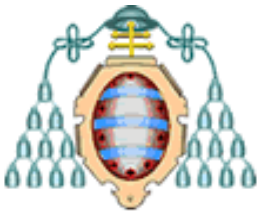

UNIVERSIDAD DE OVIEDO

ESCUELA POLITÉCNICA DE MIERES

MÁSTER UNIVERSITARIO EN GEOTECNOLOGÍA Y DESARROLLO DE PROYECTOS SIG

DEPARTAMENTO DE BIOLOGÍA DE ORGANISMOS Y SISTEMAS
ÁREA DE ECOLOGÍA

TRABAJO FIN DE MÁSTER

MODELADO DE LA DISTRIBUCIÓN DE LOS ECOSISTEMAS FORESTALES DEL PARQUE NACIONAL PICOS DE EUROPA: UNA APROXIMACIÓN METODOLÓGICA BASADA EN TÉCNICAS DE TELEDETECCIÓN

ANEXOS

AUTOR: Daniel Pfitzer López

TUTOR: Susana Suárez Seoane

COTUTORES: Francisco De Borja Jiménez-Alfaro

José Valentín Roces Díaz

JULIO, 2023

Índice

A. Código del complemento de QGIS **1**
A.1. Dependencias 1
A.2. Main 2
A.3. Módulos, funciones y objetos 37
A.4. Análisis 68
A.4.1. Incertidumbre 68
A.4.2. Exactitud y sensibilidad 69
A.4.3. Incertidumbre 70

B. Salidas generadas por el complemento de QGIS **72**
B.1. Archivos principales 72
B.2. Directorios 72

A. Código del complemento de QGIS

A.1. Dependencias

El código se ejecuta desde el entorno de desarrollo integrado en QGIS (v.3.28.5-Firenze). Es necesario instalar los siguientes módulos desde la consola de OSGeo4W previamente:

1. SAGA
2. GRASS
3. mapclassify
4. contextily
5. seaborn
6. rioxarray
7. pysal (v2.5.0)
8. sklearn
9. xgboost
10. lightgbm
11. pyimpute

Para ello, se abre la consola de OSGeo4W y se escribe el comando '`python -m pip install`' seguido del modulo, por ejemplo:

```
python -m pip install sklearn
```

A.2. Main

```
from osgeo import ogr, gdal
from PyQt5 import QtGui
import sys, glob, os, inspect, shutil, time, fileinput

# INITIALIZE MESSAGE ============================================================
def msg_box(msg: str, title: str):
    msg_box = QMessageBox()
    msg_box.setWindowTitle(title)
    msg_box.setText(msg)
    msg_box.exec_()

message = ("""
Spatial modelling distribution of habitats in the N.P. of Picos de Europa:
a methodological approach based on remote sensing techniques

By Daniel Pfitzer Lopez
""")
msg_box(message, 'TFM Title')
#===============================================================================
# GET WORKING PATH
dir_path = QFileDialog.getExistingDirectory(
    iface.mainWindow(),
    "Select .py directory", os.getcwd())

sys.path.insert(1, dir_path)

os.chdir(dir_path)
#===============================================================================
# GET CUSTOM FUNCTIONS
import funciones_dani as d


extensions_raster = [".tif", ".tiff", ".img", ".rst", ".asc"]
extensions_vector = [".shp", ".dgn", ".dxf", ".dwg", ".gdb", ".geojson",
    ".gpkg", ".kml", ".kmz", ".json", ".tab", ".mif", ".sqlite", ".sxf", ".vrt",
    ".csv", ".xls", ".xlsx"]

# GLOBAL SRC ====================================================================
crs = QgsCoordinateReferenceSystem()
mySelector = QgsProjectionSelectionDialog( iface.mainWindow() )
mySelector.setCrs( crs )
if mySelector.exec():
```

```
  mCrs = mySelector.crs()

# CREATE OUTPUT PATH ============================================================
d.limpiar()                                    # Erases everything on canvas
working_dir   = 'OUTPUT'
output_number = 0
bar = True

#Checks previous outputs
if os.path.exists(os.path.abspath(working_dir)):
    while os.path.exists(os.path.abspath(working_dir)):

        try:
            number = re.findall(r'\d+', working_dir)[-1]
            next_number = str(int(number) + 1)
            working_dir = working_dir[:len(working_dir)-len(number)]+next_number

        except:
            output_number += 1
            working_dir   += '_' + str(output_number)



os.mkdir(os.path.abspath(working_dir))                  # Creates empty OUTPUT
figures_path = working_dir+'/Figures'
os.mkdir(os.path.abspath(figures_path))

# IMPORT LAYERS - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#Create dataframe for variables
data = {
    'type'       :  [],
    'name'       :  [],
    'pyvariable' :  [],
    'resolution' :  [],
    'EPSG'       :  []}
variables_df = pd.DataFrame(data)


# Set input txt names from group_names dict
input_names=(
    'study_area',           # 0
    group_names['T_mean'],  # 1
```

```
    group_names['T_max'],    # 2
    group_names['T_min'],    # 3
    group_names['Pluv'],     # 4
    group_names['DEM'],      # 5
    group_names['S2'],       # 6
    'VegMap',                # 7
    group_names['DSM'])      # 8


# Read lasts inputs from last use

try:
    input_paths=[]                    # Same order as above
    for input_name in input_names:
        input_paths.append(txt_to_list(input_name))
except:
    input_paths=((),(),(),(),(),(),(),())
    print('No previous input data found')

# Get data paths
dialog = input_data()
if dialog.exec_() == QDialog.Accepted:

    dsm_dir, minumun_canopy_height, mch_check, gauss_check, min_NDVI, NDVI_check,
    mch_gauss_iterations, mch_gauss_std, mch_gauss_kernel_radio, topo_res, S2_res,
    veg_field = dialog.get_mch_data()

    directories = dialog.get_study_area_files()

    # 0 = Study Area
    # 1 = Mean Temperature
    # 2 = Max temperature
    # 3 = Min temperature
    # 4 = Pluviometry
    # 5 = DEM
    # 6 = Satellite
    # 7 = Veg Map



    directories += (dsm_dir,)

st = time.time() #Start time

