GEDI Data Access on S3 using Python

Yu-Feng Ho

# Background

This notebook is a demonstration of retrieving [GEDI](https://gedi.umd.edu/) Level 2A & 2B data from cloud storage.

Global Ecosystem Dynamics Investigation (GEDI) is a full-waveform satellite [LiDAR (Light Detection And Ranging)](https://en.wikipedia.org/wiki/Lidar), collecting 3D measurements over land (trough recorded backscattered laser energy, i.e., waveforms) near-globally, between 51.6° N and S within a laser footprint diameter of 25 m between 2019-03-25 and 2023-03-15. To foster the uptake and mask out the complexity and large data volume of GEDI products (GEDI L2, version 2,- V002) to the OEMC and broader users, we aim to host pre-filtered and high-quality GEDI points (including their relative heights, and waveform- and derived-attributes) and offer them as a cloud service. To handle such data we use (the R bindings of) cloud-compatible software tools [Apache Arrow](https://arrow.apache.org/) and [Polars](https://docs.pola.rs/).

GEDI is organized as tracks, shot, granules. The single file usually crosses a huge area but covers loose information for a certain region. The information covers are too coarse to be analyzed in only a single track. In reality, GEDI attaches to the international Space Station (ISS) that varies its temporal resolution and is location dependent. Not only the dataset structure optimizing for tracing recording makes it extremely difficult to retrieve demanding data, but also the irregular temporal resolution in the same location thwarts the confidence for researchers to conduct further analyses.



Therefore, we pre-processed and re-chunked all available data from the start of the mission in April 2019 until the end of 2023 (with frequent updates planned). The dataset is currently stored on a Wasabi Simple Storage Service (S3) bucket, administered by [OpenGeoHub](https://opengeohub.org/) and has ~5.4 billion observations (~1TB). It is organized as a partitioned [Parquet](https://parquet.apache.org/) dataset, which consists of one file per 5x5 degree tile and year, approximately ~2000 partitions in total. Parquet is a column-oriented file format with efficient storage and high compression properties. Read more about [Parquet in Geospatial Data](https://medium.com/radiant-earth-insights/the-geoparquet-ecosystem-at-1-0-0-96dee8ce9201)

from globalearthpoint import gedil2, mapview # objects and functions for GEDI retrieval   
import polars as pl # cloud lazy loading pacakge  
from pyarrow.dataset import dataset # parquet format handler  
from shapely import Polygon # self-defined geometry   
import pandas as pd # work with dataframe  
import numpy as np # numpy array opt

mapview(lookup\_file='lookup.fgb')

<leafmap.foliumap.Map at 0x7f0353f2b760>

# Spatial temporal object construction

### gedil2: GEDI dataset class

* geometry (shapely.geometry): a given geometry to subset GEDI data to the area of interest
* year (list): integer from 2019 - 2023. Examples: years=[2019,2023], retrieving partition with year=2019 & year=2023

# define object with geometry and year  
obj = gedil2(geometry=Polygon([(17, 6), (18, 6),(18, 7), (17, 7)]),years=[2021,2022,2023])

# Query

tile\_query(): construct queries from GEDI dataset partition tile, narrowing the data retrieval to minimal partition levels.

* Pros: fast query and download
* Cons: larger amount of redundant data if querying small area

tile\_queries = obj.tile\_query()  
tile\_queries

|  | lon | lat | year | dir | n\_points | geometry |
| --- | --- | --- | --- | --- | --- | --- |
| 2091 | 15.0 | 5.0 | 2023.0 | /lon=15/lat=5/year=2023/gedi\_l2\_0.parquet | 491786 | POLYGON ((15.00000 5.00000, 20.00000 5.00000, ... |
| 2092 | 15.0 | 5.0 | 2022.0 | /lon=15/lat=5/year=2022/gedi\_l2\_0.parquet | 1896325 | POLYGON ((15.00000 5.00000, 20.00000 5.00000, ... |
| 2093 | 15.0 | 5.0 | 2021.0 | /lon=15/lat=5/year=2021/gedi\_l2\_0.parquet | 1561074 | POLYGON ((15.00000 5.00000, 20.00000 5.00000, ... |

bbox\_queries = obj.bbox\_query()  
bbox\_queries

{'GEDI\_tile\_subset\_lon=15\_lat=5\_year=2023\_gedi\_l2\_0.parquet': <LazyFrame at 0x7F0345891A90>,  
 'GEDI\_tile\_subset\_lon=15\_lat=5\_year=2022\_gedi\_l2\_0.parquet': <LazyFrame at 0x7F03458918B0>,  
 'GEDI\_tile\_subset\_lon=15\_lat=5\_year=2021\_gedi\_l2\_0.parquet': <LazyFrame at 0x7F03458918E0>}

# Download

* download\_gedi: input of queries and download the data as parquet files in **~/GEDI\_download**

obj.download\_gedi(bbox\_queries,cores=3)

Downloading bbox subsets...

Downloading subset: 100%|██████████| 3/3 [06:16<00:00, 125.42s/it]

Completed after 376.25 sec.

obj.download\_gedi(tile\_queries,cores=3)

Downloading tiles...

Downloading tiles: 100%|██████████| 3/3 [10:43<00:00, 214.61s/it]

Completed after 643.85 sec.

# Read

read downloaded parquet files from **~/GEDI\_download** by pyarrow.dataset.dataset.

pyarrow\_dataset = dataset(  
 source = 'GEDI\_download',  
 format = 'parquet'  
)

gedi\_ds = pl.scan\_pyarrow\_dataset(  
 pyarrow\_dataset,  
)

gedi\_df = gedi\_ds.collect().to\_pandas()

# Rescale

Finally, we rescale GEDI data to the original value, applying the function rescale(x, scl) to get real value for each columns

def \_rescale(n, x, scl):  
 fac = scl[scl['Variable'] == n]  
 if not fac.empty and not pd.isna(fac.iloc[0]['NoData']):  
 x[n] = x[n].replace(fac.iloc[0]['NoData'], np.nan)  
 if not fac.empty and fac.iloc[0]['Scale'] < 1:  
 x[n] = x[n] \* fac.iloc[0]['Scale']  
 return x[n]  
  
def rescale\_gedi(x, scl):  
 scl = scl[scl['Variable'].isin(x.columns)]  
 for n in x.columns:  
 x[n] = \_rescale(n, x, scl)  
 return x

scl = pd.read\_csv('gedi\_columns.csv')  
rescale\_gedi(gedi\_df,scl)

|  | delta\_time | beamname | shotnumber | latitude | longitude | elev\_lowestmode | rh100 | rh99 | rh98 | rh97 | ... | selected\_rg\_algorithm | rhov | selected\_l2a\_algorithm | fhd\_normal | surface\_flag | leaf\_off\_flag | l2b\_quality\_flag | lon | lat | year |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 102973632 | 4 | 131210500300340288 | 6.999335 | 17.947599 | 453.82 | 1.5328 | 1.0827 | 0.7828 | 0.5527 | ... | 1 | 0.6 | 2 | 2.57 | True | True | True | 15 | 5 | 2021 |
| 1 | 102973632 | 4 | 131210500300340288 | 6.997658 | 17.948807 | 449.68 | 0.3291 | -0.0408 | -0.3008 | -0.5608 | ... | 1 | 0.6 | 2 | 2.99 | True | True | True | 15 | 5 | 2021 |
| 2 | 102973634 | 4 | 131210500300340432 | 6.935666 | 17.993559 | 449.64 | 2.8427 | 2.5427 | 2.2828 | 2.0527 | ... | 1 | 0.6 | 2 | 2.78 | True | True | True | 15 | 5 | 2021 |
| 3 | 102973633 | 5 | 131210600300632416 | 6.976102 | 17.957745 | 443.54 | -1.6573 | -2.0972 | -2.4372 | -2.7773 | ... | 1 | 0.6 | 2 | 2.39 | True | True | True | 15 | 5 | 2021 |
| 4 | 102973633 | 5 | 131210600300632432 | 6.974008 | 17.959260 | 446.95 | -2.4045 | -2.9244 | -3.2644 | 3.0292 | ... | 1 | 0.6 | 2 | 2.98 | True | True | True | 15 | 5 | 2021 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 158156 | 163322614 | 6 | 239510800300331552 | 6.943298 | 17.970978 | 437.09 | -1.3937 | -1.8036 | -2.1037 | -2.3636 | ... | 1 | 0.6 | 2 | 1.55 | True | True | True | 15 | 5 | 2023 |
| 158157 | 163322614 | 6 | 239510800300331584 | 6.933223 | 17.978221 | 459.47 | 2.9427 | 2.5728 | 2.2328 | 1.8228 | ... | 1 | 0.6 | 2 | 2.71 | True | True | True | 15 | 5 | 2023 |
| 158158 | 163322615 | 6 | 239510800300331584 | 6.922747 | 17.985766 | 463.89 | -2.6009 | -2.9309 | -3.2308 | 3.0227 | ... | 1 | 0.6 | 2 | 2.76 | True | True | True | 15 | 5 | 2023 |
| 158159 | 163322615 | 6 | 239510800300331616 | 6.920234 | 17.987574 | 459.09 | 2.7964 | 2.3463 | 2.0163 | 1.6763 | ... | 1 | 0.6 | 2 | 2.23 | True | True | True | 15 | 5 | 2023 |
| 158160 | 163322615 | 6 | 239510800300331648 | 6.904733 | 17.998745 | 453.99 | -2.3372 | -2.5572 | -2.7073 | -2.8972 | ... | 1 | 0.6 | 2 | 2.37 | True | True | True | 15 | 5 | 2023 |