```
In [1]: %load_ext autoreload
%reload_ext autoreload
%autoreload 2
import pytz
import os
import numpy as np
import pickle
import warnings
warnings.filterwarnings("ignore")
import matplotlib
matplotlib.use('Qt5Agg')
import matplotlib.pyplot as plt
from matplotlib import gridspec
plt.ion()
import pandas as pd
from datetime import datetime
from coastsat import SDS_download, SDS_preprocess, SDS_shoreline, SDS_tools, SDS_transects
```

```
In [2]: # region of interest (longitude, latitude)
polygon = [[[-96.97886,27.97414],
            [-96.95842,27.95499],
            [-97.03835,27.83565],
            [-97.05713,27.84780],
            [-96.97886,27.97414]]]
# it's recommended to convert the polygon to the smallest rectangle (sides parallel to coordinate axes)
polygon = SDS_tools.smallest_rectangle(polygon)
# date range
dates = ['2018-12-01', '2018-12-31']
# satellite missions ['L5','L7','L8','L9','S2']
sat_list = ['L8','L9','S2']
# choose Landsat collection 'C01' or 'C02'
collection = 'C02'
# name of the site
sitename = 'San_Jose_TX_3'
# directory where the data will be stored
filepath = os.path.join(os.getcwd(), 'data')
# put all the inputs into a dictionnary
inputs = {'polygon': polygon, 'dates': dates, 'sat_list': sat_list, 'sitename': sitename, 'filepath':filepath,
         'landsat_collection': collection}

# before downloading the images, check how many images are available for your inputs
SDS_download.check_images_available(inputs);
```

```
Number of images available between 2018-12-01 and 2018-12-31:
- In Landsat Tier 1 & Sentinel-2 Level-1C:
     L8: 1 images
     L9: 0 images
     S2: 16 images
  Total to download: 17 images
- In Landsat Tier 2 (not suitable for time-series analysis):
     L8: 0 images
  Total Tier 2: 0 images
```

```
In [3]: # inputs['include_T2'] = True
metadata = SDS_download.retrieve_images(inputs)
```

```
Number of images available between 2018-12-01 and 2018-12-31:
- In Landsat Tier 1 & Sentinel-2 Level-1C:
     L8: 1 images
     L9: 0 images
     S2: 16 images
  Total to download: 17 images
- In Landsat Tier 2 (not suitable for time-series analysis):
     L8: 0 images
  Total Tier 2: 0 images