```

```
131
132 # Save last input parameters into txt files
133 for list, txt_name in zip(directories, input_names):
134     with open('last_input/'+txt_name+'.txt', 'w') as fp:
135         for item in list:
136             fp.write("%s\n" % item)
137
138
139 # Load layers (name and object)
140 T_max, T_max_layers           = open_files(
141     files       =  directories[2],
142     group_name  =  group_names['T_max'],
143     trazar      =  False)
144
145 T_min, T_min_layers           = open_files(
146     files       =  directories[3],
147     group_name  =  group_names['T_min'],
148     trazar      =  False)
149
150 T_mean, T_mean_layers          = open_files(
151     files       =  directories[1],
152     group_name  =  group_names['T_mean'],
153     trazar      =  False)
154
155 Pluv, Pluv_layers            = open_files(
156     files       =  directories[4],
157     group_name  =  group_names['Pluv'],
158     trazar      =  False)
159
160 # Add climate layers to climate group
161 create_group_from_groups(
162     group_names = [
163         group_names['T_max'] ,
164         group_names['T_min'] ,
165         group_names['T_mean'],
166         group_names['Pluv']] ,
167         new_group_name = group_names['Climate_group'])
168
169 # Load DEM
170 if len(directories[5])==1: #Only one raster file (is merged)
171     DEM, DEM_layers              = open_files(
172         files           =  directories[5],
173         group_name      =  group_names['DEM'],
174         trazar          =  False)
175 else:
```

```
    DEM, DEM_layers              = merge_raster(
        group_name      =  group_names['DEM'],
        dir_merged_name = 'Set directory for merged DEM ',
        files           =  directories[5],
        name_merged     = 'DEM_Merged',
        alias           = 'DEM_Merged',
        trazar          =  False)

#Load S2
S2, S2_layers            = open_files(
    files       =  directories[6],
    group_name  =  group_names['S2'],
    trazar      =  False)

# Import study Area - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
StudyArea, StudyArea_layers  = open_files(
    files           =  directories[0],
    group_name    = None,
    trazar        = True)

# Study Area Style (might move this to the bottom)
apply_style(
    layer      = QgsProject.instance().mapLayersByName(StudyArea[0])[0],
    style_file = os.path.abspath('styles/StudyAreaStyle.qml'))

deactivate_group(group_names['Climate_group'])


#Tuple with all variables
'''
    Variables -> qgis names (str)
    layers    -> objects    (QgsRasterLayer)
'''
variables = (None, T_mean, T_max, T_min, Pluv, DEM, S2)
layers    = (None, T_mean_layers, T_max_layers, T_min_layers, Pluv_layers, DEM_layers, S2_layers)

# MASK data to study area  - - - - - - - - - - - - - - - - - - - - - - - - - -
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#Mask parameters
parameters_clip = {
        'ALPHA_BAND'      : False,
        'CROP_TO_CUTLINE' : True,
        'DATA_TYPE'       : 0,
```

```
        'EXTRA'           : '',
        'KEEP_RESOLUTION' : False,
        'MASK' : get_layer_URI_by_name(StudyArea[0]),
        'MULTITHREADING'  : False,
        'NODATA'          : None,
        'OPTIONS'         : '',
        'SET_RESOLUTION'  : False,
        'SOURCE_CRS'      : mCrs,
        'TARGET_CRS'      : mCrs,
        'X_RESOLUTION'    : None,
        'Y_RESOLUTION'    : None }

temp_variables = []
for group_name, variables in zip(group_names.values(), layers):
    if group_name != 'Climate':

        # Create folders for the new layers
        folder_name = group_name
        folder_path = os.path.abspath(working_dir + '/' +folder_name)
        os.mkdir(folder_path)

        layers_names = list_processing_raster(
            layer_list       = variables,
            parameters       = parameters_clip,
            output_dir       = folder_path,
            proccesing_name  = 'gdal:cliprasterbymasklayer',
            proccesing_alias = 'clipped')
        temp_variables.append(layers_names)
        time.sleep(0.05)


# Update our variables
T_mean, T_max, T_min, Pluv, DEM, S2 = temp_variables
variables = (None, T_mean, T_max, T_min, Pluv, DEM, S2)


# Load proccesed layers
root = QgsProject.instance().layerTreeRoot()

temp_variables = []

for group_name, variables in zip(group_names.values(), variables):
    if group_name != 'Climate':

        # Get folder names
```

```
265         folder_name = group_name
266         folder_path = os.path.abspath(working_dir + '/' + folder_name)
267 
268         #Get the group instances and proccess
269         group = root.findGroup(group_name)
270 
271         layers = []
272 
273         for layer in variables:
274 
275             if group_name == group_names['DSM']:
276                 trazar = False
277             else:
278                 trazar = True
279 
280             layers.append(d.abrir_raster(
281                 directorio = folder_path,
282                 capa       = layer+'.tiff',
283                 alias      = layer,
284                 trazar     = trazar,
285                 group      = group
286             ))
287             print('')
288         temp_variables.append(layers)
289 
290 # Update layers object
291 T_mean_layers, T_max_layers, T_min_layers, Pluv_layers, DEM_layers, S2_layers = temp_variables
292 layers = (None, T_mean_layers, T_max_layers, T_min_layers, Pluv_layers, DEM_layers, S2_layers)
293 
294 # Add climatic to dataframe
295 for lys in layers[1:5]:
296     for layer in lys:
297         new_row = {
298             'type'       : 'Climate',
299             'name'       :  layer.name(),
300             'resolution' :  layer.rasterUnitsPerPixelX(),
301             'pyvariable' :  layer,
302             'EPSG'       :  layer.crs().authid()}
303 
304         variables_df.loc[len(variables_df)] = new_row
305 
306 
307 # GET TOPO VARIABLES FROM CLIPPED DEM - - - - - - - - - - - - - - - - - - - - - -
308 # - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
309 
```

```
310 # SLOPE
311 parameters = {
312     'AS_PERCENT': False,
313     'BAND': 1,
314     'COMPUTE_EDGES': False,
315     'EXTRA': '',
316     'INPUT': os.path.abspath(working_dir + '/' +group_names['DEM']+'/'+DEM[0]+'.tiff'),
317     'OPTIONS': '',
318     'OUTPUT': os.path.abspath(working_dir + '/' +group_names['DEM']+'/slope.tiff'),
319     'SCALE': 1,
320     'ZEVENBERGEN': False
321 }
322 processing.run('gdal:slope', parameters)
323
324 slope_path = os.path.abspath(working_dir + '/'+group_names['DEM']+'/slope.tiff')
325
326 slope = d.abrir_raster(
327     directorio = slope_path,
328     capa       = 'slope',
329     alias      = 'slope',
330     trazar     = True,
331     group      = root.findGroup('Topo'))
332
333 # ASPECT
334 parameters = {
335     'BAND' : 1,
336     'COMPUTE_EDGES' : False,
337     'EXTRA' : '',
338     'INPUT': os.path.abspath(working_dir + '/'+group_names['DEM']+'/'+DEM[0]+'.tiff') ,
339     'OPTIONS' : '',
340     'OUTPUT': os.path.abspath(working_dir + '/'+group_names['DEM']+'/aspect_degs.tiff'),
341     'TRIG_ANGLE' : True,
342     'ZERO_FLAT' : False,
343     'ZEVENBERGEN' : False }
344
345 processing.run('gdal:aspect', parameters)
346
347
348 aspect_file = os.path.abspath(working_dir + '/' +group_names['DEM']+'/aspect_degs.tiff')
349
350 #Lee el archivo de salida de gdal:aspect
351 ds = gdal.Open(aspect_file)
352 aspect_array = ds.ReadAsArray()
353
354 # Convierte el ángulo del aspecto de grados a radianes
```

```
355 aspect_array = np.deg2rad(aspect_array)
356
357 # Calcula la componente de northness y eastness
358 northness_array = np.cos(aspect_array)
359 eastness_array = np.sin(aspect_array)
360
361 # Define la transformación a utilizar en el archivo de salida
362 geotransform = ds.GetGeoTransform()
363 projection = ds.GetProjection()
364
365 # Guarda la componente de northness en un archivo
366 driver = gdal.GetDriverByName('GTiff')
367 northness_ds = driver.Create(
368     working_dir + '/topo/northness.tiff',
369     ds.RasterXSize,
370     ds.RasterYSize,
371     1,
372     gdal.GDT_Float32)
373 northness_ds.SetGeoTransform(geotransform)
374 northness_ds.SetProjection(projection)
375 northness_ds.GetRasterBand(1).WriteArray(northness_array)
376 northness_ds.FlushCache()
377
378 # Guarda la componente de eastness en otro archivo
379 eastness_ds = driver.Create(
380     working_dir +'/topo/eastness.tiff',
381     ds.RasterXSize,
382     ds.RasterYSize,
383     1,
384     gdal.GDT_Float32)
385
386 eastness_ds.SetGeoTransform(geotransform)
387 eastness_ds.SetProjection(projection)
388 eastness_ds.GetRasterBand(1).WriteArray(eastness_array)
389 eastness_ds.FlushCache()
390
391 # Cierra los datasets abiertos
392 ds = None
393 northness_ds = None
394 eastness_ds = None
395
396 northness_path = os.path.abspath(working_dir + '/' +group_names['DEM']+'/northness.tiff')
397 eastness_path  = os.path.abspath(working_dir + '/' +group_names['DEM']+'/eastness.tiff')
398
```

```
399 #Load northness and eastness
400 northness = d.abrir_raster(
401     directorio = northness_path,
402     capa       = 'northness',
403     alias      = 'northness',
404     trazar     = True,
405     group      = root.findGroup('Topo'))
406 
407 eastness = d.abrir_raster(
408     directorio = eastness_path,
409     capa       = 'eastness',
410     alias      = 'eastness',
411     trazar     = True,
412     group      = root.findGroup('Topo'))
413 
414 
415 # RESAMPLING AND TEXTURE - - - - - - - - - - - - - - - - - - - - - - - - - - - -
416 # - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
417 resampled_name       = 'resampled'
418 texture_name         = 'texture'
419 
420 # Topo resampling ================================================================
421 resampled_dir        = working_dir + '/' +group_names['DEM']+'/'+resampled_name
422 
423 os.mkdir(resampled_dir)
424 
425 resample_pixel = topo_res
426 
427 params_avg = {
428     '-n'                            : False,
429     '-w'                            : True,
430     'GRASS_RASTER_FORMAT_META'      : '',
431     'GRASS_RASTER_FORMAT_OPT'       : '',
432     'GRASS_REGION_CELLSIZE_PARAMETER' : resample_pixel,
433     'GRASS_REGION_PARAMETER'        : None,
434     'method'                        : 0}
435 
436 params_var = {
437     '-n'                            : False,
438     '-w'                            : True,
439     'GRASS_RASTER_FORMAT_META'      : '',
440     'GRASS_RASTER_FORMAT_OPT'       : '',
441     'GRASS_REGION_CELLSIZE_PARAMETER' : resample_pixel,
442     'GRASS_REGION_PARAMETER'        : None,
443     'method'                        : 10}
```

```
444
445 parameters = (
446     params_avg,
447     params_var)
448
449 resampled_aggretations = (
450     '_average',
451     '_variance')
452
453 topo_resampled_names = []
454
455 for params, aggregation_type in zip(parameters,resampled_aggretations):
456     topo_resampled_names.extend(group_processing_raster(
457         group_name       = group_names['DEM'],
458         parameters       = params,
459         output_dir       = resampled_dir,
460         proccesing_name  = 'grass7:r.resamp.stats',
461         proccesing_alias = resampled_name+aggregation_type))
462
463 resampled_group_name = group_names['DEM']+'R'+str(resample_pixel)
464 a = root.addGroup(resampled_group_name)
465
466 topo_resampled=[]
467
468 for resampled_path in topo_resampled_names:
469
470     filename_with_ext = os.path.basename(resampled_path)
471     name = os.path.splitext(filename_with_ext)[0]
472
473     layer = d.abrir_raster(
474         directorio = resampled_path,
475         capa       = name,
476         alias      = name,
477         trazar     = True,
478         group      = root.findGroup(resampled_group_name))
479
480     topo_resampled.append(layer)
481
482     new_row = {
483         'type'       : 'Topo',
484         'name'       :  layer.name(),
485         'resolution' :  int(round(layer.rasterUnitsPerPixelX(),0)),
486         'pyvariable' :  layer,
487         'EPSG'       :  layer.crs().authid()}
488
```

```
    variables_df.loc[len(variables_df)] = new_row

# Erase auxiliary files that annoys me a lot
erase_files_on_dir(resampled_dir, '*.xml')
erase_files_on_dir(resampled_dir, '*.tfw')




# S2 resampling ======================================================================
resampled_dir      = working_dir + '/' +group_names['S2']+'/'+ resampled_name
texture_dir        = working_dir + '/' +group_names['S2']+'/'  + texture_name

os.mkdir(resampled_dir)

resample_pixel = S2_res

params_avg = {
    '-n'                            : False,
    '-w'                            : True,
    'GRASS_RASTER_FORMAT_META'      : '',
    'GRASS_RASTER_FORMAT_OPT'       : '',
    'GRASS_REGION_CELLSIZE_PARAMETER' : resample_pixel,
    'GRASS_REGION_PARAMETER'        : None,
    'method'                        : 0}

S2_resampled_names = []

S2_resampled_names.extend(group_processing_raster(
        group_name       = group_names['S2'],
        parameters       = params_avg,
        output_dir       = resampled_dir,
        proccesing_name  = 'grass7:r.resamp.stats',
        proccesing_alias = resampled_name))

resampled_group_name = group_names['S2']+'R'+str(resample_pixel)
a = root.addGroup(resampled_group_name)

S2_resampled=[]
S2_testured= []

for resampled_path in S2_resampled_names:

    filename_with_ext = os.path.basename(resampled_path)
```

```
533         name = os.path.splitext(filename_with_ext)[0]
534 
535         layer = d.abrir_raster(
536             directorio = resampled_path,
537             capa       = name,
538             alias      = name,
539             trazar     = True,
540             group      = root.findGroup(resampled_group_name))
541 
542         topo_resampled.append(layer)
543 
544         new_row = {
545             'type'       : 'S2',
546             'name'       :  layer.name(),
547             'resolution' :  int(layer.rasterUnitsPerPixelX()),
548             'pyvariable' :  layer,
549             'EPSG'       :  layer.crs().authid()}
550 
551         variables_df.loc[len(variables_df)] = new_row
552 
553     # Erase auxiliary files that annoys me a lot
554     erase_files_on_dir(resampled_dir, '*.xml')
555     erase_files_on_dir(resampled_dir, '*.tfw')
556 
557 
558 
559 
560     # MCH - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
561     # - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
562     mch_dir = working_dir + '/' +group_names['DSM']
563     os.mkdir(mch_dir)
564 
565     parameters_clip['OUTPUT'] = mch_dir+'/DSM_clipped.tiff'
566     parameters_clip['INPUT']  = dsm_dir[0]
567     processing.run('gdal:cliprasterbymasklayer', parameters_clip)
568 
569     dsm = d.abrir_raster(
570         directorio = parameters_clip['OUTPUT'],
571         capa       = 'DSM_clipped',
572         alias      = 'DSM_clipped',
573         trazar     =  False,
574         group      =  None)
575 
576     # Resample parameters:
```

```
577 params_avg = {
578     '-n'                            : False,
579     '-w'                            : True,
580     'GRASS_RASTER_FORMAT_META'      : '',
581     'GRASS_RASTER_FORMAT_OPT'       : '',
582     #'GRASS_REGION_CELLSIZE_PARAMETER' : resample_pixel,
583     'GRASS_REGION_PARAMETER'        : None,
584     'method'                        : 0}
585
586 # Get DEM and DSM resolutions
587 dsm_res = dsm.rasterUnitsPerPixelX()
588 dem_res = DEM_layers[0].rasterUnitsPerPixelX()
589
590 if dsm_res > dem_res:
591     params_avg['input']  = DEM_layers[0].dataProvider().dataSourceUri()
592     params_avg['output'] = mch_dir+'/'+DEM_layers[0].name()+'_AvgResampled'+str(dsm_res)+'.tiff'
593     params_avg['GRASS_REGION_CELLSIZE_PARAMETER'] = dsm_res
594     processing.run('grass7:r.resamp.stats', params_avg)
595     mch_top = dsm
596     mch_bot = d.abrir_raster(
597         directorio = params_avg['output'],
598         capa       = 'mhc_bot'+'_AvgResampled'+str(dsm_res),
599         alias      = 'mhc_bot'+'_AvgResampled'+str(dsm_res),
600         trazar     = False,
601         group      = None)
602
603 elif dsm_res < dem_res:
604     params_avg['input']  = dsm.dataProvider().dataSourceUri()
605     params_avg['output'] = mch_dir+'/'+dsm.name()+'_AvgResampled'+str(DEM_layers[0])+'.tiff'
606     params_avg['GRASS_REGION_CELLSIZE_PARAMETER'] = dem_res
