Moreno, Melissa

December 3, 2019

GEO6938

Final Project Write-Up

**Project Summary and Outcome:**

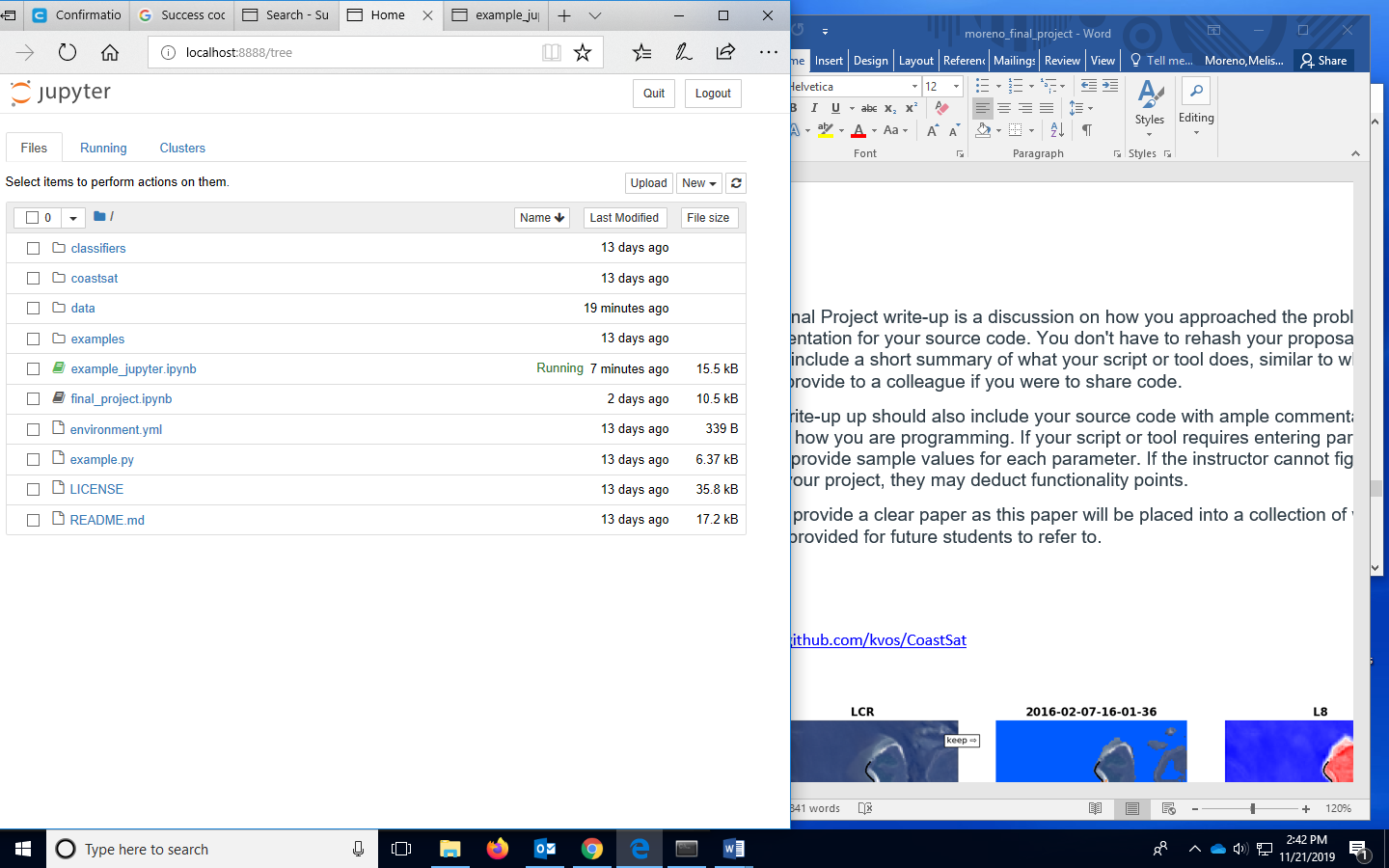
For my project I intend to use Google Earth Engine to do a shoreline analysis of my study area, which is Cedar Key, Florida. This project, for the class, directly relates to my graduate research, since my graduate research entails finding the easiest and most reproducible way to do shoreline analysis with available data. The issues that arise with using satellite imagery is that organizing the imagery is a nightmare. It can take a very long time to retrieve and locate images, especially if you are dealing with a very specific study site. Using Google Earth Engine can standardize the process, and collect imagery with specifications.

The script that I used was created by Kilian Vos, <https://github.com/kvos/CoastSat>. I used their Python code because it gathers satellite imagery from Google Earth Engine through an API. This API allows the user to retrieve imagery from specific dates and satellites and store the imagery in a folder to use for the coastline analysis. The API can also filter through the imagery metadata to select images with specific cloud cover, and other variables. This code is on Github, so it can be easily shared with other people.

This package is a global shoreline tool written in Python to “obtain time-series of shoreline position at any coastline worldwide from 30+ years (and growing) of publicly available satellite imagery”. The package CoastSat is published and described in Vos et al. 2019. In the publication they used the sand/water interface to automatically map a sub-pixel resolution shoreline detection technique. I used this package because it is new and I wanted to test if it would work for my study area.

The script allows you to choose your area of interest, add additional imagery variables for filtering, and then create a plot.lib to digitize the shoreline for the imagery selected. The shoreline lines that you draw out can then be used to analyze the changes to the shoreline. The analysis can also depend on how many image you select to digitize.

The final output (Figure 2) will include a figure with the digitized shoreline, a shoreline with the differences in sand, water, shoreline, and white water, and lastly a direct image from Landsat 8. In the second map, there is a clear idea of how much sand is in the study are during that year.

Figure 1- Screen- shot of the Jupyter Notebook for CoastSat.

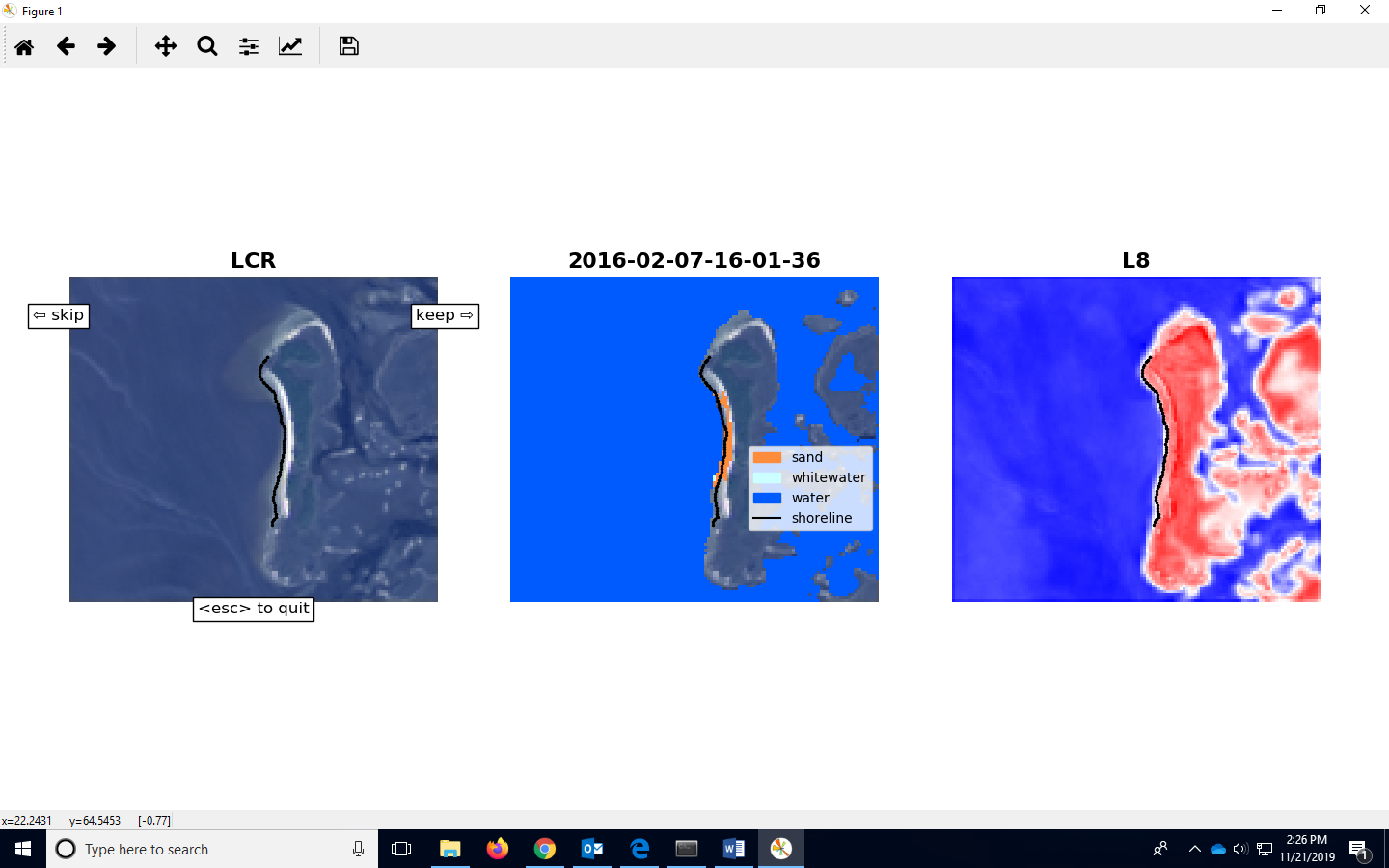


Figure 2- Final output of the code and digitizing,

**Methods: Code**

*1. Initial Settings*

# Load Modules

import os

import numpy as np

import pickle

import warnings

warnings.filterwarnings("ignore")

import matplotlib.pyplot as plt

from coastsat import SDS\_download, SDS\_preprocess, SDS\_shoreline, SDS\_tools, SDS\_transects

# Region of interest (longitude, latitude) in WGS84. Make sure you add 5 coordinate points. Think of the coordinates being a box and the last one is the first coordinate you added, so it wraps up the box nicely.

polygon = [[[-83.089512, 29.242809],

[-83.091572, 29.228579],

[-83.071918, 29.228427],

[-83.071829, 29.243931],

[-83.089512, 29.242809]]]

# Date Range entered in YYYY-MM-DD format

dates = ['2016-01-01', '2017-12-01']

# Satellite missions that can be selected ['L5', 'L7', 'L8', 'S2']

sat\_list = ['L8']

# Name of the site, can be changed to anything

sitename = 'LCR'

# Directory where the data will be stored specified

filepath = os.path.join(os.getcwd(), 'data')

# Putting all the inputs into a dictionary

inputs = {'polygon': polygon, 'dates': dates, 'sat\_list': sat\_list, 'sitename': sitename, 'filepath':filepath}

# put all the inputs into a dictionary

inputs = {

'polygon': polygon,

'dates': dates,

'sat\_list': sat\_list,

'sitename': sitename,

'filepath': filepath\_data

}

*2. Retrieve images*

# retrieve satellite images from GEE

metadata = SDS\_download.retrieve\_images(inputs)

# if you have already downloaded the images, just load the metadata file

metadata = SDS\_download.get\_metadata(inputs)

*3. Batch shoreline detection*

# settings for the shoreline extraction

settings = {

# general parameters:

'cloud\_thresh': 0.5, # threshold on maximum cloud cover that is acceptable

'output\_epsg': 28356, # epsg code of spatial reference system desired for the output

# quality control:

'check\_detection': True, # if True, shows each shoreline detection to the user for validation

'save\_figure': True, # if True, saves a figure showing the mapped shoreline for each image

# add the inputs defined previously

'inputs': inputs,

# [ONLY FOR ADVANCED USERS] shoreline detection parameters:

'min\_beach\_area': 4500, # minimum area (in metres^2) for an object to be labelled as a beach

'buffer\_size': 150, # radius (in meters) of the buffer around sandy pixels considered in the shoreline detection

'min\_length\_sl': 200, # minimum length (in meters) of shoreline perimeter to be valid

'cloud\_mask\_issue': False, # switch this parameter to True if sand pixels are masked (in black) on many images

'sand\_color': 'default', # 'default', 'dark' (for grey/black sand beaches) or 'bright' (for white sand beaches)

}

# [OPTIONAL] preprocess images (cloud masking, pansharpening/down-sampling)

SDS\_preprocess.save\_jpg(metadata, settings)

# [OPTIONAL] create a reference shoreline (helps to identify outliers and false detections)

settings['reference\_shoreline'] = SDS\_preprocess.get\_reference\_sl(metadata, settings)

# set the max distance (in meters) allowed from the reference shoreline for a detected shoreline to be valid

settings['max\_dist\_ref'] = 100

# extract shorelines from all images (also saves output.pkl and shorelines.kml)

output = SDS\_shoreline.extract\_shorelines(metadata, settings)

# plot the mapped shorelines

fig = plt.figure()

plt.axis('equal')

plt.xlabel('Eastings')

plt.ylabel('Northings')

plt.grid(linestyle=':', color='0.5')

for i in range(len(output['shorelines'])):

sl = output['shorelines'][i]

date = output['dates'][i]

plt.plot(sl[:,0], sl[:,1], '.', label=date.strftime('%d-%m-%Y'))

plt.legend()

mng = plt.get\_current\_fig\_manager()

mng.window.showMaximized()

fig.set\_size\_inches([15.76, 8.52])

*4. Shoreline analysis*

# If you have already mapped the shorelines, load the output.pkl file

filepath = os.path.join(inputs['filepath'], sitename)

with open(os.path.join(filepath, sitename + '\_output' + '.pkl'), 'rb') as f:

output = pickle.load(f)

# Now, we have to define cross-shore transects over which to quantify the shoreline changes

# Each transect is defined by two points, its origin and a second point that defines its orientation

# There are 3 options to create the transects:

# Option 1: draw the shore-normal transects along the beach

# Option 2: load the transect coordinates from a .kml file

# Option 3: create the transects manually by providing the coordinates

# Option 1: draw origin of transect first and then a second point to define the orientation

transects = SDS\_transects.draw\_transects(output, settings)

# Option 2: load the transects from a .geojson file

#geojson\_file = os.path.join(os.getcwd(), 'examples', 'NARRA\_transects.geojson')

#transects = SDS\_tools.transects\_from\_geojson(geojson\_file)

# Option 3: create the transects by manually providing the coordinates of two points

#transects = dict([])

#transects['Transect 1'] = np.array([[342836, 6269215], [343315, 6269071]])

#transects['Transect 2'] = np.array([[342482, 6268466], [342958, 6268310]])

#transects['Transect 3'] = np.array([[342185, 6267650], [342685, 6267641]])

# Intersect the transects with the 2D shorelines to obtain time-series of cross-shore distance

settings['along\_dist'] = 25

cross\_distance = SDS\_transects.compute\_intersection(output, transects, settings)

# Plot the time-series

from matplotlib import gridspec

fig = plt.figure()

gs = gridspec.GridSpec(len(cross\_distance),1)

gs.update(left=0.05, right=0.95, bottom=0.05, top=0.95, hspace=0.05)

for i,key in enumerate(cross\_distance.keys()):

if np.all(np.isnan(cross\_distance[key])):

continue

ax = fig.add\_subplot(gs[i,0])

ax.grid(linestyle=':', color='0.5')

ax.set\_ylim([-50,50])

ax.plot(output['dates'], cross\_distance[key]- np.nanmedian(cross\_distance[key]), '-^', markersize=6)

ax.set\_ylabel('distance [m]', fontsize=12)

ax.text(0.5,0.95,'Transect ' + key, bbox=dict(boxstyle="square", ec='k',fc='w'), ha='center',

va='top', transform=ax.transAxes, fontsize=14)

mng = plt.get\_current\_fig\_manager()

mng.window.showMaximized()

fig.set\_size\_inches([15.76, 8.52])

References

Vos, K., Splinter, K. D., Harley, M. D., Simmons, J. A., & Turner, I. L. (2019). CoastSat: A Google Earth Engine-enabled Python toolkit to extract shorelines from publicly available satellite imagery. Environmental Modelling & Software, 122, 104528.