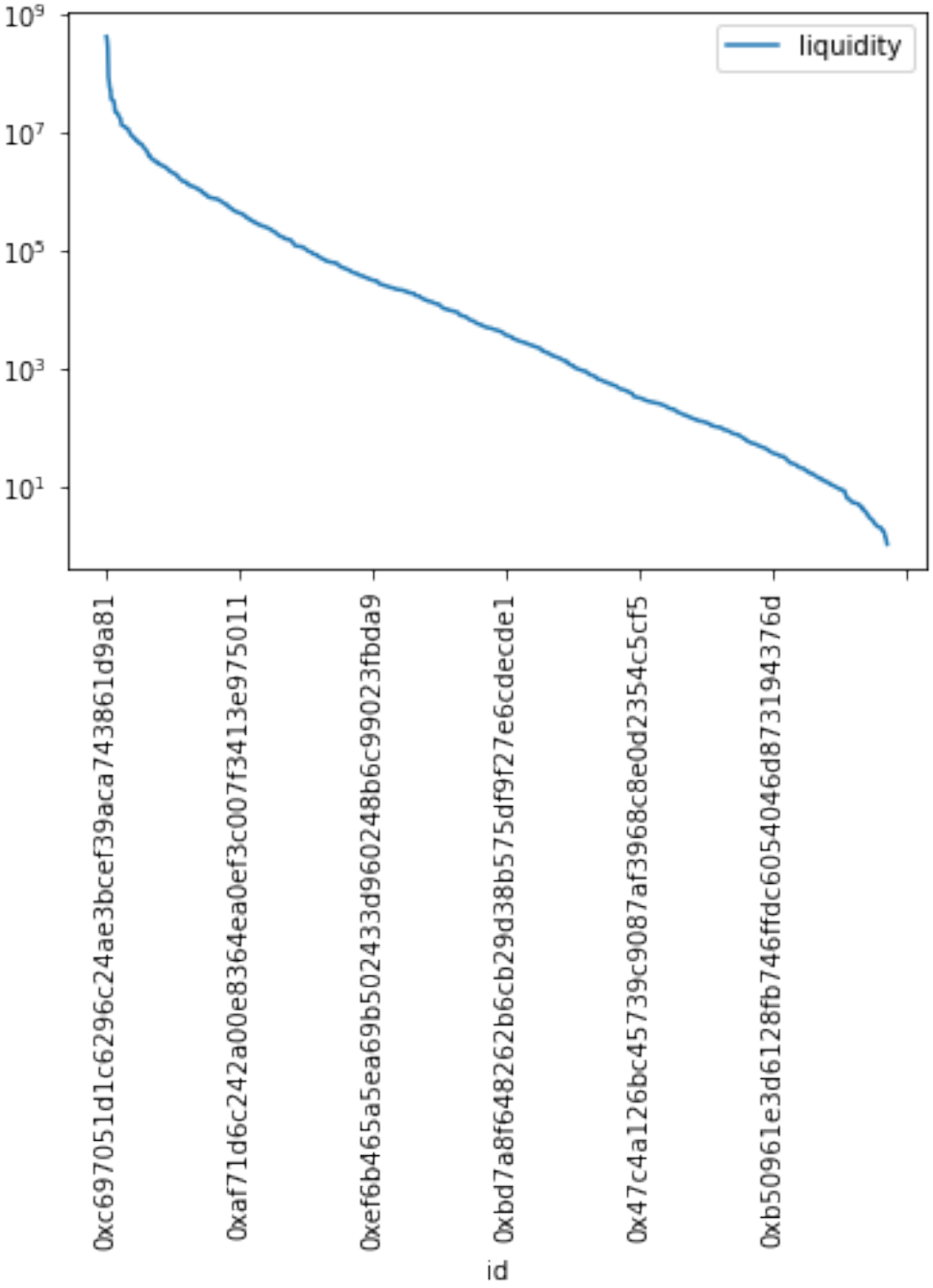
[15]:
```
          liquidity
count  3.004000e+03
mean   7.347331e+05
std    1.123784e+07
min    0.000000e+00
25%    0.000000e+00
50%    0.000000e+00
75%    5.634744e+02
max    4.004134e+08
```

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

[16]:
```
                                            id  liquidity
344   0x1d261ec7ab834fedb01602c5b7ffc6fc68362bbf   1.577654
1000  0x53f160490d7e48ba2c31be4790f3d87a2f4dc662   1.371422
1870  0x9e4a4b53e19410ae519be74f92659e5b0ef9489b   1.330313
2382  0xcb8ec8236aff8e112517f4e9a9ffb413a237e6b7   1.153105
1313  0x6d42692518c8b09c883e7c1e69c97518107f2185   1.030083
```

