```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
Mounted at /content/drive
```

```
!ls drive/MyDrive/Bathy/Survey_Data/
```

```
Bakreshwar   Getalsud_full  Kandala_creek      Kolkata                   Rihand_Dam
Dhurwa_full  Hirakud        Kandla_creek       Kolkata_port_sea_channel  Sholayar
Dibang_Dam   Hirakud_dam    Khadakwasala       Mahi_Bajaj_Sagar_Dam      Tenughat
Getalsud     Kadana_dam     Khamaria_Jabalpur  Rihand                    Tenughat_full
```

```
!ls drive/MyDrive/Bathy/Survey_Data/Bakreshwar/7_nov_to_14_nov_depth_1.xyz
# !ls drive/MyDrive/Bathy/Survey_Data/Bakreshwar
```

```
drive/MyDrive/Bathy/Survey_Data/Bakreshwar/7_nov_to_14_nov_depth_1.xyz
```

```
import pandas as pd
df = pd.read_csv('drive/MyDrive/Bathy/Survey_Data/Bakreshwar/7_nov_to_14_nov_depth_1.xyz', header=None)
df
```

```
#!ls drive/MyDrive/Bathy/Survey_Data/Rihand
```

```
ip_path = '/content/drive/MyDrive/Dam_data/'
dams = ['Tenughat', 'Getalsud', 'Mahi river', 'Kadana', 'Bakreshwar', 'Rihand']
start_dates = ['2021-09-02', '2021-09-02', '2019-12-17', '2020-10-28', '2022-03-02', '2021-10-20']
end_dates = ['2021-09-20', '2021-09-27', '2020-01-31', '2021-01-03', '2022-09-20', '2021-10-30']
```

```
# INSTALLING REQUIRED LIBRARIES
!pip install -U pyproj -q
!pip install -U utm -q
```

```
  Preparing metadata (setup.py) ... done
  Building wheel for utm (setup.py) ... done
```

```
!pip install -U google-colab -q
!pip install -U tornado -q
!pip install geemap -q
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 17.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 437.2/437.2 kB 7.4 MB/s eta 0:00:00
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is
google-colab 1.0.0 requires tornado==6.3.3, but you have tornado 6.4.2 which is incompatible.
```

```
!pip install wxee
```

```
Collecting wxee
  Downloading wxee-0.4.2-py3-none-any.whl.metadata (6.8 kB)
Requirement already satisfied: earthengine-api in /usr/local/lib/python3.10/dist-packages (from wxee) (1.4.3)
```

```
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from wxee) (1.4.2)
Collecting rasterio (from wxee)
  Downloading rasterio-1.4.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (9.1 kB)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from wxee) (2.32.3)
Collecting rioxarray (from wxee)
  Downloading rioxarray-0.18.1-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from wxee) (4.67.1)
Requirement already satisfied: xarray in /usr/local/lib/python3.10/dist-packages (from wxee) (2024.11.0)
Requirement already satisfied: google-cloud-storage in /usr/local/lib/python3.10/dist-packages (from earthengine-api->wxee)
Requirement already satisfied: google-api-python-client>=1.12.1 in /usr/local/lib/python3.10/dist-packages (from earthengine
Requirement already satisfied: google-auth>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from earthengine-api->wxee) (2
Requirement already satisfied: google-auth-httplib2>=0.0.3 in /usr/local/lib/python3.10/dist-packages (from earthengine-api-
Requirement already satisfied: httplib2<1dev,>=0.9.2 in /usr/local/lib/python3.10/dist-packages (from earthengine-api->wxee)
Collecting affine (from rasterio->wxee)
  Downloading affine-2.4.0-py3-none-any.whl.metadata (4.0 kB)
Requirement already satisfied: attrs in /usr/local/lib/python3.10/dist-packages (from rasterio->wxee) (24.3.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from rasterio->wxee) (2024.12.14)
Requirement already satisfied: click>=4.0 in /usr/local/lib/python3.10/dist-packages (from rasterio->wxee) (8.1.7)
Collecting cligj>=0.5 (from rasterio->wxee)
  Downloading cligj-0.7.2-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: numpy>=1.24 in /usr/local/lib/python3.10/dist-packages (from rasterio->wxee) (1.26.4)
Collecting click-plugins (from rasterio->wxee)
  Downloading click_plugins-1.1.1-py2.py3-none-any.whl.metadata (6.4 kB)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.10/dist-packages (from rasterio->wxee) (3.2.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->wxee) (3.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->wxee) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->wxee) (2.2.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from rioxarray->wxee) (24.2)
Requirement already satisfied: pyproj>=3.3 in /usr/local/lib/python3.10/dist-packages (from rioxarray->wxee) (3.7.0)
Requirement already satisfied: pandas>=2.1 in /usr/local/lib/python3.10/dist-packages (from xarray->wxee) (2.2.2)
Requirement already satisfied: google-api-core!=2.0.*,!=2.1.*,!=2.2.*,!=2.3.0,<3.0.0.dev0,>=1.31.5 in /usr/local/lib/python3
Requirement already satisfied: uritemplate<5,>=3.0.1 in /usr/local/lib/python3.10/dist-packages (from google-api-python-clie
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth>=1.4.1->e
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth>=1.4.1->ea
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth>=1.4.1->earthengin
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=2.1->xarray->
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=2.1->xarray->wxee) (202
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=2.1->xarray->wxee) (2
Requirement already satisfied: google-cloud-core<3.0dev,>=2.3.0 in /usr/local/lib/python3.10/dist-packages (from google-clou
Requirement already satisfied: google-resumable-media>=2.7.2 in /usr/local/lib/python3.10/dist-packages (from google-cloud-s
Requirement already satisfied: google-crc32c<2.0dev,>=1.0 in /usr/local/lib/python3.10/dist-packages (from google-cloud-stor
Requirement already satisfied: googleapis-common-protos<2.0.dev0,>=1.56.2 in /usr/local/lib/python3.10/dist-packages (from g
Requirement already satisfied: protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0.dev0,>
Requirement already satisfied: proto-plus<2.0.0dev,>=1.22.3 in /usr/local/lib/python3.10/dist-packages (from google-api-core
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas>=2.1
Downloading wxee-0.4.2-py3-none-any.whl (26 kB)
Downloading rasterio-1.4.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (22.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 22.2/22.2 MB 89.3 MB/s eta 0:00:00
Downloading rioxarray-0.18.1-py3-none-any.whl (61 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.9/61.9 kB 6.0 MB/s eta 0:00:00
