# Open Cosmos Challenge

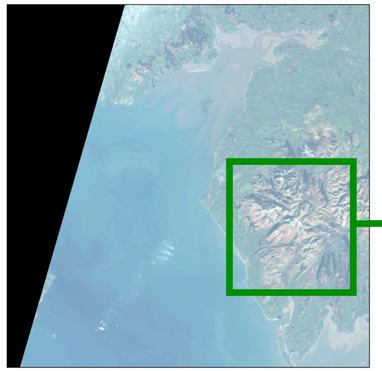
## Data Scientist Role

**Mikel Sagardia, 2023-03-28**

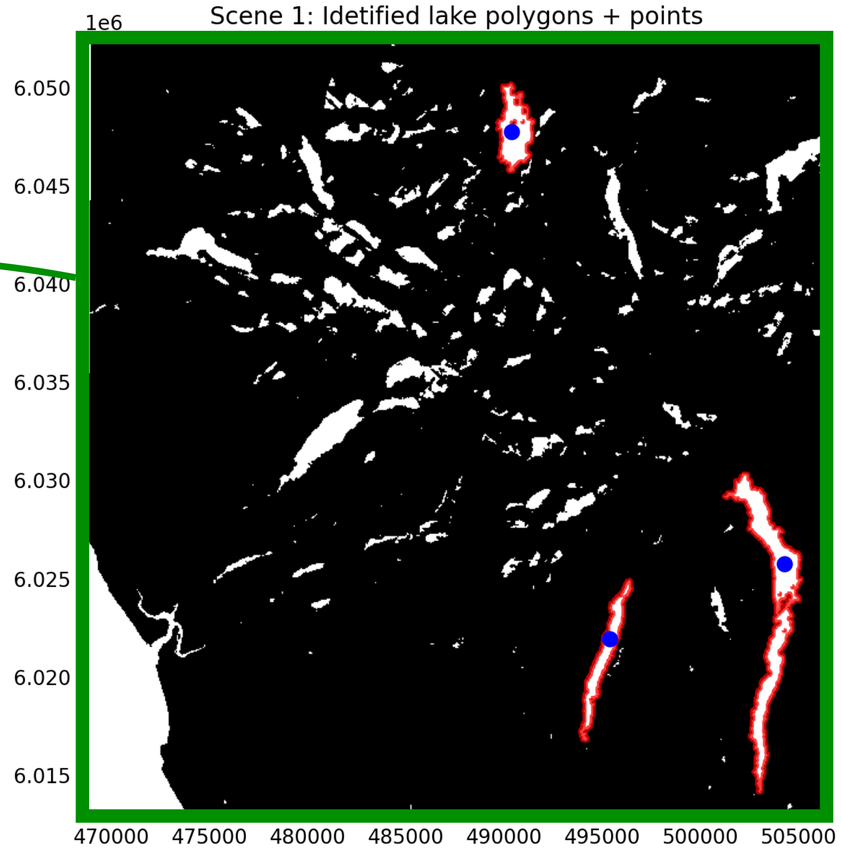
# Let’s start from the results…

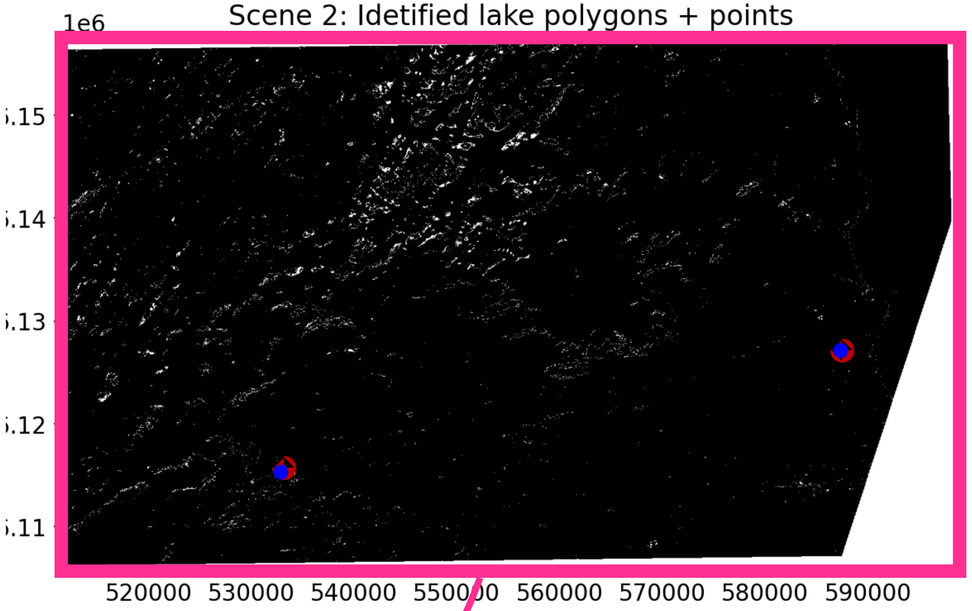
## Scenes 1 & 2: RGB Bands and Lake Polygons