607     processing.run('grass7:r.resamp.stats', params_avg)
608     mch_top = DEM_layers[0]
609     mch_bot = d.abrir_raster(
610         directorio = params_avg['output'],
611         capa       = 'mhc_top'+'_AvgResampled'+str(DEM_layers[0]),
612         alias      = 'mhc_top'+'_AvgResampled'+str(DEM_layers[0]),
613         trazar     = False,
614         group      = None)
615 else:
616     mch_top = dsm
617     mch_bot = DEM_layers[0]
618
619 # Compute canopy height
620 canopy_dir = mch_dir+'/canopy_height_r'+str(mch_top.rasterUnitsPerPixelX())+'.tiff'
```

```
621
622 # Add raster calculations entries
623 mch_entries = []
624
625 append_raster_to_calc_entry(
626     entries    =  mch_entries,
627     ref_name   = 'mch_top@1',
628     raster     =  mch_top,
629     bandnumber =  1)
630
631 append_raster_to_calc_entry(
632     entries    =  mch_entries,
633     ref_name   = 'mch_bot@1',
634     raster     =  mch_bot,
635     bandnumber =  1)
636
637 calculation = QgsRasterCalculator(
638     formulaString    = 'mch_top@1 - mch_bot@1',
639     outputFile       = working_dir + '/MCH/canopy_height_r5.tiff',
640     outputFormat     = 'GTiff',
641     outputExtent     = mch_top.extent(),
642     nOutputColumns   = mch_top.width(),
643     nOutputRows      = mch_top.height(),
644     rasterEntries    = mch_entries)
645
646 calculation.processCalculation()
647
648 mch_group = create_group('mch')
649
650 canopy_height = d.abrir_raster(
651     directorio = working_dir + '/MCH/canopy_height_r5.tiff',
652     capa       = 'canopy_height_r5',
653     alias      = 'canopy_height_r5',
654     trazar     = True,
655     group      = mch_group)
656
657 new_row = {
658     'type'       : 'Topo',
659     'name'       :  canopy_height.name(),
660     'resolution' :  int(canopy_height.rasterUnitsPerPixelX()),
661     'pyvariable' :  canopy_height,
662     'EPSG'       :  canopy_height.crs().authid()}
663
664 variables_df.loc[len(variables_df)] = new_row
665
```

```
666
667 # MUESTREO - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
668 # - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
669 dir_veg_map = directories[7][0]
670
671 veg_map = d.abrir_shape(
672     directorio =  dir_veg_map,
673     capa       = 'vegPNPE_2010',
674     alias      = 'VegetationMap',
675     trazar     =  True,
676     group      =  None)
677
678 #Seleccionar only bosque
679 expression = f'"{veg_field}" is not \'No_bosque\''
680 selection = veg_map.getFeatures(QgsFeatureRequest().setFilterExpression(expression))
681 veg_map.selectByIds([s.id() for s in selection])
682
683 #Generar sampling
684 sampling_dir = working_dir + '/sampling'
685 os.mkdir(sampling_dir)
686
687 min_dis = np.arange(150, 1205, 5) #Distancias desde 150 a 1200 de 5 en 5
688 samplings = pd.DataFrame({
689     'MinDistance': min_dis,
690     'path': ''})
691
692 # Generate a dataframe with samplings from 50 to 1000 (min, distance)
693 for i, row in samplings.iterrows():
694     parameters = {
695         'POINTS':os.path.abspath(sampling_dir + '/sampleo' + str(i) + '_' + str(row['MinDistance']) + '
696         'EXTENT':3,
697         'SHAPES':QgsProcessingFeatureSourceDefinition(
698             dir_veg_map,
699             selectedFeaturesOnly=True,
700             featureLimit=-1,
701             geometryCheck=QgsFeatureRequest.GeometryAbortOnInvalid),
702         'POLYGONS':QgsProcessingFeatureSourceDefinition(
703             dir_veg_map,
704             selectedFeaturesOnly=True,
705             featureLimit=-1,
706             geometryCheck=QgsFeatureRequest.GeometryAbortOnInvalid),
707         'XMIN':0,
708         'XMAX':0,
709         'YMIN':0,
710         'YMAX':0,
```

```
        'NX':1,
        'NY':1,
        'BUFFER':0,
        'COUNT':2000,
        'DISTRIBUTE':0,
        'ITERATIONS':1000,
        'DISTANCE': row['MinDistance']}



    processing.run("saga:createrandompoints", parameters)
    samplings.at[i, 'path'] = parameters['POINTS']

    samplings.at[i, 'PointLayer'] = QgsVectorLayer(samplings.at[i, 'path'])
    samplings.at[i, 'NPoints'] = samplings.at[i, 'PointLayer'].featureCount()

samplings['sampling_data_object'] = None #Aqui van los objetos gdf para cada uno


# EXTRACCION DE DATOS  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
def mch_filter(df, column_name, min_value):
    # Obtener el número de filas en el DataFrame original
    original_row_count = len(df)

    # Crear una máscara booleana para seleccionar las filas en las que el valor en la columna especifica
    mask = df[column_name] >= min_value

    # Filtrar el DataFrame usando la máscara booleana
    result_df = df[mask]

    # Obtener el número de filas en el DataFrame filtrado
    filtered_row_count = len(result_df)

    # Calcular el número de filas eliminadas
    removed_row_count = original_row_count - filtered_row_count

    # Imprimir el número de filas eliminadas
    #print(f'Number of sampling points filtered by mch: {removed_row_count}, {removed_row_count/origina

    # Devolver el DataFrame filtrado
    return result_df


#Calcular las is de moran para TODOS los muestreos
```

```
for fila in range(len(samplings)):
    test_df = extract_all(samplings.at[fila, 'PointLayer'])


    if mch_check:
        test_df = mch_filter(
            df          = test_df,
            column_name = 'canopy_height_r5',
            min_value   = minumun_canopy_height)

    samplings.at[fila, 'NPoints_MCH'] = len(test_df.index)

    samplings.at[fila, 'sampling_data_object'] = test_df

    if NDVI_check:
        test_df['NDVI'] = (test_df['SENTINEL2A_20170820-111220-771_L2A_T30TUN_D_V1-7_SRE_B8_clipped_resa
        test_df = test_df[test_df['NDVI'] >= min_NDVI]

    samplings.at[fila, 'NPoints_NDVI'] = len(test_df.index)

    for col in test_df.columns:
        if col != 'coords':
            (moran, moran_pval) = moranI(
                gdf = test_df,
                col = col)
            samplings.at[fila, f'moran_{col}'] = moran


# shift column at position 1 to first position
mdt_avg = samplings.iloc[:,55]
samplings.drop(samplings.columns[55], axis=1, inplace=True)
samplings.insert(0, mdt_avg.name, mdt_avg)

mdt_avg = samplings.iloc[:,56]
samplings.drop(samplings.columns[56], axis=1, inplace=True)
samplings.insert(1, mdt_avg.name, mdt_avg)

#ELEGIR EL MEJOR MUESTREO ============================================================
os.mkdir(working_dir+'/GoodSampling')

row = samplings.loc[(samplings.iloc[:, 57:] < 0.3).all(axis=1)]
selected_sampling = row['PointLayer'].iloc[0]
min_distance = row['MinDistance'].iloc[0]
```

```
800 print(f"\n Minimun distance sampling: {min_distance} \n")
801
802
803
804
805
806
807 # Generar no_bosque ================================================================
808 veg_map.deselect(veg_map.selectedFeatureIds())
809 # Se ha decidido emplear solo datos de presencia y no presencia en zonas
810 # de bosque!
811
812 #----------------------------------------
813
814
815
816 # DATAFRAME WITH SELECTED SAMPLING -----------------------
817
818 #get samppling n3
819 vector_sampling = selected_sampling
820
821
822 #Get all forest on that sampling
823 good_sampling = os.path.abspath(working_dir+'/GoodSampling/sampling.shp')
824 params = {
825     'INPUT'               :vector_sampling,
826     'INPUT_2'             :veg_map,
827     'FIELDS_TO_COPY'      :[veg_field],
828     'DISCARD_NONMATCHING' :True,
829     'PREFIX'              :'',
830     'NEIGHBORS'           :1,
831     'MAX_DISTANCE'        : 0,
832     'OUTPUT'              : good_sampling}
833
834 processing.run("native:joinbynearest", params)
835
836 vlayer = QgsVectorLayer(good_sampling, 'good_sampling', 'ogr')
837 test_data = extract_all_with_forest(vlayer)
838 #no_bosque_data = extract_all_with_forest(no_bosque)
839
840 #Erase NA
841 test_data = test_data.dropna(how='any')
842
843 # MCH Filter
```

```
844 if mch_check:
845     test_data = mch_filter(
846         df          = test_data,
847         column_name = 'canopy_height_r5',
848         min_value   = minumun_canopy_height)
849 
850 # NDVI Filter
851 min_NDVI = 0.6
852 test_data['NDVI'] = (test_data['SENTINEL2A_20170820-111220-771_L2A_T30TUN_D_V1-7_SRE_B8_clipped_resample
853 if NDVI_check:
854     test_data = test_data[test_data['NDVI'] >= min_NDVI]
855 
856 #test_data = test_data.append(no_bosque_data)
857 test_data = test_data.dropna(how='any')
858 
859 save_df(test_data, 'muestreo')
860 
861 
862 #FEATURE SELECTION ================================================================
863 climate_cols = [x for x in range(49) if x!=  1]
864 topo_cols    = [x for x in range(57) if x > 48]
865 S2_cols      = [x for x in range(67) if x > 56] + [68]
866 
867 #Valores para climograma
868 def get_climatic_mean(raster_list):
869     '''
870     Input a list of QgsRasterLayer
871     Returns a list with each average
872 
873     Used to get average of each month in the study area for climatic rasters
874     '''
875     stats=[]
876     for month_variable in raster_list:
877         stats.append(month_variable.dataProvider().bandStatistics(1, QgsRasterBandStats.Mean).mean)
878     return stats
879 
880 def correlation(matrix, recorte_etiqueta, file_name, show:bool, valores:bool):
881     plt.figure()
882     sns.heatmap(matrix, vmax=1, vmin=-1, center=0, cmap='vlag', annot=valores)
883     # Truncar las etiquetas del eje x
884     x_labels = [label.get_text().split(recorte_etiqueta)[0] for label in plt.gca().get_xticklabels()]
885     plt.gca().set_xticklabels(x_labels)
886     # Truncar las etiquetas del eje y
887     y_labels = [label.get_text().split(recorte_etiqueta)[0] for label in plt.gca().get_yticklabels()]
888     plt.gca().set_yticklabels(y_labels)
```

```
889         plt.savefig(os.path.abspath(figures_path+'/'+ file_name +'.svg'), format='svg')
890         if show:
891             plt.show()
892 
893     T_mean_Month_Average = get_climatic_mean(T_mean_layers)
894     T_max_Month_Average = get_climatic_mean(T_max_layers)
895     T_min_Month_Average = get_climatic_mean(T_min_layers)
896     T_pluv_Month_Average = get_climatic_mean(Pluv_layers)
897 
898     lista_Tmean = test_data.columns[0:12].tolist()
899     lista_Tmax  = test_data.columns[12:24].tolist()
900     lista_Tmin  = test_data.columns[24:36].tolist()
901     lista_pluv  = test_data.columns[36:48].tolist()
902 
903     hot_month = lista_Tmax[T_max_Month_Average.index(max(T_max_Month_Average))]
904     cold_month = lista_Tmin[T_min_Month_Average.index(min(T_min_Month_Average))]
905     dry_month = lista_pluv[T_pluv_Month_Average.index(min(T_pluv_Month_Average))]
906 
907     climate_features = [hot_month, cold_month, dry_month]
908 
909     # Correlation ========================================================================
910     ''' ---------------------------------------
911     CLIMA-> criterio experto; por el mes mas frio, mas caliente y más seco
912     ---------------------------------------'''
913     climate_matrix = test_df.iloc[:, climate_cols].corr().round(2)
914 
915     # Figura clima full correlaciones
916     correlation(
917         matrix            = climate_matrix,
918         recorte_etiqueta  = 'r',
919         file_name         = 'climate_correlation_full',
920         show              = False,
921         valores           = False)
922 
923     climate_matrix = test_df[climate_features].corr().round(2)
924 
925     correlation(
926         matrix            = climate_matrix,
927         recorte_etiqueta  = 'r',
928         file_name         = 'climate_correlation_selected_features',
929         show              = True,
930         valores           = True)
931 
932     hot_path = QgsProject.instance().mapLayersByName(hot_month)[0].source()
```

```
933 cold_path = QgsProject.instance().mapLayersByName(cold_month)[0].source()
934 dry_path = QgsProject.instance().mapLayersByName(dry_month)[0].source()
935
936 ''' ---------------------------------------
937 TOPO
938 ---------------------------------------'''
939 topo_matrix = test_df.iloc[:, topo_cols].corr().round(2)
940
941 correlation(
942     matrix            = topo_matrix,
943     recorte_etiqueta  = ' ',
944     file_name         = 'topo_correlation_full',
945     show              = True,
946     valores           = True)
947
948 topo_features = test_df.columns.values[topo_cols].tolist()
949     # 0 DEM   Resampled (avg)
950     # 1 Slope Resampled (avg)
951     # 2 Northness Res.  (avg)
952     # 3 Eastness  Res.  (avg)
953     # 4 DEM Resampled   (var)
954     # 5 Slope Resampled (var)
955     # 6 Northness Res.  (var)
956     # 7 Eastness  Res.  (var)
957
958 topo_features = topo_features[1:4] + [topo_features[5]]
959
960 topo_matrix = test_df[topo_features].corr().round(2)
961
962 correlation(
963     matrix            = topo_matrix,
964     recorte_etiqueta  = ' ',
965     file_name         = 'topo_correlation_selected_features',
966     show              = True,
967     valores           = True)
968
969
970 topo_paths = []
971
972 for topo_feature in topo_features:
973     topo_paths.append(QgsProject.instance().mapLayersByName(topo_feature)[0].source())
974
975 ''' ---------------------------------------
976 SENTINEL 2
```

```
977 ---------------------------------------'''
978 S2_matrix = test_df.iloc[:, S2_cols].corr().round(2)
979
980 correlation(
981     matrix            = S2_matrix,
982     recorte_etiqueta  = '_clip',
983     file_name         = 'S2_correlation_full_valores',
984     show              = True,
985     valores           = True)
986
987 correlation(
988     matrix            = S2_matrix,
989     recorte_etiqueta  = '_clip',
990     file_name         = 'S2_correlation_full',
991     show              = True,
992     valores           = False)
993
994 S2_features = test_df.columns.values[S2_cols].tolist()
995 S2_features = S2_features[0:len(S2_features)-1]
996
997     # 0  B2  Blue
998     # 1  B3  Green
999     # 2  B4  Red
1000    # 3  B5  Visible and Near Infrared (VNIR)
1001    # 4  B6  Visible and Near Infrared (VNIR)
1002    # 5  B7  Visible and Near Infrared (VNIR)
1003    # 6  B8  Visible and Near Infrared (VNIR)
1004    # 7  B8A Visible and Near Infrared (VNIR)
1005    # 8  B11 Short Wave Infrared (SWIR)
1006    # 9  B12 Short Wave Infrared (SWIR)
1007    # 10 NDVI
1008
1009 ndvi = get_NDVI_layer()
1010 ratio_swir = get_ratio_swir_layer()
1011
1012
1013
1014
1015
1016 S2_matrix = test_df[S2_features].corr().round(2)
1017
1018 correlation(
1019     matrix            = S2_matrix,
1020     recorte_etiqueta  = ' ',
1021     file_name         = 'S2_correlation_selected_features',
```

```
    show              = True,
    valores           = True)

S2_features = [S2_features[0]]+[S2_features[1]]+[S2_features[2], S2_features[3], S2_features[4], S2_feat
S2_names = ('B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'NVDI', 'ratio_swir')

S2_paths = []

for S2_feature in S2_features:
    try:
        S2_paths.append(QgsProject.instance().mapLayersByName(S2_feature)[0].source())
    except:
        pass

S2_paths.append(ndvi.source())
S2_paths.append(ratio_swir.source())




#Folders for models ------------------------------
input_folder = working_dir+'/InputModel'
if os.path.exists(input_folder):
    shutil.rmtree(input_folder)
os.mkdir(input_folder)

#Climate
climate_folder = input_folder + '/Climate'
os.mkdir(climate_folder)
shutil.copy(hot_path , climate_folder+ '/' + hot_month  + '.tiff')
shutil.copy(cold_path, climate_folder+ '/' + cold_month + '.tiff')
shutil.copy(dry_path , climate_folder+ '/' + dry_month  + '.tiff')

#Topo
topo_folder = input_folder + '/Topo'
os.mkdir(topo_folder)
for topo_feature, topo_path in zip(topo_features,topo_paths):
    shutil.copy(topo_path , topo_folder+ '/' + topo_feature  + '.tiff')

#S2
S2_folder = input_folder + '/S2'
os.mkdir(S2_folder)

for S2_feature, S2_path in zip(S2_features,S2_paths):
    try:
```

```
        shutil.copy(S2_path , S2_folder+ '/' + S2_feature  + '.tiff')
    except:
        shutil.copy(S2_path , S2_folder+ '/' + S2_feature.name()  + '.tiff')

#Match all S2 rasters
os.mkdir(S2_folder+'/matched')
raster_features = sorted(glob.glob(os.path.abspath(S2_folder+ '/*.tiff')))
for raster_feature, band in zip(raster_features, S2_names):
        match_raster(
            input_path   = raster_feature ,
            target_path  = variables_df.loc[variables_df['type'] == 'S2', 'pyvariable'].iloc[0].source()
            output       = S2_folder+'/matched/'+band+'.tiff')


### ========================================================================
def eliminar_carpetas_vacias(ruta):
    for root, dirs, files in os.walk(ruta, topdown=False):
        for dir in dirs:
            dir_path = os.path.join(root, dir)
            if not os.listdir(dir_path):
                os.rmdir(dir_path)



### MODEL BUILDING
def df_presence(df, column_name, value):
    new_df = df.copy()
    new_df[column_name] = new_df[column_name] == value
    return new_df

forest_types = sorted(test_data[veg_field].unique())

best_model = {}
    # AOO + [forest_type] -> path
    # AO2
    # ARO

df = pd.DataFrame(columns=['bosque', 'path', 'modeltype'])

df_AOO_P = pd.DataFrame()
df_AOO_L = pd.DataFrame()
df_AOO_R = pd.DataFrame()
```

```
1111
1112  for forest_type in forest_types:
1113
1114      #1. Climatic AOO
1115      forest_presence_test_df = df_presence(
1116          df           = test_data,
1117          column_name  = veg_field,
1118          value        = forest_type)
1119
1120      npoints = forest_presence_test_df[veg_field].sum()
1121
1122      # AOO: Genera los modelos y guarda el path del mejor:
1123      best_model['AOO'+forest_type], dict1 = predict(
1124          input_folder   = climate_folder,
1125          gdf            = forest_presence_test_df,
1126          gdf_target     = veg_field,
1127          type           = 'climate',
1128          forest_name    = forest_type,
1129          add_model_path = False)
1130
1131      dict1['forest']=forest_type
1132
1133      print('\n Forest type: '+forest_type+ 'best AOO model path -> '+best_model['AOO'+forest_type])
1134
1135      match_raster(
1136          input_path   = best_model['AOO'+forest_type]+'/probability_True.tif' ,
1137          target_path  = variables_df.loc[variables_df['type'] == 'Topo', 'pyvariable'].iloc[0].source(),
1138          output       = best_model['AOO'+forest_type]+'/probability_True.tiff')
1139
1140      # AO2:
1141      best_model['AO2'+forest_type], dict2 = predict(
1142          input_folder   = topo_folder,
1143          gdf            = forest_presence_test_df,
1144          gdf_target     = veg_field,
1145          type           = 'topo',
1146          forest_name    = forest_type,
1147          add_model_path = best_model['AOO'+forest_type]+'/probability_True.tiff')
1148
1149      dict2['AOO_P'] = dict2.pop('probability_True')
1150      dict2['forest']=forest_type
1151      print('\n Forest type: '+forest_type+ 'best AO2 model path -> '+ best_model['AO2'+forest_type])
1152
1153      match_raster(
1154          input_path   = best_model['AO2'+forest_type]+'/probability_True.tif' ,
1155          target_path  = variables_df.loc[variables_df['type'] == 'S2', 'pyvariable'].iloc[0].source(),
```

```
        output       = best_model['AO2'+forest_type]+'/probability_True.tiff')

    # REAL:
    best_model['REAL_'+forest_type],dict3 = predict(
        input_folder   = S2_folder+'/matched',
        gdf            = forest_presence_test_df,
        gdf_target     = veg_field,
        type           = 'S2',
        forest_name    = forest_type,
        add_model_path = best_model['AO2'+forest_type]+'/probability_True.tiff')

    dict3['AOO_L'] = dict3.pop('probability_True')
    dict3['forest']=forest_type

    df_AOO_P = df_AOO_P.append(dict1, ignore_index=True)
    df_AOO_L = df_AOO_L.append(dict2, ignore_index=True)
    df_AOO_R = df_AOO_R.append(dict3, ignore_index=True)

del(list)
eliminar_carpetas_vacias(working_dir+'/models')
AOO_REAL_PATHS = {key.replace('REAL_', ''): value for key, value in best_model.items() if key.startswith
df_AOO_REAL = pd.DataFrame(list(AOO_REAL_PATHS.items()), columns=['forest', 'path'])
df_AOO_REAL['index'] = range(1, len(df_AOO_REAL) + 1)
raster_paths = df_AOO_REAL['path'].tolist()

raster_paths = list(map(lambda path: path + '/probability_True.tif', raster_paths))

df_AOO_REAL.to_csv(working_dir+'/modelos_lista_AOO_real.csv', index=False)

df_AOO_P.to_csv(working_dir+'/AOO_P.csv', index=False)
df_AOO_L.to_csv(working_dir+'/AOO_L.csv', index=False)
df_AOO_R.to_csv(working_dir+'/AOO_R.csv', index=False)

'''
COMPUTE AOO REAL
'''
# Load the input rasters as NumPy arrays
arrays = [rasterio.open(raster_path).read(1) for raster_path in raster_paths]
# Stack the arrays along a new axis
stacked = np.stack(arrays, axis=-1)
# Find the index of the input raster with the highest value at each pixel
output_array = np.argmax(stacked, axis=-1) + 1
# Create a mask that identifies pixels where all input rasters have a nodata value
nodata_mask = np.all(np.isnan(stacked), axis=-1)
```

```
# Set the value of the output array at these pixels to a specific value (e.g., 0)
output_array[nodata_mask] = 0
# Get the profile of the first input raster (assuming all rasters have the same profile)
with rasterio.open(raster_paths[0]) as src:
    profile = src.profile

# Update the profile to set the correct data type for the output raster
profile.update(dtype=rasterio.int32)
# Save the output array as a new raster
try:
    with rasterio.open(working_dir+'/models/AOO_real.tif', 'w', **profile) as dst:
        dst.write(output_array.astype(rasterio.int32), 1)
except:
    pass
# Load the output raster as a QgsRasterLayer
AOO_real_rlayer = QgsRasterLayer(working_dir+'/models/AOO_real.tif', 'AOO_real')
# Add the output raster to the map
#QgsProject.instance().addMapLayer(output_raster)

'''
INCERTIDUMBRE
'''

# Load the input rasters as NumPy arrays
arrays = [rasterio.open(raster_path).read(1) for raster_path in raster_paths]
# Stack the arrays along a new axis
stacked = np.stack(arrays, axis=0)
# Find the maximum and second highest values at each pixel
maximum = np.max(stacked, axis=0)
sorted_values = np.sort(stacked, axis=0)
second_highest = sorted_values[-2]

# Calculate the difference between the maximum and second highest values
output_array = 1-(maximum - second_highest)
# Get the profile of the first input raster (assuming all rasters have the same profile)
with rasterio.open(raster_paths[0]) as src:
    profile = src.profile

# Update the profile to set the correct data type for the output raster
profile.update(dtype=rasterio.float32)
# Save the output array as a new raster
try:
    with rasterio.open(working_dir+'/models/incertidumbre.tif', 'w', **profile) as dst:
        dst.write(output_array, 1)
```

```
except:
    pass
# Load the output raster as a QgsRasterLayer
incertidumbre = QgsRasterLayer(working_dir+'/models/incertidumbre.tif', 'difference')




'''
FINAL CROPPED MODEL
'''
####################
try:
    os.mkdir(working_dir+'/forest_mask')
except:
    pass


canopy = QgsRasterLayer(working_dir + '/MCH/canopy_height_r5.tiff', 'canopy')
s2_res = ndvi.rasterUnitsPerPixelX()

output_dir = working_dir + '/forest_mask'
os.makedirs(output_dir, exist_ok=True)
output = output_dir + '/canopy_resampled.tif'

params_avg = {
    '-n': False,
    '-w': True,
    'GRASS_RASTER_FORMAT_META': '',
    'GRASS_RASTER_FORMAT_OPT': '',
    'GRASS_REGION_CELLSIZE_PARAMETER': int(s2_res),
    'GRASS_REGION_PARAMETER': None,
    'method': 0,
    'input': canopy,
    'output': output
}

processing.run("grass7:r.resamp.stats", params_avg)

# Carga el resultado como un QgsRasterLayer
height = QgsRasterLayer(output, 'Resampled Canopy')


def compute_forest_mask():
    entries = []
```

```
    temp_file = working_dir+'/forest_mask/forest_mask.tiff'

    AppendRaster2CalcEntry(
        entries    = entries,
        ref_name   = 'ndvi@1',
        raster     = ndvi,
        bandnumber = 1)

    AppendRaster2CalcEntry(
        entries    = entries,
        ref_name   = 'height@1',
        raster     = height,
        bandnumber = 1)

    formula = "(('ndvi@1' > 0.7) AND ('height@1' > 3)) * 1.0"

    NDVI = QgsRasterCalculator(
        formulaString    = formula,
        outputFile       = temp_file,
        outputFormat     = 'GTiff',
        outputExtent     = ndvi.extent(),
        nOutputColumns   = ndvi.width(),
        nOutputRows      = ndvi.height(),
        rasterEntries    = entries)

    NDVI.processCalculation()
    layer =  QgsRasterLayer(temp_file, 'f_mask')
    return layer

forest_mask=compute_forest_mask()
output = output_dir + '/forest_mask_r20.tif'
params_avg = {
    '-n': False,
    '-w': True,
    'GRASS_RASTER_FORMAT_META': '',
    'GRASS_RASTER_FORMAT_OPT': '',
    'GRASS_REGION_CELLSIZE_PARAMETER': 20,
    'GRASS_REGION_PARAMETER': None,
    'method': 0,
    'input': forest_mask,
    'output': output
}

processing.run("grass7:r.resamp.stats", params_avg)
```

```
1333
1334
1335  def mask_forest(raster, output):
1336      entries = []
1337
1338
1339      AppendRaster2CalcEntry(
1340          entries    = entries,
1341          ref_name   = 'raster@1',
1342          raster     = raster,
1343          bandnumber = 1)
1344
1345      AppendRaster2CalcEntry(
1346          entries    = entries,
1347          ref_name   = 'mask@1',
1348          raster     = forest_mask,
1349          bandnumber = 1)
1350
1351      formula = "'mask@1'*'raster@1'"
1352
1353      NDVI = QgsRasterCalculator(
1354          formulaString    = formula,
1355          outputFile       = output,
1356          outputFormat     = 'GTiff',
1357          outputExtent     = raster.extent(),
1358          nOutputColumns   = raster.width(),
1359          nOutputRows      = raster.height(),
1360          rasterEntries    = entries)
1361
1362      NDVI.processCalculation()
1363      layer =  QgsRasterLayer(output, 'final_')
1364      return layer
1365
1366
1367  #FINAL MODEL.
1368  compute_forest_mask()
1369  final_model=mask_forest(AOO_real_rlayer,
1370      working_dir+'/Clasification.tif')
1371
1372  #FINAL INCERTIDUMBRE.
1373  final_incert=mask_forest(incertidumbre,
1374      working_dir+'/Incertidumbre.tif')
1375
1376
1377
```

```
# Navigation and Canvas view parameters --------------------
# ----------------------------------------
#Zoom to StudyArea
time.sleep(0.5)
zoom_to_layer(StudyArea[0])                                    # zoom a zona de estudio
collapse_all_layers()                                          # Colapsa todo
deactivate_group(group_names['Climate_group'])                 # Desactiva clima
deactivate_group(resampled_group_name)                         # Desactiva resampled
deactivate_group('mch')                                        # Desactiva DSM
deactivate_group('S2')                                         # Desactiva S2
expand_group(group_names['Climate_group'])                     # Expande clima

apply_style(                                                   # Aplica hillshade
    layer      = QgsProject.instance().mapLayersByName(DEM[0])[0],
    style_file = os.path.abspath('styles/DEMStyle.qml'))

apply_style(                                                   # Aplica hillshade
    layer      = veg_map,
    style_file = os.path.abspath('styles/VegMap.qml'))

modify_qml_mch('styles/canopy_height.qml', minumun_canopy_height)
for mch_layer in mch_group.children():
    apply_style(                                                # MCH
        layer      = mch_layer.layer(),
        style_file = os.path.abspath('styles/canopy_height.qml'))

move_group(
    group_name     = group_names['DEM'],
    group_position = 4)

move_group(
    group_name     = 'mch',
    group_position = 2)

#--- --- --- --- --- --- --- --- --- --- --- --- --- ---
#--- --- --- --- --- --- --- --- --- --- --- --- --- ---
#--- --- --- --- --- --- --- --- --- --- --- --- --- ---




# Save variables
variables_df.to_csv(os.path.abspath(working_dir + '/variables_df.csv'), index=False)

# Save all samplings
```

```
numeric_columns = samplings.select_dtypes(include=['int32', 'float64'])
# Save the selected columns to a CSV file
numeric_columns.to_csv(os.path.abspath(working_dir + '/samplings.csv'), index=False)


et = time.time()
elapsed_time = et - st
print('Execution time:', elapsed_time, 'seconds \n')

d.autor()



#Climogram - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

meses=['Ene','Feb','Mar','Abr','May','Jun','Jul','Ago','Sep','Oct','Nov','Dic']

# Climogram figure - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#plt.figure()
fig, ax1 = plt.subplots()
ax2 = ax1.twinx()

ax1.bar(meses, T_pluv_Month_Average, label='Precipitación')

ax2.plot(meses, T_mean_Month_Average, '#2f2d2d', marker='o', markersize=2.5, label='Tž Media')
ax2.plot(meses, T_max_Month_Average, '#e60000',linestyle='dashed',linewidth=1.1, marker='o', markersize=
ax2.plot(meses, T_min_Month_Average, 'c-',linestyle='dashed',linewidth=1.1, marker='o', markersize=2.5,

ax2.legend(loc='upper center', bbox_to_anchor=(0.68, -0.05),
          fancybox=True, shadow=True, ncol=5)
ax1.legend(loc='upper center', bbox_to_anchor=(0.11, -0.05),
          fancybox=True, shadow=True, ncol=5)

ax2.set_ylabel('Temperatura (žC)')
ax1.set_ylabel('Precipitación (mm)')

ax1.set_yticks([-200,0,200,400,600,800,1000,1200,1400,1600,1800,2000])
ax2.set_yticks([-100,0,100,200,300,400,500,600,700,80,900,1000])

ax2.axes.get_xaxis().set_visible(False)
ax1.spines['bottom'].set_position(('data', 0.))
ax2.spines['bottom'].set_position(('data', 0.))
ax1.set_xticklabels(meses, rotation = 30, ha="right")
```

```

plt.title("Climograma del PNPE")

plt.savefig(os.path.abspath(figures_path+'/climogram.svg'), format='svg')
plt.show()

# Métricas climograma

# Mes más frío
print('> Minimo de Tmin:', str(round(min(T_min_Month_Average),2))+'žC',
    'en', meses[T_min_Month_Average.index(min(T_min_Month_Average))], '\n')

# Mes más hot
print('> Máximo de Tmax:', str(round(max(T_max_Month_Average),2))+'žC',
    'en', meses[T_max_Month_Average.index(max(T_max_Month_Average))], '\n')

# Mes más seco
print('> Mes más seco:', str(round(min(T_pluv_Month_Average),2))+' mm',
    'en', meses[T_pluv_Month_Average.index(min(T_pluv_Month_Average))],"\n")

# Moran's I figure (only presence points)- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
def reg_line(df, dfx_name, dfy_name, point_label, x_label, y_label, filename, titulo):
    x = df[dfx_name]
    y = df[dfy_name]

    # Fit a polynomial of degree 2 to the data
    coefficients = np.polyfit(x, y, 1)
    polynomial = np.poly1d(coefficients)

    # Calculate the residuals
    residuals = y - polynomial(x)

    # Calculate R^2
    r_squared = 1 - (np.sum(residuals**2) / ((len(y) - 1) * np.var(y, ddof=1)))

    # Plot the data and the polynomial curve
    plt.figure()
    plt.plot(x, polynomial(x), c='r', zorder=1, linewidth=0.75)
    plt.scatter(x, y, s=7, zorder=2, label = point_label)

    # Add vertical lines
    for xi, yi in zip(x, y):
        plt.plot([xi, xi], [yi, polynomial(xi)], c='k', linestyle='dashed', linewidth=0.5)

    plt.xlabel(x_label)
```

```
    plt.ylabel(y_label)
    plt.title(titulo)
    plt.plot(x, polynomial(x), c='r', zorder=1, linewidth=0.75, label=f'Regresión lineal (R2 = {r_square
    plt.legend(fontsize='small', loc='lower left')
    plt.savefig(os.path.abspath(figures_path+'/'+filename+'.svg'), format='svg')
    plt.show()

for climate_feature in climate_features:
    reg_line(
        df           = samplings,
        dfx_name     = 'MinDistance',
        dfy_name     = 'moran_'+climate_feature,
        point_label  = 'Muestreos',
        x_label      = 'Distancia mínima de muestreo',
        y_label      = 'I de Moran',
        filename     = 'Moran_Climate_'+climate_feature,
        titulo       = 'Autocorrelación espacial de '+ climate_feature)

for topo_feature in topo_features:
    reg_line(
        df           = samplings,
        dfx_name     = 'MinDistance',
        dfy_name     = 'moran_'+topo_feature,
        point_label  = 'Muestreos',
        x_label      = 'Distancia mínima de muestreo',
        y_label      = 'I de Moran',
        filename     = 'Moran_Topo_'+climate_feature,
        titulo       = 'Autocorrelación espacial de '+ topo_feature)

try:
    for s2_feat in S2_features:
        reg_line(
            df           = samplings,
            dfx_name     = 'MinDistance',
            dfy_name     = 'moran_'+topo_feature,
            point_label  = 'Muestreos',
            x_label      = 'Distancia mínima de muestreo',
            y_label      = 'I de Moran',
            filename     = 'Moran_S2_'+s2_feat,
            titulo       = 'Autocorrelación espacial de '+ s2_feat)
except:
    pass
```

A.3. Módulos, funciones y objetos

```
from qgis.core import QgsVectorLayer,QgsRasterLayer, QgsVectorFileWriter, QgsGeometry, QgsFeature, QgsPr
from PyQt5.QtWidgets import QFileDialog
import random




from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
import geopandas as gpd

import rasterio
from rasterio.crs import CRS

#Moran's I
from libpysal.weights import Queen
from pysal.lib import weights
from esda.moran import Moran

# Correlation
import seaborn as sns

#Random Forest
from pyimpute import load_training_vector
from pyimpute import load_targets

# import machine learning classifiers
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier #relative importance

from pyimpute import impute
from sklearn import model_selection


#QgsProject.instance().removeAllMapLayers() #Borra capas
#print()

def nube_puntos(n_puntos, x1, y1, x2, y2):
    '''Devuelve un objeto QgsGeometry con N puntos entre las coord deseadas
    Generado desde una lista (funcion genrerar_numeros)'''
```

```
43      def generar_numeros(n_puntos, min_, max_):
44          '''Genera lista de N puntos entre valores min_ y max_'''
45          return list(map(lambda n: random.randint(min_,max_), [None]*n_puntos))
46      return QgsGeometry.fromPolygonXY([list(map(lambda x,y: QgsPointXY(x,y),
47      generar_numeros(n_puntos, x1, x2),generar_numeros(n_puntos, y1, y2)))])
48
49
50  def trazar_puntos(puntos):
51      '''puntos > QgsGeometry (polygon)'''
52      layer = QgsVectorLayer('Point?crs=epsg:25830&field=id:integer', 'puntos' ,'memory')
53      vpr = layer.dataProvider()
54      for v in puntos.vertices():
55          feat =QgsFeature()      # crea una entidad para almacenar cada punto, ==cada fila
56          feat.setGeometry(v)     # asigna al objeto feat la geometría de cada punto.
57          vpr.addFeatures([feat]) # añade cada entidad al proveedor de datos
58
59      QgsProject.instance().addMapLayer(layer) #añade capa a la vista
60      iface.zoomToActiveLayer()   #zoom a la capa activa
61
62  #Ej trazar puntos:
63  #trazar_puntos(nube_puntos(5,0,0,50,50)) #5 puntos entre (0,0) y (50,50)
64
65  def QgsGeometry_a_QgsPointXY(geometria):
66      '''QgsGeometry -> QgsPointXY:
67      Solo funciona si es un punto.
68      Util para pasar centroide (QgsGeometry) a QgspointXY'''
69      QgsPointXY(list((map(lambda x: x, geometria.vertices())))[0])
70
71
72
73  def cargar_wms(url,nombre_capa,crs,formato,alias):
74      uri = "url="+url+"?&crs="+crs+"&format=image/"+formato+"&layers="+nombre_capa+"&styles"
75
76      #carga la capa:
77      rlayer = QgsRasterLayer(uri, alias, "wms")
78      print(uri)
79      if not rlayer.isValid():
80          print ("Failed to load.")
81
82      #añade la capa:
83      return QgsProject.instance().addMapLayer(rlayer)
84
85  #Ej cargar wms
86  #Parámetros de la función
```

```
87
88  # url='http://www.ign.es/wms-inspire/pnoa-ma?Request=GetCapabilities&Service=WMS'
89  # crs='EPSG:25830'
90  # formato='png'
91  # nombre_capa='OI.OrthoimageCoverage'
92  # cargar_wms(url,nombre_capa,crs,formato, 'Cobertura')
93
94
95  def abrir_shape(directorio,capa,alias, trazar, group):
96      '''Abre .shp de un directorio con un alias'''
97      try:
98          layer=QgsVectorLayer(directorio +'/'+ capa, alias, 'ogr')
99          if not layer.isValid():
100             layer = QgsVectorLayer(directorio, capa)
101         if not layer.isValid():
102             print ('> Error al cargar capa', capa, 'desde', directorio)
103         else:
104             if trazar:
105                 print('> Capa', alias, 'cargada desde',directorio)
106                 if group:
107                     QgsProject.instance().addMapLayers([layer], False)
108                     group.addLayer(layer)
109                 else:
110                     QgsProject.instance().addMapLayers([layer])
111
112                 return layer
113             else:
114                 print('> Capa', alias, 'cargada (sin trazar) desde',directorio)
115                 return layer
116     except:
117         print(' >')
118
119 #EJ
120
121 # abrir_shape(directorio = 'C:/PAGEOP',
122             # capa       = 'Comunidades_autonomas.shp',
123             # alias      = 'CCAA')
124
125 def abrir_raster(directorio, capa, alias, trazar, group):
126     rlayer = QgsRasterLayer(directorio +"/"+capa, alias)
127     if not rlayer.isValid():
128         rlayer = QgsRasterLayer(directorio, capa)
129     if not rlayer.isValid():
130         print ('> Error al cargar capa', capa, 'desde', directorio)
131     else:
```

```
132         if trazar:
133             print('> Capa', alias, 'cargada desde',directorio)
134             if group:
135                 QgsProject.instance().addMapLayers([rlayer], False)
136                 group.addLayer(rlayer)
137             else:
138                 QgsProject.instance().addMapLayers([rlayer])
139 
140             return rlayer
141         else:
142             print('> Capa', alias, 'cargada (sin trazar) desde',directorio)
143             return rlayer
144 
145 
146 
147 def abrir_csv(nombre, dir):
148     csv_url = "file:///" + dir + '/'+ nombre + "?delimiter=,"
149     nombre, extension = os.path.splitext(nombre)
150     csv = QgsVectorLayer(csv_url, nombre, "delimitedtext")
151     QgsProject.instance().addMapLayer(csv)
152     if not csv.isValid():
153         print("\n El csv no se ha cargado. Comprobar '.csv' en nombre")
154     else:
155         print('\n > CSV '+nombre+extension+ ' cargado exitosamente')
156 
157 def cargar_csv(directorio,nombre,epsg):
158     dir_csv = "file:///"+directorio+"/"+nombre+'?delimiter={}&crs=epsg:4258&xField={}&yField={}'.format
159     a = QgsVectorLayer(dir_csv, "Nucleos urbanos", "delimitedtext")
160     QgsProject.instance().addMapLayer(a)
161     return a
162 
163 def clonar_shape(directorio,capa,nombre_archivo,CRS_destino):
164     capa_dest = directorio +'/'+nombre_archivo+'.shp'
165     print('> Capa',capa.name(),'clonada a', nombre_archivo+'.shp','en', directorio)
166     QgsVectorFileWriter.writeAsVectorFormat(capa, capa_dest,'UTF-8',CRS_destino,'ESRI Shapefile')
167 
168 def abrir_archivos(iface,directorio):
169         title_window = 'Seleccionar archivos'
170         archivos = QFileDialog.getOpenFileNames(None, title_window, directorio)
171         return(archivos)
172 
173 
174 
175 def autor():
176     print('\n Autor: Daniel Pfitzer')
```

```
177
178 def limpiar():
179         QgsProject.instance().removeAllMapLayers()
180         root = QgsProject.instance().layerTreeRoot()
181         for group in [child for child in root.children() if child.nodeType() == 0]:
182             root.removeChildNode(group)
183
184
185         print('\n' * 50)
186
187
188 # CLASSES   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
189 class SquareWidget(QWidget):
190     def __init__(self, dem_spinbox, s2_spinbox, vegmap_edit, mean_edit, *args, **kwargs):
191         super().__init__(*args, **kwargs)
192         self.dem_spinbox = dem_spinbox
193         self.s2_spinbox = s2_spinbox
194         self.vegmap_edit = vegmap_edit
195         self.mean_edit = mean_edit
196         self.dem_spinbox.valueChanged.connect(self.update)
197         self.s2_spinbox.valueChanged.connect(self.update)
198         self.vegmap_edit.textChanged.connect(self.update)
199         self.mean_edit.textChanged.connect(self.update)
200
201     def paintEvent(self, event):
202         painter = QPainter(self)
203         dem_size = self.dem_spinbox.value()
204         s2_size = self.s2_spinbox.value()
205         vegmap_size = 200
206         max_size = max(dem_size, s2_size, vegmap_size)
207         scale = min(self.width(), self.height()) / max_size
208         painter.scale(scale, scale)
209
210         # Set pen color to match brush color for each square
211         painter.setPen(QColor("#4c56ae"))
212         painter.setBrush(QColor("#4c56ae"))
213         painter.drawRect(0, 0, vegmap_size, vegmap_size)
214
215         painter.setPen(QColor("#e16e73"))
216         painter.setBrush(QColor("#e16e73"))
217         painter.drawRect(0, 0, dem_size, dem_size)
218
219         painter.setPen(QColor("#94ba4e"))
220         painter.setBrush(QColor("#94ba4e"))
221         painter.drawRect(0, 0, s2_size, s2_size)
```

```
222
223         # Draw grid
224         grid_size = int(s2_size)
225         grid_color = QColor("#94ba4e")
226         painter.setPen(grid_color)
227         for x in range(0, max_size + 1, grid_size):
228             painter.drawLine(x, 0, x, max_size)
229             painter.drawLine(0, x, max_size, x)
230
231
232
233     def get_vegmap_resolution(self):
234         try:
235             raster_path = self.mean_edit.text().split(';')[0]
236             with rasterio.open(raster_path) as src:
237                 res = src.res[0]
238             return res
239         except:
240             return 0
241
242
243 class RichTextLabel(QLabel):
244     def __init__(self, *args, **kwargs):
245         super().__init__(*args, **kwargs)
246         self.doc = QTextDocument(self)
247
248     def setHtml(self, html):
249         self.doc.setHtml(html)
250         self.adjustSize()
251
252     def paintEvent(self, event):
253         painter = QPainter(self)
254         self.doc.drawContents(painter)
255
256 class input_data(QDialog):
257     def __init__(self, parent=None):
258         super().__init__(parent)
259         self.setWindowTitle(" ")
260
261         # Set the default width of the window
262         self.setFixedWidth(443)
263
264
265         # Create tab widget and add tabs
266         self.tab_widget = QTabWidget(self)
```

```
267         self.input_tab = QWidget(self)
268         self.mch_tab = QWidget(self)
269         self.NDVI_tab = QWidget(self)
270         self.vegmap_tab = QWidget(self)
271 
272 
273         self.tab_widget.addTab(self.input_tab, "Input data")
274         self.tab_widget.addTab(self.vegmap_tab, "Parameters")
275         self.tab_widget.addTab(self.mch_tab, "MCH Filter")
276         self.tab_widget.addTab(self.NDVI_tab, "NDVI Filter")
277 
278 
279         # Create layouts for each tab
280         self.input_layout = QGridLayout()
281         self.mch_layout = QGridLayout()
282         self.NDVI_layout = QGridLayout()
283         self.vegmap_layout = QGridLayout()
284 
285         self.input_tab.setLayout(self.input_layout)
286         self.mch_tab.setLayout(self.mch_layout)
287         self.NDVI_tab.setLayout(self.NDVI_layout)
288         self.vegmap_tab.setLayout(self.vegmap_layout)
289 
290         # A create widgets for first tab = = = = = = = = = = = = = = = = = = = = =
291 
292 
293         # A 1.1Study area
294         self.area_label  = QLabel("Study area             ")
295         self.area_edit   = QLineEdit()
296         self.area_button = QPushButton("...", clicked =self.select_area_files)
297 
298         # A 1.2 Mean temperature
299         self.mean_label = QLabel("Mean temperatures")
300         self.mean_edit = QLineEdit()
301         self.mean_button = QPushButton("...", clicked=self.select_mean_files)
302 
303         # A 1.3 Max temperature
304         self.max_label = QLabel("Max temperature")
305         self.max_edit = QLineEdit()
306         self.max_button = QPushButton("...", clicked=self.select_max_files)
307 
308         # A 1.4 Min temperature
309         self.min_label = QLabel("Min temperature")
310         self.min_edit = QLineEdit()
```

```
311         self.min_button = QPushButton("...", clicked=self.select_min_files)
312
313         # A 1.5 Pluviometry
314         self.pluv_label = QLabel("Pluviometry")
315         self.pluv_edit = QLineEdit()
316         self.pluv_button = QPushButton("...", clicked=self.select_pluv_files)
317
318         # A 1.6 DEM
319         self.dem_label = QLabel("DEM(s) ")
320         self.dem_edit = QLineEdit()
321         self.dem_button = QPushButton("...", clicked=self.select_dem_files)
322
323         # A 1.7 Satellite
324         self.ls_label = QLabel("Sentinel 2 ")
325         self.ls_edit = QLineEdit()
326         self.ls_button = QPushButton("...", clicked=self.select_s2_files)
327
328         # A 1.7 VegMap
329         self.vegmap_label = QLabel("Vegetation Map ")
330         self.vegmap_edit = QLineEdit()
331         self.vegmap_button = QPushButton("...", clicked=self.select_vegmap_files)
332
333         self.study_area_ok_button = QPushButton("Ok", clicked =self.study_area_ok)
334
335         # A 2 Add widgets to first layout
336         self.input_layout.addWidget(self.area_label, 0, 0)
337         self.input_layout.addWidget(self.area_edit, 0, 1)
338         self.input_layout.addWidget(self.area_button, 0, 2)
339
340         self.input_layout.addWidget(self.mean_label, 1, 0)
341         self.input_layout.addWidget(self.mean_edit, 1, 1)
342         self.input_layout.addWidget(self.mean_button, 1, 2)
343
344         self.input_layout.addWidget(self.max_label, 2, 0)
345         self.input_layout.addWidget(self.max_edit, 2, 1)
346         self.input_layout.addWidget(self.max_button, 2, 2)
347
348         self.input_layout.addWidget(self.min_label, 3, 0)
349         self.input_layout.addWidget(self.min_edit, 3, 1)
350         self.input_layout.addWidget(self.min_button, 3, 2)
351
352         self.input_layout.addWidget(self.pluv_label, 4, 0)
353         self.input_layout.addWidget(self.pluv_edit, 4, 1)
354         self.input_layout.addWidget(self.pluv_button, 4, 2)
```

```
355
356         self.input_layout.addWidget(self.dem_label, 5, 0)
357         self.input_layout.addWidget(self.dem_edit, 5, 1)
358         self.input_layout.addWidget(self.dem_button, 5, 2)
359
360         self.input_layout.addWidget(self.ls_label, 6, 0)
361         self.input_layout.addWidget(self.ls_edit, 6, 1)
362         self.input_layout.addWidget(self.ls_button, 6, 2)
363
364         self.input_layout.addWidget(self.vegmap_label, 7, 0)
365         self.input_layout.addWidget(self.vegmap_edit, 7, 1)
366         self.input_layout.addWidget(self.vegmap_button, 7, 2)
367
368         self.input_layout.addWidget(self.study_area_ok_button, 8, 2)
369
370
371         # B create widgets for second tab = = = = =  = = = = = = = = = = = = = =
372
373         # B 1.1 DSM
374         self.file_label = QLabel('DSM')
375         self.file_edit = QLineEdit()
376         self.file_btn = QPushButton('...')
377         self.file_btn.clicked.connect(self.select_file)
378
379         # B 1.2 Minimun canopy height + slider
380         self.mch_label = QLabel('Minimun Canopy Height')
381         self.mch_spinbox = QDoubleSpinBox()
382         self.mch_spinbox.setMinimum(1)
383         self.mch_spinbox.setMaximum(30)
384         self.mch_spinbox.setDecimals(1)
385         self.mch_spinbox.setValue(3)
386         self.mch_spinbox.setAlignment(Qt.AlignCenter)
387
388         self.mch_slider = QSlider(Qt.Horizontal)
389         self.mch_slider.setMinimum(10 * self.mch_spinbox.minimum())
390         self.mch_slider.setMaximum(10 * self.mch_spinbox.maximum())
391         self.mch_slider.setValue(10 * self.mch_spinbox.value())
392         self.mch_slider.setTickPosition(QSlider.TicksBelow)
393         self.mch_slider.setTickInterval(1)
394         self.mch_slider.valueChanged.connect(lambda value: self.mch_spinbox.setValue(value / 10))
395         self.mch_slider.setTickPosition(QSlider.NoTicks)
396
397         # B 1.3 Gaussian iterations number
398         self.gauss_label = QLabel('Gaussian filter iterations')
399         self.gauss_spinbox = QSpinBox()
```

```
400             self.gauss_spinbox.setMinimum(1)
401             self.gauss_spinbox.setMaximum(30)
402             self.gauss_spinbox.setValue(1)
403             self.gauss_spinbox.setAlignment(Qt.AlignCenter)
404
405             self.gauss_slider = QSlider(Qt.Horizontal)
406             self.gauss_slider.setMinimum(10 * self.gauss_spinbox.minimum())
407             self.gauss_slider.setMaximum(10 * self.gauss_spinbox.maximum())
408             self.gauss_slider.setValue(10 * self.gauss_spinbox.value())
409
410             self.gauss_slider.setTickPosition(QSlider.TicksBelow)
411             self.gauss_slider.setTickInterval(1)
412             self.gauss_slider.valueChanged.connect(lambda value: self.gauss_spinbox.setValue(value / 10))
413             self.gauss_slider.setTickPosition(QSlider.NoTicks)
414
415             # B 1.4 STD
416             self.std_label = QLabel('STD')
417             self.std_spinbox = QDoubleSpinBox()
418             self.std_spinbox.setMinimum(1)
419             self.std_spinbox.setMaximum(200)
420             self.std_spinbox.setDecimals(2)
421             self.std_spinbox.setValue(41.85)
422             self.std_spinbox.setAlignment(Qt.AlignCenter)
423
424             self.std_slider = QSlider(Qt.Horizontal)
425             self.std_slider.setMinimum(10 * self.std_spinbox.minimum())
426             self.std_slider.setMaximum(10 * self.std_spinbox.maximum())
427             self.std_slider.setValue(10 * self.std_spinbox.value())
428             self.std_slider.setTickPosition(QSlider.TicksBelow)
429             self.std_slider.setTickInterval(1)
430             self.std_slider.valueChanged.connect(lambda value: self.std_spinbox.setValue(value / 10))
431             self.std_slider.setTickPosition(QSlider.NoTicks)
432
433             # B 1.5 Kernel radio
434             self.radio_label = QLabel('Kernel radius')
435             self.radio_spinbox = QSpinBox()
436             self.radio_spinbox.setMinimum(1)
437             self.radio_spinbox.setMaximum(200)
438             self.radio_spinbox.setValue(1)
439             self.radio_spinbox.setAlignment(Qt.AlignCenter)
440
441             self.rad_slider = QSlider(Qt.Horizontal)
442             self.rad_slider.setMinimum(1)
443             self.rad_slider.setMaximum(50)
```

```
444         self.rad_slider.setValue(10 * self.radio_spinbox.value())
445         self.rad_slider.setTickPosition(QSlider.TicksBelow)
446         self.rad_slider.setTickInterval(1)
447         self.rad_slider.valueChanged.connect(lambda value: self.radio_spinbox.setValue(value))
448         self.rad_slider.setTickPosition(QSlider.NoTicks)
449 
450 
451         self.mch_checkbox = QCheckBox('Enable MCH filter')
452         self.mch_checkbox.setChecked(True)
453 
454         #self.mch_checkbox2 = QCheckBox('Enable Gaussian filter')
455         #self.mch_checkbox2.setChecked(False)
456 
457         # B 2 Add widgets to mean temp layout
458 
459         self.collapsibleBox = QGroupBox("Enable Gaussian filter")
460         self.collapsibleBox.setCheckable(True)
461         self.collapsibleBox.setChecked(False)
462 
463         collapsibleLayout = QGridLayout()
464         collapsibleLayout.addWidget(self.gauss_label, 0, 0)
465         collapsibleLayout.addWidget(self.gauss_spinbox , 0, 1)
466         collapsibleLayout.addWidget(self.gauss_slider , 0, 2)
467 
468         collapsibleLayout.addWidget(self.std_label, 1, 0)
469         collapsibleLayout.addWidget(self.std_spinbox, 1, 1)
470         collapsibleLayout.addWidget(self.std_slider , 1, 2)
471 
472         collapsibleLayout.addWidget(self.radio_label, 2, 0)
473         collapsibleLayout.addWidget(self.radio_spinbox, 2, 1)
474         collapsibleLayout.addWidget(self.rad_slider , 2, 2)
475 
476         self.collapsibleBox.setLayout(collapsibleLayout)
477 
478         self.mch_layout.addWidget(self.mch_checkbox, 0, 0)
479 
480         self.mch_layout.addWidget(self.file_label, 2, 0)
481         self.mch_layout.addWidget(self.file_edit, 2, 1)
482         self.mch_layout.addWidget(self.file_btn, 2, 2)
483 
484         self.mch_layout.addWidget(self.mch_label, 3, 0)
485         self.mch_layout.addWidget(self.mch_spinbox , 3, 1)
486         self.mch_layout.addWidget(self.mch_slider , 3, 2)
487 
```

```
488         spacer = QLabel()
489         spacer.setFixedHeight(20)
490         self.mch_layout.addWidget(spacer ,4 ,0)
491 
492         self.mch_layout.addWidget(self.collapsibleBox ,5 ,0 ,1 ,3)
493 
494 
495         # C Add tab widget to main layout
496         main_layout = QVBoxLayout()
497         main_layout.addWidget(self.tab_widget)
498         self.setLayout(main_layout)
499 
500         # D create widgets for second tab = = = = =  = = = = = = = = = = = = = =
501 
502         # D 1.1 Minimun NDVI + slider + add
503         self.NDVI_label = QLabel('Minimun NDVI')
504         self.NDVI_spinbox = QDoubleSpinBox()
505         self.NDVI_spinbox.setMinimum(0)
506         self.NDVI_spinbox.setMaximum(1)
507         self.NDVI_spinbox.setDecimals(2)
508         self.NDVI_spinbox.setValue(0.6)
509         self.NDVI_spinbox.setSingleStep(0.01)
510         self.NDVI_spinbox.setAlignment(Qt.AlignCenter)
511 
512         self.NDVI_slider = QSlider(Qt.Horizontal)
513         self.NDVI_slider.setMinimum(100 * self.NDVI_spinbox.minimum())
514         self.NDVI_slider.setMaximum(100 * self.NDVI_spinbox.maximum())
515         self.NDVI_slider.setValue(100 * self.NDVI_spinbox.value())
516         self.NDVI_slider.setTickPosition(QSlider.TicksBelow)
517         self.NDVI_slider.setTickInterval(1)
518         self.NDVI_slider.valueChanged.connect(lambda value: self.NDVI_spinbox.setValue(value / 100))
519         self.NDVI_slider.setTickPosition(QSlider.NoTicks)
520 
521         self.NDVI_layout.addWidget(self.NDVI_label, 1, 0)
522         self.NDVI_layout.addWidget(self.NDVI_spinbox , 1, 1)
523         self.NDVI_layout.addWidget(self.NDVI_slider , 1, 2)
524 
525 
526         self.NDVI_checkbox = QCheckBox('Enable NDVI filter')
527         self.NDVI_checkbox.setChecked(True)
528         self.NDVI_layout.addWidget(self.NDVI_checkbox, 0, 0)
529 
530 
531          # Params tab
```

```
532         print(self.vegmap_edit.text().split(';'))
533 
534         self.Climate_Res_label = RichTextLabel()
535         self.Climate_Res_label.setHtml('<span style="color:#4c56ae">\u25A0</span> Climate resolution [m]
536         self.Climate_Res_value = QLabel()
537         self.Climate_Res_value.setText('              200')
538 
539         self.DEM_Res_label = RichTextLabel()
540         self.DEM_Res_label.setHtml('<span style="color:#e16e73">\u25A0</span> Resampled DEM [m]')
541         self.DEM_Res_spinbox = QDoubleSpinBox()
542         self.DEM_Res_spinbox.setMinimum(0)
543         self.DEM_Res_spinbox.setMaximum(1000)
544         self.DEM_Res_spinbox.setDecimals(0)
545         self.DEM_Res_spinbox.setValue(100)
546         self.DEM_Res_spinbox.setSingleStep(1)
547         self.DEM_Res_spinbox.setAlignment(Qt.AlignCenter)
548 
549         self.S2_Res_label = RichTextLabel()
550         self.S2_Res_label.setHtml('<span style="color:#94ba4e">\u25A0</span> Resampled S2   [m]')
551         self.S2_Res_spinbox = QDoubleSpinBox()
552         self.S2_Res_spinbox.setMinimum(0)
553         self.S2_Res_spinbox.setMaximum(1000)
554         self.S2_Res_spinbox.setDecimals(0)
555         self.S2_Res_spinbox.setValue(20)
556         self.S2_Res_spinbox.setSingleStep(1)
557         self.S2_Res_spinbox.setAlignment(Qt.AlignCenter)
558 
559         self.vegfield_label  = QLabel("Forest field")
560         self.vegfield_edit   = QLineEdit()
561         self.vegfield_edit.setText("DEN_COD2_E")
562 
563 
564         self.square_widget = SquareWidget(self.DEM_Res_spinbox, self.S2_Res_spinbox, self.dem_edit, sel
565 
566 
567         # Set the minimum size of the labels and spinboxes
568         self.Climate_Res_label.setMinimumHeight(25)  # Set the minimum height explicitly
569         self.Climate_Res_value.setMinimumHeight(self.Climate_Res_value.sizeHint().height())
570         self.DEM_Res_label.setMinimumHeight(self.DEM_Res_label.sizeHint().height())
571         self.DEM_Res_spinbox.setMinimumHeight(self.DEM_Res_spinbox.sizeHint().height())
572         self.S2_Res_label.setMinimumHeight(self.S2_Res_label.sizeHint().height())
573         self.S2_Res_spinbox.setMinimumHeight(self.S2_Res_spinbox.sizeHint().height())
574         self.vegfield_label.setMinimumHeight(self.vegfield_label.sizeHint().height())
575         self.vegfield_edit.setMinimumHeight(self.vegfield_edit.sizeHint().height())
```

```
576
577
578         # Create a horizontal layout for each pair of label and spinbox
579         climate_layout = QHBoxLayout()
580         climate_layout.addWidget(self.Climate_Res_label)
581         climate_layout.addWidget(self.Climate_Res_value)
582
583         dem_layout = QHBoxLayout()
584         dem_layout.addWidget(self.DEM_Res_label)
585         dem_layout.addWidget(self.DEM_Res_spinbox)
586
587         s2_layout = QHBoxLayout()
588         s2_layout.addWidget(self.S2_Res_label)
589         s2_layout.addWidget(self.S2_Res_spinbox)
590
591         field_layout = QHBoxLayout()
592         field_layout.addWidget(self.vegfield_label)
593         field_layout.addWidget(self.vegfield_edit)
594
595         # Create a vertical layout for the elements to the left of the square
596         left_layout = QVBoxLayout()
597         left_layout.setSpacing(10)  # Set the spacing between elements
598         area_layout = QHBoxLayout()
599         area_layout.addWidget(self.vegfield_label)
600         area_layout.addWidget(self.vegfield_edit)
601         left_layout.addLayout(area_layout)
602         left_layout.addStretch()
603         left_layout.addLayout(climate_layout)
604         left_layout.addLayout(dem_layout)
605         left_layout.addLayout(s2_layout)
606
607         left_layout.addStretch()  # Add a stretch element to occupy any additional space
608
609         # Create a new horizontal layout for self.area_label and self.area_edit
610
611
612         # Set a fixed size for the square widget
613         self.square_widget.setFixedSize(150, 150)
614
615         # Add the new layout to the left_layout
616
617
618
619         # Create a horizontal layout and add the vertical layout and the square
620         h_layout = QHBoxLayout()
```

```
621         h_layout.addLayout(left_layout)
622         h_layout.addWidget(self.square_widget)
623 
624         # Add the horizontal layout to the main layout
625         self.vegmap_layout.addLayout(h_layout, 1, 0)
626 
627 
628 
629 
630         # E Prev inputs:
631         try:
632             if len(input_paths[0])!=0:
633                 self.area_edit.setText(';'.join(input_paths[0]))
634 
635             if len(input_paths[1])!=0:
636                 self.mean_edit.setText(';'.join(input_paths[1]))
637 
638             if len(input_paths[2])!=0:
639                 self.max_edit.setText(';'.join(input_paths[2]))
640 
641             if len(input_paths[3])!=0:
642                 self.min_edit.setText(';'.join(input_paths[3]))
643 
644             if len(input_paths[4])!=0:
645                 self.pluv_edit.setText(';'.join(input_paths[4]))
646 
647             if len(input_paths[5])!=0:
648                 self.dem_edit.setText(';'.join(input_paths[5]))
649 
650             if len(input_paths[6])!=0:
651                 self.ls_edit.setText(';'.join(input_paths[6]))
652 
653             if len(input_paths[8])!=0:
654                 self.file_edit.setText(';'.join(input_paths[8]))
655 
656             if len(input_paths[7])!=0:
657                 self.vegmap_edit.setText(';'.join(input_paths[7]))
658 
659 
660         except:
661             print('\n > Check older input .txt data versions \n')
662 
663     def select_area_files(self):
664         files, _ = QFileDialog.getOpenFileNames(self, "Select area of study","","Vector files (*.shp *.g
```

```
665             if files:
666                 self.area_edit.setText(';'.join(files))
667 
668         def select_mean_files(self):
669             files, _ = QFileDialog.getOpenFileNames(self, "Select Mean Temperature files"," " ,"Raster files
670             if files:
671                 self.mean_edit.setText(';'.join(files))
672 
673         def select_max_files(self):
674             files, _ = QFileDialog.getOpenFileNames(self, "Select Max Temperature files"," " ,"Raster files 
675             if files:
676                 self.max_edit.setText(';'.join(files))
677 
678         def select_min_files(self):
679             files, _ = QFileDialog.getOpenFileNames(self, "Select Min Temperature files"," " ,"Raster files 
680             if files:
681                 self.min_edit.setText(';'.join(files))
682 
683         def select_pluv_files(self):
684             files, _ = QFileDialog.getOpenFileNames(self, "Select Pluviometry Files"," " ,"Raster files (*.t
685             if files:
686                 self.pluv_edit.setText(';'.join(files))
687 
688         def select_dem_files(self):
689             files, _ = QFileDialog.getOpenFileNames(self, "Select DEM  Files"," " ,"Raster files (*.tif *.ti
690             if files:
691                 self.dem_edit.setText(';'.join(files))
692 
693         def select_vegmap_files(self):
694             files, _ = QFileDialog.getOpenFileNames(self, "Select Vegetation Map"," " ,"Vector files (*.shp)
695             if files:
696                 self.vegmap_edit.setText(';'.join(files))
697 
698         def study_area_ok(self):
699             self.accept()
700 
701         def get_study_area_files(self):
702             return self.area_edit.text().split(';'), self.mean_edit.text().split(';'), self.max_edit.text()
703 
704 
705         def get_mean_temp_files(self):
706             return self.mean_edit.text().split(';')
707 
708         def select_file(self):
709             files, _ = QFileDialog.getOpenFileNames(self, 'Select DSM', '', 'Todos los archivos (*);;')
```

```
710             if files:
711                 self.file_edit.setText(';'.join(files))
712 
713 
714 
715         def get_mch_data(self):
716             file  = self.file_edit.text().split(';')
717             mch   = self.mch_spinbox.value()
718             mch_check = self.mch_checkbox.isChecked()
719             gauss_check = False
720             NDVI  = self.NDVI_spinbox.value()
721             NDVI_check = self.NDVI_checkbox.isChecked()
722             iter  = self.gauss_spinbox.value()
723             std   = self.std_spinbox.value()
724             rad   = self.radio_spinbox.value()
725             topo_res = self.DEM_Res_spinbox.value()
726             S2_res = self.S2_Res_spinbox.value()
727             veg_field = self.vegfield_edit.text()
728 
729             return file, mch, mch_check, gauss_check, NDVI, NDVI_check, iter, std, rad, topo_res, S2_res, ve
730 
731         def select_area_files(self):
732             files, _ = QFileDialog.getOpenFileNames(self, "Select area of study","","Vector files (*.shp *.c
733             if files:
734                 self.area_edit.setText(';'.join(files))
735 
736         def select_mean_files(self):
737             files, _ = QFileDialog.getOpenFileNames(self, "Select Mean Temperature files"," " ,"Raster files
738             if files:
739                 self.mean_edit.setText(';'.join(files))
740 
741         def select_max_files(self):
742             files, _ = QFileDialog.getOpenFileNames(self, "Select Max Temperature files"," " ,"Raster files
743             if files:
744                 self.max_edit.setText(';'.join(files))
745 
746         def select_min_files(self):
747             files, _ = QFileDialog.getOpenFileNames(self, "Select Min Temperature files"," " ,"Raster files
748             if files:
749                 self.min_edit.setText(';'.join(files))
750 
751         def select_pluv_files(self):
752             files, _ = QFileDialog.getOpenFileNames(self, "Select Pluviometry Files"," " ,"Raster files (*.t
753             if files:
754                 self.pluv_edit.setText(';'.join(files))
```

```
755
756     def select_dem_files(self):
757         files, _ = QFileDialog.getOpenFileNames(self, "Select Pluviometry Files"," " ,"Raster files (*.t
758         if files:
759             self.dem_edit.setText(';'.join(files))
760
761     def select_s2_files(self):
762         files, _ = QFileDialog.getOpenFileNames(self, "Select Sentinel2 Files"," " ,"Raster files (*.tif
763         if files:
764             self.ls_edit.setText(';'.join(files))
765
766     def get_files(self):
767             #print (self.area_edit.text().split(';'))
768             return [self.area_edit.text().split(';'),
769             self.mean_edit.text().split(';'),
770             self.max_edit.text().split(';'),
771             self.min_edit.text().split(';'),
772             self.pluv_edit.text().split(';'),
773             self.dem_edit.text().split(';'),
774             self.veg_map_edit.text().split(';')]
775
776
777 # FUNCTIONS  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
778
779 def create_group(group_name:str):
780     group = QgsProject.instance().layerTreeRoot().addGroup(group_name)
781     group.setExpanded(False)  # Colapsa el grupo de capas
782     return group
783 def open_files(files:list, group_name:str, trazar:bool) -> tuple:
784     '''
785
786     '''
787
788     if files:
789         root = QgsProject.instance().layerTreeRoot()
790         if group_name is not None:
791             group = create_group(group_name)
792         else:
793             group = False             # If we dont want a group
794
795         # Itera sobre los archivos seleccionados y agrega cada uno como una capa raster a QGIS
796
797         names      = []
798         layers     = []
799
```

```
800         for file_path in files:
801             layer_name, extension = os.path.splitext(os.path.basename(file_path))  # Obtiene el nombre 
802             names.append(layer_name)
803 
804 
805             if extension in extensions_raster: #is it raster?
806                 layers.append(d.abrir_raster(
807                     directorio = file_path,
808                     capa       = layer_name,
809                     alias      = layer_name,
810                     trazar     = trazar,
811                     group      = group))
812 
813             elif extension in extensions_vector: # or is it vector type?
814                 layers.append(d.abrir_shape(
815                     directorio = file_path,
816                     capa       = layer_name,
817                     alias      = layer_name,
818                     trazar     = trazar,
819                     group      = group))
820 
821 
822             #Collapse
823             if group_name is not None and trazar is True:
824                 layer = QgsProject.instance().mapLayersByName(layer_name)[0]
825                 myLayerNode = root.findLayer(layer.id())
826                 myLayerNode.setExpanded(False)
827 
828             print(' ')
829 
830     return names, layers
831 
832 def merge_raster(message:str, group_name:str, dir_merged_name:str, files:list ,name_merged:str, alias:st
833 
834     dir_out = QFileDialog.getExistingDirectory(None, dir_merged_name, "/")
835     capaout = dir_out +'\\' + 'topo_merged.tiff'
836 
837     parameters = {
838         'INPUT' : files,
839         'OUTPUT': capaout}
840     processing.run('gdal:merge', parameters)
841 
842     group = QgsProject.instance().layerTreeRoot().addGroup(group_name)
843     group.setExpanded(False)  # Colapsa el grupo de capas
844 
```

```
845     return d.abrir_raster(
846                 directorio = capaout,
847                 capa       = capaout,
848                 alias      = alias,
849                 trazar     = trazar,
850                 group      = group)
851 
852 def dialog_box_options(opciones:list, title:str, dialog:str) -> tuple:
853     option, ok = QInputDialog.getItem(
854         None,       # cuadro de diálogo sea modal y se bloquee hasta que se cierre
855         title,
856         dialog,
857         opciones,
858         editable = False)
859 
860     return option, ok
861 
862 def move_group(group_name: str, group_position: int):
863     root = QgsProject.instance().layerTreeRoot()
864     group = root.findGroup(group_name)                    # Busca el grupo por su nombre
865     current_index = root.children().index(group)          # Obtiene el índice actual del grupo de capas
866     root.insertChildNode(group_position, group.clone())  # Inserta una copia del grupo de capas en la nu
867     root.removeChildNode(group)                           # Elimina el grupo original de su posición ant
868 
869 
870 def create_group_from_groups(group_names: list, new_group_name: str):
871     root = QgsProject.instance().layerTreeRoot()
872     new_group = root.addGroup(new_group_name)             # Crea un nuevo grupo
873 
874     for group_name in group_names:
875         group = root.findGroup(group_name)                # Busca el grupo por su nombre
876         if group:                                         # Si se encontró el grupo, lo inserta en el nu
877             new_group.insertChildNode(0, group.clone())
878             root.removeChildNode(group)                   # Elimina el grupo de su posición anterior
879 
880     return new_group
881 
882 def deactivate_group(group_name:str):
883     grupo = QgsProject.instance().layerTreeRoot().findGroup(group_name)
884     grupo.setItemVisibilityChecked(False)
885 
886 
887 def close_layers_in_group(group_name:str):
888     # Obtener el árbol de capas del proyecto
```

```
    root = QgsProject.instance().layerTreeRoot()

    # Obtener el grupo de capas por su nombre
    group = root.findGroup(group_name)

    if group:
        # Obtener una lista de todas las capas en el grupo
        layers = group.findLayers()

        # Cerrar cada capa en el grupo
        for layer in layers:
            QgsProject.instance().removeMapLayer(layer.layerId())
    else:
        print(f"No se encontró el grupo '{group_name}'.")

def collapse_all_layers():
    root = QgsProject.instance().layerTreeRoot()
    for node in root.children():
        set_layer_not_expanded(node)


def set_layer_not_expanded(node):
    node.setExpanded(False)
    for child_node in node.children():
        set_layer_not_expanded(child_node)

def expand_group(group_name:str):
    root = QgsProject.instance().layerTreeRoot()
    group = root.findGroup(group_name) # Busca el grupo por su nombre
    if group:
        group.setExpanded(True)        # Establece el valor de expanded en True para el nodo del grupo

def group_processing_raster(group_name:str, parameters:dict, output_dir:str, proccesing_name:str, procce
    '''Returns a list with the proccesed output names'''

    root = QgsProject.instance().layerTreeRoot()
    group = root.findGroup(group_name)

    layer_list =  []

    for layer in group.children():
        if isinstance(layer, QgsLayerTreeLayer):
            # Obtener la capa QgsMapLayer a partir de QgsLayerTreeLayer
            map_layer = layer.layer()
```

```
933
934             if isinstance(map_layer, QgsRasterLayer):
935
936                 # We set a different output file for each layer and append it to the layer list
937                 new_layer_name = map_layer.name() + '_' + proccesing_alias
938                 parameters['OUTPUT'] = output_dir + '/' + new_layer_name + '.tiff'
939                 parameters['output'] = output_dir + '/' + new_layer_name + '.tiff'
940
941                 map_layer_path = map_layer.dataProvider().dataSourceUri()
942                 parameters['INPUT'] = map_layer_path
943                 parameters['input'] = map_layer_path
944                 layer_list.append(parameters['output'])
945
946                 processing.run(proccesing_name, parameters)
947                 print('> '+new_layer_name+' succesfully proccesed \n')
948
949     return layer_list
950
951 def list_processing_raster(layer_list:list, parameters:dict, output_dir:str, proccesing_name:str, procce
952     '''Returns a list with the proccesed output names'''
953
954     output_layer_list =  []
955
956     for layer in layer_list:
957
958             map_layer = layer
959
960             if isinstance(map_layer, QgsRasterLayer):
961
962                 # We set a different output file for each layer and append it to the layer list
963                 new_layer_name = map_layer.name() + '_' + proccesing_alias
964                 parameters['OUTPUT'] = output_dir + '/' + new_layer_name + '.tiff'