Downloading images:
L8: 1 images
100%
L9: 0 images

S2: 16 images
100%
Satellite images downloaded from GEE and save in D:\OMU\Msc_Geoscience\Seminar\CoastSat-master\data\San_Jose_TX_3
```

```
In [4]: metadata = SDS_download.get_metadata(inputs)
```

```
In [5]: settings = {
    # general parameters:
    'cloud_thresh': 90,        # threshold on maximum cloud cover
    'dist_clouds': 300,         # ditance around clouds where shoreline can't be mapped
    'output_epsg': 32614,       # 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
    'adjust_detection': True,  # if True, allows user to adjust the postion of each shoreline by changing the threhold
    'save_figure': True,        # if True, saves a figure showing the mapped shoreline for each image
    # [ONLY FOR ADVANCED USERS] shoreline detection parameters:
    'min_beach_area': 1000,     # minimum area (in metres^2) for an object to be labelled as a beach
    'min_length_sl': 500,       # 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': 'latest',    # 'default', 'latest', 'dark' (for grey/black sand beaches) or 'bright' (for white sand beaches)
    'pan_off': False,           # True to switch pansharpening off for Landsat 7/8/9 imagery
    # add the inputs defined previously
    'inputs': inputs,
}
```

```
In [6]: SDS_preprocess.save_jpg(metadata, settings, use_matplotlib=True)
```

```
Saving images as jpg:
L8: 1 images
100%
L9: 0 images

S2: 16 images
100%
Satellite images saved as .jpg in D:\OMU\Msc_Geoscience\Seminar\CoastSat-master\data\San_Jose_TX_3\jpg_files\preprocessed
```

```
In [12]: %matplotlib qt
settings['reference_shoreline'] = SDS_preprocess.get_reference_sl(metadata, settings)
settings['max_dist_ref'] = 100 # max distance (in meters) allowed from the reference shoreline
```

```
Reference shoreline has been saved in D:\OMU\Msc_Geoscience\Seminar\CoastSat-master\data\San_Jose_TX_3
```

```
In [13]: %matplotlib qt
output = SDS_shoreline.extract_shorelines(metadata, settings)
```

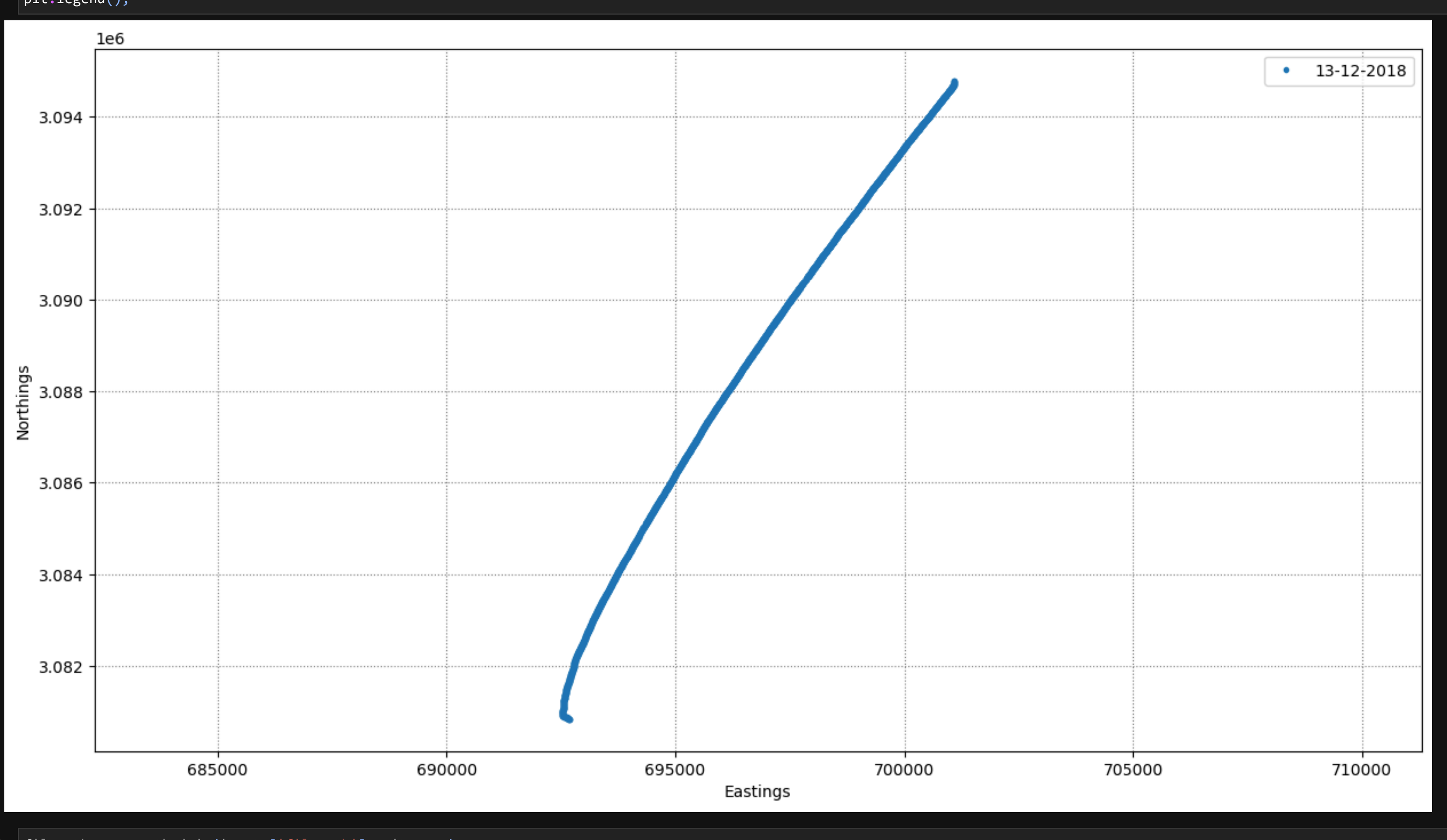
```
Mapping shorelines:
L8:   100%

S2:   18%Could not map shoreline so image was skipped
S2:   100%
```

```
In [14]: output = SDS_tools.remove_duplicates(output) # removes duplicates (images taken on the same date by the same satellite)
output = SDS_tools.remove_inaccurate_georef(output, 50) # remove inaccurate georeferencing (set threshold to 10 m)
```

```
0 duplicates
0 bad georef
```

```
In [15]: %matplotlib inline
fig = plt.figure(figsize=[15,8])
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();
```



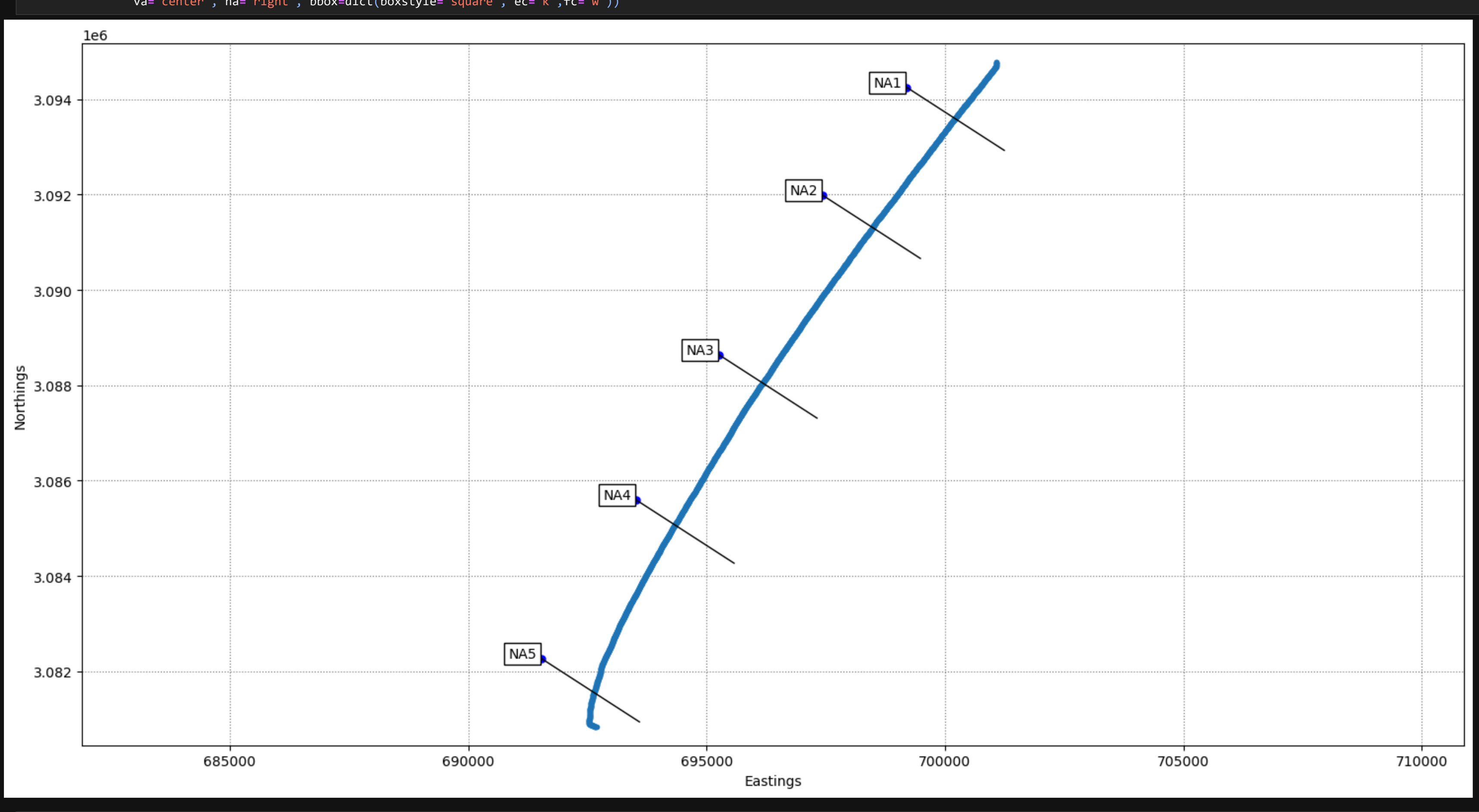

```
In [16]: filepath = os.path.join(inputs['filepath'], sitename)
with open(os.path.join(filepath, sitename + '_output' + '.pkl'), 'rb') as f:
    output = pickle.load(f)
# remove duplicates (images taken on the same date by the same satellite)
output = SDS_tools.remove_duplicates(output)
# remove inaccurate georeferencing (set threshold to 10 m)
output = SDS_tools.remove_inaccurate_georef(output, 11)
```

```
0 duplicates
0 bad georef
```

```
In [17]: from pyproj import CRS
geomtype = 'lines' # choose 'points' or 'lines' for the layer geometry
gdf = SDS_tools.output_to_gdf(output, geomtype)
if gdf is None:
    raise Exception("output does not contain any mapped shorelines")
gdf.crs = CRS(settings['output_epsg']) # set layer projection
# save GEOJSON layer to file
gdf.to_file(os.path.join(inputs['filepath'], inputs['sitename'], '%s_output_%s.geojson'%(sitename,geomtype)),
                                driver='GeoJSON', encoding='utf-8')
```

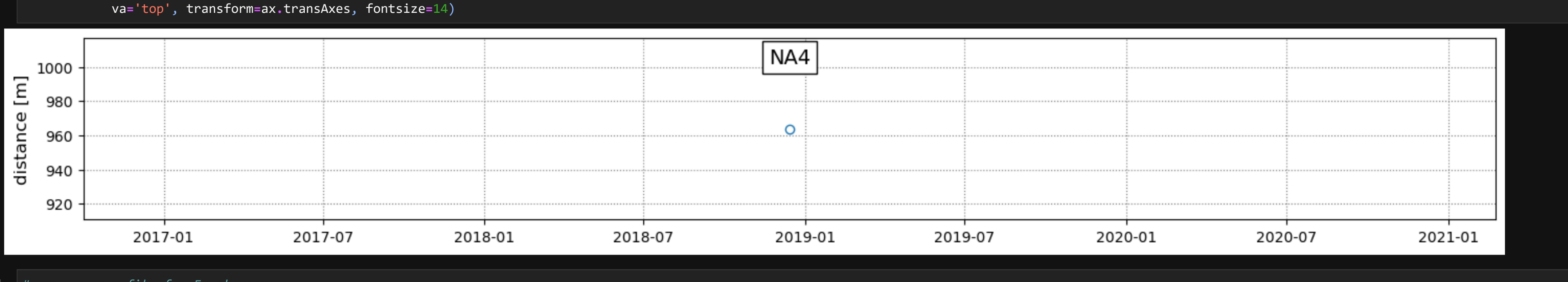
```
In [102…: transects = dict([])
transects['NA1'] = np.array([[ 699192.698243512073532, 3094252.328119106590748 ], [ 701241.923493810580112, 3092927.828859397675842 ]])
transects['NA2'] = np.array([[ 697433.871956584393047, 3091985.737088795285672 ], [ 699483.097206882899627, 3090661.237829086370766 ]])
transects['NA3'] = np.array([[ 695265.622151894494891, 3088636.657476091757417 ], [ 697314.847402193001471, 3087312.158216382842511 ]])
transects['NA4'] = np.array([[ 693526.679775676457211, 3085593.807869399430093 ], [ 695575.905025974963792, 3084269.308609690517187 ]])
transects['NA5'] = np.array([[ 691542.356133548542857, 3082265.187552568968385 ], [ 693591.581383847049437, 3080940.688292860053480 ]])
```

```
In [103…: fig = plt.figure(figsize=[15,8], tight_layout=True)
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'))
for i,key in enumerate(list(transects.keys())):
    plt.plot(transects[key][0,0],transects[key][0,1], 'bo', ms=5)
    plt.plot(transects[key][:,0],transects[key][:,1],'k-',lw=1)
    plt.text(transects[key][0,0]-100, transects[key][0,1]+100, key,
                va='center', ha='right', bbox=dict(boxstyle="square", ec='k',fc='w'))
```




```
In [104…: settings_transects = { # parameters for computing intersections
                      'along_dist':          12,        # along-shore distance to use for computing the intersection
                      'min_points':          3,         # minimum number of shoreline points to calculate an intersection
                      'max_std':             15,        # max std for points around transect
                      'max_range':           30,        # max range for points around transect
                      'min_chainage':        -100,      # largest negative value along transect (landwards of transect origin)
                      'multiple_inter':      'auto',    # mode for removing outliers ('auto', 'nan', 'max')
                      'auto_prc':             0.1,       # percentage of the time that multiple intersects are present to use the max
                     }
cross_distance = SDS_transects.compute_intersection_QC(output, transects, settings_transects)
```

```
In [105…: fig = plt.figure(figsize=[15,10], tight_layout=True)
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.plot(output['dates'], cross_distance[key], '-o', ms=6, mfc='w')
    ax.set_ylabel('distance [m]', fontsize=12)
    ax.text(0.5,0.95, key, bbox=dict(boxstyle="square", ec='k',fc='w'), ha='center',
          va='top', transform=ax.transAxes, fontsize=14)
```




```
In [88]: # save a .csv file for Excel users
out_dict = dict([])
out_dict['dates'] = output['dates']
for key in transects.keys():
    out_dict[key] = cross_distance[key]
df = pd.DataFrame(out_dict)
fn = os.path.join(settings['inputs']['filepath'],settings['inputs']['sitename'],
                  'transect_time_series_texas.csv')
df.to_csv(fn, sep=',')
print('Time-series of the shoreline change along the transects saved as:\n%s'%fn)
```

```
---------------------------------------------------------------------------
PermissionError                           Traceback (most recent call last)
Cell In[88], line 9
      6 df = pd.DataFrame(out_dict)
      7 fn = os.path.join(settings['inputs']['filepath'],settings['inputs']['sitename'],
      8                   'transect_time_series_texas.csv')
----> 9 df.to_csv(fn, sep=',')
     10 print('Time-series of the shoreline change along the transects saved as:\n%s'%fn)

File ~\anaconda3\envs\coastsat\lib\site-packages\pandas\core\generic.py:3772, in NDFrame.to_csv(self, path_or_buf, sep, na_rep, float_format, columns, header, index, index_label, mode, encoding, compression, quoting, quotechar, lineterminator, chunksize, date_format, doublequote, escapechar, decimal, errors, storage_options)
   3761 df = self if isinstance(self, ABCDataFrame) else self.to_frame()
   3763 formatter = DataFrameFormatter(
   3764     frame=df,
   3765     header=header,
   (...)
   3769     decimal=decimal,
   3770 )
-> 3772 return DataFrameRenderer(formatter).to_csv(
   3773     path_or_buf,
   3774     lineterminator=lineterminator,
   3775     sep=sep,
   3776     encoding=encoding,
   3777     errors=errors,
   3778     compression=compression,
   3779     quoting=quoting,
   3780     columns=columns,
   3781     index_label=index_label,
   3782     mode=mode,
   3783     chunksize=chunksize,
   3784     quotechar=quotechar,
   3785     date_format=date_format,
   3786     doublequote=doublequote,
   3787     escapechar=escapechar,
   3788     storage_options=storage_options,
   3789 )

File ~\anaconda3\envs\coastsat\lib\site-packages\pandas\io\formats\format.py:1186, in DataFrameRenderer.to_csv(self, path_or_buf, encoding, sep, columns, index_label, mode, compression, quoting, quotechar, lineterminator, chunksize, date_format, doublequote, escapechar, errors, storage_options)
   1165     created_buffer = False
   1167 csv_formatter = CSVFormatter(
   1168     path_or_buf=path_or_buf,
   1169     lineterminator=lineterminator,
   (...)
   1184     formatter=self.fmt,
   1185 )
-> 1186 csv_formatter.save()
   1188 if created_buffer:
   1189     assert isinstance(path_or_buf, StringIO)

File ~\anaconda3\envs\coastsat\lib\site-packages\pandas\io\formats\csvs.py:240, in CSVFormatter.save(self)
    236 """
    237 Create the writer & save.
    238 """
    239 # apply compression and byte/text conversion
--> 240 with get_handle(
    241     self.filepath_or_buffer,
    242     self.mode,
    243     encoding=self.encoding,
    244     errors=self.errors,
    245     compression=self.compression,
    246     storage_options=self.storage_options,
    247 ) as handles:
    248     # Note: self.encoding is irrelevant here
    249     self.writer = csvlib.writer(
    250         handles.handle,
    251         lineterminator=self.lineterminator,
   (...)
    256         quotechar=self.quotechar,
    257     )
    259     self._save()

File ~\anaconda3\envs\coastsat\lib\site-packages\pandas\io\common.py:859, in get_handle(path_or_buf, mode, encoding, compression, memory_map, is_text, errors, storage_options)
    854 elif isinstance(handle, str):
    855     # Check whether the filename is to be opened in binary mode.
    856     # Binary mode does not support 'encoding' and 'newline'.
    857     if ioargs.encoding and "b" not in ioargs.mode:
    858         # Encoding
--> 859         handle = open(
    860             handle,
    861             ioargs.mode,
    862             encoding=ioargs.encoding,
    863             errors=errors,
    864             newline="",
    865         )
    866     else:
    867         # Binary mode
    868         handle = open(handle, ioargs.mode)

PermissionError: [Errno 13] Permission denied: 'D:\\OMU\\Msc_Geoscience\\Seminar\\CoastSat-master\\data\\San_Jose_TX_3\\transect_time_series_texas.csv'
```

```
In [ ]: # load the measured tide data
filepath = os.path.join(os.getcwd(),'examples','1999.csv')
tide_data = pd.read_csv(filepath)
tz = pytz.timezone('GMT') #To add the UTC in the CSV


dates_ts = [tz.localize(pd.to_datetime(date_str)) for date_str in tide_data['dates']]
tides_ts = np.array(tide_data['tides'])


# get tide levels corresponding to the time of image acquisition
dates_sat = output['dates']
tides_sat = SDS_tools.get_closest_datapoint(dates_sat, dates_ts, tides_ts)

# plot the subsampled tide data
fig, ax = plt.subplots(1,1,figsize=(15,4), tight_layout=True)
ax.grid(which='major', linestyle=':', color='0.5')
ax.plot(dates_ts, tides_ts, '-', color='0.6', label='all time-series')
ax.plot(dates_sat, tides_sat, '-o', color='k', ms=6, mfc='w',lw=1, label='image acquisition')
ax.set(ylabel='tide level [m]',xlim=[dates_sat[0],dates_sat[-1]], title='Water levels at the time of image acquisition');
ax.legend();
```

```
In [ ]: # tidal correction along each transect
reference_elevation = 0 # elevation at which you would like the shoreline time-series to be
beach_slope = 0.1
cross_distance_tidally_corrected = {}
for key in cross_distance.keys():
    correction = (tides_sat-reference_elevation)/beach_slope
    cross_distance_tidally_corrected[key] = cross_distance[key] + correction

# store the tidally-corrected time-series in a .csv file
out_dict = dict([])
out_dict['dates'] = dates_sat
for key in cross_distance_tidally_corrected.keys():
    out_dict[key] = cross_distance_tidally_corrected[key]
df = pd.DataFrame(out_dict)
fn = os.path.join(settings['inputs']['filepath'],settings['inputs']['sitename'],
                  'transect_time_series_tidally_corrected_1999.csv')
df.to_csv(fn, sep=',')
print('Tidally-corrected time-series of the shoreline change along the transects saved as:\n%s'%fn)

# plot the time-series of shoreline change (both raw and tidally-corrected)
fig = plt.figure(figsize=[15,8], tight_layout=True)
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.plot(output['dates'], cross_distance[key], '-o', ms=6, mfc='w', label='raw')
    ax.plot(output['dates'], cross_distance_tidally_corrected[key], '-o', ms=6, mfc='w', label='tidally-corrected')
    ax.set_ylabel('distance [m]', fontsize=12)
    ax.text(0.5,0.95, key, bbox=dict(boxstyle="square", ec='k',fc='w'), ha='center',
          va='top', transform=ax.transAxes, fontsize=14)
ax.legend();
```

```
In [ ]:
```