Scene 1



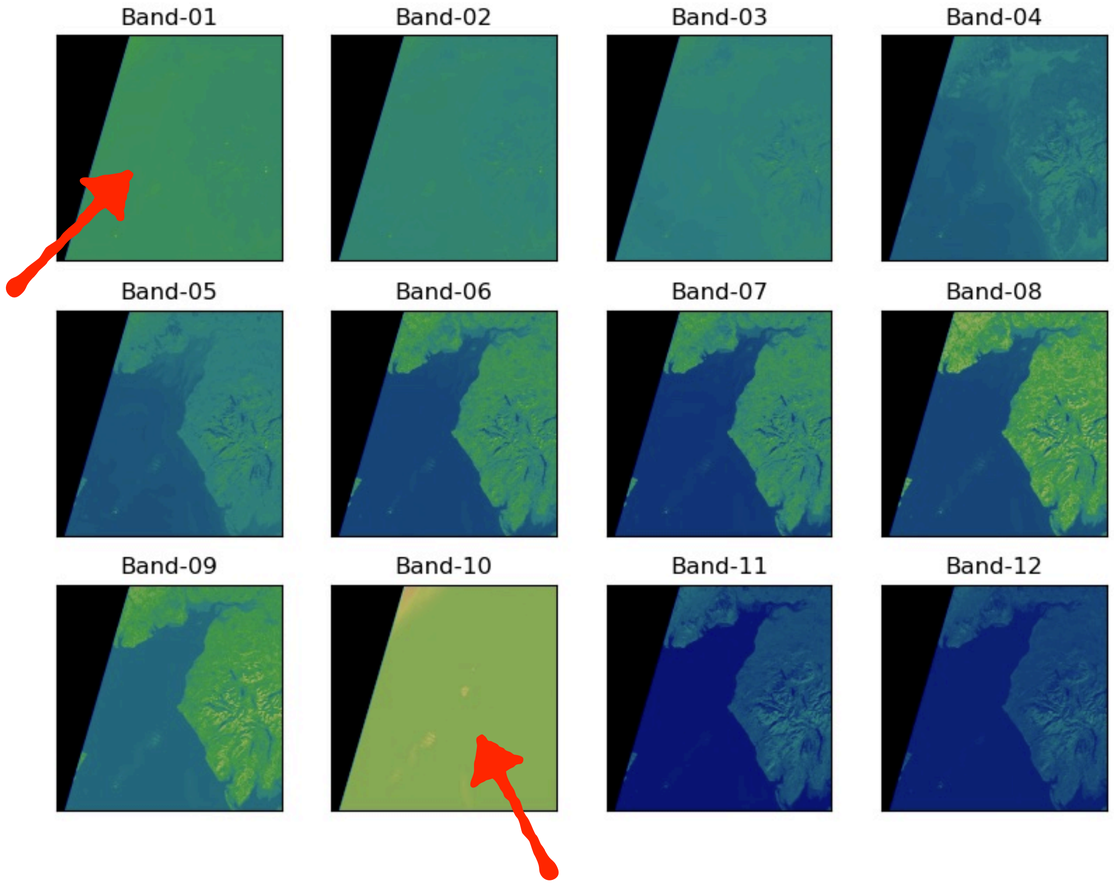
Scene 2



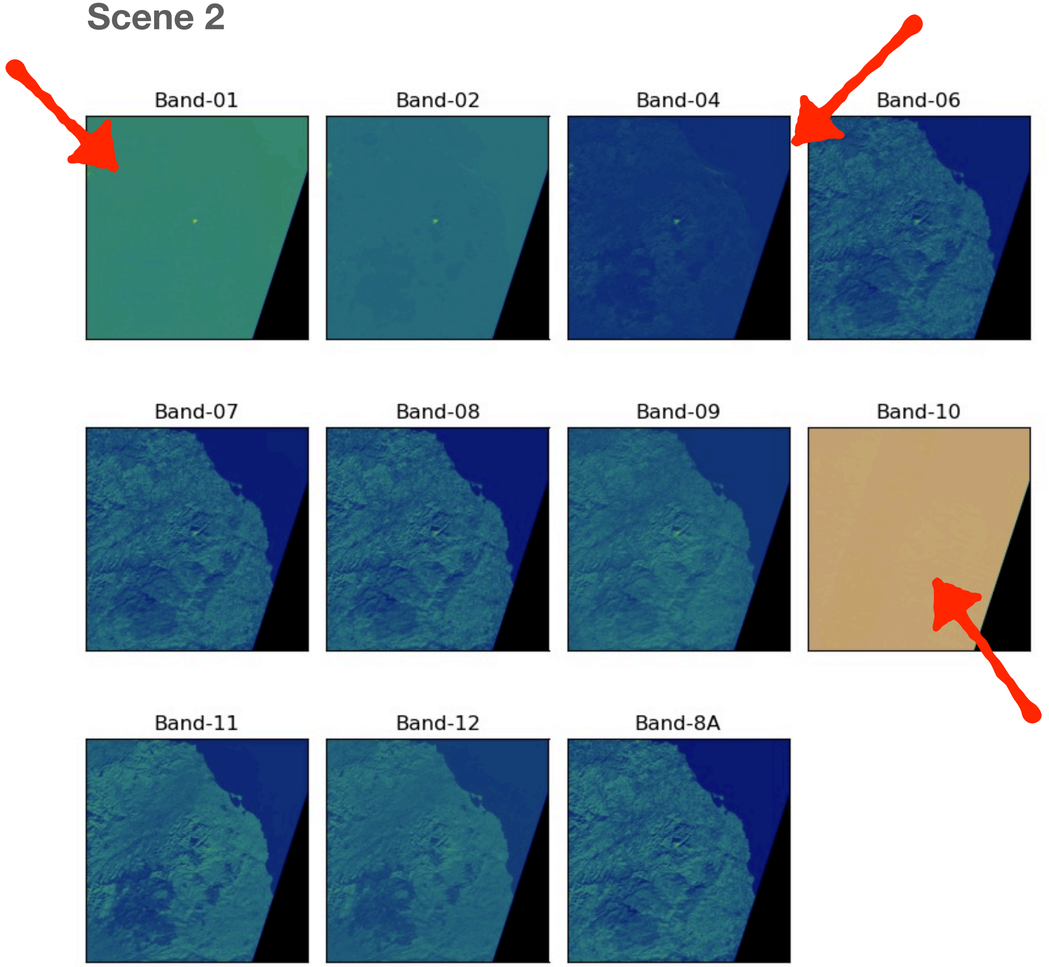



# Raster Bands

## Scene 1

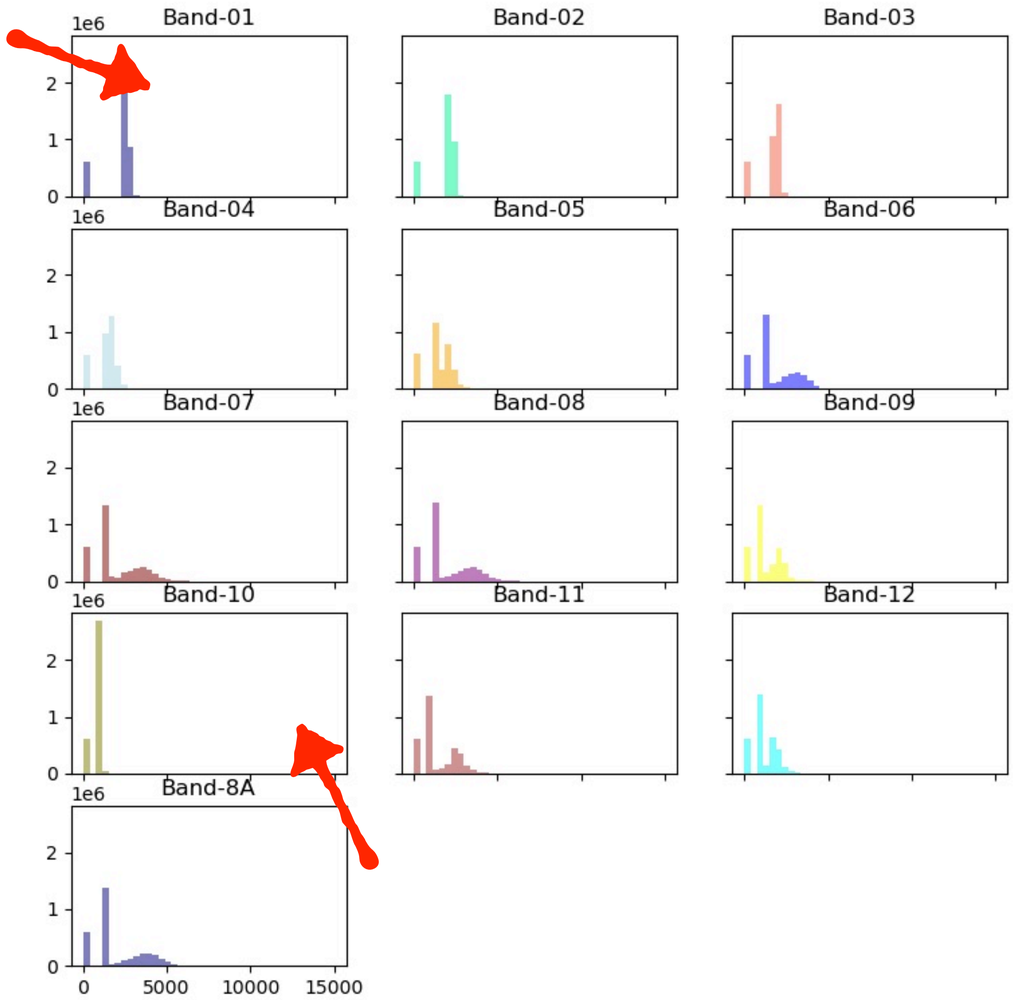



## Scene 2

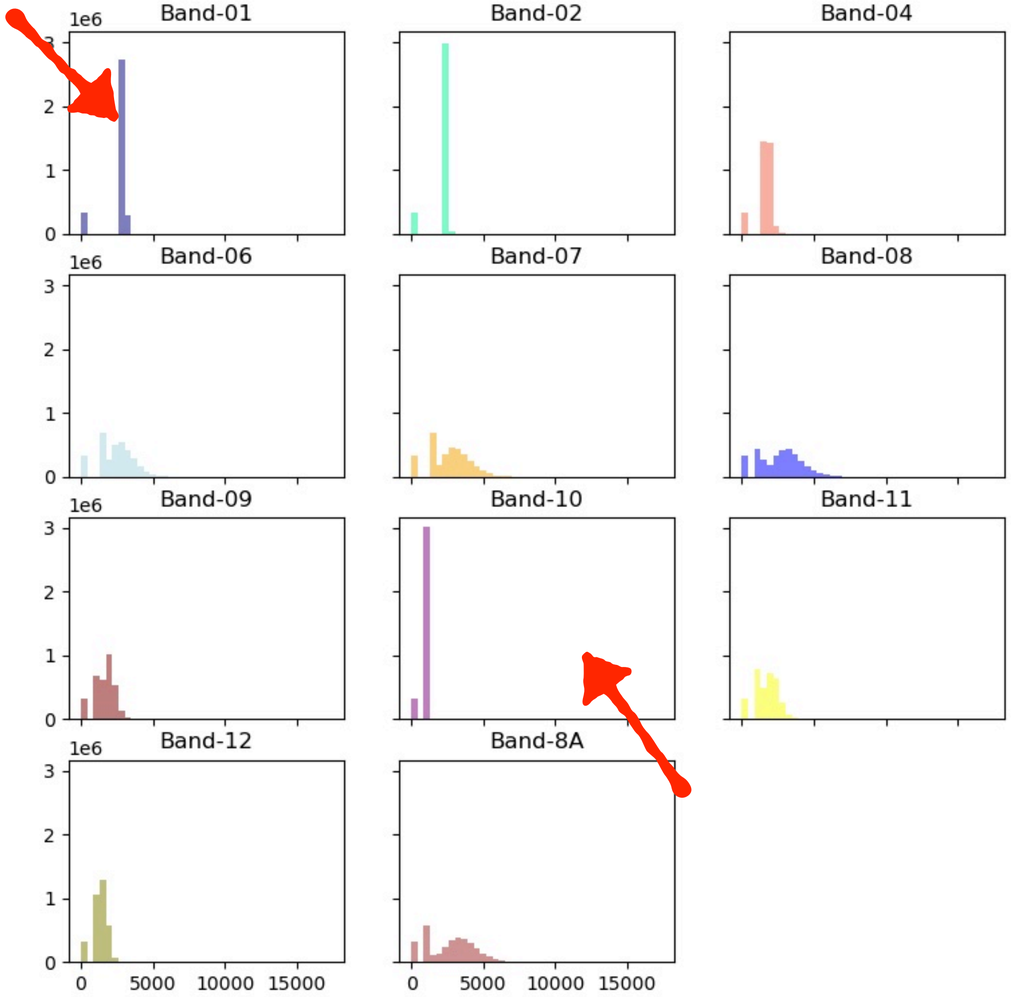


# Raster Histograms

Scene 1

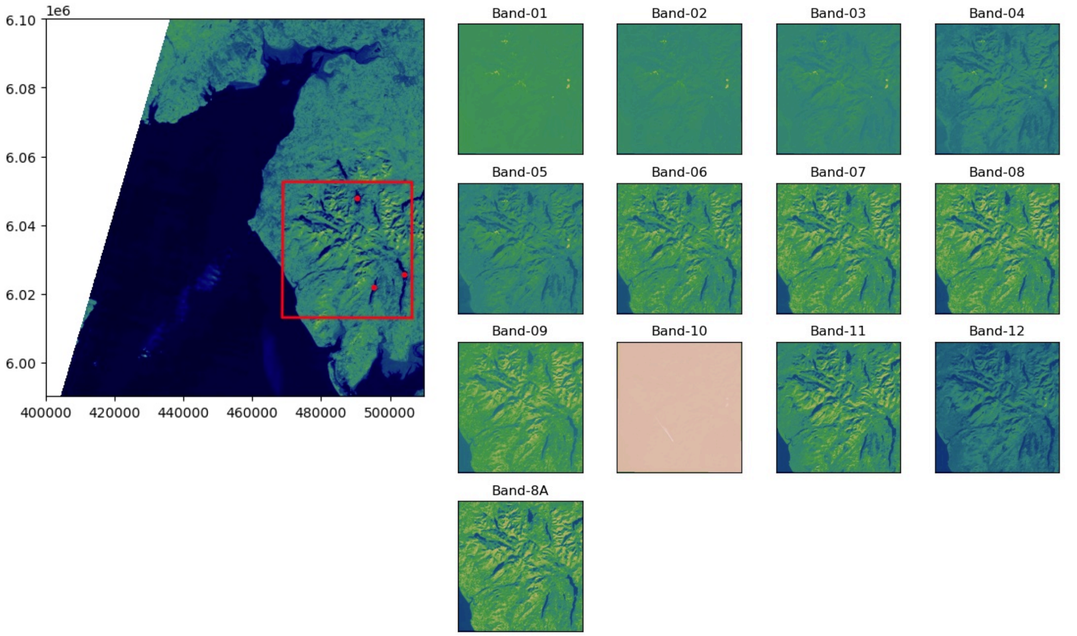



Scene 2

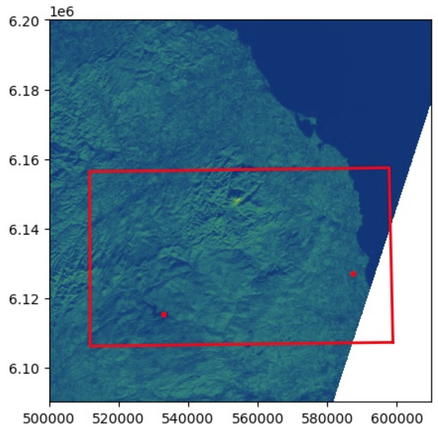


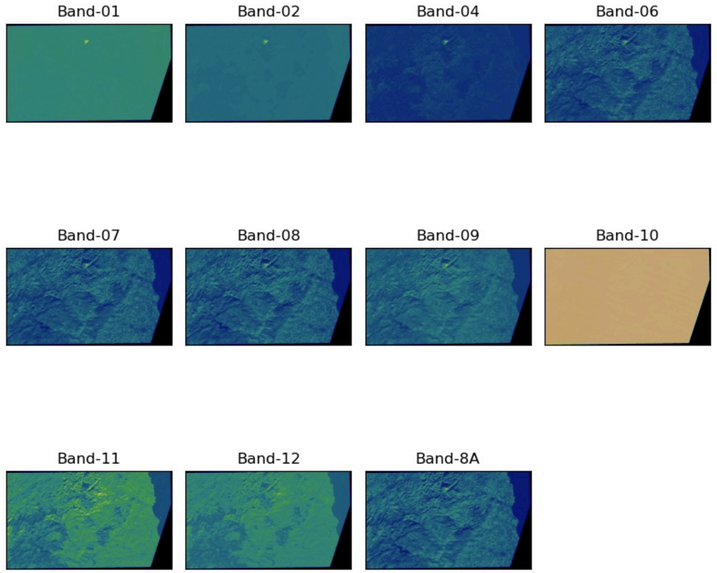
# Objective 1: Resample, Crop and Persist Rasters

## Scene 1




## Scene 2



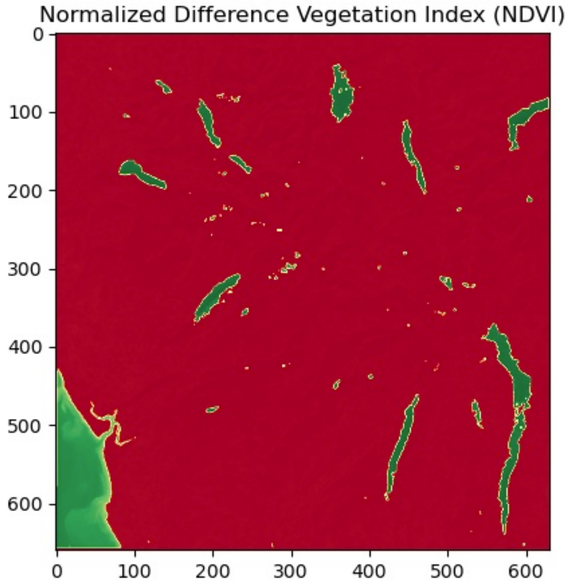
# Objective 1: Resample, Crop and Persist Rasters

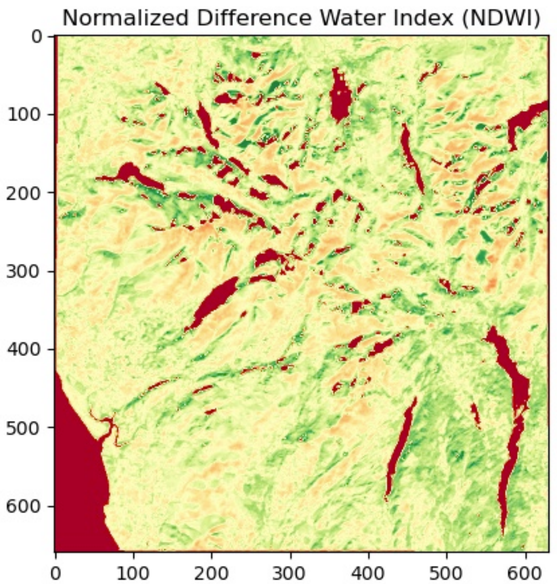
## Steps

- Implementation of the function `resample_persist_band()`: given a band path, it resamples and stores it to disk.
- Resample and save all bands.
- Implementation of the function `crop_persist_band()`: given a band path, it crops and stores it to disk.
- Crop and save all bands.
- Function `load_band_image()`: given a band path, load it and resample it if desired.
- Load all cropped bands.
- Plot cropped, saved bands.
- Plot band histograms and their characteristics (mean & std).

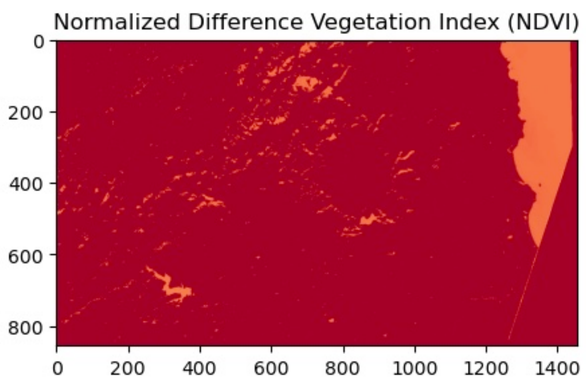
# Objective 2: Compute the NDVI and the NDWI Maps

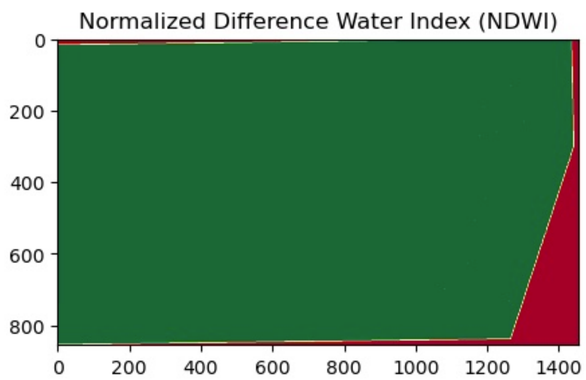
## Scene 1




## Scene 2



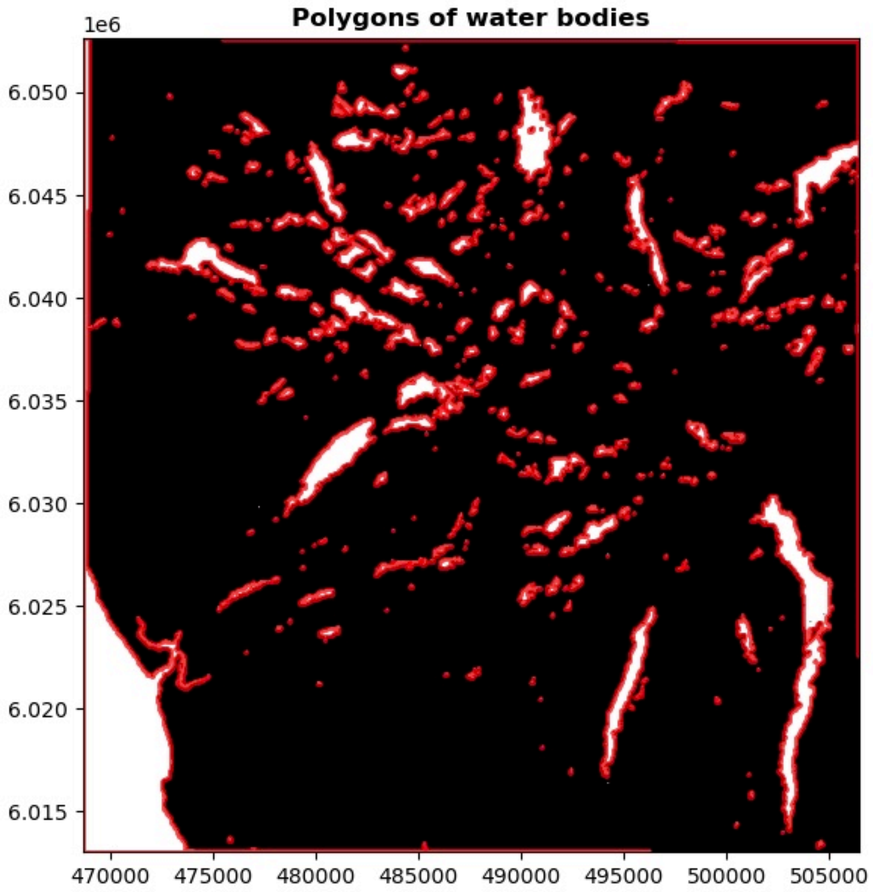
# Objective 2: Compute the NDVI and the NDWI Maps

## Steps

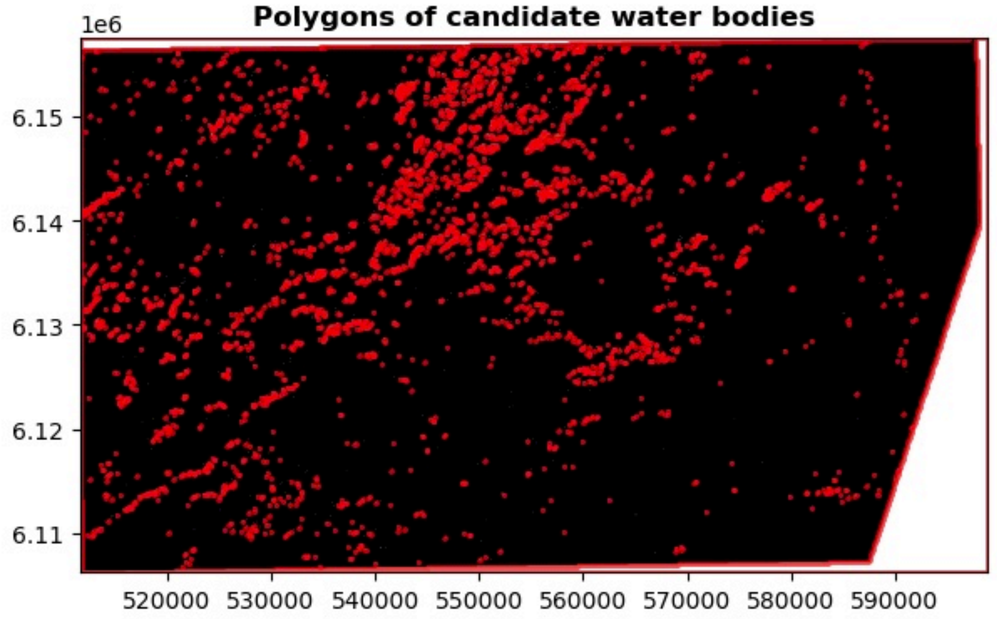
- Implementation of the function `compute_ndvi()`: given band arrays, compute the NDVI map.
- Function `compute_ndwi()`: given band arrays, compute the NDWI map.
- Function `generate_persist_ndmap()`: given band arrays, compute either the NDVI or NDWI map and save it to disk.
- Compute and persist the NDVI and NDWI maps for the scene.
- Load the NDVI and NDWI maps and visualize them.

# Objective 3: Extract Water Shapes

Scene 1




Scene 2



# Objective 3: Extract Water Shapes

## Steps

- Load the saved NDVI or NDWI maps; each scene uses a different one.
- Perform thresholding on the map to obtain a mask of the candidate water bodies.
- Convert the water body BLOBs into polygons of a GeoSeries.
- Plot the mask and the polygons.

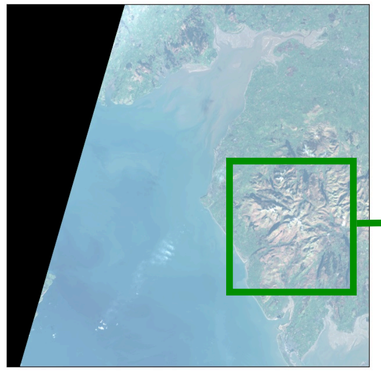
# Objective 3: Extract Water Shapes

## Issues with Scene 2

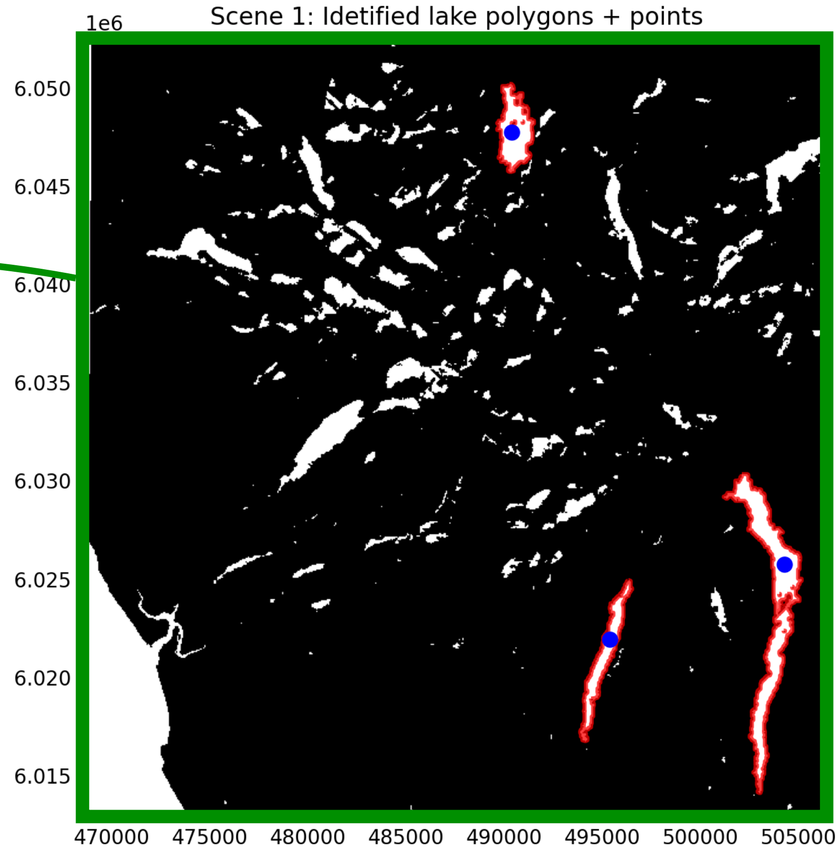
- Since the NDWI map of Scene 2 is not good enough, the NDVI was chosen, since denser vegetation is expected to be around water areas; however, the approach is not satisfactory.
- Alternative approaches (future work):
  - Modified NDWI
  - Tasseled cap transformation
  - Create a pixel-wise classification model (i.e., semantic segmentation) which is trained on labeled data.

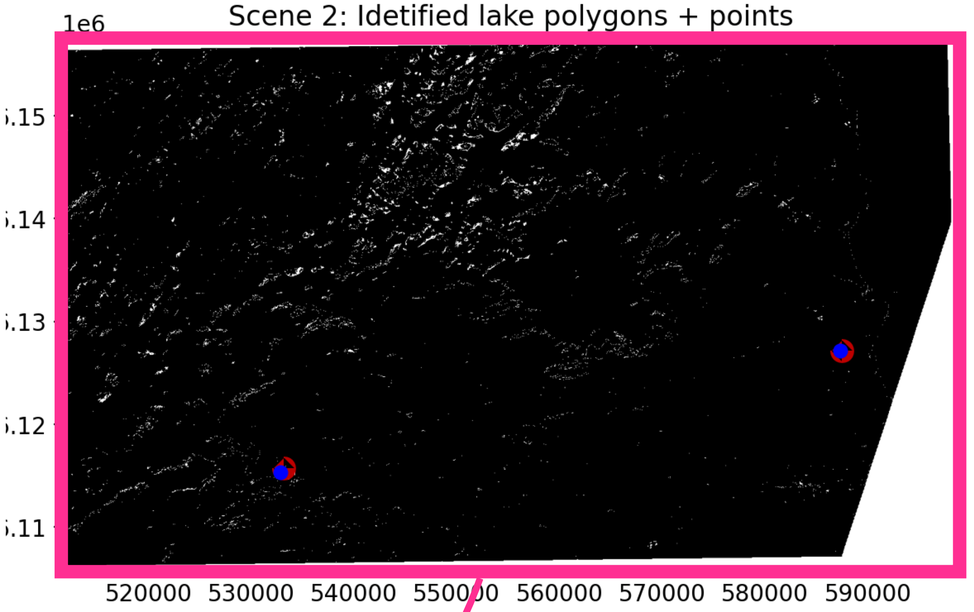
# Objective 4: Identify Lake Polygons

Scene 1



Scene 2



# Objective 4: Identify Lake Polygons

## Steps

- Filter the water body polygons: take the ones which contain or are closest to the target points using built-in `contains()` and `distance()` methods from GeoPandas.
- Assemble GeoDataFrame and save it to disk as a GeoJSON.
- Plot the final result: masked raster + select water polygons + original target points

# Extra: Production Code

- The production code is organized as follows:
  - A library/package `geo_toolkit` which contains generic and reusable functions.
  - A python script `vectorize_water_blobs.py` which uses the library and has code with fine-tuned parameters.
- Additional features:
  - PEP8-conform (linted)
  - Logging and exception handling
  - Testing with Pytest

# Limitations and Future Work

- Persisting images in different stages: not really necessary?
- Try histogram equalization for band 10?
- Resampling: make it clearly optional.
- NDWI of scene 2: seems useless; I decided to work with the NDVI, which is conceptually wrong.
- Try other approaches to detect water bodies in scene 2; see for instance [space_exploration](https://github.com/mxagar/space_exploration).
    - Tasseled cap transformation.
    - Create a pixel-wise classification model.
- Refactor: functions one-task only, OOP, SOLID principles, etc.
- Configuration file for the production environment
- Flask web app
- Containerization