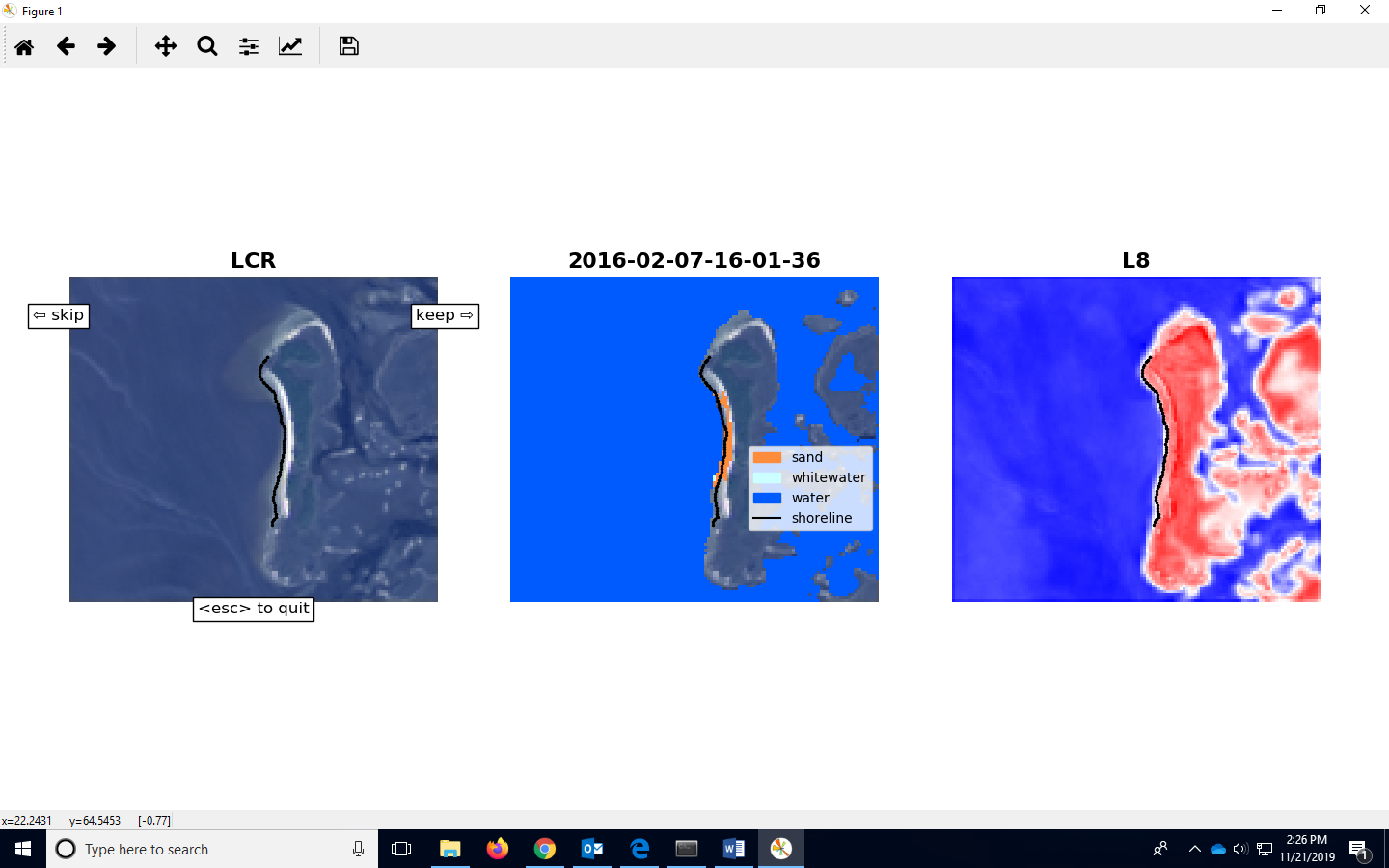
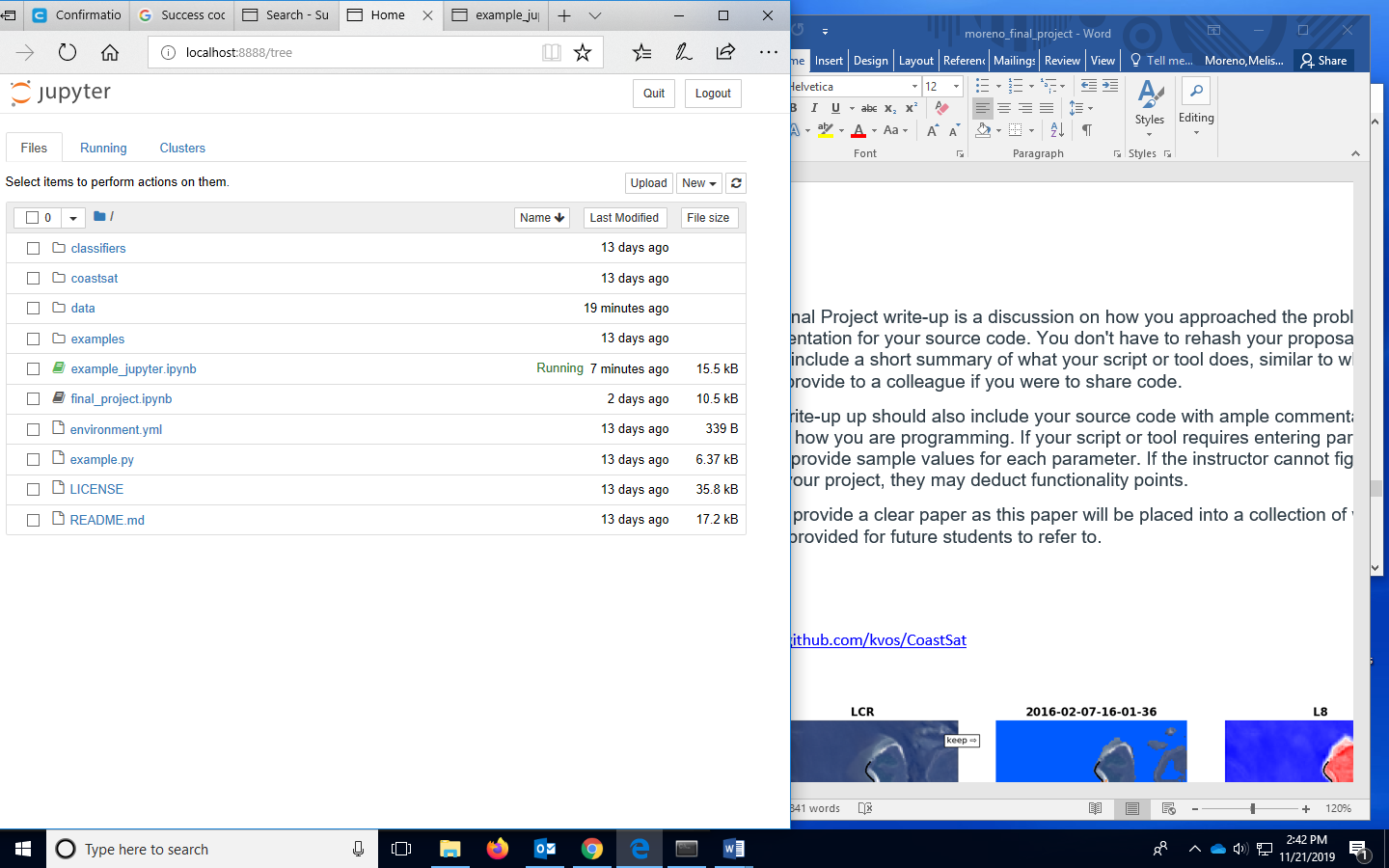
Your Final Project write-up is a discussion on how you approached the problem and the documentation for your source code. You don't have to rehash your proposal, but you should include a short summary of what your script or tool does, similar to what you would provide to a colleague if you were to share code.

Your write-up up should also include your source code with ample commentary to explain how you are programming. If your script or tool requires entering parameters, please provide sample values for each parameter. If the instructor cannot figure out how to run your project, they may deduct functionality points.

Please provide a clear paper as this paper will be placed into a collection of works that will be provided for future students to refer to.

<https://github.com/kvos/CoastSat>





#==========================================================#

# Shoreline extraction from satellite images (<https://github.com/kvos/CoastSat>)

#==========================================================#

# Kilian Vos WRL 2018

#%% 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), also need to have the first and last coordinates the same

polygon = [[[-83.089512, 29.242809],

[-83.091572, 29.228579],

[-83.071918, 29.228427],

[-83.071829, 29.243931],

[-83.089512, 29.242809]]]

# date range, YYYY-MM-DD

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

# satellite missions, ['L5', 'L7', 'L8', 'S2']

sat\_list = ['L8']

# name of the site

sitename = 'LCR'

# directory where the data will be stored

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

# put 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

'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 metres) of the buffer around sandy pixels considered in the shoreline detection

'min\_length\_sl': 200, # minimum length (in metres) 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])