thegraph_data_access

March 21, 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 972
total 2972
```

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

[2]:

	id	liquidity
0	0x002d3737e074fb4521036f2c41beba05d221ba69	0
1	0x003a70265a3662342010823bea15dc84c6f7ed54	1595731.132743630150772111474830064
2	0x004e74ff81239c8f2ec0e2815defb970f3754d86	680.928486911431236447041487663722
3	0x0077732357ac0f29e26ea629b79ab3b266ddb796	0.865314042046488881442681859118312 5
4	0x0092b2d25d76d84d27b999fe93d5e1c70511cd2b	8.417288376820507167011285071450 78

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 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 972
total 2972
```

[5]:
```
subgraph_tvl2
```

[5]:
```
                                                  id  \
0     0x002d3737e074fb4521036f2c41beba05d221ba69
1     0x003a70265a3662342010823bea15dc84c6f7ed54
2     0x004e74ff81239c8f2ec0e2815defb970f3754d86
3     0x0077732357ac0f29e26ea629b79ab3b266ddb796
4     0x0092b2d25d76d84d27b999fe93d5e1c70511cd2b
…                                           …
```

```
2967  0xffe8c31fb0ab62c99fc6e8c724d0f1949dbaa44f
2968  0xfff293e1f6c174867f23351c1510833c8087fecb
2969  0xfff29c8bce4fbe8702e9fa16e0e6c551f364f420
2970  0xfff2a5f81d14729408201341df42af29f3b30458
2971  0xfff82910d352abe04d00d542f0ded0bfc8516f78

                                  liquidity
0                                         0
1       1595731.13274363015077211474830064
2        680.92848691143123644704148766372 2
3      0.865314042046488881442681859118312 5
4        8.41728837682050716701128507145078
…                                       …
2967   3572.46246072109407797495712269991 3
2968                                      0
2969                                      0
2970   4525743.52312092837461642867593017 8
2971                                      0

[2972 rows x 2 columns]
```

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

[6]:
```
                                          id  \
0  0x002d3737e074fb4521036f2c41beba05d221ba69
1  0x003a70265a3662342010823bea15dc84c6f7ed54
2  0x004e74ff81239c8f2ec0e2815defb970f3754d86
3  0x0077732357ac0f29e26ea629b79ab3b266ddb796
4  0x0092b2d25d76d84d27b999fe93d5e1c70511cd2b

                              liquidity
0                                     0
1   1595731.13274363015077211474830064
2    680.92848691143123644704148766372
3  0.865314042046488881442681859118312 5
4    8.41728837682050716701128507145078
```

[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     2972
unique       1
```

```
top       True
freq      2972
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]:
```
0       0.000000e+00
1725    0.000000e+00
1723    0.000000e+00
1722    0.000000e+00
1721    0.000000e+00
             …
1079    6.912330e+07
1624    8.177993e+07
1059    2.709410e+08
2292    3.237292e+08
366     3.516588e+08
Name: liquidity, Length: 2972, dtype: float64
```

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

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

[15]:
```
          liquidity
count  2.972000e+03
mean   7.033577e+05
std    1.056827e+07
min    0.000000e+00
25%    0.000000e+00
50%    0.000000e+00
75%    5.461584e+02
max    3.516588e+08
```

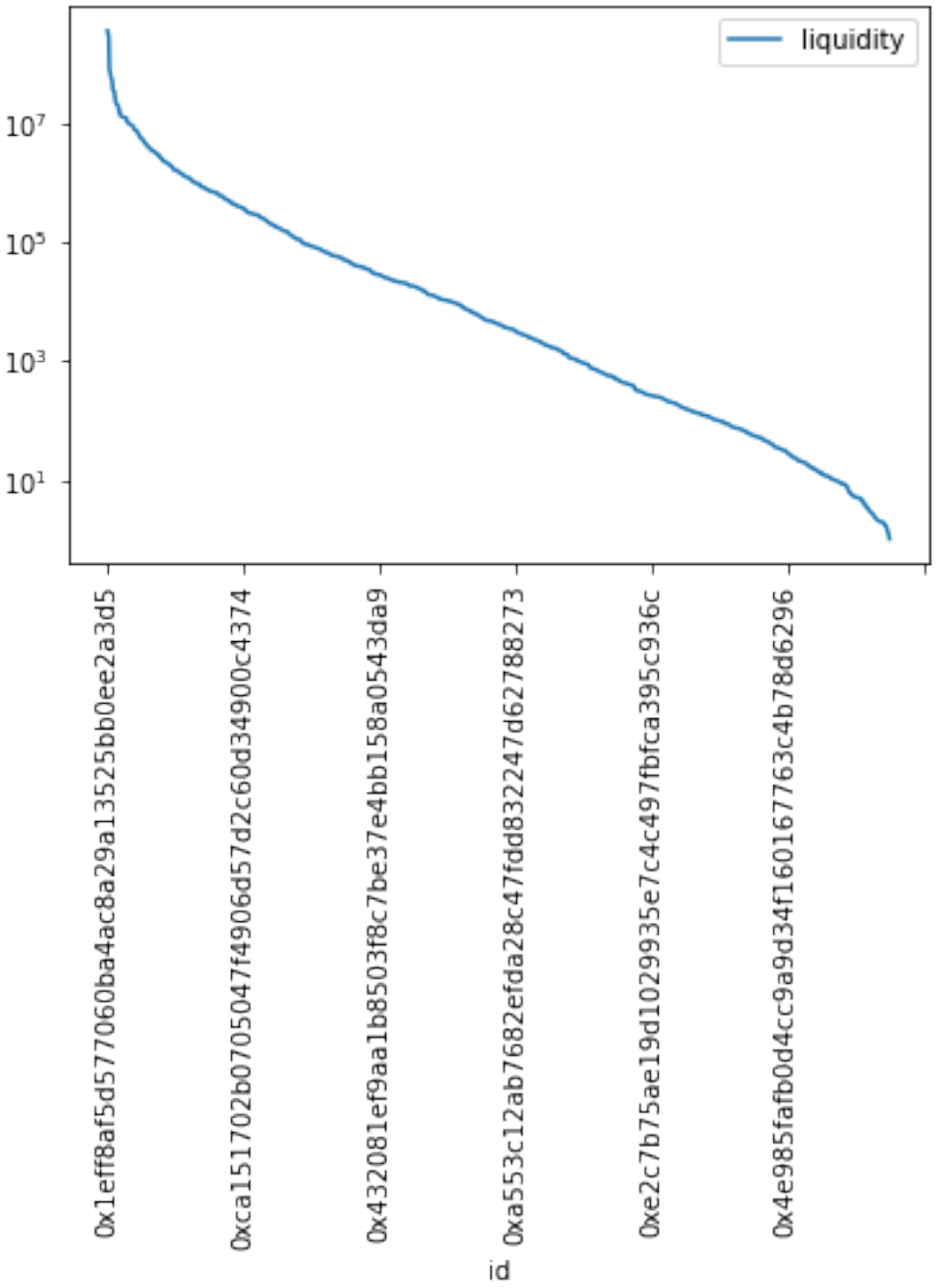
[16]:
```
plt_df.tail()
```

[16]:
```
                                            id  liquidity
340   0x1d261ec7ab834fedb01602c5b7ffc6fc68362bbf   1.577654
987   0x53f160490d7e48ba2c31be4790f3d87a2f4dc662   1.371422
1850  0x9e4a4b53e19410ae519be74f92659e5b0ef9489b   1.330313
2355  0xcb8ec8236aff8e112517f4e9a9ffb413a237e6b7   1.153105
```

```
1297  0x6d42692518c8b09c883e7c1e69c97518107f2185   1.030083
```

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

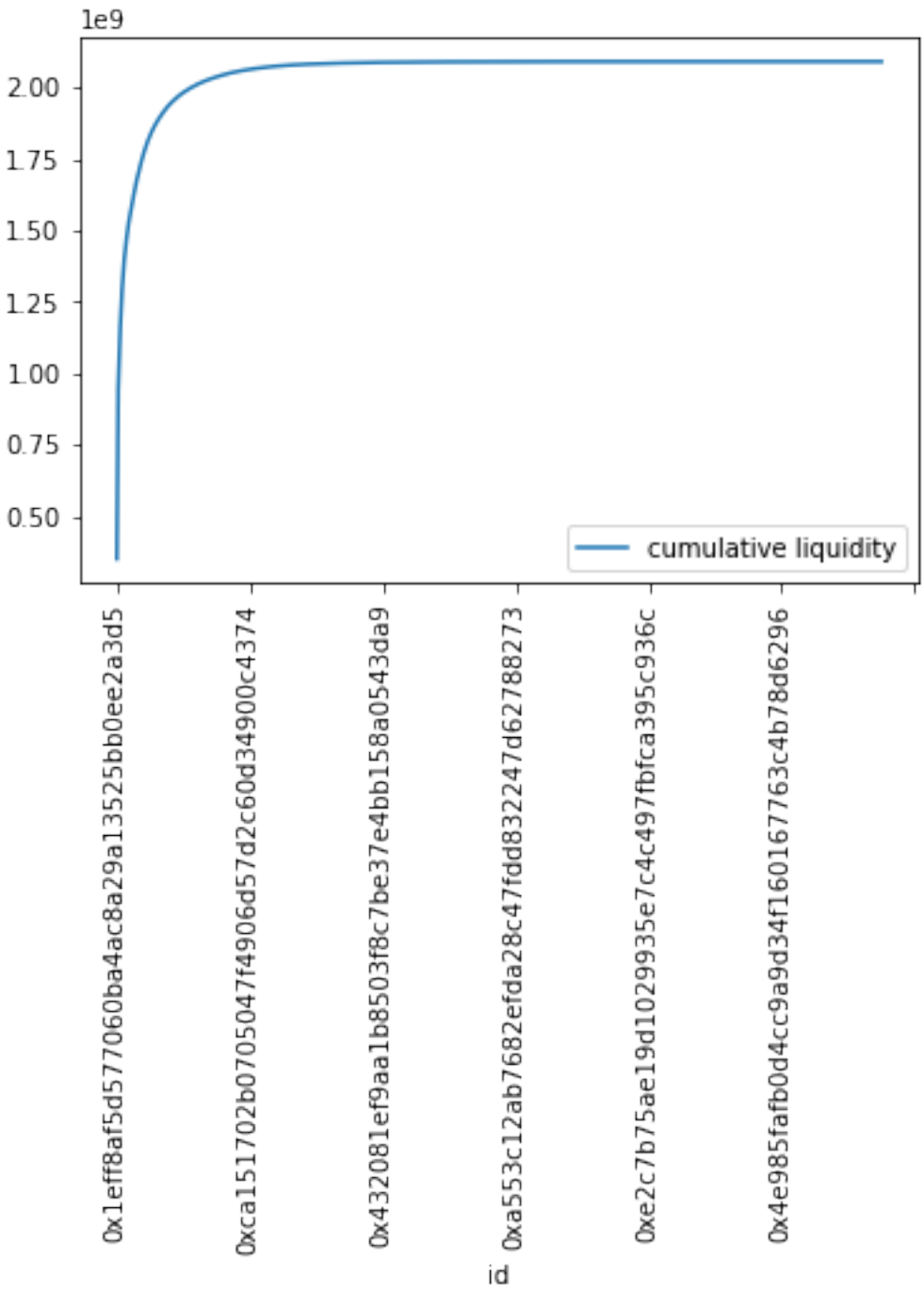
[17]:
```
(array([-200.,    0.,  200.,  400.,  600.,  800., 1000., 1200., 1400.]),
 [Text(-200.0, 0, '0x292329c7259771df82e5db9ab7eff3f53aeed450'),
  Text(0.0, 0, '0x1eff8af5d577060ba4ac8a29a13525bb0ee2a3d5'),
  Text(200.0, 0, '0xca151702b0705047f4906d57d2c60d34900c4374'),
  Text(400.0, 0, '0x432081ef9aa1b8503f8c7be37e4bb158a0543da9'),
  Text(600.0, 0, '0xa553c12ab7682efda28c47fdd832247d62788273'),
  Text(800.0, 0, '0xe2c7b75ae19d1029935e7c4c497fbfca395c936c'),
  Text(1000.0, 0, '0x4e985fafb0d4cc9a9d34f160167763c4b78d6296'),
  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, '0x292329c7259771df82e5db9ab7eff3f53aeed450'),
        Text(0.0, 0, '0x1eff8af5d577060ba4ac8a29a13525bb0ee2a3d5'),
        Text(200.0, 0, '0xca151702b0705047f4906d57d2c60d34900c4374'),
        Text(400.0, 0, '0x432081ef9aa1b8503f8c7be37e4bb158a0543da9'),
        Text(600.0, 0, '0xa553c12ab7682efda28c47fdd832247d62788273'),
        Text(800.0, 0, '0xe2c7b75ae19d1029935e7c4c497fbfca395c936c'),
        Text(1000.0, 0, '0x4e985fafb0d4cc9a9d34f160167763c4b78d6296'),
        Text(1200.0, 0, ''),
        Text(1400.0, 0, '')])
```


1e9
2.00
1.75
1.50
1.25
1.00
0.75
0.50
cumulative liquidity
0x1eff8af5d577060ba4ac8a29a13525bb0ee2a3d5
0xca151702b0705047f4906d57d2c60d34900c4374
0x432081ef9aa1b8503f8c7be37e4bb158a0543da9
0xa553c12ab7682efda28c47fdd832247d62788273
0xe2c7b75ae19d1029935e7c4c497fbfca395c936c
0x4e985fafb0d4cc9a9d34f160167763c4b78d6296
id

[ ]:

[ ]: