UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

Development of a tool for the estimation of rooftop solar photovoltaic generation

FINAL DEGREE PROJECT

Document:
Annex

Author:
Maria Maiques Garcia

Director - Co-director:
Alvaro Luna Alloza – Jordi Cipriano

Degree:
Grau en Enginyeria en Tecnologies Industrials (GrETI)

Convocation:
Spring, 2022

Table of contents

1 Tool for the estimation of rooftop solar photovoltaic generation

The complete Python code capable of estimating the solar photovoltaic generation considering shading over the rooftop has been committed in a GitHub project. What's more, the code is shown as follows:

```
import functools
import math
import os
import random
import shutil
import subprocess
import time
import sys
import numpy as np
import pandas as pd
import pdal
import pyproj
import shapely.geometry
import trimesh
import requests
import json


from PySAM.PySSC import PySSC

import datetime


# Constants
repertoire = 'data/output'


def dist2plan(pt, a, b, c, d, liste):
    """
    determine the distance given the index of the point in the list liste and the 4 parameters of a plane
    """
    point = liste.loc[pt, :].values
    x, y, z = point[0], point[1], point[2]
    dis = a * x + b * y + c * z + d
    return (dis)


def pts2plane(q, w, e):
    """determine the equation of a plane with 3 given points"""
    x1, y1, z1 = q[0], q[1], q[2]
    x2, y2, z2 = w[0], w[1], w[2]
    x3, y3, z3 = e[0], e[1], e[2]
    a = (y2 - y1) * (z3 - z1) - (y3 - y1) * (z2 - z1)
    b = (z2 - z1) * (x3 - x1) - (z3 - z1) * (x2 - x1)
    c = (x2 - x1) * (y3 - y1) - (x3 - x1) * (y2 - y1)
    d = -(a * x1 + b * y1 + c * z1)  # d = - ρ
    return (a, b, c, d)


def getAngles(normal):
    """Get the azimuth and altitude from the normal vector."""
    azimuth = 90 - math.degrees(math.atan2(normal[1], normal[0]))
    if azimuth >= 360.0:
```

```
        azimuth -= 360.0
    elif azimuth < 0.0:
        azimuth += 360.0
    t = math.sqrt(normal[0] ** 2 + normal[1] ** 2)
    if t == 0:
        tilt = 0.0
    else:
        tilt = 90 - math.degrees(math.atan(normal[2] / t))  # 0 for flat roof, 90 for wall/vertical roof
    tilt = round(tilt, 3)

    return (tilt, azimuth)


def vector(azimut, elevation):
    # https://math.stackexchange.com/questions/1150232/finding-the-unit-direction-vector-given-azimuth-and-
elevation
    """ return the coordinates of direction vector from azimuth and elevation"""
    x = math.sin(math.radians(azimut)) * math.cos(math.radians(elevation))
    y = math.cos(math.radians(azimut)) * math.cos(math.radians(elevation))
    z = math.sin(math.radians(elevation))
    return (x, y, z)


def create_polygon(tuple_coords_polygon):
    wsg84 = pyproj.Proj(init='epsg:4326')
    ETR = pyproj.Proj(init="EPSG:25831")
    coords = [pyproj.transform(wsg84, ETR, x, y) for (y, x) in tuple_coords_polygon]

    return shapely.geometry.Polygon(coords)


def create_output_folder(path='/output'):
    if os.path.isdir(repertoire):
        shutil.rmtree(repertoire)
        os.makedirs(repertoire)
    else:
        os.makedirs(repertoire)


def create_building_laz(polygon, b, laz_file):
    crop_building = {
        "pipeline": [laz_file,  # needs to be already downloaded (for now)
                     {"type": "filters.crop",
                      'bounds': str(([b[0], b[2]], [b[1], b[3]]))},
                     {"type": "filters.crop",
                      'polygon': polygon.wkt},
                     {
                         "type": "filters.outlier",
                         "method": "statistical",
                         "mean_k": 8,
                         "multiplier": 1},
                     {
                         "type": "filters.neighborclassifier",
                         "domain": "Classification[5:5]",
                         "k": 8},
                     {
                         "type": "filters.range",
                         "limits": "Classification![7:7]"},
                     {"type": "filters.eigenvalues",
                      "knn": 16},
                     {"type": "filters.normal",
                      "knn": 16},
                     {"type": "filters.nndistance",
                      "k": 8
                      },
                     {
```

```
                "type": "filters.approximatecoplanar",
                "knn": 15,
                "thresh1": 25,
                "thresh2": 6
            },
            {
                "filename": repertoire + "/building.laz"  # if I want to export it
            }
            ]
}

pipeline_crop_building = pdal.Pipeline(json.dumps(crop_building))
pipeline_crop_building.validate()
return pipeline_crop_building


def results_test(TMY_filename, tiltS,  azimuthS, nb_roofS,matrix_shading_SAM, areaS):


text = open(matrix_shading_SAM[0], "r")
text = ''.join([i for i in text])
text = text.replace(",9,18", "0,9,18")
x = open('data/output/matrix_shading_SAM_in1.csv', "w")
x.writelines(text)
x.close()
matrix_shading_SAM_in1='data/output/matrix_shading_SAM_in1.csv'


# Format in which we want to display the date
fmt = "%Y-%m-%d %H:%M"
# fmt = '%Y-%m-%d %H:%M'
# Create the start and end times and a 1 hour interval for increments.
start_day = datetime.datetime(2019, 1, 1, 00, 00)
end_day = datetime.datetime(2019, 12, 31, 23, 00)
# inicio = datetime.datetime(2019, 1, 1, 0, 0)
# fin = datetime.datetime(2019, 12, 31, 0, 0)
_1h = datetime.timedelta(hours=1)

# Generar la lista de horas
t = start_day
count = 0
list_of_hours = []
while t <= end_day:
    list_of_hours.append(t.strftime(fmt))
    t += _1h

#Constants
Pmp = 335.205
area_PV = 1.631
ssc = PySSC()
path='data'+'/'+TMY_filename
#path=repertoire+'/'+TMY_filename
path_binary = bytes(path, 'utf-8')
ssc.module_exec_set_print(0)
f = open(repertoire + '/outputs_SAM', "w")
data = ssc.data_create()
ssc.data_set_string( data, b'solar_resource_file', path_binary );
#NEW MODULE : PVSAM1
#Determination of the number of panels and nameplate capacity
nb_panels= [ 0 for a in range(nb_roofS)]
area_SAM=areaS.copy()
tilt_SAM=tiltS.copy()
azimuth_SAM=azimuthS.copy()
```

```
    #easy estimation of the number of solar panels
    for a in range(nb_roofS):
        nb_panels[a]=int(0.8*area_SAM[a]/area_PV)#ratio of 0.8 for the area available
        nb_panels[a]=7*int(nb_panels[a]/7) #because there are at maximum 7 modules per string
#       if tilt_SAM[a]<15 :
#           tilt_SAM[a]= latitude_SAM-10
#           azimuth_SAM[a]=180
    dc_system_capacity=  0
    for a in range(nb_roofS):
        dc_system_capacity=dc_system_capacity+nb_panels[a]*Pmp/1000
    ssc.data_set_number( data, b'system_capacity', dc_system_capacity ) #DC Nameplate capacity (kWdc)
    area_one_module=[ 0 for a in range(nb_roofS)]
    # a more precise estimation of number of PV panels
    for a in range(nb_roofS):
        h=1.046* math.sin(math.radians(24)) #elevation the 21th December (12a.m), 1.046 = width module
        l2=h/math.sin(math.radians(tilt_SAM[a]))
        x=np.sqrt(l2**2-h**2) #space between 2 modules (meters)
        b=np.sqrt(1.559**2-h**2) #1.559 = length of the module
        area_one_module[a]=(x+b)*(1.559+0.2)#space for one module (to not make shading on other modules)
    ssc.data_set_number( data, b'use_wf_albedo', 0 )
    albedo =[ 0.20000000298023224, 0.20000000298023224, 0.20000000298023224, 0.20000000298023224,
0.20000000298023224, 0.20000000298023224, 0.20000000298023224, 0.20000000298023224,
0.20000000298023224, 0.20000000298023224, 0.20000000298023224, 0.20000000298023224 ];
    ssc.data_set_array( data, b'albedo',  albedo) #albedo = reflection from a surface
    ssc.data_set_number( data, b'irrad_mode', 0 )
    #0=beam&diffuse,1=total&beam,2=total&diffuse,3=poa_reference,4=poa_pyranometer
    ssc.data_set_number( data, b'sky_model', 2 ) #0=isotropic,1=hkdr,2=perez
    nb_inverters=int(int(sum(nb_panels)/7)*1/1.56) # 1.56 after tests
    ssc.data_set_number( data, b'inverter_count', nb_inverters )
    ssc.data_set_number( data, b'enable_mismatch_vmax_calc', 0 )
    ssc.data_set_number( data, b'subarray1_nstrings', int(nb_panels[0]/7) ) #after tests, depends on the
characterictics of the PV module and inverter
    ssc.data_set_number( data, b'subarray1_modules_per_string',  7)
    ssc.data_set_number( data, b'subarray1_mppt_input', 1 )
    #if tilt_SAM[0]<15 : #if flat
        #tilt_SAM[0]= latitude_SAM-10 #the best tilt is the value of the latitude-10
        #azimuth_SAM[0]=180 #and face south
    ssc.data_set_number( data, b'subarray1_tilt',tilt_SAM[0])
    ssc.data_set_number( data, b'subarray1_tilt_eq_lat', 0 )
    ssc.data_set_number( data, b'subarray1_azimuth', azimuth_SAM[0] )
    ssc.data_set_number( data, b'subarray1_track_mode', 0 )
    ssc.data_set_number( data, b'subarray1_rotlim', 45 )
    ssc.data_set_number( data, b'subarray1_shade_mode', 0 )
    ssc.data_set_number( data, b'subarray1_gcr', 0.30000001192092896 ) # Ground coverage ratio
    # ratio between the PV modules area and the total ground area
    ssc.data_set_number( data, b'subarray1_shading:string_option', -1 )
    subarray1_shading_azal = matrix_shading_SAM_in1
    ssc.data_set_matrix_from_csv( data, b'subarray1_shading:azal', subarray1_shading_azal );
    subarray1_soiling =[ 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5 ];
    ssc.data_set_array( data, b'subarray1_soiling',  subarray1_soiling);
    ssc.data_set_number( data, b'subarray1_rear_irradiance_loss', 0 )
    ssc.data_set_number( data, b'subarray1_mismatch_loss', 2 )
    #ssc.data_set_number( data, b'subarray1_mismatch_loss', 2 )
    ssc.data_set_number( data, b'subarray1_diodeconn_loss', 0.5 )
    #ssc.data_set_number( data, b'subarray1_diodeconn_loss', 0.5 )
    ssc.data_set_number( data, b'subarray1_dcwiring_loss', 2 )
    ssc.data_set_number( data, b'subarray1_tracking_loss', 0 )
    ssc.data_set_number( data, b'subarray1_nameplate_loss', 0 )
    ssc.data_set_number( data, b'subarray2_rear_irradiance_loss', 0 )
    ssc.data_set_number( data, b'subarray2_mismatch_loss', 2 )
    #ssc.data_set_number( data, b'subarray2_mismatch_loss', 2 )
    ssc.data_set_number( data, b'subarray2_diodeconn_loss', 0.5 )
    #ssc.data_set_number( data, b'subarray2_diodeconn_loss', 0.5 )
    ssc.data_set_number( data, b'subarray2_dcwiring_loss', 2 )
    ssc.data_set_number( data, b'subarray2_tracking_loss', 0 )
    ssc.data_set_number( data, b'subarray2_nameplate_loss', 0 )
```

```
ssc.data_set_number( data, b'subarray3_rear_irradiance_loss', 0 )
ssc.data_set_number( data, b'subarray3_mismatch_loss', 2 )
#ssc.data_set_number( data, b'subarray3_mismatch_loss', 2 )
ssc.data_set_number( data, b'subarray3_diodeconn_loss', 0.5 )
#ssc.data_set_number( data, b'subarray3_diodeconn_loss', 0.5 )
ssc.data_set_number( data, b'subarray3_dcwiring_loss', 2 )
ssc.data_set_number( data, b'subarray3_tracking_loss', 0 )
ssc.data_set_number( data, b'subarray3_nameplate_loss', 0 )
ssc.data_set_number( data, b'subarray4_rear_irradiance_loss', 0 )
ssc.data_set_number( data, b'subarray4_mismatch_loss', 2 )
#ssc.data_set_number( data, b'subarray4_mismatch_loss', 2 )
ssc.data_set_number( data, b'subarray4_diodeconn_loss', 0.5 )
#ssc.data_set_number( data, b'subarray4_diodeconn_loss', 0.5 )
ssc.data_set_number( data, b'subarray4_dcwiring_loss', 2 )
ssc.data_set_number( data, b'subarray4_tracking_loss', 0 )
ssc.data_set_number( data, b'subarray4_nameplate_loss', 0 )
ssc.data_set_number( data, b'dcoptimizer_loss', 0 )
ssc.data_set_number( data, b'acwiring_loss', 1 )
ssc.data_set_number( data, b'transmission_loss', 0 )
ssc.data_set_number( data, b'subarray1_mod_orient', 0 )
ssc.data_set_number( data, b'subarray1_nmodx', 7 ) #Sub-array 1 Number of modules along bottom of row
ssc.data_set_number( data, b'subarray1_nmody', 2 ) #Sub-array 1 Number of modules along side of row
ssc.data_set_number( data, b'subarray1_backtrack', 0 )
if nb_roofS>1:

    text1 = open(matrix_shading_SAM[1], "r")
    text1 = ''.join([i for i in text1])
    text1 = text1.replace(",9,18", "0,9,18")
    x = open('data/output/matrix_shading_SAM_in2.csv', "w")
    x.writelines(text1)
    x.close()
    matrix_shading_SAM_in2 = 'data/output/matrix_shading_SAM_in2.csv'


    ssc.data_set_number( data, b'subarray2_enable', 1 )
    ssc.data_set_number( data, b'subarray2_modules_per_string', 7)
    ssc.data_set_number( data, b'subarray2_nstrings', int(nb_panels[1]/7)  )
    ssc.data_set_number( data, b'subarray2_mppt_input', 1 )
    ssc.data_set_number( data, b'subarray2_tilt', tilt_SAM[1] )
    ssc.data_set_number( data, b'subarray2_tilt_eq_lat', 0 )
    ssc.data_set_number( data, b'subarray2_azimuth', azimuth_SAM[1] )
    ssc.data_set_number( data, b'subarray2_track_mode', 0 )
    ssc.data_set_number( data, b'subarray2_rotlim', 45 )
    ssc.data_set_number( data, b'subarray2_shade_mode', 0 )
    ssc.data_set_number( data, b'subarray2_gcr', 0.30000001192092896 )
    ssc.data_set_number( data, b'subarray2_shading:string_option', -1 )
    subarray2_shading_azal =  matrix_shading_SAM_in2
    ssc.data_set_matrix_from_csv(data, b'subarray2_shading:azal', subarray2_shading_azal);
    #ssc.data_set_matrix( data, b'subarray2_shading:azal', subarray2_shading_azal );
    subarray2_soiling =[ 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5 ];
    ssc.data_set_array( data, b'subarray2_soiling',  subarray2_soiling);
    ssc.data_set_number( data, b'subarray2_mod_orient', 0 )
    ssc.data_set_number( data, b'subarray2_nmodx', 9 )
    ssc.data_set_number( data, b'subarray2_nmody', 2 )
    ssc.data_set_number( data, b'subarray2_backtrack', 0 )
else :
    ssc.data_set_number( data, b'subarray2_enable', 0 )
    ssc.data_set_number( data, b'subarray2_modules_per_string', 0 )
    ssc.data_set_number( data, b'subarray2_nstrings', 0 )
    ssc.data_set_number( data, b'subarray2_mppt_input', 1 )
    ssc.data_set_number( data, b'subarray2_tilt', 0 )
    ssc.data_set_number( data, b'subarray2_tilt_eq_lat', 0 )
    ssc.data_set_number( data, b'subarray2_azimuth', 0 )
    ssc.data_set_number( data, b'subarray2_track_mode', 0 )
    ssc.data_set_number( data, b'subarray2_rotlim', 45 )
    ssc.data_set_number( data, b'subarray2_shade_mode', 0 )
```

```
    ssc.data_set_number( data, b'subarray2_gcr', 0.30000001192092896 )
    subarray2_soiling =[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    #subarray2_soiling =[ 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5 ];
    ssc.data_set_array( data, b'subarray2_soiling',  subarray2_soiling);
    ssc.data_set_number( data, b'subarray2_mod_orient', 0 )
    ssc.data_set_number( data, b'subarray2_nmodx', 9 )
    ssc.data_set_number( data, b'subarray2_nmody', 2 )
    ssc.data_set_number( data, b'subarray2_backtrack', 0 )
if nb_roofS>2:

    text2 = open(matrix_shading_SAM[2], "r")
    text2 = ''.join([i for i in text2])
    text2 = text2.replace(",9,18", "0,9,18")
    x = open('data/output/matrix_shading_SAM_in3.csv', "w")
    x.writelines(text2)
    x.close()
    matrix_shading_SAM_in3 = 'data/output/matrix_shading_SAM_in3.csv'

    ssc.data_set_number( data, b'subarray3_enable', 1 )
    ssc.data_set_number( data, b'subarray3_modules_per_string',  7)
    ssc.data_set_number( data, b'subarray3_nstrings', int(nb_panels[2]/7) )
    ssc.data_set_number( data, b'subarray3_mppt_input', 1 )
    ssc.data_set_number( data, b'subarray3_tilt', tilt_SAM[2] )
    ssc.data_set_number( data, b'subarray3_tilt_eq_lat', 0 )
    ssc.data_set_number( data, b'subarray3_azimuth', azimuth_SAM[2] )
    ssc.data_set_number( data, b'subarray3_track_mode', 0 )
    ssc.data_set_number( data, b'subarray3_rotlim', 45 )
    ssc.data_set_number( data, b'subarray3_shade_mode', 0 )
    ssc.data_set_number( data, b'subarray3_gcr', 0.30000001192092896 )
    ssc.data_set_number( data, b'subarray3_shading:string_option', -1 )
    #subarray3_shading_azal = matrix_shading_SAM[2];
    #ssc.data_set_matrix( data, b'subarray3_shading:azal', subarray3_shading_azal );
    subarray3_shading_azal = matrix_shading_SAM_in3
    ssc.data_set_matrix_from_csv(data, b'subarray3_shading:azal', subarray3_shading_azal);
    subarray3_soiling =[ 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5 ];
    ssc.data_set_array( data, b'subarray3_soiling',  subarray3_soiling);
    ssc.data_set_number( data, b'subarray3_mod_orient', 0 )
    ssc.data_set_number( data, b'subarray3_nmodx', 9 )
    ssc.data_set_number( data, b'subarray3_nmody', 2 )
    ssc.data_set_number( data, b'subarray3_backtrack', 0 );
else :
    ssc.data_set_number( data, b'subarray3_enable', 0 )
    ssc.data_set_number( data, b'subarray3_modules_per_string', 0 )
    ssc.data_set_number( data, b'subarray3_nstrings', 0 )
    ssc.data_set_number( data, b'subarray3_mppt_input', 1 )
    ssc.data_set_number( data, b'subarray3_tilt', 0 )
    ssc.data_set_number( data, b'subarray3_tilt_eq_lat', 0 )
    ssc.data_set_number( data, b'subarray3_azimuth', 0 )
    ssc.data_set_number( data, b'subarray3_track_mode', 0 )
    ssc.data_set_number( data, b'subarray3_rotlim', 45 )
    ssc.data_set_number( data, b'subarray3_shade_mode', 0 )
    ssc.data_set_number( data, b'subarray3_gcr', 0.30000001192092896 )
    subarray3_soiling =[ 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5 ];
    ssc.data_set_array( data, b'subarray3_soiling',  subarray3_soiling);
    ssc.data_set_number( data, b'subarray3_mod_orient', 0 )
    ssc.data_set_number( data, b'subarray3_nmodx', 9 )
    ssc.data_set_number( data, b'subarray3_nmody', 2 )
    ssc.data_set_number( data, b'subarray3_backtrack', 0 );
if nb_roofS>3 :

    text3 = open(matrix_shading_SAM[3], "r")
    text3 = ''.join([i for i in text3])
    text3 = text3.replace(",9,18", "0,9,18")
    x = open('data/output/matrix_shading_SAM_in4.csv', "w")
    x.writelines(text3)
    x.close()
```

```
    matrix_shading_SAM_in4 = 'data/output/matrix_shading_SAM_in4.csv'


    ssc.data_set_number( data, b'subarray4_enable', 1 )
    ssc.data_set_number( data, b'subarray4_modules_per_string', 7 )
    ssc.data_set_number( data, b'subarray4_nstrings', int(nb_panels[3]/7) )
    ssc.data_set_number( data, b'subarray4_mppt_input', 1 )
    ssc.data_set_number( data, b'subarray4_tilt', tilt_SAM[3] )
    ssc.data_set_number( data, b'subarray4_tilt_eq_lat', 0 )
    ssc.data_set_number( data, b'subarray4_azimuth',azimuth_SAM[3] )
    ssc.data_set_number( data, b'subarray4_track_mode', 0 )
    ssc.data_set_number( data, b'subarray4_rotlim', 45 )
    ssc.data_set_number( data, b'subarray4_shade_mode', 0 )
    ssc.data_set_number( data, b'subarray4_gcr', 0.30000001192092896 )
    ssc.data_set_number( data, b'subarray4_shading:string_option', -1 )
    #subarray4_shading_azal = matrix_shading_SAM[3];
    #ssc.data_set_matrix( data, b'subarray4_shading:azal', subarray4_shading_azal )
    subarray4_shading_azal = matrix_shading_SAM_in4
    ssc.data_set_matrix_from_csv(data, b'subarray4_shading:azal', subarray4_shading_azal);
    subarray4_soiling =[ 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5 ];
    ssc.data_set_array( data, b'subarray4_soiling',  subarray4_soiling);
    ssc.data_set_number( data, b'subarray4_mod_orient', 0 )
    ssc.data_set_number( data, b'subarray4_nmodx', 9 )
    ssc.data_set_number( data, b'subarray4_nmody', 2 )
    ssc.data_set_number( data, b'subarray4_backtrack', 0 )
else :
    ssc.data_set_number( data, b'subarray4_enable', 0 )
    ssc.data_set_number( data, b'subarray4_modules_per_string', 0 )
    ssc.data_set_number( data, b'subarray4_nstrings', 0 )
    ssc.data_set_number( data, b'subarray4_mppt_input', 1 )
    ssc.data_set_number( data, b'subarray4_tilt', 0)
    ssc.data_set_number( data, b'subarray4_tilt_eq_lat', 0 )
    ssc.data_set_number( data, b'subarray4_azimuth', 0 )
    ssc.data_set_number( data, b'subarray4_track_mode', 0 )
    ssc.data_set_number( data, b'subarray4_rotlim', 45 )
    ssc.data_set_number( data, b'subarray4_shade_mode', 0 )
    ssc.data_set_number( data, b'subarray4_gcr', 0.30000001192092896 )
    subarray4_soiling =[ 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5 ];
    ssc.data_set_array( data, b'subarray4_soiling',  subarray4_soiling);
    ssc.data_set_number( data, b'subarray4_mod_orient', 0 )
    ssc.data_set_number( data, b'subarray4_nmodx', 9 )
    ssc.data_set_number( data, b'subarray4_nmody', 2 )
    ssc.data_set_number( data, b'subarray4_backtrack', 0 )
ssc.data_set_number( data, b'module_model', 1 )
ssc.data_set_number( data, b'module_aspect_ratio', 1.7000000476837158 )
ssc.data_set_number( data, b'spe_area', 0.74074000120162964 )
ssc.data_set_number( data, b'spe_rad0', 200 )
ssc.data_set_number( data, b'spe_rad1', 400 )
ssc.data_set_number( data, b'spe_rad2', 600 )
ssc.data_set_number( data, b'spe_rad3', 800 )
ssc.data_set_number( data, b'spe_rad4', 1000 )
ssc.data_set_number( data, b'spe_eff0', 13.5 )
ssc.data_set_number( data, b'spe_eff1', 13.5 )
ssc.data_set_number( data, b'spe_eff2', 13.5 )
ssc.data_set_number( data, b'spe_eff3', 13.5 )
ssc.data_set_number( data, b'spe_eff4', 13.5 )
ssc.data_set_number( data, b'spe_reference', 4 )
ssc.data_set_number( data, b'spe_module_structure', 0 )
ssc.data_set_number( data, b'spe_a', -3.559999942779541 )
ssc.data_set_number( data, b'spe_b', -0.075000002980232239 )
ssc.data_set_number( data, b'spe_dT', 3 )
ssc.data_set_number( data, b'spe_temp_coeff', -0.5 )
ssc.data_set_number( data, b'spe_fd', 1 )
ssc.data_set_number( data, b'spe_vmp', 30 )
ssc.data_set_number( data, b'spe_voc', 36 )
ssc.data_set_number( data, b'spe_is_bifacial', 0 )
```

```
ssc.data_set_number( data, b'spe_bifacial_transmission_factor', 0.013000000268220901 )
ssc.data_set_number( data, b'spe_bifaciality', 0.64999997615814209 )
ssc.data_set_number( data, b'spe_bifacial_ground_clearance_height', 1 )
ssc.data_set_number( data, b'cec_area', area_PV )
ssc.data_set_number( data, b'cec_a_ref', 2.4203310012817383 )
ssc.data_set_number( data, b'cec_adjust', 4.8697919845581055 )
ssc.data_set_number( data, b'cec_alpha_sc', 0.0024920001160353422 )
ssc.data_set_number( data, b'cec_beta_oc', -0.16975000500679016 )
ssc.data_set_number( data, b'cec_gamma_r', -0.31000000238418579 )
ssc.data_set_number( data, b'cec_i_l_ref', 6.2368049621582031 )
ssc.data_set_number( data, b'cec_i_mp_ref', 5.8499999046325684 )
ssc.data_set_number( data, b'cec_i_o_ref', 3.988314019320871e-12 )
ssc.data_set_number( data, b'cec_i_sc_ref', 6.2300000190734863 )
ssc.data_set_number( data, b'cec_n_s', 96 )
ssc.data_set_number( data, b'cec_r_s', 0.49938899278640747 )
ssc.data_set_number( data, b'cec_r_sh_ref', 457.185302734375 )
ssc.data_set_number( data, b'cec_t_noct', 46.400001525878906 )
ssc.data_set_number( data, b'cec_v_mp_ref', 57.299999237060547 )
ssc.data_set_number( data, b'cec_v_oc_ref', 67.900001525878906 )
ssc.data_set_number( data, b'cec_temp_corr_mode', 0 )
ssc.data_set_number( data, b'cec_is_bifacial', 0 )
ssc.data_set_number( data, b'cec_bifacial_transmission_factor', 0.013000000268220901 )
ssc.data_set_number( data, b'cec_bifaciality', 0.64999997615814209 )
ssc.data_set_number( data, b'cec_bifacial_ground_clearance_height', 1 )
ssc.data_set_number( data, b'cec_standoff', 6 )
ssc.data_set_number( data, b'cec_height', 0 )
ssc.data_set_number( data, b'cec_mounting_config', 0 )
ssc.data_set_number( data, b'cec_heat_transfer', 0 )
ssc.data_set_number( data, b'cec_mounting_orientation', 0 )
ssc.data_set_number( data, b'cec_gap_spacing', 0.05000000074505806 )
ssc.data_set_number( data, b'cec_module_width', 1 )
ssc.data_set_number( data, b'cec_module_length', 1.6310000419616699 )
ssc.data_set_number( data, b'cec_array_rows', 1 )
ssc.data_set_number( data, b'cec_array_cols', 10 )
ssc.data_set_number( data, b'cec_backside_temp', 20 )
ssc.data_set_number(data, b'cec_transient_thermal_model_unit_mass', 11.091900000000001)
ssc.data_set_number( data, b'6par_celltech', 1 )
ssc.data_set_number( data, b'6par_vmp', 30 )
ssc.data_set_number( data, b'6par_imp', 6 )
ssc.data_set_number( data, b'6par_voc', 37 )
ssc.data_set_number( data, b'6par_isc', 7 )
ssc.data_set_number( data, b'6par_bvoc', -0.10999999940395355 )
ssc.data_set_number( data, b'6par_aisc', 0.0040000001899898052 )
ssc.data_set_number( data, b'6par_gpmp', -0.40999999642372131 )
ssc.data_set_number( data, b'6par_nser', 60 )
ssc.data_set_number( data, b'6par_area', 1.2999999523162842 )
ssc.data_set_number( data, b'6par_tnoct', 46 )
ssc.data_set_number( data, b'6par_standoff', 6 )
ssc.data_set_number( data, b'6par_mounting', 0 )
ssc.data_set_number( data, b'6par_is_bifacial', 0 )
ssc.data_set_number( data, b'6par_bifacial_transmission_factor', 0.013000000268220901 )
ssc.data_set_number( data, b'6par_bifaciality', 0.64999997615814209 )
ssc.data_set_number( data, b'6par_bifacial_ground_clearance_height', 1 )
ssc.data_set_number( data, b'snl_module_structure', 0 )
ssc.data_set_number( data, b'snl_a', -3.619999885559082 )
ssc.data_set_number( data, b'snl_b', -0.075000002980232239 )
ssc.data_set_number( data, b'snl_dtc', 3 )
ssc.data_set_number( data, b'snl_ref_a', -3.619999885559082 )
ssc.data_set_number( data, b'snl_ref_b', -0.075000002980232239 )
ssc.data_set_number( data, b'snl_ref_dT', 3 )
ssc.data_set_number( data, b'snl_fd', 1 )
ssc.data_set_number( data, b'snl_a0', 0.94045001268386841 )
ssc.data_set_number( data, b'snl_a1', 0.052641000598669052 )
ssc.data_set_number( data, b'snl_a2', -0.0093897003680467606 )
ssc.data_set_number( data, b'snl_a3', 0.00072622997686266899 )
ssc.data_set_number( data, b'snl_a4', -1.9937999240937643e-05 )
```

```
    ssc.data_set_number( data, b'snl_aimp', -0.0003800000122282654 )
    ssc.data_set_number( data, b'snl_aisc', 0.00060999998822808266 )
    ssc.data_set_number( data, b'snl_area', 1.2439999580383301 )
    ssc.data_set_number( data, b'snl_b0', 1 )
    ssc.data_set_number( data, b'snl_b1', -0.0024379999376833439 )
    ssc.data_set_number( data, b'snl_b2', 0.00031030000536702573 )
    ssc.data_set_number( data, b'snl_b3', -1.2460000107239466e-05 )
    ssc.data_set_number( data, b'snl_b4', 2.1099999969464989e-07 )
    ssc.data_set_number( data, b'snl_b5', -1.3600000015046021e-09 )
    ssc.data_set_number( data, b'snl_bvmpo', -0.13899999856948853 )
    ssc.data_set_number( data, b'snl_bvoco', -0.13600000739097595 )
    ssc.data_set_number( data, b'snl_c0', 1.0039000511169434 )
    ssc.data_set_number( data, b'snl_c1', -0.0038999998942017555 )
    ssc.data_set_number( data, b'snl_c2', 0.29106599092483521 )
    ssc.data_set_number( data, b'snl_c3', -4.7354598045349121 )
    ssc.data_set_number( data, b'snl_c4', 0.99419999122619629 )
    ssc.data_set_number( data, b'snl_c5', 0.0057999999262392521 )
    ssc.data_set_number( data, b'snl_c6', 1.0722999572753906 )
    ssc.data_set_number( data, b'snl_c7', -0.072300001978874207 )
    ssc.data_set_number( data, b'snl_impo', 5.25 )
    ssc.data_set_number( data, b'snl_isco', 5.75 )
    ssc.data_set_number( data, b'snl_ixo', 5.6500000953674316 )
    ssc.data_set_number( data, b'snl_ixxo', 3.8499999046325684 )
    ssc.data_set_number( data, b'snl_mbvmp', 0 )
    ssc.data_set_number( data, b'snl_mbvoc', 0 )
    ssc.data_set_number( data, b'snl_n', 1.2209999561309814 )
    ssc.data_set_number( data, b'snl_series_cells', 72 )
    ssc.data_set_number( data, b'snl_vmpo', 40 )
    ssc.data_set_number( data, b'snl_voco', 47.700000762939453 )
    ssc.data_set_number( data, b'sd11par_nser', 116 )
    ssc.data_set_number( data, b'sd11par_area', 0.72000002861022949 )
    ssc.data_set_number( data, b'sd11par_AMa0', 0.94169998168945312 )
    ssc.data_set_number( data, b'sd11par_AMa1', 0.065159998834133148 )
    ssc.data_set_number( data, b'sd11par_AMa2', -0.020220000296831131 )
    ssc.data_set_number( data, b'sd11par_AMa3', 0.0021899999119341373 )
    ssc.data_set_number( data, b'sd11par_AMa4', -9.1000001702923328e-05 )
    ssc.data_set_number( data, b'sd11par_glass', 0 )
    ssc.data_set_number( data, b'sd11par_tnoct', 44.900001525878906 )
    ssc.data_set_number( data, b'sd11par_standoff', 6 )
    ssc.data_set_number( data, b'sd11par_mounting', 0 )
    ssc.data_set_number( data, b'sd11par_Vmp0', 64.599998474121094 )
    ssc.data_set_number( data, b'sd11par_Imp0', 1.0500000715255737 )
    ssc.data_set_number( data, b'sd11par_Voc0', 87 )
    ssc.data_set_number( data, b'sd11par_Isc0', 1.1799999475479126 )
    ssc.data_set_number( data, b'sd11par_alphaIsc', 0.00047200100379996002 )
    ssc.data_set_number( data, b'sd11par_n', 1.4507100582122803 )
    ssc.data_set_number( data, b'sd11par_Il', 1.1895099878311157 )
    ssc.data_set_number( data, b'sd11par_Io', 2.0852199966725493e-09 )
    ssc.data_set_number( data, b'sd11par_Egref', 0.73766797780990601 )
    ssc.data_set_number( data, b'sd11par_d1', 13.550399780273438 )
    ssc.data_set_number( data, b'sd11par_d2', -0.076973497867584229 )
    ssc.data_set_number( data, b'sd11par_d3', 0.23732699453830719 )
    ssc.data_set_number( data, b'sd11par_c1', 1930.1500244140625 )
    ssc.data_set_number( data, b'sd11par_c2', 474.6400146484375 )
    ssc.data_set_number( data, b'sd11par_c3', 1.4874600172042847 )
    #Info about inverter
    ssc.data_set_number( data, b'inverter_model', 0 )
#0=cec,1=datasheet,2=partload,3=coefficientgenerator,4=PVYield
    #Values coming from the characteristics of the inverter : Inv_snl values
    ssc.data_set_number( data, b'mppt_low_inverter', 100 )
    ssc.data_set_number( data, b'mppt_hi_inverter', 480 )
    ssc.data_set_number( data, b'inv_snl_c0', -3.0813801004114794e-06 )
    ssc.data_set_number( data, b'inv_snl_c1', -4.8000001697801054e-05 )
    ssc.data_set_number( data, b'inv_snl_c2', 0.00012399999832268804 )
    ssc.data_set_number( data, b'inv_snl_c3', -0.0016319999704137444 )
    ssc.data_set_number( data, b'inv_snl_paco', 3850 )
```

```
ssc.data_set_number( data, b'inv_snl_pdco', 3964.40576171875 )
ssc.data_set_number( data, b'inv_snl_pnt', 1.1549999713897705 )
ssc.data_set_number( data, b'inv_snl_pso', 17.885602951049805 )
ssc.data_set_number( data, b'inv_snl_vdco', 400 )
ssc.data_set_number( data, b'inv_snl_vdcmax', 480 )
inv_tdc_cec_db = [[ 1,   52.799999237060547,   -0.020999999716877937 ]]
# 52.8 = Tstart, -0.021 = slope (%/ºC)
ssc.data_set_matrix( data, b'inv_tdc_cec_db', inv_tdc_cec_db )
ssc.data_set_number( data, b'adjust:constant', 0 )
ssc.data_set_number( data, b'dc_adjust:constant', 0 )
dc_adjust_periods = [[ 0,   0,   0 ]];
ssc.data_set_matrix( data, b'dc_adjust:periods', dc_adjust_periods )
module = ssc.module_create(b'pvsamv1')
ssc.module_exec_set_print( 0 );
if ssc.module_exec(module, data) == 0:
    print ('pvsamv1 simulation error')
    idx = 1
    msg = ssc.module_log(module, 0)
    while (msg != None):
        print ('    : ' + msg.decode("utf - 8"))
        msg = ssc.module_log(module, idx)
        idx = idx + 1
    SystemExit( "Simulation Error" );
ssc.module_free(module)
# PRINT THE INTERESTING VALUES :
annual_energy = ssc.data_get_number(data, b'annual_energy');
print('area_one_module = ', area_one_module, file=f)
print('Number of panels = ', nb_panels, file=f)
print('Tilt of solar panels= ', tilt_SAM, file=f)
print('Tilt of rooftop(s)= ', tiltS, file=f)
print('Azimuth of PV = ', azimuth_SAM, file=f)
print('Azimuth of rootop(s) = ', azimuthS, file=f)
print('Area of rooftop(s) = ', area_SAM, file=f)
print('Number of inverters = ', nb_inverters, file=f)
print('Nameplate capacity = %s kWdc' %(dc_system_capacity), file=f)
print ('Annual energy (year 1) = %s kWh' % (annual_energy), file=f)
monthly = ssc.data_get_array( data, b'monthly_energy' );
print ('Monthly energy (year 1) = ', monthly, file=f)
capacity_factor = ssc.data_get_number(data, b'capacity_factor');
print ('Capacity factor (year 1) = ', capacity_factor, file=f)
kwh_per_kw = ssc.data_get_number(data, b'kwh_per_kw');
print ('Energy yield (year 1) = %s kWh/kW' %(kwh_per_kw), file=f)
performance_ratio = ssc.data_get_number(data, b'performance_ratio');
print ('Performance ratio (year 1) = ', performance_ratio, file=f)
gen= ssc.data_get_array(data, b'gen')
radiation_after_shading = ssc.data_get_array(data, b'poa_shaded')
radiation_before_shading = ssc.data_get_array(data, b'poa_nom')
gen_new=[]

for element in gen:
    if element < 0:
      element = 0
    gen_new.append(float(element))
print(gen_new)
x=gen_new

mat_gen = []
while x != []:
    mat_gen.append(x[:1])
    x = x[1:]

mat_radiation_after_shading =[]
while radiation_after_shading != []:
    mat_radiation_after_shading.append(radiation_after_shading[:1])
    radiation_after_shading = radiation_after_shading[1:]
```

```
    mat_radiation_before_shading =[]
    while radiation_before_shading != []:
        mat_radiation_before_shading.append(radiation_before_shading[:1])
        radiation_before_shading = radiation_before_shading[1:]


    df_gen = pd.DataFrame(mat_gen, columns=['System Power generated (kW)'],index=list_of_hours)
    df_rad = pd.DataFrame(mat_radiation_after_shading, columns=['Radiation after shading (kW)'],
index=list_of_hours)
    df_radnom = pd.DataFrame(mat_radiation_before_shading, columns=['Radiation before shading (kW)'],
index=list_of_hours)

    df_gen.to_csv('data/output/Hourly_generation.csv',float_format='%.2f') # rounded to two decimals
    df_rad.to_csv('data/output/Hourly_radiation_after_shading.csv', float_format='%.2f')  # rounded to two
decimals
    df_radnom.to_csv('data/output/Hourly_radiation_before_shading.csv', float_format='%.2f')  # rounded to two
decimals

    #   df_gen_1 = pd.DataFrame(mat_gen)
    #   df_rad_1 = pd.DataFrame(mat_radiation_after_shading)

    #  df_gen_1.to_csv('data/output/Hourly_generation_list.csv',float_format='%.2f') # rounded to two decimals
    #  df_rad_1.to_csv('data/output/Hourly_radiation_after_shading_list.csv', float_format='%.2f')  # rounded to
two decimals

    print ('System power generated (kW) = ', mat_gen ,file=f)

    return(gen)
    #ssc.data_free(data);
    f.close()


    #results_test('data/output','tmy.epw', 1.393,  2.1487998591855586, 1,'shading.csv',
335.205,381.96574164317633, 1.631,41.512)




    #print(lista_de_horas)

if __name__ == '__main__':
    """
        @input_1: List of tuples that represent the coords of the polygon. [(lat1,long1),....]
        @input_2: File.laz # Before Execution -> docker cp <file> <docker_id>:/solar_potencial_estimation/data
http://centrodedescargas.cnig.es/CentroDescargas/catalogo.do?Serie=LIDAR
    """
    t1 = time.time()
    # Args
    # Input 1: List of coords
    import re

    tuple_coords_polygon = eval(sys.argv[1])

    # Input 2: List of Urls
    laz_path = "/solar_potencial_estimation/data/"

    laz_files = ""
    for file in os.listdir(laz_path):
        if file.endswith('.laz'):
            laz_files += " -i " + '/solar_potencial_estimation/data/' + file

    # Merge Laz
    args = ['wine /LAStools/bin/lasmerge %s -o /solar_potencial_estimation/data/joined_laz.laz' % laz_files]
    proc = subprocess.Popen(args, stdout=subprocess.PIPE, stderr=subprocess.PIPE, shell=True)
    output, error = proc.communicate()
```

```
    laz_file = "data/joined_laz.laz"

    polygon = create_polygon(tuple_coords_polygon)
    b = polygon.bounds

    create_output_folder(repertoire)

    pipeline_crop_building = create_building_laz(polygon, b, laz_file)

    n_points = pipeline_crop_building.execute()

    # Load Pipeline output in  a dataframe
    # n_points_m2 = n_points / polygon.area
    arr = pipeline_crop_building.arrays[0]
    description = arr.dtype.descr
    cols = [col for col, __ in description]
    df = pd.DataFrame({col: arr[col] for col in cols})

    # # To make the x,y,z values starting from 0 (for the plot and the classification)
    df['X_0'] = df['X']
    df['Y_0'] = df['Y']
    df['Z_0'] = df['Z']
    df['X'] = df['X'] - df['X_0'].min()
    df['Y'] = df['Y'] - df['Y_0'].min()
    df['Z'] = df['Z'] - df['Z_0'].min()

    # Modification of the point density
    n_points_m2 = (n_points - sum(df['Classification'] == 12)) / polygon.area  # new point density

    # TEST WITH SEVERAL ITERATIONS
    df['tree_potential'] = (df['Classification'] == 6) & (df['Z'] >= 2) & (df['Eigenvalue0'] > .05) & (
            df['NumberOfReturns'] - df['ReturnNumber'] >= 1)
    z_threshold = 5  # begin with 5m
    df['building_potential'] = (df['Classification'] == 6) & (df['Z'] > z_threshold) & (~df['tree_potential'])

    while len(df.loc[df['building_potential'], ['X']]) < 0.4 * len(df['X']):
        if z_threshold < 0:
            break
        else:
            df['building_potential'] = (df['Classification'] == 6) & (df['Z'] > z_threshold - 1) & (
                ~df['tree_potential'])
            z_threshold = z_threshold - 1
    # tree_potential is used to try to avoid that tree points are classified as building points
    if z_threshold < 0:
        z_threshold = df['Z'].mean() - 1
    df['building_potential'] = (df['Classification'] == 6) & (df['Z'] > z_threshold) & (~df['tree_potential'])

    # Initiailization
    p = []  # nbre of point_list
    nb = []  # nbre of roofs
    nb_r = []  # nbre of points per roofs
    list_index = []
    std_filt = []
    length = sum(df['building_potential'])

    for comp in range(10):  # number of RANSAC iterations
        point_list = df.loc[df['building_potential'], ['X', 'Y', 'Z']].copy()
        n = len(point_list)
        nb_max_roof_desired = 4
        nb_roof = 0  # number of detected rooftops
        nb_roof2 = 0  # iterator
        nb_max_roof = 3 * n
        PN_S = 0.7 * n / nb_max_roof_desired  # 0.7*n/nb_roof_desired #random, opt possible -> minimal
number of points for creating a plane
        roof = [[] for i in
                range(
```

```
            nb_max_roof + 1)]  # having the coordinates for the points in each plane (a list of points per
plane)
        nb_point_roof = [0 for i in range(nb_max_roof + 1)]  # number of points per plane
        coord_plane = [[] for i in range(nb_max_roof + 1)]  # equation of each plane
        index_roof = [[] for i in range(nb_max_roof + 1)]  # index of the points in each plane
        std = [[] for i in range(nb_max_roof + 1)]
        while len(point_list) > 0.05 * n and nb_roof2 < nb_max_roof:  # 0.05*n is arbitrary
            bestSupport = 0
            bestPlane = [0, 0, 0, 0]
            bestStd = np.inf
            i = 0
            best = []
            bestIndex = []
            eps = 0.1  # proba that a point is an outlier
            alpha = 0.97  # must be between 0.9 and 0.99
            N = np.round(np.log(1 - alpha) / (np.log(1 - (1 - eps) ** 3)))  # formula coming from a paper
            t = 20  # threshold for the euclidean distance
            while (i <= N):
                rand1, rand2, rand3 = random.choice(point_list.index), random.choice(point_list.index),
random.choice(
                    point_list.index)
                # take 3 index from the index available
                while rand1 == rand2 or rand1 == rand3 or rand2 == rand3:
                    # check that we have 3 different points
                    rand1, rand2, rand3 = random.choice(point_list.index), random.choice(
                        point_list.index), random.choice(
                        point_list.index)
                pt1, pt2, pt3 = point_list.loc[rand1, :].values, point_list.loc[rand2, :].values, point_list.loc[rand3,
                                                                                                                 :].values
                if rand1 != rand2 and rand1 != rand3 and rand2 != rand3:  # just in case of
                    a, b, c, d = pts2plane(pt1, pt2, pt3)  # return the parameters of the created plane
                    dis = []
                    s = []
                    index = []
                    for ind in point_list.index:
                        dis.append(dist2plan(ind, a, b, c, d, point_list))
                        if abs(dist2plan(ind, a, b, c, d, point_list)) <= t:
                            # if the distance of a point is below the threshold t
                            # we take the value and the index
                            s.append(dist2plan(ind, a, b, c, d, point_list))
                            index.append(ind)
                    st = np.std(s)
                    if st < bestStd and len(s) > PN_S:  # to only select the best planes
                        bestSupport = len(s)
                        bestPlane = [a, b, c, d]
                        bestStd = st
                        best = s
                        bestIndex = index
                    i = i + 1

            if len(bestIndex) > 0:  # try to avoid the index_roof issue
                # here we put the values of index, parameters of plane and coordinates of points in the plane
nb_roof
                for ind in point_list.index:
                    if ind in bestIndex:
                        roof[nb_roof].append(point_list.loc[ind, :].values)
                        point_list = point_list.drop(ind)  # we delete the index
                nb_point_roof[nb_roof] = len(bestIndex)
                coord_plane[nb_roof] = bestPlane
                index_roof[nb_roof] = bestIndex
                std[nb_roof] = st
                nb_roof = nb_roof + 1
            nb_roof2 = nb_roof2 + 1
        p.append(len(point_list))
        nb.append(nb_roof)
        nb_point_roof = [nb_point_roof[i] for i in range(len(nb_point_roof)) if nb_point_roof[i] > 0]
```

```
            nb_r.append(nb_point_roof)
            index_roof = [index_roof[i] for i in range(len(nb_point_roof)) if len(index_roof[i]) > 0]
            list_index.append(index_roof)
            std = [std[i] for i in range(len(nb_point_roof)) if len(index_roof[i]) > 0]
            std_filt.append(std)

        # Keep the "best" segmentation, based on standard deviation
        std_RANSAC = np.inf
        index_RANSAC = 0
        for a in range(len(p)):
            mean_std = sum(std_filt[a]) / len(std_filt[a])
            if mean_std < std_RANSAC:  # better standard deviation
                index_RANSAC = a
                std_RANSAC = mean_std

        # Add a column 'roof_id', which gives an id to the points which are part of a plane
        df['roof_id'] = np.nan
        for a in range(nb[index_RANSAC]):
            for ind in list_index[index_RANSAC][a]:
                df['roof_id'][ind] = a  # change the index value by the id of the detected rooftop

        # One df per plane detected, the idea is if I want to get the slope, area, azimut of each part of the rooftops
        df_roof = [[] for i in range(nb[index_RANSAC])]
        for a in range(nb[index_RANSAC]):
            df_roof[a] = df[df['roof_id'] == a]
            df_roof[a] = pd.DataFrame(df_roof[a])

        # Plot the results
        df_final = functools.reduce(lambda left, right: pd.merge(left, right, how='outer'), df_roof)
        df_final.reset_index(drop=True, inplace=True)
        for a in range(len(df_roof)):
            df_roof[a].reset_index(drop=True, inplace=True)

        # found in https://github.com/tudelft3d/Solar3Dcity/blob/master/polygon3dmodule.py, which works on solar
estimation on 3D city models

        # Delete points that have a too low height (to delete woooden hut, terrace, etc)
        for a in range(len(df_roof)):
            df_roof[a] = df_roof[a][df_roof[a]['Z'] > df_final['Z'].mean() - 7]
            if len(df_roof[a]) == 0:
                df_roof.pop(a)

        # Determine the area -> for now, the only option I found is to use the number of points detected
        # Determine also tilt + azimuth
        tilt_beforejoin = [0 for i in range(len(df_roof))]
        area_beforejoin = [0 for i in range(len(df_roof))]
        azimuth_beforejoin = [0 for i in range(len(df_roof))]
        for i in range(len(df_roof)):
            normal = [df_roof[i]['NormalX'].mean(), df_roof[i]['NormalY'].mean(), df_roof[i]['NormalZ'].mean()]
            tilt_beforejoin[i], azimuth_beforejoin[i] = getAngles(normal)
            area_beforejoin[i] = 0.95 * len(df_roof[i]['X']) / n_points_m2  # *0.95 comes from the report "Fixed
parameters"
        area_wholebuilding = [0.95 * len(df_final) / n_points_m2]  # if we want the area of the whole building

        # Join df_roof when tilt<12º or if difference of azimuth <10º :
        df_roof_test = df_roof.copy()
        nb_roof = len(df_roof_test)
        for a in range(nb_roof - 1):
            b = a + 1
            while b < len(df_roof_test):
                if tilt_beforejoin[a] < 12 and tilt_beforejoin[b] < 12 and abs(
                        df_roof[a]['Z'].mean() - df_roof[b]['Z'].mean()) < 5:  # difference of max 5meters for z
                    df_roof_test[a] = functools.reduce(lambda left, right: pd.merge(left, right, how='outer'),
                                                       [df_roof_test[a], df_roof_test[b]])
                    print("Join Tilt")
                    df_roof_test.pop(b)
```

```
            for i in range(len(df_roof_test)):
                normal = [df_roof_test[i]['NormalX'].mean(), df_roof_test[i]['NormalY'].mean(),
                          df_roof_test[i]['NormalZ'].mean()]
                tilt_beforejoin[i], azimuth_beforejoin[i] = getAngles(normal)
        elif abs(azimuth_beforejoin[a] - azimuth_beforejoin[b]) < 10 and abs(
                tilt_beforejoin[a] - tilt_beforejoin[b]) < 5 and abs(
            df_roof[a]['Z'].mean() - df_roof[b]['Z'].mean()) < 5:
            df_roof_test[a] = functools.reduce(lambda left, right: pd.merge(left, right, how='outer'),
                                               [df_roof_test[a], df_roof_test[b]])
            df_roof_test.pop(b)
            for i in range(len(df_roof_test)):
                normal = [df_roof_test[i]['NormalX'].mean(), df_roof_test[i]['NormalY'].mean(),
                          df_roof_test[i]['NormalZ'].mean()]
                tilt_beforejoin[i], azimuth_beforejoin[i] = getAngles(normal)
                print("Join Az")
        else:
            b = b + 1

    # Tilt, azimuth and area after the joining algorithm
    tilt = [0 for i in range(len(df_roof_test))]
    area = [0 for i in range(len(df_roof_test))]
    azimuth = [0 for i in range(len(df_roof_test))]
    for i in range(len(df_roof_test)):
        normal = [df_roof_test[i]['NormalX'].mean(), df_roof_test[i]['NormalY'].mean(),
                  df_roof_test[i]['NormalZ'].mean()]
        tilt[i], azimuth[i] = getAngles(normal)
        area[i] = 0.95 * len(df_roof_test[i]['X']) / n_points_m2  # *0.95 comes from the report "Fixed parameters"

    # Write the characteirstics in a text file :
    with open(repertoire + '/characteristics_building.txt', 'w') as f:
        f.write("Before join: \n")
        f.write("Number of roofs : %s roof(s), Slope : %s º, Azimuth : %s º ,Area rooftops : %s m2 \n" % (
            len(tilt_beforejoin), tilt_beforejoin, azimuth_beforejoin, area_beforejoin))
        f.write("\n")
        f.write("After join: \n")
        f.write("Number of roofs : %s roof(s), Slope : %s º, Azimuth : %s º ,Area rooftops : %s m2" % (
            len(tilt), tilt, azimuth, area))
        f.write("\n")

    # Delete not-suitable roofs :
    # Values coming from“A GIS-based method for identification of wide area rooftop suitability for minimum
size PV systems using LiDAR data and photogrammetry”
    # values for the UK, maybe they should be modified
    # if we look in https://www.sunearthtools.com/dp/tools/pos_sun.php?lang=en, we can keep it or just do
between 280 and 80
    index = []
    df_test = df_roof_test.copy()
    nb_roof = len(df_roof_test)
    i = 0
    while i < len(df_test):
        if tilt[i] > 60:
            # delete this roof of the suitable roofs list
            index.append([i, tilt[i], azimuth[i], area[i]])
            df_test.pop(i)
            tilt.pop(i)
            azimuth.pop(i)
            area.pop(i)
            nb_roof = nb_roof - 1
        elif (azimuth[i] > 270 or azimuth[i] < 90) and (
                tilt[i] > 20):  # or more 300 and 60º according to https://www.gaisma.com/en/location/malaga.html
            # delete this roof of the suitable roofs list
            index.append([i, tilt[i], azimuth[i], area[i]])
            df_test.pop(i)
            tilt.pop(i)
            azimuth.pop(i)
            area.pop(i)
```

```
            nb_roof = nb_roof - 1
            index.append(i)
        else:
            i = i + 1

    # Add this information in the text file :
    with open(repertoire + '/characteristics_building.txt', 'a') as f:
        f.write("\n")
        f.write("Deleted roof(s) : \n")
        f.write("[Number/Index, Slope, Azimuth, Area] in this order : %s  \n" % index)

    # See the differences
    df_final = functools.reduce(lambda left, right: pd.merge(left, right, how='outer'), df_roof)

    df_final.reset_index(drop=True, inplace=True)
    for a in range(len(df_roof)):
        df_roof[a].reset_index(drop=True, inplace=True)

    # Take more area for the shading (I draw a rectangle around the building)
    # for information : b = [minx, miny, maxx, maxy]
    b = polygon.bounds
    las_with_neighborhood = {
        "pipeline": [laz_file,
                     {"type": "filters.crop",
                      'bounds': str(([b[0] - 50, b[2] + 50], [b[1] - 80, b[3] + 20]))},
                     # cover more area in the southern part
                     {
                         "type": "filters.range",
                         "limits": "Classification![7:7]"},
                     {
                         "filename": 'data/cropped_neighborhood.laz'
                     }
                     ]
    }
    # only delete the points already classified as noise points = Class 7
    pipeline_las_with_neighborhood = pdal.Pipeline(json.dumps(las_with_neighborhood))
    pipeline_las_with_neighborhood.validate()
    n_points = pipeline_las_with_neighborhood.execute()

    # Load Pipeline output in python objects
    arr = pipeline_las_with_neighborhood.arrays[
        0]  # modify if we want to use a filtered las file or the original cropped las
    description = arr.dtype.descr
    cols = [col for col, __ in description]
    df2 = pd.DataFrame({col: arr[col] for col in cols})
    # To make the x,y,z values starting from 0
    df2['X_0'] = df2['X']
    df2['Y_0'] = df2['Y']
    df2['Z_0'] = df2['Z']
    df2['X'] = df2['X'] - df2['X_0'].min()
    df2['Y'] = df2['Y'] - df2['Y_0'].min()
    df2['Z'] = df2['Z'] - df2['Z_0'].min()
    df2['Z'].max()

    # Creation of this gap_z because the minimum height for z can be different between only the drawing and
the neighborhood.
    # And so the values of Z is not the same in these 2 dataframes.
    gap_z = (df2[(df2['X_0'] == df_roof[0]['X_0'].iloc[0]) & (df2['Y_0'] == df_roof[0]['Y_0'].iloc[0])].iloc[0]['Z']) - \
            df_roof[0]['Z'].iloc[0]

    # LAS2SHP
    args = [
        'wine /LAStools/bin/las2shp -i /solar_potencial_estimation/data/cropped_neighborhood.laz -single_points
-o /solar_potencial_estimation/data/shape_neighborhood.shp -record_size 10000']
    # the path/folder where is las2shp needs to be modified !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    proc = subprocess.Popen(args, stdout=subprocess.PIPE, stderr=subprocess.PIPE, shell=True)
```

```
    output, error = proc.communicate()

    from qgis.core import QgsApplication, QgsVectorLayer, QgsRasterLayer, QgsRasterBandStats
    from qgis.analysis import QgsNativeAlgorithms
    from processing.core.Processing import Processing

    # export  QT_QPA_PLATFORM=offscreen

    # Initialize QGIS Application and the processing plugin
    os.environ["QT_QPA_PLATFORM"] = "offscreen"
    QgsApplication.setPrefixPath("/usr", False)
    qgs = QgsApplication([], False)
    qgs.initQgis()

    Processing.initialize()
    qgs.processingRegistry().addProvider(QgsNativeAlgorithms())

    # Read shp to obtain layer extent
    shp_file = QgsVectorLayer("/solar_potencial_estimation/data/shape_neighborhood.shp",
                              "cropped_neighborhood", "ogr")

    shp_extension = shp_file.extent()

    # TIN interpolation

    x = "%s,%s,%s,%s" % (
        shp_extension.xMinimum(), shp_extension.xMaximum(), shp_extension.yMinimum(),
shp_extension.yMaximum())

    job2 = Processing.runAlgorithm("qgis:tininterpolation", {
        'INTERPOLATION_DATA': '/solar_potencial_estimation/data/shape_neighborhood.shp::~::1::~::-1::~::0',
        'METHOD': 0, 'EXTENT': x,
        'PIXEL_SIZE': 0.1,
        'OUTPUT': '/solar_potencial_estimation/data/raster_neighborhood.tiff'})

    # Get the minimum height
    raster_file = QgsRasterLayer("/solar_potencial_estimation/data/raster_neighborhood.tiff",
                                 "raster_neighborhood")
    raster_extent = raster_file.extent()
    min_height = raster_file.dataProvider().bandStatistics(1, QgsRasterBandStats.All, raster_extent,
0).minimumValue

    parameters = {
        "layer": "/solar_potencial_estimation/data/raster_neighborhood.tiff",
        "roi_x_max": raster_extent.xMaximum(),
        "roi_x_min": raster_extent.xMinimum(),
        "roi_y_max": raster_extent.yMaximum(),
        "roi_y_min": raster_extent.yMinimum(),
        "roi_rect_Param": {
            "center": [
                raster_extent.center().x(),
                raster_extent.center().y()
            ],
            "width": abs(raster_extent.xMaximum() - raster_extent.xMinimum()),
            "height": abs(raster_extent.yMaximum() - raster_extent.yMinimum()),
            "rotation": 0
        },
        "spacing_mm": 0.5,
        "height": abs(raster_extent.yMaximum() - raster_extent.yMinimum()),
        "width": abs(raster_extent.xMaximum() - raster_extent.xMinimum()),
        "z_scale": 1.0,
        "scale": 999.999978,
        "scale_h": 1000.0000093614896,
        "scale_w": 999.9999459220444,
        "z_inv": False,
        "z_base": min_height,
```

```
    "divideRow": 1,
    "divideCols": 1,
    "projected": False,
    "crs_layer": "",
    "crs_map": ""
}

sys.path.append('/solar_potencial_estimation/DEMto3D-QGIS-Plugin/model_builder')
import Model_Builder, STL_Builder

model = Model_Builder.Model(parameters)
model.run()
matrix_dem = model.get_model()

stl_file = '/solar_potencial_estimation/data/stl_neighborhood.stl'

stl = STL_Builder.STL(parameters, stl_file, model.get_model())
stl.run()

mesh = trimesh.load('data/stl_neighborhood.stl')

# These bounds are used for the shading, because the mesh range is not the same than the df range
xmin, xmax = mesh.bounds[:, 0]
ymin, ymax = mesh.bounds[:, 1]
zmin, zmax = mesh.bounds[:, 2]

# CREATE THE MATRIX OF SHADING : ONE PER DETECTED SUITABLE PLANE

# Initialise values
nb_roof = len(df_roof_test)
matrix = [[] for i in range(nb_roof)]
matrix_shading = [[] for i in range(nb_roof)]
#matrix_3by3 = [[] for i in range(nb_roof)]
#matrix_shading_3by3 = [[] for i in range(nb_roof)]
compteur = 0  # total of points with high shading (until 87°)
outside_polygon = 0  # total of building points outside the polygon : should not happen, but it seems to
come sometimes, because of the estimation of reprojection
matrix_SAM = [[] for i in range(nb_roof)]  # the input for SAM

# Loop to create the shading matrix
# 2 matrix for each roof -> one in a range of 3º of azimuth and 3º of angle of sun = matrix_shading_3by
# and one of range of 9º, to have a better visualisation = matrix_shading

from shapely.geometry import Point

# List of matrix (3x3) and (9x9)
#m3 = []
m9 = []

for i in range(nb_roof):
    list_index = df_roof_test[i].index  # list of the index not yet choosen
    for nb_points in range(min(30, len(df_roof_test[i]))):  # to be sure it works for all size of roofs
        indice = random.choice(list_index)
        mat_stl_3by3 = np.zeros((int(90 / 3) + 1, int(360 / 3) + 1))
        x, y, z = vector(azimuth[i], tilt[i])
        ray_origins = np.array([[df_roof_test[i]['X'][indice] + xmin + 50, df_roof_test[i]['Y'][indice] + ymin + 80,
                                 df_roof_test[i]['Z'][indice] + zmin + gap_z + 0]])
        # + 50 for x and 80 for y because of the dimensions of the 3D model
        # + xmin, ymin, zmin because the mesh and the df begins not from the same values
        pt_test = Point(ray_origins[0][0], ray_origins[0][1])
        p_minx, p_miny = polygon.exterior.coords.xy

        p_minx = min(p_minx)
        p_miny = min(p_miny)
        point1, point2 = pt_test.xy
        point1 = float(point1[0]) + p_minx - 50
```

```
            point2 = float(point2[0]) + p_miny - 80

            pt_test_aux = Point(point1, point2)

            # plt.scatter(point1,point2)
            if pt_test_aux.within(polygon) == False:  # check if points are within polygon
                outside_polygon = outside_polygon + 1
            else:
                for a in range(0, 361, 3):  # or more (60,303,3) to gain time
                    for b in range(0, 90, 3):
                        x, y, z = vector(a, b)
                        ray_directions = np.array([[x, y, z]])
                        loc_intersects, n_intersection, index_intersect = mesh.ray.intersects_location(ray_origins,
                                                                                                       ray_directions)
                        if len(n_intersection) > 0:
                            for j in range(len(n_intersection)):
                                p1 = Point(loc_intersects[j][0] + p_minx - 50, loc_intersects[j][1] + p_miny - 80)
                                if p1.within(polygon) == False:
                                    mat_stl_3by3[b // 3, a // 3] = 100
                # create a second matrix (for better visualisation)
                mat_stl_mean = np.zeros((11, 41))
                for a in range(1, 41):
                    for b in range(1, 11):
                        mat_stl_mean[b, a] = mat_stl_3by3[3 * b - 3:3 * b, 3 * a - 3:3 * a].mean()
                        mat_stl_mean[b, 0] = 9 * b
                test = 0
 #                for a in range(1, int(360 / 3) + 1):
  #                   if mat_stl_3by3[30, a] > 90:  # if there is a point with shading for high angle of sun
   #                      test = 1
                if test < 1:
                    mat_stl_mean[0] = [9 * i for i in range(41)]
                    matrix[i].append(mat_stl_mean)
                    #matrix_3by3[i].append(mat_stl_3by3)
                else:
                    compteur = compteur + 1
            list_index = list_index.drop(indice)  # this indice will not be used one more time

        # Export the results : 3by3 and 9by9
        # 1) 9 by 9 (just for visualization)
        matrix_shading[i] = sum(matrix[i]) / len(matrix[i])
        index_mat = [9 * i for i in range(1, 11)]
        columns_mat = [9 * i for i in range(1, 41)]
        df_mat = pd.DataFrame(matrix_shading[i][1:, 1:], index=index_mat, columns=columns_mat)
        pd.DataFrame(df_mat).to_csv(repertoire + '/shading_matrix_9by9_add0m_' + str(i) + '.csv', sep=',')
        ## 2) by 3 matrix (input for SAM)
        #matrix_shading_3by3[i] = sum(matrix_3by3[i]) / len(matrix_3by3[i])
        #index_mat_3by3 = [3 * i for i in range(31)]
        #columns_mat_3by3 = [3 * i for i in range(121)]
        #df_mat_3by3 = pd.DataFrame(matrix_shading_3by3[i], index=index_mat_3by3,
columns=columns_mat_3by3)
        #pd.DataFrame(df_mat_3by3).to_csv(repertoire + '/shading_matrix_3by3_add0m_' + str(i) + '.csv',
sep=',')

        #m3.append(df_mat_3by3.to_json(orient="records"))
        m9.append(df_mat.to_json(orient="records"))



    print(tilt,nb_roof, area)
    results_test(TMY_filename='tmy.epw', tiltS=tilt,  azimuthS=azimuth,
nb_roofS=nb_roof,matrix_shading_SAM=['data/output/shading_matrix_9by9_add0m_0.csv'], areaS=area)

    print(time.time() - t1)


    exit(0)
```

2 Code for the comparation of the results from the tool with the ones obtained through the SAM software.

The Python code created for comparing the results of the tool with the ones provided by the SAM software has been committed to a GitHub project. What's more, the code is shown as follows:

```
import sys
import csv
import numpy as np

import pandas as pd
import plotly.graph_objects as go
import plotly.express as px

csv.field_size_limit(sys.maxsize)

from csv import reader

import csv

with
open('/Users/mariamaaiques/PycharmProjects/solar_potencial_estimation/COMPARATION/SAM_radiation.cs
v') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    lists_from_csv = []
    for row in csv_reader:
        lists_from_csv.append(row)
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        else:
            line_count += 1
    print(f'Processed {line_count} lines.')
    # print(lists_from_csv)

with
open('/Users/mariamaaiques/PycharmProjects/solar_potencial_estimation/COMPARATION/SAM_generation.c
sv') as csv_file3:
    csv_reader3 = csv.reader(csv_file3, delimiter=',')
    line_count3 = 0
    lists_from_csv3 = []
    for row in csv_reader3:
        lists_from_csv3.append(row)
        if line_count3 == 0:
            print(f'Column names are {", ".join(row)}')
            line_count3 += 1
        else:
            line_count3 += 1
    print(f'Processed {line_count3} lines.')
    # print(lists_from_csv)

def column(matrix, i):
    return [row[i] for row in matrix]


SAM_RADIATION_VALUES_0 = column(lists_from_csv, 1)
SAM_GENERATION_VALUES_0 = column(lists_from_csv3, 1)
```

```
SAM_RADIATION_VALUES = []
SAM_GENERATION_VALUES = []
line_counter = 0
line_counter3 = 0

for element in SAM_RADIATION_VALUES_0:
    if line_counter > 0:
        SAM_RADIATION_VALUES.append(float(element))
    line_counter = line_counter+1

for element in SAM_GENERATION_VALUES_0:
    if line_counter3 > 0:
        SAM_GENERATION_VALUES.append(float(element))
    line_counter3 = line_counter3+1

with
open('/Users/mariamaaiques/PycharmProjects/solar_potencial_estimation/COMPARATION/Hourly_radiation_
after_shading.csv') as csv_file_2:
    csv_reader_2 = csv.reader(csv_file_2, delimiter=',')
    line_count = 0
    lists_from_csv_2 = []
    for row in csv_reader_2:
        lists_from_csv_2.append(row)
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        else:
            line_count += 1
    print(f'Processed {line_count} lines.')
# print(lists_from_csv_2)

with
open('/Users/mariamaaiques/PycharmProjects/solar_potencial_estimation/COMPARATION/Hourly_generation
.csv') as csv_file_4:
    csv_reader_4 = csv.reader(csv_file_4, delimiter=',')
    line_count = 0
    lists_from_csv_4 = []
    for row in csv_reader_4:
        lists_from_csv_4.append(row)
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        else:
            line_count += 1
    print(f'Processed {line_count} lines.')
# print(lists_from_csv_4)

PYSAM_RADIATION_VALUES_0 = column(lists_from_csv_2, 1)
PYSAM_GENERATION_VALUES_0 = column(lists_from_csv_4, 1)
PYSAM_RADIATION_VALUES = []
PYSAM_GENERATION_VALUES = []
line_counter = 0
line_counter4 = 0

for element in PYSAM_RADIATION_VALUES_0:
    if line_counter > 0:
        element=float(element)
        if element < 0:
            element = 0
        PYSAM_RADIATION_VALUES.append(float(element))
    line_counter = line_counter+1
print("elements", line_counter)

for element in PYSAM_GENERATION_VALUES_0:
    if line_counter4 > 0:
        element=float(element)
```

```
        if element < 0:
            element = 0
        PYSAM_GENERATION_VALUES.append(float(element))
    line_counter4 = line_counter4+1
print("elements", line_counter4)

df = pd.DataFrame({'Value1': SAM_RADIATION_VALUES, 'Value2': PYSAM_RADIATION_VALUES})
date = pd.date_range('2022-01-01', periods=8760, freq='H')
print(date)
date = pd.DataFrame(date, columns=['Date'])

df = pd.concat([date, df], axis=1)
print(df)

fig = go.Figure()
fig.add_trace(go.Scatter(x=df["Date"], y=df["Value1"], name="SAM"))
fig.add_trace(go.Scatter(x=df["Date"], y=df["Value2"], name="PYSAM"))
fig.update_layout(hovermode="x unified")
fig.update_layout(
    title="RADIATION AFTER SHADING SAM vs PYSAM",
    xaxis_title="Time",
    yaxis_title="Radiation after shading [kW]",
    legend=dict(
        yanchor="top",
        y=0.99,
        xanchor="left",
        x=0.01))
fig.show()
fig.write_html("/Users/mariamaaiques/PycharmProjects/solar_potencial_estimation/COMPARATION/radiation
SAMandPYSAM.html")

fig=px.scatter(x=SAM_RADIATION_VALUES, y=PYSAM_RADIATION_VALUES, opacity=0.65,
trendline="ols", trendline_color_override='darkblue')
fig.update_layout(
    title="RADIATION AFTER SHADING SAM vs PYSAM",
    xaxis_title="Hourly SAM radiation [kW]",
    yaxis_title="Hourly radiation after shading [kW]",
    legend=dict(
        yanchor="top",
        y=0.99,
        xanchor="left",
        x=0.01))
fig.show()
fig.write_html("/Users/mariamaaiques/PycharmProjects/solar_potencial_estimation/COMPARATION/REG_radi
ationSAMandPYSAM.html")

df = pd.DataFrame({'Value1': SAM_GENERATION_VALUES, 'Value2': PYSAM_GENERATION_VALUES})
date = pd.date_range('2022-01-01', periods=8760, freq='H')
print(date)
date = pd.DataFrame(date, columns=['Date'])

df = pd.concat([date, df], axis=1)
print(df)

fig = go.Figure()
fig.add_trace(go.Scatter(x=df["Date"], y=df["Value1"], name="SAM"))
fig.add_trace(go.Scatter(x=df["Date"], y=df["Value2"], name="PYSAM"))
fig.update_layout(hovermode="x unified")
fig.update_layout(
    title="POWER GENERATION SAM vs PYSAM",
    xaxis_title="Time",
    yaxis_title="Generation [kW]",
    legend=dict(
        yanchor="top",
        y=0.99,
        xanchor="left",
```

```
        x=0.01))
fig.show()
fig.write_html("/Users/mariamaaiques/PycharmProjects/solar_potencial_estimation/COMPARATION/generatio
nSAMandPYSAM.html")

fig=px.scatter(x=SAM_GENERATION_VALUES, y=PYSAM_GENERATION_VALUES, opacity=0.65,
trendline="ols", trendline_color_override='darkblue')
fig.update_layout(
    title="Power generation SAM vs PYSAM",
    xaxis_title="Hourly SAM generation [kW]",
    yaxis_title="Hourly PYSAM generation [kW]",
    legend=dict(
        yanchor="top",
        y=0.99,
        xanchor="left",
        x=0.01))
fig.show()
fig.write_html("/Users/mariamaaiques/PycharmProjects/solar_potencial_estimation/COMPARATION/REG_gen
erationSAMandPYSAM.html")
```

3 Code for the validation of the tool for the estimation of rooftop solar photovoltaic generation

The Python code created for validating the results has been committed to a GitHub project. What's more, the code is shown as follows:

```
import sys
import csv
import numpy as np

import pandas as pd
import plotly.graph_objects as go

csv.field_size_limit(sys.maxsize)

from csv import reader

import csv

#area de superficie amb panells
area=285.25

with
open('/Users/mariamaaiques/PycharmProjects/solar_potencial_estimation/COMPARATION/PVGIS_radiation.c
sv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    lists_from_csv = []
    for row in csv_reader:
        lists_from_csv.append(row)
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        else:
            line_count += 1
    print(f'Processed {line_count} lines.')
    # print(lists_from_csv)

ini_array = np.array(lists_from_csv)
#z = [False, True, True, False, True, True, True, True, True, True]
#z = [False, False, True, True, True, True, True, True]
z = [False, False, True, True, True, True]

lists_from_csv = ini_array[:, np.logical_not(z)]
print(lists_from_csv)

def column(matrix, i):
    return [row[i] for row in matrix]


PVGIS_RADIATION_VALUES_0 = column(lists_from_csv, 1)

PVGIS_RADIATION_VALUES = []
line_counter = 0

for element in PVGIS_RADIATION_VALUES_0:
    if line_counter > 0:
        element=float(element)*area/1000
        PVGIS_RADIATION_VALUES.append(element)
    line_counter = line_counter+1
```

```
with
open('/Users/mariamaaiques/PycharmProjects/solar_potencial_estimation/COMPARATION/Hourly_radiation_
after_shading.csv') as csv_file_2:
    csv_reader_2 = csv.reader(csv_file_2, delimiter=',')
    line_count = 0
    lists_from_csv_2 = []
    for row in csv_reader_2:
        lists_from_csv_2.append(row)
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        else:
            line_count += 1
    print(f'Processed {line_count} lines.')
# print(lists_from_csv_2)

PYSAM_RADIATION_VALUES_0 = column(lists_from_csv_2, 1)

PYSAM_RADIATION_VALUES = []
line_counter = 0

for element in PYSAM_RADIATION_VALUES_0:
    if line_counter > 0:
        element=float(element)
        if element < 0:
            element = 0
        PYSAM_RADIATION_VALUES.append(float(element))
    line_counter = line_counter+1

print("elements", line_counter)


df = pd.DataFrame({'Value1': PVGIS_RADIATION_VALUES, 'Value2': PYSAM_RADIATION_VALUES})
date = pd.date_range('2022-01-01', periods=8760, freq='H')
#print(date)
date = pd.DataFrame(date, columns=['Date'])

df = pd.concat([date, df], axis=1)
#print(df)

fig = go.Figure()
fig.add_trace(go.Scatter(x=df["Date"], y=df["Value1"], name="PVGIS"))
fig.add_trace(go.Scatter(x=df["Date"], y=df["Value2"], name="PYSAM"))
fig.update_layout(hovermode="x unified")
fig.update_layout(
    title="RADIATION AFTER SHADING PVGIS vs PYSAM",
    xaxis_title="Time",
    yaxis_title="Radiation after shading [kW]",
    legend=dict(
        yanchor="top",
        y=0.99,
        xanchor="left",
        x=0.01))
fig.show()
fig.write_html("/Users/mariamaaiques/PycharmProjects/solar_potencial_estimation/COMPARATION/radiation
PVGISandPYSAM.html")
import plotly.express as px

fig=px.scatter(x=PVGIS_RADIATION_VALUES, y=PYSAM_RADIATION_VALUES, opacity=0.65,
trendline="ols", trendline_color_override='darkblue')
fig.update_layout(
    title="RADIATION AFTER SHADING PVGIS vs PYSAM",
    xaxis_title="Hourly PVGIS radiation [kW]",
    yaxis_title="Hourly radiation after shading [kW]",
    legend=dict(
        yanchor="top",
```

```
        y=0.99,
        xanchor="left",
        x=0.01))
fig.show()
fig.write_html("/Users/mariamaaiques/PycharmProjects/solar_potencial_estimation/COMPARATION/REGRES
ION_radiationPVGISandPYSAM.html")
matrix=np.column_stack((PVGIS_RADIATION_VALUES,PYSAM_RADIATION_VALUES))
#print("matrix is", matrix)
df_mat=pd.DataFrame(matrix)
df_mat.to_csv("/Users/mariamaaiques/PycharmProjects/solar_potencial_estimation/COMPARATION/matrixR
ESULTS.csv", sep=',',index=False)
```