Downloading cligj-0.7.2-py3-none-any.whl (7.1 kB)
Downloading affine-2.4.0-py3-none-any.whl (15 kB)
Downloading click_plugins-1.1.1-py2.py3-none-any.whl (7.5 kB)
```

```
!pip install plotly --upgrade
```

```
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (5.24.1)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly) (9.0.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from plotly) (24.2)
```

```
!pip install -U kaleido
```

```
Collecting kaleido
  Downloading kaleido-0.2.1-py2.py3-none-manylinux1_x86_64.whl.metadata (15 kB)
Downloading kaleido-0.2.1-py2.py3-none-manylinux1_x86_64.whl (79.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 79.9/79.9 MB 27.8 MB/s eta 0:00:00
Installing collected packages: kaleido
Successfully installed kaleido-0.2.1
```

```
# !earthengine authenticate
import ee
import os

# Set the path to the service account key file
service_account = 'editor@ee-manmeet20singh15-wbis.iam.gserviceaccount.com'
key_file = 'drive/MyDrive/earth_engine/ee-manmeet20singh15-wbis-fab7f1ca35e0.json'

# Use the service account for authentication
credentials = ee.ServiceAccountCredentials(service_account, key_file)
ee.Initialize(credentials)
```

```
import ee
import wxee
#wxee.Initialize()
```

```
from geopy.geocoders import Nominatim
from pyproj import CRS
import utm
#import proj
import pandas as pd
import numpy as np
import plotly.io as pio
```

```
def get_epsg(dam_name, country):
  geolocator = Nominatim(user_agent='my_user_agent')
  # get location in lat lon
  loc = geolocator.geocode(dam_name  + ',' + country)
  # get utm zone and utm letter from lat lon
  x, y, utm_zone, letter = utm.from_latlon(loc.latitude, loc.longitude)
  # predefined utm letters
  # https://www.maptools.com/tutorials/grid_zone_details#:~:text=Each%20zone%20is%20divided%20into,spans%2012%C2%B0%20of%20la
  nothern_letters = ('N','P','Q','R','S','T','U','V','W','X')
  southern_letters = ('M','L','K','J','H','G','F','E','D','C')
  # check hemisphere using utm letter
  if letter in nothern_letters:
    hemisphere = 'nothern'
    north_south = True
  else:
    hemisphere = 'southern'
    north_south = False
  # get crs info using utm zone and hemisphere
  crs = CRS.from_dict({'proj': 'utm', 'zone': utm_zone, north_south: True})
  # formatting epsg code into required fromat
  epsg = crs.to_authority()[0] + ':' + crs.to_authority()[1]
  return epsg, hemisphere, north_south
```

```
dam = dams[0]
i_dam = 0

#epsg, hemisphere, north_south = get_epsg(dam_name=dam, country='India')
start_date, end_date = start_dates[i_dam], end_dates[i_dam]
```

```
# start_date, end_date = '2022-04-01', '2022-04-30'
# df = pd.DataFrame()
# for i in range(1,7):
#     df_ = pd.read_csv('drive/MyDrive/Bathy/Survey_Data/Rihand_Dam/RIHAND_RESERVOIR_DATA_SOFTA_GEOTECHNICAL_PVT_LTD/Rihand_d
#     df = pd.concat([df, df_])
df = pd.read_csv('drive/MyDrive/Bathy/Survey_Data/Bakreshwar/7_nov_to_14_nov_depth_1.xyz', header=None)
```

```
#print(get_epsg(dam_name='Tenughat', country='India'))
```

```
epsg, hemisphere, north_south = '32645', 'northern', True
#https://epsg.io/32644
```

```
#epsg, hemisphere, north_south = get_epsg(dam_name='Bakreshwar', country='India')
```

```
zone = int(epsg[-2:])
```

```
zone
```

```
45
```

```
df
```

```
from tqdm import tqdm

df_ = df.copy()
df_[[0, 1, 2]] = df_[0].str.split(' ', expand=True)
print(pd.isnull(df_.iloc[0,2]))
for i in tqdm(range(df_.shape[0])):
    #print(pd.isnull(df_.iloc[i,2]))
    if len(df_.iloc[i,2])==0:
        df_.iloc[i,2] = df_.iloc[i,3]
df_
```

```
# df_.drop(df_.columns[3], axis=1, inplace=True)
# df_
```

```
df = df_.copy()
df = df.apply(pd.to_numeric, errors='coerce')
df.iloc[0,2]
```

```
-75.23
```

```
x_coord = df.values[:,0]
y_coord = df.values[:,1]
```

```
latitude, longitude = utm.to_latlon(x_coord, y_coord, zone, northern=north_south)
```

```
df['lat'] = latitude
df['lon'] = longitude
```

```
df['bathy'] = -1*(np.min(df.iloc[:,2].values) - df.iloc[:,2].values)
```

```
df_survey = df.copy()
df_survey
```

```
print(np.min(df.lat.values), np.max(df.lat.values))
print(np.min(df.lon.values), np.max(df.lon.values))
print((np.max(df.lat.values) - np.min(df.lat.values))/0.0006)
print((np.max(df.lon.values) - np.min(df.lon.values))/0.0006)
print(np.max(df.bathy.values))
```

```
23.820442018891118 23.85890792375116
87.38651833765994 87.42203410386985
64.10984143340552
59.19294368318618
23.089999999999996
```

```
import plotly.express as px
# df = px.data.carshare()
fig = px.scatter_mapbox(df_survey, lat="lat", lon="lon", color="bathy",
                  color_continuous_scale=px.colors.cyclical.IceFire, size_max=5, zoom=12,
                  mapbox_style="carto-positron")
pio.write_image(fig, 'figure.png', scale=5)
fig.show()
```

```
!ls
!mv figure.png drive/MyDrive/Bathy/Survey_Data/Bakreshwar
```

```
drive  figure.png  sample_data
```

```
# import plotly.express as px
# fig = px.scatter_mapbox(df, lat="lat", lon="lon", color="bathy",
#                   color_continuous_scale=px.colors.cyclical.IceFire, size_max=5, zoom=12,
#                   mapbox_style="carto-positron")
# #pio.write_image(fig, 'test.png')
# fig.write_image("test.png", engine="kaleido")
```

```
from scipy import stats
```

```
lon_diff = np.max(df.lon.values) - np.min(df.lon.values)
lat_diff = np.max(df.lat.values) - np.min(df.lat.values)
lat_diff = 0
lon_diff = 0
aoi = ee.Geometry.Polygon(
        [[[np.min(df.lon.values) - lon_diff, np.min(df.lat.values) - lat_diff],
          [np.max(df.lon.values) + lon_diff, np.min(df.lat.values) - lat_diff],
          [np.max(df.lon.values) + lon_diff, np.max(df.lat.values) + lat_diff],
          [np.min(df.lon.values) + lon_diff, np.max(df.lat.values) + lat_diff]]])
# aoi = ee.Geometry.Polygon(
#         [[[np.min(df.lon.values) - lon_diff, np.min(df.lat.values) - lat_diff],
#           [np.max(df.lon.values) , np.min(df.lat.values) - lat_diff],
#           [np.max(df.lon.values) , np.max(df.lat.values) ],
#           [np.min(df.lon.values) - lon_diff, np.max(df.lat.values) ]]])
```

```
coords = aoi.coordinates().getInfo()[0]
coords
```

```
[[87.38651833765994, 23.820442018891118],
 [87.42203410386985, 23.820442018891118],
 [87.42203410386985, 23.85890792375116],
 [87.38651833765994, 23.85890792375116],
 [87.38651833765994, 23.820442018891118]]
```