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

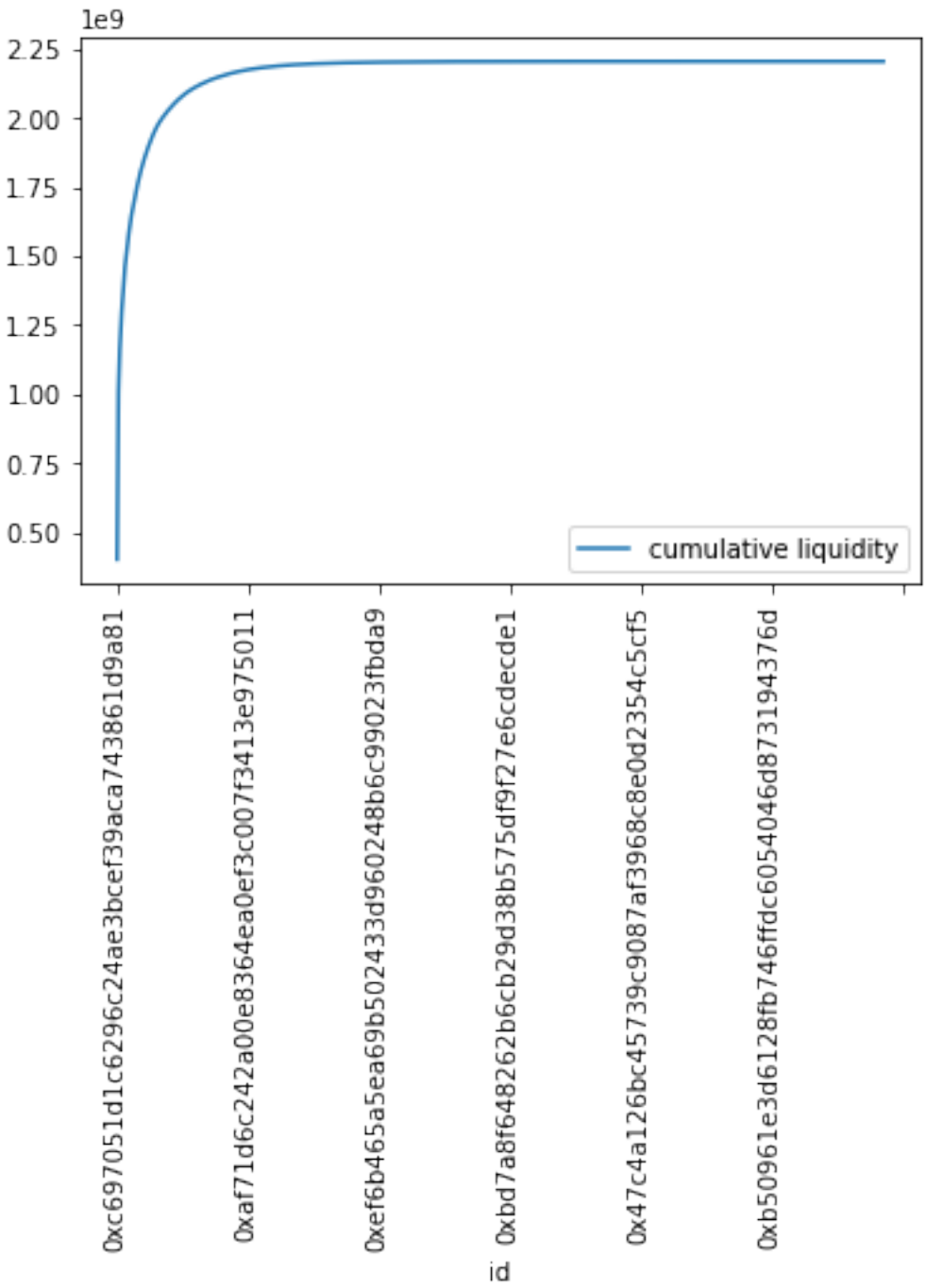
[17]:
```
(array([-200.,    0.,  200.,  400.,  600.,  800., 1000., 1200., 1400.]),
 [Text(-200.0, 0, '0x5df340aee23aee3feedeead6890fa1aaa94c19ed'),
  Text(0.0, 0, '0xc697051d1c6296c24ae3bcef39aca743861d9a81'),
  Text(200.0, 0, '0xaf71d6c242a00e8364ea0ef3c007f3413e975011'),
  Text(400.0, 0, '0xef6b465a5ea69b502433d960248b6c99023fbda9'),
  Text(600.0, 0, '0xbd7a8f648262b6cb29d38b575df9f27e6cdecde1'),
  Text(800.0, 0, '0x47c4a126bc45739c9087af3968c8e0d2354c5cf5'),
  Text(1000.0, 0, '0xb50961e3d6128fb746ffdc6054046d873194376d'),
  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, '0xaf71d6c242a00e8364ea0ef3c007f3413e975011'),
        Text(400.0, 0, '0xef6b465a5ea69b502433d960248b6c99023fbda9'),
        Text(600.0, 0, '0xbd7a8f648262b6cb29d38b575df9f27e6cdecde1'),
        Text(800.0, 0, '0x47c4a126bc45739c9087af3968c8e0d2354c5cf5'),
        Text(1000.0, 0, '0xb50961e3d6128fb746ffdc6054046d873194376d'),
        Text(1200.0, 0, ''),
        Text(1400.0, 0, '')])
```


1e9
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
cumulative liquidity
0xc697051d1c6296c24ae3bcef39aca743861d9a81
0xaf71d6c242a00e8364ea0ef3c007f3413e975011
0xef6b465a5ea69b502433d960248b6c99023fbda9
0xbd7a8f648262b6cb29d38b575df9f27e6cdecde1
0x47c4a126bc45739c9087af3968c8e0d2354c5cf5
0xb50961e3d6128fb746ffdc6054046d8731943 76d
id

[ ]:

[ ]: