```
In [1]: import os
        import requests
        import json
        import itertools
```

```
In [2]: #Every request begins with the server's URL
        SERVER = 'http://data.neonscience.org/api/v0/'
```

```
In [4]: SITE_CODE = 'UNDE'
```

```
In [5]: # plant presence and percent cover
        PRODUCTCODE = 'DP1.10058.001'
```

```
In [8]: url = SERVER+'sites/'+SITE_CODE
```

```
In [9]: #Request the url
        site_request = requests.get(url)

        #Convert the request to Python JSON object
        site_json = site_request.json()
```

get latest available month, and print it

```
In [50]: months = []

         for product in site_json['data']['dataProducts']:
             #if a list item's 'dataProductCode' dict element equals the product code string
             if(product['dataProductCode'] == PRODUCTCODE):
                 #print the available months
                 for month in product['availableMonths']:
                     months.append(month)

         months.sort()

         months[-5:]
```

```
Out[50]: ['2021-06', '2021-07', '2022-06', '2022-07', '2022-08']
```

```
In [51]: month_idx = -1
         latest_month = months[-1]

         latest_month
```

```
Out[51]: '2022-08'
```

get list of files

```
In [52]: data_request = requests.get(SERVER+'data/'+PRODUCTCODE+'/'+SITE_CODE+'/'+latest_month)
         data_json = data_request.json()
```

```
In [53]: # filter through files and get the URL for the one we want (1m^2)
         url = None
         name = None

         for file in data_json['data']['files']:
             if '1m2' in file['name']:
                 for key in file.keys():
```

```
            print(key,':\t', file[key])

        url = file['url']
        name = file['name']

url
```

```
name :   NEON.D05.UNDE.DP1.10058.001.div_1m2Data.2022-08.expanded.20230410T163816Z.csv
size :   38694
md5 :    6371282b2c603abdafc108ed0a88be90
crc32 :  None
crc32c :         None
url :    https://storage.googleapis.com/neon-publication/NEON.DOM.SITE.DP1.10058.001/UND
E/20220801T000000--20220901T000000/expanded/NEON.D05.UNDE.DP1.10058.001.div_1m2Data.2022
-08.expanded.20230410T163816Z.csv
name :   NEON.D05.UNDE.DP1.10058.001.div_1m2Data.2022-08.basic.20230410T163816Z.csv
size :   38694
md5 :    6371282b2c603abdafc108ed0a88be90
crc32 :  None
crc32c :         None
url :    https://storage.googleapis.com/neon-publication/NEON.DOM.SITE.DP1.10058.001/UND
E/20220801T000000--20220901T000000/basic/NEON.D05.UNDE.DP1.10058.001.div_1m2Data.2022-0
8.basic.20230410T163816Z.csv
```

Out[53]:

```
'https://storage.googleapis.com/neon-publication/NEON.DOM.SITE.DP1.10058.001/UNDE/202208
01T000000--20220901T000000/basic/NEON.D05.UNDE.DP1.10058.001.div_1m2Data.2022-08.basic.2
0230410T163816Z.csv'
```

download data from URL

In [54]:

```
import urllib.request
urllib.request.urlretrieve(url, 'data/' + name)
```

Out[54]:

```
('data/NEON.D05.UNDE.DP1.10058.001.div_1m2Data.2022-08.basic.20230410T163816Z.csv',
 <http.client.HTTPMessage at 0x1f0f3375430>)
```

import CSV

In [55]:

```
import pandas as pd

df = pd.read_csv('data/' + name)

df
```

Out[55]:

	uid	namedLocation	domainID	siteID	decimalLatitude	decimalLongitude	geodeticDatum
0	4740265f-e549-4df5-b4dd-9d6510e1e8b1	UNDE_022.basePlot.div	D05	UNDE	46.230668	-89.569550	WGS84
1	6b3aee6d-99ef-44b0-ad9f-b7d0f9170394	UNDE_036.basePlot.div	D05	UNDE	46.253708	-89.516869	WGS84
2	459cd4d6-48cf-4236-b706-c3e1d8e0832b	UNDE_016.basePlot.div	D05	UNDE	46.245970	-89.525485	WGS84
3	96a4f65b-9063-49f6-81d5-84670e2d76e3	UNDE_014.basePlot.div	D05	UNDE	46.225960	-89.513299	WGS84

4	0fa628ac-1f63-4d77-8794-b78793ded145	UNDE_029.basePlot.div	D05	UNDE	46.251554	-89.516809	WGS84
...	...	...	...	...	...	...	...
103	4b52ed7d-476b-4915-b35a-9b377ed1faba	UNDE_018.basePlot.div	D05	UNDE	46.243048	-89.534710	WGS84
104	5c23661f-bbe3-42f5-9b9e-fc2e72a33dcb	UNDE_035.basePlot.div	D05	UNDE	46.218532	-89.507910	WGS84
105	c24e3f06-b5cd-4b0c-8ca3-56e35110ff14	UNDE_025.basePlot.div	D05	UNDE	46.234098	-89.573198	WGS84
106	89a29e93-64a5-44e2-8134-bff55038c32e	UNDE_022.basePlot.div	D05	UNDE	46.230668	-89.569550	WGS84
107	2e4889fc-3c2e-4a44-a0f2-ea37ec247e37	UNDE_023.basePlot.div	D05	UNDE	46.230008	-89.501807	WGS84

108 rows × 41 columns

check to make sure data for family and percent cover is not missing

In [56]:
```python
# try an earlier month
def redownload_data():
    global month_idx
    global df
    global latest_month

    month_idx -= 1
    latest_month = months[month_idx]
    latest_month = months[month_idx]
    data_request = requests.get(SERVER+'data/'+PRODUCTCODE+'/'+SITE_CODE+'/'+latest_mont
    data_json = data_request.json()

    url = None
    name = None

    for file in data_json['data']['files']:
        if '1m2' in file['name']:
            for key in file.keys():
                print(key,':\t', file[key])

            url = file['url']
            name = file['name']

    urllib.request.urlretrieve(url, 'data/' + name)

    df = pd.read_csv('data/' + name)
```

```python
# redownload earlier data if more than half of family/percentcover data is Nan
```

In [57]:
```
while True:
    num_rows = df.shape[0]
    num_nan = max(df['family'].isnull().sum(), df['percentCover'].isnull().sum())

    print(num_nan, num_rows)

    if 2 * num_nan > num_rows:
        # try an earlier month
        redownload_data()
    else:
        break
```

```
108 108
name :   NEON.D05.UNDE.DP1.10058.001.div_1m2Data.2022-07.expanded.20230313T204832Z.csv
size :   196288
md5 :    7c6f845216cbbfc99083a88b398c9258
crc32 :  None
crc32c :         None
url :    https://storage.googleapis.com/neon-publication/NEON.DOM.SITE.DP1.10058.001/UND
E/20220701T000000--20220801T000000/expanded/NEON.D05.UNDE.DP1.10058.001.div_1m2Data.2022
-07.expanded.20230313T204832Z.csv
name :   NEON.D05.UNDE.DP1.10058.001.div_1m2Data.2022-07.basic.20230313T204832Z.csv
size :   196288
md5 :    7c6f845216cbbfc99083a88b398c9258
crc32 :  None
crc32c :         None
url :    https://storage.googleapis.com/neon-publication/NEON.DOM.SITE.DP1.10058.001/UND
E/20220701T000000--20220801T000000/basic/NEON.D05.UNDE.DP1.10058.001.div_1m2Data.2022-0
7.basic.20230313T204832Z.csv
185 505
```

In [58]:
```
latest_month
```

Out[58]:
```
'2022-07'
```

get average percent cover for each family

In [71]:
```
import math

avg_tracker = dict()

def isnan(val):
    return type(val) == float and math.isnan(val)

for index, row in df.iterrows():
    if isnan(row['family']) or isnan(row['percentCover']):
        continue

    family = row['family']
    percCover = row['percentCover']

    if family not in avg_tracker:
        avg_tracker[family] = (0, 0)

    avg_tracker[family] = (avg_tracker[family][0] + percCover, avg_tracker[family][1] +

avgPercCover = []
sumAvgPercCover = 0

for key in avg_tracker:
    avgPercCover.append((key, avg_tracker[key][0] / avg_tracker[key][1]))
    sumAvgPercCover += avgPercCover[-1][1]

avgPercCover
```

```
Out[71]: [('Caprifoliaceae', 4.428571428571429),
          ('Rosaceae', 1.28125),
          ('Liliaceae', 1.7647058823529411),
          ('Pyrolaceae', 1.1666666666666667),
          ('Ranunculaceae', 4.75),
          ('Oleaceae', 1.6875),
          ('Grossulariaceae', 8.0),
          ('Primulaceae', 1.5),
          ('Betulaceae', 31.333333333333332),
          ('Pinaceae', 3.625),
          ('Onagraceae', 0.6111111111111112),
          ('Orchidaceae', 0.5),
          ('Asteraceae', 1.9444444444444444),
          ('Salicaceae', 0.5),
          ('Aceraceae', 2.7941176470588234),
          ('Cyperaceae', 2.660377358490566),
          ('Cupressaceae', 0.5),
          ('Clusiaceae', 1.0),
          ('Monotropaceae', 0.5),
          ('Cornaceae', 4.625),
          ('Polygonaceae', 1.5),
          ('Osmundaceae', 34.5),
          ('Poaceae', 2.25),
          ('Rubiaceae', 0.8),
          ('Lamiaceae', 4.25),
          ('Violaceae', 2.0),
          ('Dryopteridaceae', 12.5),
          ('Thelypteridaceae', 13.6),
          ('Dennstaedtiaceae', 13.0),
          ('Scrophulariaceae', 0.5),
          ('Thymelaeaceae', 8.0),
          ('Oxalidaceae', 5.75),
          ('Araceae', 5.583333333333333),
          ('Lycopodiaceae', 3.0),
          ('Iridaceae', 0.5),
          ('Equisetaceae', 3.0),
          ('Droseraceae', 0.5),
          ('Ericaceae', 1.75),
          ('Sparganiaceae', 8.0),
          ('Apiaceae', 0.5),
          ('Saxifragaceae', 1.0),
          ('Balsaminaceae', 6.0),
          ('Brassicaceae', 0.5)]
```

```
In [72]: sumAvgPercCover
```

```
Out[72]: 204.15541120536267
```

```
In [ ]:
```