Sentinel 2A

```
def maskS2clouds(image):
    qa = image.select('QA60')
```

```
    cloudBitMask = 1 << 10
    cirrusBitMask = 1 << 11
    mask = qa.bitwiseAnd(cloudBitMask).eq(0) and (qa.bitwiseAnd(cirrusBitMask).eq(0))
    return image.updateMask(mask).divide(10000)
```

```
start_date = '2022-11-01'
end_date = '2022-11-30'
dataset = ee.ImageCollection('COPERNICUS/S2_SR_HARMONIZED') \
            .filterDate(start_date, end_date) \
            .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE',20)) \
            .map(maskS2clouds).filterBounds(aoi) \
            .mean()
```

```
dataset = dataset.set('system:time_start', 0)
ds_sentinel = dataset.wx.to_xarray(region=aoi.bounds(), scale=30)#,  crs='EPSG:32645')
```

```
ds_sentinel.B8.plot()
```

```
df_B1_sentinel_df = ds_sentinel.isel(time=0).B1.to_dataframe().reset_index()
```

```
df_B1_sentinel_df
```

Next steps: Generate code with df_B1_sentinel_df | View recommended plots | New interactive sheet

```
# import plotly.express as px
# # df = px.data.carshare()
# fig = px.scatter_mapbox(df_B1_sentinel_df, lat="y", lon="x", color="B1",
#                   color_continuous_scale=px.colors.cyclical.IceFire, size_max=5, zoom=12,
#                   mapbox_style="carto-positron")
# #fig.write_image("sentinel_b1.png", engine="kaleido")
# fig.show()
```

Landsat-8

```
def applyScaleFactors(image):
    opticalBands = image.select('SR_B.').multiply(0.0000275).add(-0.2)
    thermalBands = image.select('ST_B.*').multiply(0.00341802).add(149.0)
    return image.addBands(opticalBands, None, True).addBands(thermalBands, None, True)
```

```
start_date = '2022-11-01'
end_date = '2022-11-30'
dataset = ee.ImageCollection('LANDSAT/LC08/C02/T1_L2').filterDate(start_date, end_date).filterBounds(aoi)
dataset = dataset.map(applyScaleFactors)
dataset = dataset.set('system:time_start', 0)
ds_landsat = dataset.wx.to_xarray(region=aoi.bounds(), scale=30)#,  crs='EPSG:32645')
```

```
ds_landsat.SR_B5.isel(time=0).plot(cmap='coolwarm')
```

Icesat-2

```
!pip install --upgrade icepyx
```

```
Requirement already satisfied: multipledispatch in /usr/local/lib/python3.10/dist-packages (from panel>=1.0->holoviews->icepy
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from panel>=1.0->holoviews->icepyx) (4.67.1)
Requirement already satisfied: locket in /usr/local/lib/python3.10/dist-packages (from partd>=1.4.0->dask[dataframe]->icepyx
Collecting bounded-pool-executor (from pqdm>=0.1->earthaccess>=0.5.1->icepyx)
  Downloading bounded_pool_executor-0.0.3-py3-none-any.whl.metadata (2.7 kB)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib->i
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.10/dist-packages (from numba->datashader
Collecting jmespath<2.0.0,>=0.7.1 (from botocore<1.35.82,>=1.35.74->aiobotocore<3.0.0,>=2.5.4->s3fs->icepyx)
  Downloading jmespath-1.0.1-py3-none-any.whl.metadata (7.6 kB)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2>=2.9->bokeh>=3.1->hol
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->panel>=1.0->holoviews->
Requirement already satisfied: uc-micro-py in /usr/local/lib/python3.10/dist-packages (from linkify-it-py->panel>=1.0->holov
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py->panel>=1.0->holov
Downloading icepyx-1.3.0-py3-none-any.whl (77 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 77.1/77.1 kB 6.2 MB/s eta 0:00:00
Downloading earthaccess-0.12.0-py3-none-any.whl (60 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 60.5/60.5 kB 5.9 MB/s eta 0:00:00
Downloading s3fs-2024.12.0-py3-none-any.whl (30 kB)
Downloading fsspec-2024.12.0-py3-none-any.whl (183 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 183.9/183.9 kB 8.1 MB/s eta 0:00:00
Downloading backoff-2.2.1-py3-none-any.whl (15 kB)
Downloading datashader-0.16.3-py2.py3-none-any.whl (18.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.3/18.3 MB 6.9 MB/s eta 0:00:00
Downloading fiona-1.10.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (17.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.3/17.3 MB 101.0 MB/s eta 0:00:00
Downloading hvplot-0.11.2-py3-none-any.whl (161 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 161.9/161.9 kB 15.3 MB/s eta 0:00:00
Downloading aiobotocore-2.16.0-py3-none-any.whl (77 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 77.8/77.8 kB 7.1 MB/s eta 0:00:00
Downloading dask_expr-1.1.16-py3-none-any.whl (243 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 243.2/243.2 kB 23.2 MB/s eta 0:00:00
Downloading multimethod-1.12-py3-none-any.whl (10 kB)
Downloading pqdm-0.2.0-py2.py3-none-any.whl (6.8 kB)
Downloading python_cmr-0.13.0-py3-none-any.whl (14 kB)
Downloading tinynetrc-1.3.1-py2.py3-none-any.whl (3.9 kB)
Downloading pyct-0.5.0-py2.py3-none-any.whl (15 kB)
Downloading aioitertools-0.12.0-py3-none-any.whl (24 kB)
Downloading botocore-1.35.81-py3-none-any.whl (13.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.3/13.3 MB 90.1 MB/s eta 0:00:00
Downloading bounded_pool_executor-0.0.3-py3-none-any.whl (3.4 kB)
Downloading jmespath-1.0.1-py3-none-any.whl (20 kB)
Installing collected packages: tinynetrc, bounded-pool-executor, pyct, pqdm, multimethod, jmespath, fsspec, backoff, aioiter
  Attempting uninstall: fsspec
    Found existing installation: fsspec 2024.10.0
    Uninstalling fsspec-2024.10.0:
      Successfully uninstalled fsspec-2024.10.0
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is
gcsfs 2024.10.0 requires fsspec==2024.10.0, but you have fsspec 2024.12.0 which is incompatible.
Successfully installed aiobotocore-2.16.0 aioitertools-0.12.0 backoff-2.2.1 botocore-1.35.81 bounded-pool-executor-0.0.3 das
```

```
import icepyx as ipx
import numpy as np
!touch /root/.netrc
!echo "machine urs.earthdata.nasa.gov login ayantika03 password Climate2010" > /root/.netrc
!chmod 0600 /root/.netrc

earthdata_uid = 'ayantika03'
earthdata_email = 'ayantika.dey@gmail.com'

short_name = 'ATL13'

#spatial_extent = [-55, 68, -48, 71]
spatial_extent = [85.76922504150433, 23.70632937798437, 85.83765628246628, 23.748408865450276]

# Given as longitude, latitude coordinate pairs
date_range = ['2022-11-01','2022-12-30']

region_a = ipx.Query(short_name, spatial_extent, date_range)

# # search for available granules using icepyx
# region_a.earthdata_login(earthdata_uid, earthdata_email)

path = './download'
!rm -rf ./download/*
region_a.download_granules(path)
```

```
Total number of data order requests is  1  for  2  granules.
Data request  1  of  1  is submitting to NSIDC
Enter your Earthdata Login username: manmeet.singh@utexas.edu
Enter your Earthdata password: ··········
order ID:  5000005890543
```

```
Initial status of your order request at NSIDC is:  processing
Your order status is still  processing  at NSIDC. Please continue waiting... this may take a few moments.
Your order is: complete
NSIDC returned these messages
['Granule 296533479 contained no data within the spatial and/or temporal '
 'subset constraints to be processed',
 'Granule 296533486 contained no data within the spatial and/or temporal '
 'subset constraints to be processed']
Beginning download of zipped output...
Unable to download  5000005890543 . Check granule order for messages.
Download complete
```

```
!ls ./download
```

```
ls: cannot access './download': No such file or directory
```

```
path = './download/'
pattern = "processed_ATL{product:2}_{datetime:%Y%m%d%H%M%S}_{rgt:4}{cycle:2}{orbitsegment:2}_{version:3}_{revision:2}.h5"
reader = ipx.Read(path, "ATL13", pattern)

lats_ = []
lons_ = []
ht_water_surface_ = []
err_ht_water_surface_ = []
import glob
import h5py

for filepath in glob.iglob('./download/*.h5'):
    print(filepath)
    with h5py.File(filepath, mode='r') as f:

        latvar = f['/gt1l/segment_lat']
        latitude = latvar[:]

        lonvar = f['/gt1l/segment_lon']
        longitude = lonvar[:]

        dset_name = '/gt1l/segment_geoid'
        datavar = f[dset_name]
        geoid = datavar[:]

        datavar = f['/gt1l/ht_ortho']
        ht_ortho = datavar[:]

        datavar = f['/gt1l/ht_water_surf']
        ht_water_surf = datavar[:]


        datavar = f['/gt1l/err_ht_water_surf']
        err_ht_water_surf = datavar[:]

        units = datavar.attrs['units']
        long_name = datavar.attrs['long_name']
        _FillValue = datavar.attrs['_FillValue']

        # Handle FillValue
        geoid[geoid == _FillValue] = np.nan
        ht_water_surf[ht_water_surf == _FillValue] = np.nan
        ht_ortho[ht_ortho == _FillValue] = np.nan
        err_ht_water_surf[err_ht_water_surf == _FillValue] = np.nan
        lats_.append(latitude)
        lons_.append(longitude)
        ht_water_surface_.append(ht_water_surf)
        err_ht_water_surface_.append(err_ht_water_surf)

lats__ = np.array([item for sublist in lats_ for item in sublist])
lons__ = np.array([item for sublist in lons_ for item in sublist])
ht_water_surface__ = np.array([item for sublist in ht_water_surface_ for item in sublist])
err_ht_water_surface__ = np.array([item for sublist in err_ht_water_surface_ for item in sublist])
```

```
You have 1 files matching the filename pattern to be read in.
./download/processed_ATL13_20211024045412_04851301_005_01.h5
```

```
import matplotlib.pyplot as plt
#ax = plt.axes(projection='3d')
```

```
plt.scatter(lons__, lats__, c=ht_water_surface__, cmap='Greens')
plt.colorbar()
```

```
import plotly.express as px
# df = px.data.carshare()
fig = px.scatter_mapbox(df_icesat, lat="x", lon="y", color="icesat_data",
                  color_continuous_scale=px.colors.cyclical.IceFire, size_max=5, zoom=12,
                  mapbox_style="carto-positron")
fig.show()
```

```
ht_water_surface__
```

```
array([204.31064, 204.29868, 204.27911, 204.26178, 204.2799 , 204.29974,
       204.28564, 204.31549, 204.31906, 204.28677, 204.2832 , 204.26495,
       204.27728, 204.26544, 204.24957, 204.25519, 204.29956, 204.31601,
       204.25607, 204.26071, 204.24197, 204.23494, 204.23668, 204.22298,
       204.20885, 204.22096, 204.23354, 204.2086 , 204.20433, 204.20424,
       204.19595, 204.22049, 204.2262 , 204.2182 , 204.21802, 204.21884,
       204.25262, 204.25815, 204.26147, 204.23456, 204.21423, 204.2312 ,
       204.23288, 204.2201 , 204.23846, 204.23318, 204.21696, 204.23988,
       204.24815, 204.2375 , 204.24092, 204.20767, 204.23056, 204.22458,
       204.24243, 204.22803, 204.23856, 204.2528 , 204.26901, 204.26898,
       204.25621, 204.23227, 204.26205, 204.27142, 204.25128, 204.26425,
       204.2713 , 204.27118, 204.26907, 204.25368, 204.26459, 204.25899,
```

```
       204.25415, 204.24596, 204.27339, 204.24701, 204.24158, 204.27336,
       204.23787, 204.27281, 204.26994, 204.26596, 204.27744, 204.2754 ,
       204.26399, 204.25922, 204.28214, 204.28836, 204.28256, 204.27448,
       204.27284, 204.24586, 204.28693, 204.28299, 204.25772, 204.26593,
       204.27884, 204.23184, 204.26106, 204.28181, 204.23335, 204.2654 ,
       204.24287, 204.25185, 204.27199, 204.23941, 204.22205, 204.2496 ,
       204.22314, 204.26843, 204.27185, 204.27646, 204.25925, 204.26245,
       204.27353, 204.26465, 204.28253, 204.26785, 204.25745, 204.27954,
       204.27563, 204.27615, 204.27426, 204.29172, 204.28406, 204.26382,
       204.2601 , 204.27689, 204.28162, 204.27203, 204.2663 , 204.26837,
       204.24374, 204.23915, 204.27351, 204.24757, 204.24373, 204.25902,
       204.249  , 204.2753 , 204.21866, 204.20999, 204.25735, 204.23358,
       204.24033, 204.25667, 204.2415 , 204.22804, 204.23607, 204.2699 ,
       204.25433, 204.22815, 204.23471, 204.22395, 204.2574 , 204.25269,
       204.22934, 204.24115, 204.24446, 204.2267 , 204.25366, 204.26433,
       204.26608, 204.2489 , 204.2397 , 204.24113, 204.24615, 204.24637,
       204.19415, 204.23642, 204.23189, 204.26901, 204.23221, 204.22974,
       204.24954, 204.23346, 204.25719, 204.2264 , 204.2656 , 204.2192 ,
       204.24126, 204.27354, 204.26709, 204.27252, 204.25485, 204.25589,
       204.24059, 204.23753, 204.22699, 204.27951, 204.24704, 204.2467 ,
       204.26257, 204.27454, 204.24585, 204.24596, 204.24684, 204.23119,
       204.21364, 204.22716, 204.21858, 204.2359 , 204.255  , 204.24432,
       204.21368, 204.24777, 204.25519, 204.2489 , 204.27005, 204.26535,
       204.25098, 204.26599, 204.2459 , 204.26353, 204.25554, 204.24475,
       204.25699, 204.24918, 204.20355, 204.22298, 204.23294, 204.23093,
       204.22852, 204.21498, 204.2381 , 204.22104, 204.19385, 204.24655,
       204.23575, 204.24277, 204.2518 , 204.22832, 204.22008, 204.24573,
       204.22705, 204.23505, 204.22389, 204.22012, 204.21904, 204.21399,
       204.24144, 204.21402, 204.24155, 204.2163 , 204.2297 , 204.2174 ,
       204.2052 , 204.20255, 204.1788 , 204.199  , 204.19879, 204.18967,
       204.212  , 204.17538, 204.18913, 204.2118 , 204.20535, 204.21725,
       204.21817, 204.29785, 204.28076, 204.18738, 204.1749 , 204.18898,
       204.21233, 204.17578, 204.20554, 204.21468, 204.20755, 204.2333 ,
       204.20451, 204.20963, 204.22736, 204.23097, 204.23907, 204.22604,
       204.18669, 204.18732, 204.19095, 204.20607, 204.1505 , 204.18442],
      dtype=float32)
```

```
import pandas as pd
dict_ = {'y':lons__, 'x':lats__, 'icesat_data':ht_water_surface__}
df_icesat = pd.DataFrame(dict_)
df_icesat
```

```
import xarray as xr
```

```
# Create an xarray Dataset
ds_icesat = xr.Dataset(
    {
        "icesat_data": (["points"], df_icesat.icesat_data.values)
    },
    coords={
        "latitude": (["points"], df_icesat.x.values[:len(df_icesat.icesat_data.values)]),
        "longitude": (["points"], df_icesat.y.values[:len(df_icesat.icesat_data.values)])
    }
)
ds_icesat
```

```
ds_icesat.icesat_data.plot()
```

```
def distance(lat1, lon1, lat2, lon2):
    p = 0.017453292519943295
    hav = 0.5 - np.cos((lat2-lat1)*p)/2 + np.cos(lat1*p)*np.cos(lat2*p) * (1-np.cos((lon2-lon1)*p)) / 2
    return 12742 * np.arcsin(np.sqrt(hav))

def closest(data, v):
    return min(data, key=lambda p: distance(v['y'],v['x'],p['y'],p['x']))
# tempDataList = [{'lat': 39.7612992, 'lon': -86.1519681},
#                 {'lat': 39.762241,  'lon': -86.158436 },
#                 {'lat': 39.7622292, 'lon': -86.1578917}]

v = {'y': 23.710020, 'x':   85.779668}
#print(closest(tempDataList, v))
```

```
distance(23.710020, 85.779668, 23.710096, 85.779659)
```

```
0.008501290591972816
```

```
def closest(df_survey, df_icesat, lat_, lon_):
    closest_lat_lon_index = 0
    dist__ = distance(df_survey['lat'].iloc[0], df_survey['lon'].iloc[0], lat_, lon_)
    for i_ in range(1,df_survey.shape[0]):
        dist_ = distance(df_survey['lat'].iloc[i_], df_survey['lon'].iloc[i_], lat_, lon_)
        if dist_ < dist__:
            closest_lat_lon_index = i_
            dist__ = dist_
    return closest_lat_lon_index
```

```
survey_data_closest_to_icesat = []
for i_ in range(df_icesat.shape[0]):
    survey_data_closest_to_icesat.append(df_survey.iloc[closest(df_survey, df_icesat, df_icesat['x'].iloc[-1], df_icesat['y']
```

```
1 survey_data_closest_to_icesat
```

```
[0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
 0.44999999999998863,
```

```
1 plt.plot(survey_data_closest_to_icesat)
```

So, we can get the linear scaling relationship between the bathymetry survey and Icesat-2 data

```
# import plotly.express as px
# # df = px.data.carshare()
# fig = px.scatter_mapbox(df_icesat, lat="y", lon="x", color="icesat_data",
#                   color_continuous_scale=px.colors.cyclical.IceFire, size_max=5, zoom=12,
#                   mapbox_style="carto-positron")
# fig.show()
```

```
# Todo
# Get the bathy survey data for the Icesat-2 track and compare it with the volume of water in the reservoir
# Experiment with ATL03 and ATL08 products as well
```

GEDI

```
def qualityMask(im):
    return im.updateMask(im.select('quality_flag').eq(1)).select('rh98').toInt()
```

```
date_range = ['2022-11-01','2022-12-30']
```

```
gedi = ee.ImageCollection('LARSE/GEDI/GEDI02_A_002_MONTHLY').map(qualityMask).filterBounds(aoi).filterDate('2022-11-01','2022
ds = gedi.wx.to_xarray(region=aoi.bounds(), scale=25)
```

```
ds.rh98.values.shape
```

```
(2, 172, 160)
```

```
ds.isel(time=1).rh98.plot()
```

```
import pandas as pd

# Initialize an empty list to store DataFrames
dfs = []

# Iterate over the range and append DataFrames to the list
for i in range(ds.rh98.values.shape[0]):
    df = ds.rh98.isel(time=i).to_dataframe().reset_index().dropna()
    dfs.append(df)

# Concatenate all DataFrames in the list into a single DataFrame
df_ = pd.concat(dfs, ignore_index=True)

```

```
df_gedi = df_.copy()
df_gedi
```

```
import matplotlib.pyplot as plt
#ax = plt.axes(projection='3d')
#plt.scatter(lons__, lats__, c=ht_water_surface__, cmap='Greens')
plt.scatter(df_gedi['x'], df_gedi['y'], c=df_gedi['rh98'], cmap='Greens')
plt.colorbar()
```

```
import plotly.express as px
# df = px.data.carshare()
fig = px.scatter_mapbox(df_gedi, lat="y", lon="x", color="rh98",
                  color_continuous_scale=px.colors.cyclical.IceFire, size_max=5, zoom=11,
                  mapbox_style="carto-positron")
fig.show()
```

```
# Get the bathy survey data for the GEDI track and compare it with the volume of water in the reservoir
```

```
# survey_data_closest_to_gedi = []
# for i_ in range(df_gedi.shape[0]):
#     survey_data_closest_to_gedi.append(df_survey.iloc[closest(df_survey, df_gedi, df_gedi['y'].iloc[-1], df_gedi['x'].iloc[
```

```
# survey_data_closest_to_gedi
```

Values are coming because we are selecting the nearest survey data, we need to put a threshold on the distance

Sentinel 3

```
dataset = ee.ImageCollection('COPERNICUS/S3/OLCI').filterDate('2022-01-01', '2022-01-30').filterBounds(aoi).select('Oa07_radi
             .multiply(ee.Image([0.00879161]))
```

```
dataset = dataset.set('system:time_start', 0)
ds = dataset.wx.to_xarray(region=aoi.bounds(), scale=300)
```

```
ds
```

```
ds.Oa07_radiance.plot()
```

Time series of sedimentation using Sentinel 3

```
from tqdm import tqdm
from datetime import date
sedimentation_ = []
precipitation_ = []
times_ = []
```

```
for year in tqdm(range(2017,2023)):
    for mon in range(1,13):
        try:
            dataset = ee.ImageCollection('COPERNICUS/S3/OLCI').filterDate(date(year, mon, 1).strftime("%Y-%m-%d"), date(year,
                    .multiply(ee.Image([0.00879161]))
            dataset = dataset.set('system:time_start', 0)
            ds = dataset.wx.to_xarray(region=aoi.bounds(), scale=300)
            sedimentation_.append(ds.Oa07_radiance.mean(dim='time').mean(dim='x').mean(dim='y').values.flatten()[0])
            times_.append(date(year, mon, 1).strftime("%Y-%m-%d"))
            dataset = ee.ImageCollection('JAXA/GPM_L3/GSMaP/v6/operational').filterDate(date(year, mon, 1).strftime("%Y-%m-%c
            dataset = dataset.set('system:time_start', 0)
            ds = dataset.wx.to_xarray(region=aoi.bounds(), scale=11132)
            precipitation_.append(ds.hourlyPrecipRate.sum(dim='time').sum(dim='x').sum(dim='y').values.flatten()[0])
        except:
            continue
```

```
import datetime as dt
dates_list = [dt.datetime.strptime(date, "%Y-%m-%d").date() for date in times_]
sedimentation_ = np.array(sedimentation_)
precipitation_ = np.array(precipitation_)
```

```
import seaborn as sns
dict_ = {'Time': dates_list, 'Sedimentation':sedimentation_, 'Precipitation': precipitation_}
df = pd.DataFrame(dict_)
```

```
sns.lineplot(x = "Time", y = "Sedimentation", data = df)
```

```
sns.lineplot(x = "Time", y = "Precipitation", data = df)
```

Dynamic World

```
# #COL_FILTER = ee.Filter.and.( ee.Filter.bounds(aoi), ee.Filter.date('2021-09-01', '2021-09-30'))
# dwCol = ee.ImageCollection('GOOGLE/DYNAMICWORLD/V1').filterDate('2021-09-01', '2021-09-30').filterBounds(aoi)
# s2Col = ee.ImageCollection('COPERNICUS/S2').filterDate('2021-09-01', '2021-09-30').filterBounds(aoi)
# #DwS2Col = ee.Join.saveFirst('s2_img').apply(dwCol, s2Col,ee.Filter.equals({leftField: 'system:index', rightField: 'system:
# CLASS_NAMES = ['water', 'trees', 'grass', 'flooded_vegetation', 'crops', 'shrub_and_scrub', 'built', 'bare', 'snow_and_ice'
# dwImage = ee.Image(dwCol.first())
# VIS_PALETTE = [ '419BDF', '397D49', '88B053', '7A87C6', 'E49635', 'DFC35A', 'C4281B', 'A59B8F', 'B39FE1']
```

```
# dwRgb = dwImage.select('label').visualize({'min': 0, 'max': 8, 'palette': VIS_PALETTE}).divide(255)
# top1Prob = dwImage.select(CLASS_NAMES).reduce(ee.Reducer.max())
# top1ProbHillshade = ee.Terrain.hillshade(top1Prob.multiply(100)).divide(255)
```

```
# dwRgbHillshade = dwRgb.multiply(top1ProbHillshade)
```

```
# ds = dwRgbHillshade.wx.to_xarray(region=aoi.bounds(), scale=10)
```

NDWI

```
# dataset = ee.ImageCollection('LANDSAT/LE07/C01/T1_32DAY_NDWI').filterDate('2022-01-01', '2022-04-30').filterBounds(aoi)
# ds_ndwi = dataset.wx.to_xarray(region=aoi.bounds(), scale=30)
```

```
# # Define the image collection.
# collection = ee.ImageCollection('LANDSAT/LC09/C02/T1_L2')

# # Define a function to calculate NDWI.
# def calculate_ndwi(image):
#     ndwi = image.normalizedDifference(['SR_B3', 'SR_B5']).rename('NDWI')
#     return image.addBands(ndwi)

# # Map the function over the collection.
# ndwi_collection = collection.map(calculate_ndwi).filterDate('2022-01-01', '2022-01-30').filterBounds(aoi)
# ds_ndwi = ndwi_collection.wx.to_xarray(region=aoi.bounds(), scale=30)
```

```
# ds_ndwi.NDWI
```

```
# ds_ndwi.isel(time=0).NDWI.plot()
```

```
# ndwi_ = ds_ndwi.isel(time=0).NDWI.values.copy()
# ndwi_[:,:] = 0.0
# bool_ = ds_ndwi.isel(time=0).NDWI.values>0.001
# np.sum(bool_)
# ndwi_[bool_] = 1.0
# ds_ndwi['NDWI_'] = (('y', 'x'), ndwi_)
# ds_ndwi.NDWI_.plot(cmap='Reds')
# print(np.nanmax(ds_ndwi.NDWI.values),np.nanmin(ds_ndwi.NDWI.values))
```

Dynamic World

```
# Define the image collection.
collection = ee.ImageCollection('GOOGLE/DYNAMICWORLD/V1')

# Select the 'water' band.
water_collection = collection.select('water').filterDate('2022-11-01', '2022-11-30').filterBounds(aoi)
ds_dyn_water = water_collection.wx.to_xarray(region=aoi.bounds(), scale=30)
```

```
ds_dyn_water.water.isel(time=2).plot()
```

```
water = ds_dyn_water.water.values[2,:,:].copy()
water[:,:] = 0.0
bool_ = ds_dyn_water.water.values[2,:,:]>0.2
water[bool_] = 1.0
ds_dyn_water['water_mask'] = (( 'y', 'x'), water)
```

```
ds_dyn_water.water_mask.plot()
```

```
ds_ndwi = ds_dyn_water.copy()
ds_ndwi['NDWI_'] = ds_dyn_water['water_mask']
```

```
from tqdm import tqdm
```

```
# df = pd.read_csv('drive/MyDrive/Bathy/Survey_Data/Tenughat/Tenughat_Dam.csv')
# epsg, hemisphere, north_south = get_epsg(dam_name='Tenughat', country='India')
# x_coord = df.EASTING.values
# y_coord = df.NORTHING.values
# latitude, longitude = utm.to_latlon(x_coord, y_coord, zone, northern=north_south)
# df['lat'] = latitude
# df['lon'] = longitude
# df['bathy'] = np.max(df.DEPTH.values) - df['DEPTH'].values
```

```
df = pd.read_csv('drive/MyDrive/Bathy/Survey_Data/Bakreshwar/7_nov_to_14_nov_depth_1.xyz', header=None)
#epsg, hemisphere, north_south = get_epsg(dam_name='Getalsud', country='India')
epsg, hemisphere, north_south = '32645', 'northern', True
zone = int(epsg[-2:])
from tqdm import tqdm

df_ = df.copy()
df_[[0, 1, 2]] = df_[0].str.split(' ', expand=True)
print(pd.isnull(df_.iloc[0,2]))
for i in tqdm(range(df_.shape[0])):
    #print(pd.isnull(df_.iloc[i,2]))
    if len(df_.iloc[i,2])==0:
        df_.iloc[i,2] = df_.iloc[i,3]
# df_.drop(df_.columns[3], axis=1, inplace=True)
df = df_.copy()
df = df.apply(pd.to_numeric, errors='coerce')
x_coord = df.values[:,0]
y_coord = df.values[:,1]
latitude, longitude = utm.to_latlon(x_coord, y_coord, zone, northern=north_south)
df['lat'] = latitude
df['lon'] = longitude
df['bathy'] = -1*(np.min(df.iloc[:,2].values) - df.iloc[:,2].values)
```

```
False
100%|██████████| 171245/171245 [00:03<00:00, 45221.59it/s]
```

```
lats_ = []
lons_ = []
bathy_ = []
sentinel_B1 = []
sentinel_B2 = []
sentinel_B3 = []
sentinel_B4 = []
sentinel_B5 = []
sentinel_B6 = []
sentinel_B7 = []
sentinel_B8 = []
sentinel_B8A = []
sentinel_B9 = []
sentinel_B11 = []
sentinel_B12 = []

landsat_B1 = []
landsat_B2 = []
landsat_B3 = []
landsat_B4 = []
landsat_B5 = []
landsat_B6 = []
landsat_B7 = []

sentinel_data = [sentinel_B1, sentinel_B2, sentinel_B3, sentinel_B4, sentinel_B5, sentinel_B6, sentinel_B7, \
                 sentinel_B8, sentinel_B8A, sentinel_B9, sentinel_B11, sentinel_B12]

landsat_data = [landsat_B1, landsat_B2, landsat_B3, landsat_B4, landsat_B5, landsat_B6, landsat_B7]
bands = ['B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7', \
         'B7', 'B8', 'B8A', 'B9', 'B11', 'B12']
landsat_bands = ['SR_B1', 'SR_B2', 'SR_B3', 'SR_B4', 'SR_B5', 'SR_B6', 'SR_B7']

box_size = 0.0001
for lat_ in tqdm(np.arange(np.min(df.lat.values), np.max(df.lat.values), box_size)):
    #print(lat_, lat_+0.0006)
    for lon_ in np.arange(np.min(df.lon.values), np.max(df.lon.values), box_size):
        #print(lon_, lon_+0.0006)
            cond = (df.lat>lat_) & (df.lat<lat_+box_size) & (df.lon>lon_) & (df.lon<lon_+box_size)

            if not df[cond].empty:
                #print(df[cond])
                #print(np.mean(df[cond].bathy.values))
                lat_gee = np.mean(df[cond].lat.values)
                lon_gee = np.mean(df[cond].lon.values)
                bathy_values = np.mean(df[cond].bathy.values)
                lats_.append(lat_gee)
                lons_.append(lon_gee)
                bathy_.append(bathy_values)
                for i_sen,sen in enumerate(sentinel_data):
```

```
                    sen.append(ds_sentinel.sel(y=lat_gee, method='nearest').sel(x=lon_gee, method='nearest')[bands[i_sen]].va
                for i_land,land in enumerate(landsat_data):
                    land.append(ds_landsat.sel(y=lat_gee, method='nearest').sel(x=lon_gee, method='nearest')[landsat_bands[i_
```

```
100%|██████████| 385/385 [21:57<00:00,  3.42s/it]
```

```
print(len(sentinel_B1), len(landsat_B1))
```

```
27142 27142
```

```
dict_ = {'sen_B1': sentinel_B1, \
         'sen_B2': sentinel_B2, \
         'sen_B3': sentinel_B3, \
         'sen_B4': sentinel_B4, \
         'sen_B5': sentinel_B5, \
         'sen_B6': sentinel_B6, \
         'sen_B7': sentinel_B7, \
         'sen_B8': sentinel_B8, \
         'sen_B8A': sentinel_B8A, \
         'sen_B9': sentinel_B9, \
         'sen_B11': sentinel_B11, \
         'sen_B12': sentinel_B12, \
         'land_B1': landsat_B1, \
         'land_B2': landsat_B2, \
         'land_B3': landsat_B3, \
         'land_B4': landsat_B4, \
         'land_B5': landsat_B5, \
         'land_B6': landsat_B6, \
         'land_B7': landsat_B7, \
         'bathy': bathy_, \
         'lat': lats_, \
         'lon': lons_}
df_training = pd.DataFrame(dict_)
```

```
np.sum(np.isnan(df_training))
```

```
sen_B1     0
sen_B2     0
sen_B3     0
sen_B4     0
sen_B5     0
sen_B6     0
sen_B7     0
sen_B8     0
sen_B8A    0
sen_B9     0
sen_B11    0
sen_B12    0
land_B1    0
land_B2    0
land_B3    0
land_B4    0
land_B5    0
land_B6    0
land_B7    0
bathy      0
lat        0
lon        0
dtype: int64
```

```
df_training
```

```
df_training.to_csv('bakreshwar_full_training_sentinel_landsat.csv')
```

```
!mv bakreshwar_full_training_sentinel_landsat.csv /content/drive/MyDrive/Bathy/Training
```

```
df_training = pd.read_csv('/content/drive/MyDrive/Bathy/Training/bakreshwar_full_training_sentinel_landsat.csv')
```

GFS 16 day lead forecast

available from 2015-07-01T00:00:00Z to 2023-04-03T12:00:00

Compute the cumulative amount of rainfall for the entire lead time - the time series of the amount of rainfall at every time for the next 16 days.
A time series of this field will give an estimate of the next 16 days outlook - how much rainfall will occur using GFS

Algorithm is to take the 16 day lead forecast for each day from 2015 and then compute the monthly average time series - It should look like the Sentinel-3 image

Keep the creation date fixed and extract all the forecasts at different times

```
dataset = ee.ImageCollection('NOAA/GFS0P25').filterBounds(aoi).filter(ee.Filter.eq('creation_time',ee.Date(0).update(2018,2,1
ds_ = dataset.wx.to_xarray(region=aoi.bounds(), scale=27830)
```

```
ds_
```

```
ds_.temperature_2m_above_ground.plot()
```

```
dataset = ee.ImageCollection('NOAA/GFS0P25').filterBounds(aoi).filter(ee.Filter.eq('creation_time',ee.Date(0).update(2018,2,1
ds = dataset.wx.to_xarray(region=aoi.bounds(), scale=27830)
```

```
ds.temperature_2m_above_ground.plot()
```

```
dds = ds - ds_
dds.temperature_2m_above_ground.plot()
```

```
# Start date is 2021-09-01
# Start date is 2021-08-15
```

```
# for i in range(0,120):
#     print(i)
import datetime as dt
import xarray as xr
```

```
# 360, 363, 366, 369, 372, 375, 378, 381, 384

dataset = ee.ImageCollection('NOAA/GFS0P25').filterBounds(aoi).filter(ee.Filter.eq('creation_time',ee.Date(0).update(2021,9,1
ds_gfs = dataset.wx.to_xarray(region=aoi.bounds(), scale=27830)
if 'time' in ds_gfs.variables:
    ds_gfs = ds_gfs.drop_vars('time')

# Now you can assign datetime values to the 'time' dimension
time_values = [dt.datetime(2021, 9, 1) + dt.timedelta(hours = 3)]  # your datetime values
ds_gfs = ds_gfs.assign_coords(time=time_values)

```

```
for forecast_hr in range(6,384, 3):
    print(forecast_hr)
    dataset = ee.ImageCollection('NOAA/GFS0P25').filterBounds(aoi).filter(ee.Filter.eq('creation_time',ee.Date(0).update(2019
    ds_gfs_ = dataset.wx.to_xarray(region=aoi.bounds(), scale=27830)
    if 'time' in ds_gfs_.variables:
        ds_gfs_ = ds_gfs_.drop_vars('time')

    # Now you can assign datetime values to the 'time' dimension
    time_values = [dt.datetime(2021, 9, 1) + dt.timedelta(hours = forecast_hr)]  # your datetime values
    ds_gfs_ = ds_gfs_.assign_coords(time=time_values)
    ds_gfs = xr.concat([ds_gfs, ds_gfs_], dim='time')
```

```
import matplotlib.pyplot as plt
```

```
fig,ax = plt.subplots(ncols=1,nrows=1, figsize=(10,5))
ds_gfs.total_precipitation_surface.sum(dim='x').sum(dim='y').plot(ax=ax)
ax.set_xlabel('Forecast time')
```

```
ds_gfs
```

```
ds_gfs_
```

```
ds_gfs_day = ds_gfs.resample(time='1D').sum()
fig,ax = plt.subplots(ncols=1,nrows=1, figsize=(10,5))
ds_gfs_day.total_precipitation_surface.sum(dim='x').sum(dim='y').plot(ax=ax)
ax.set_xlabel('Forecast time')
```

```
# 360, 363, 366, 369, 372, 375, 378, 381, 384

dataset = ee.ImageCollection('NOAA/GFS0P25').filterBounds(aoi).filter(ee.Filter.eq('creation_time',ee.Date(0).update(2021,9,1
ds_gfs = dataset.wx.to_xarray(region=aoi.bounds(), scale=27830)
if 'time' in ds_gfs.variables:
    ds_gfs = ds_gfs.drop_vars('time')

# Now you can assign datetime values to the 'time' dimension
time_values = [dt.datetime(2021, 9, 1) + dt.timedelta(hours = 3)]  # your datetime values
ds_gfs = ds_gfs.assign_coords(time=time_values)

for forecast_hr in range(6,120, 1):
    print(forecast_hr)
    dataset = ee.ImageCollection('NOAA/GFS0P25').filterBounds(aoi).filter(ee.Filter.eq('creation_time',ee.Date(0).update(2019
    ds_gfs_ = dataset.wx.to_xarray(region=aoi.bounds(), scale=27830)
    if 'time' in ds_gfs_.variables:
        ds_ = ds_gfs_.drop_vars('time')

    # Now you can assign datetime values to the 'time' dimension
    time_values = [dt.datetime(2021, 9, 1) + dt.timedelta(hours = forecast_hr)]  # your datetime values
```

```
    ds_gfs_ = ds_gfs_.assign_coords(time=time_values)
    ds_gfs = xr.concat([ds_gfs, ds_gfs_], dim='time')
```