965
966                 map_layer_path = map_layer.dataProvider().dataSourceUri()
967                 parameters['INPUT'] = map_layer_path
968                 output_layer_list.append(new_layer_name)
969
970                 processing.run(proccesing_name, parameters)
971                 print('> '+new_layer_name+' succesfully proccesed \n')
972
973     return output_layer_list
974
975 def apply_style(layer, style_file):
976     # Cargue el archivo .qml (o .sld) de estilo en la capa
977     layer.loadNamedStyle(style_file)
```

```
978
979         # Aplique el estilo cargado a la capa
980         layer.triggerRepaint()
981
982     def zoom_to_layer(layer_name):
983         vLayer = QgsProject.instance().mapLayersByName(layer_name)[0]
984         canvas = iface.mapCanvas()
985         extent = vLayer.extent()
986         canvas.setExtent(extent)
987         canvas.refresh()
988
989     def get_layer_URI_by_name(layer_name:str) -> str:
990         return QgsProject.instance().mapLayersByName(layer_name)[0].dataProvider().dataSourceUri()
991
992         # Busca la capa por su nombre
993         layer = project_instance.mapLayersByName(layer_name)[0]
994
995         # Obtiene la extensión de la capa y ajusta la vista del mapa
996         canvas = iface.mapCanvas()
997         canvas.setExtent(layer.extent())
998         canvas.refresh()
999
1000    def txt_to_list(txt_name):
1001        my_file = open(dir_path+'/last_input/'+txt_name+".txt", "r")
1002        data = my_file.read()
1003        data_into_list = data.split("\n")
1004        data_into_list = data_into_list[:len(data_into_list)-1]
1005        my_file.close()
1006        if (data_into_list) ==[]:
1007            return [data]
1008
1009        return data_into_list
1010
1011    def erase_files_on_dir(dir:str, extension:str):
1012
1013        # Find files with certain extension
1014        files_to_erase = glob.glob(os.path.join(dir, extension))
1015
1016        # Erase
1017        for file in files_to_erase:
1018            os.remove(file)
1019
1020    def append_raster_to_calc_entry(entries:list, ref_name:str, raster:QgsRasterLayer, bandnumber:int):
1021        ras = QgsRasterCalculatorEntry()
1022        ras.ref = ref_name                  # Example: 'ras@1'
```

```
1023        ras.raster = raster
1024        ras.bandNumber = bandnumber
1025        entries.append(ras)
1026        return entries
1027
1028    def modify_qml_mch(qml_file_path, new_value):
1029        '''
1030        Changes MCH style depending on the minimun height (new_value)
1031        '''
1032        # Abrir el archivo .qml en modo lectura y escritura
1033        with fileinput.FileInput(qml_file_path, inplace=True) as file:
1034            # Recorrer todas las líneas del archivo
1035            for line in file:
1036                # Buscar las líneas que contienen el valor numérico a cambiar
1037                if '<item label="&lt;= 3,0000"' in line or '<item label="> 3,0000"' in line:
1038                    # Reemplazar el valor numérico por el nuevo valor
1039                    line = line.replace('3', str(new_value))
1040                # Imprimir la línea modificada en el archivo
1041                print(line, end='')
1042
1043    def extraction(raster, vlayer):
1044        # Crear una lista vacía para almacenar los datos
1045        data = []
1046        geometry = []
1047        for feature in vlayer.getFeatures():
1048            # Obtener la geometría del punto
1049            geom = feature.geometry()
1050            # Obtener las coordenadas del punto
1051            x = geom.asPoint().x()
1052            y = geom.asPoint().y()
1053            # Extraer el valor del raster en la ubicación del punto
1054            value, res = raster.dataProvider().sample(QgsPointXY(x, y), 1)
1055            # Agregar el valor a la lista de datos
1056            data.append([value])
1057            # Agregar la geometría a la lista de geometrías
1058            geometry.append(geom)
1059        # Crear un DataFrame con los datos
1060        df = pd.DataFrame(data, columns=[raster.name()])
1061        # Convertir el DataFrame en un GeoDataFrame
1062        gdf = gpd.GeoDataFrame(df, geometry=geometry)
1063        return gdf
1064
1065    def extract_all(vlayer):
1066
1067        first = True
```

```
    for raster in variables_df['pyvariable']:
        if first:
            first = False
            test_df = extraction(raster=raster, vlayer=vlayer)
        else:
            temp_df = extraction(raster=raster, vlayer=vlayer)
            temp_df = temp_df.drop(columns=['geometry'])
            test_df = test_df.join(temp_df, rsuffix=f'_{raster.name()}')
    test_df = test_df.rename(columns={'geometry': 'coords'})
    test_df = test_df.set_geometry("coords")
    return test_df.set_crs(mCrs.toWkt())

def extract_all_with_forest(vlayer):
    # Crear una lista vacía para almacenar los datos
    data = []
    geometry = []
    den_cod2_e = []

    for feature in vlayer.getFeatures():
        # Obtener la geometría del punto
        geom = feature.geometry()
        # Obtener las coordenadas del punto
        x = geom.asPoint().x()
        y = geom.asPoint().y()
        # Agregar la geometría a la lista de geometrías
        geometry.append(geom)
        # Agregar el valor del campo 'DEN_COD2_E' a la lista den_cod2_e
        den_cod2_e.append(feature[veg_field])

        row_data = []
        for raster in variables_df['pyvariable']:
            # Extraer el valor del raster en la ubicación del punto
            value, res = raster.dataProvider().sample(QgsPointXY(x, y), 1)
            # Agregar el valor a la lista de datos
            row_data.append(value)

        data.append(row_data)

    # Crear un DataFrame con los datos
    columns = [raster.name() for raster in variables_df['pyvariable']]
    df = pd.DataFrame(data, columns=columns)
    # Convertir el DataFrame en un GeoDataFrame
    gdf = gpd.GeoDataFrame(df, geometry=geometry)
    # Agregar la columna 'DEN_COD2_E'
```

```
1112        gdf[veg_field] = den_cod2_e
1113
1114        return gdf.set_crs(mCrs.toWkt())
1115
1116    def save_df(test_data, name):
1117        df = test_data.copy()
1118
1119        # Extraer las coordenadas x e y de la geometría y agregarlas como nuevas columnas
1120        df['x'] = df.geometry.x
1121        df['y'] = df.geometry.y
1122
1123
1124        # Obtener el EPSG del GeoDataFrame
1125        epsg = df.crs.to_epsg()
1126
1127        # Eliminar la columna de geometría
1128        df = df.drop('geometry', axis=1)
1129
1130        # Crear el nombre del archivo con el EPSG incluido
1131        filename = f'{name}{epsg}.csv'
1132        filepath = os.path.abspath(os.path.join(working_dir, filename))
1133
1134        # Guardar el DataFrame en un archivo CSV
1135        df.to_csv(filepath, index=False)
1136
1137
1138    def moranI(gdf, col):
1139
1140        # Crea una copia del GeoDataFrame sin las filas que contienen valores NaN en la columna 'plu08r200e
1141        clean = gdf.dropna(subset=['plu08r200esp_clipped'])
1142
1143        # Crea la matriz de pesos espaciales utilizando el GeoDataFrame limpio
1144        w = Queen.from_dataframe(clean)
1145
1146        # Calcula el índice de Moran utilizando el GeoDataFrame limpio
1147        mi = Moran(clean[col], w)
1148
1149        # Verifica los resultados del índice de Moran
1150        #print(mi.I)
1151        #print("{:.5f}".format(mi.p_norm))
1152
1153        return mi.I, mi.p_norm
1154
1155
```

```
from sklearn.impute import SimpleImputer

def predict(input_folder, gdf, gdf_target, type, forest_name, add_model_path):
    raster_features = sorted(glob.glob(os.path.abspath(input_folder+ '/*.tiff')))
    if add_model_path:
        raster_features.append(add_model_path)

    print('\nThere are', len(raster_features), 'raster features.')

    train_xs, train_y = load_training_vector(gdf, raster_features, response_field=gdf_target)
    target_xs, raster_info = load_targets(raster_features)
    train_xs.shape, train_y.shape # check shape, does it match the size above of the observations?

    # Create an instance of the SimpleImputer with the mean imputation strategy
    imputer = SimpleImputer(strategy='mean')

    # Compute the imputation values based on the training data
    imputer.fit(train_xs)

    # Fill in missing values in the training data
    train_xs_imputed = imputer.transform(train_xs)

    CLASS_MAP = {
        'rf': (RandomForestClassifier()),
        'et': (ExtraTreesClassifier()),
        'xgb': (XGBClassifier()),
        'lgbm': (LGBMClassifier())
        }

    best_cross = 0

    # model fitting and spatial range prediction
    for name, (model) in CLASS_MAP.items():
        # cross validation for accuracy scores (displayed as a percentage)
        k = 10 # k-fold
        kf = model_selection.KFold(n_splits=k)
        accuracy_scores = model_selection.cross_val_score(model, train_xs_imputed, train_y, cv=kf, scori
        print(name + " %d-fold Cross Validation Accuracy: %0.2f (+/- %0.2f)"
              % (k, accuracy_scores.mean() * 100, accuracy_scores.std() * 200))

        # Predicción espacial
        model.fit(train_xs_imputed, train_y)

        # Output directory for this iteration
        outdir = os.path.abspath(os.path.join(working_dir, 'models', type + '_' +forest_name+ '_'+ name
```

```
1201
1202
1203            #impute(target_xs, model, raster_info, outdir=outdir,
1204            #       class_prob=True, certainty=True)
1205
1206            if accuracy_scores.mean() > best_cross:
1207                best_cross = accuracy_scores.mean()
1208                path = outdir
1209                os.makedirs(path, exist_ok=True)
1210                best_model = model
1211
1212        impute(
1213            target_xs,
1214            best_model,
1215            raster_info,
1216            outdir=path,
1217            class_prob=True,
1218            certainty=True)
1219
1220        # get feature importances for the best model
1221        importances = best_model.feature_importances_
1222
1223        # create a dictionary to store the feature importances
1224        importances_dict = {}
1225
1226        for i in range(len(importances)):
1227            feature_name = os.path.basename(raster_features[i]).replace('.tiff', '')
1228            importances_dict[feature_name] = importances[i]
1229
1230        return path.replace('tif', 'tiff'), importances_dict
1231
1232
1233
1234    def match_raster(input_path, target_path, output):
1235
1236        rlayer = QgsRasterLayer(input_path,    "malo")
1237        bueno  = QgsRasterLayer(target_path, "bueno")
1238
1239        resolx= bueno.rasterUnitsPerPixelX()
1240        resoly= bueno.rasterUnitsPerPixelY()
1241
1242        file_writer = QgsRasterFileWriter(output)
1243        pipe = QgsRasterPipe()
1244        provider = bueno.dataProvider()
1245        pipe.set(rlayer.dataProvider().clone())
```

```
1246
1247        # calculate desired width and height in pixels
1248        desired_width = float(bueno.extent().width() / resolx)
1249        desired_height = float(bueno.extent().height() / resoly)
1250
1251        # write raster with desired resolution
1252        file_writer.writeRaster(pipe, desired_width, desired_height, bueno.extent(), bueno.crs())
1253
1254    def AppendRaster2CalcEntry(
1255        entries    : list,
1256        ref_name   : str,
1257        raster     : QgsRasterLayer,
1258        bandnumber : int):
1259            '''
1260            Just to use QgsRasterCalculator
1261            '''
1262
1263            ras            = QgsRasterCalculatorEntry()
1264            ras.ref        = ref_name                      # Example: 'ras@1'
1265            ras.raster     = raster                        # QgsRasterLayer
1266            ras.bandNumber = bandnumber                    # First: 1
1267
1268            entries.append(ras)
1269
1270            return entries
1271
1272    def get_NDVI_layer():
1273        entries = []
1274        temp_file = working_dir+'/S2/resampled/ndvi.tiff'
1275
1276        AppendRaster2CalcEntry(
1277            entries    = entries,
1278            ref_name   = 'nir@1',
1279            raster     = S2_layers[6],
1280            bandnumber = 1)
1281
1282        AppendRaster2CalcEntry(
1283            entries    = entries,
1284            ref_name   = 'red@1',
1285            raster     = S2_layers[2],
1286            bandnumber = 1)
1287
1288        formula = "('nir@1'-'red@1')/('nir@1'+'red@1')"
1289
1290        NDVI = QgsRasterCalculator(
```

```
1291            formulaString    = formula,
1292            outputFile       = temp_file,
1293            outputFormat     = 'GTiff',
1294            outputExtent     = S2_layers[2].extent(),
1295            nOutputColumns   = S2_layers[2].width(),
1296            nOutputRows      = S2_layers[2].height(),
1297            rasterEntries    = entries)
1298
1299
1300
1301        NDVI.processCalculation()
1302        layer =  QgsRasterLayer(temp_file, 'NDVI')
1303
1304
1305    #    parameters_clip = {
1306    #        'ALPHA_BAND'      : False,
1307    #        'CROP_TO_CUTLINE' : True,
1308    #        'DATA_TYPE'       : 0,
1309    #        'EXTRA'           : '',
1310    #        'KEEP_RESOLUTION' : False,
1311    #        'MASK' : get_layer_URI_by_name(StudyArea[0]),
1312    #        'MULTITHREADING'  : False,
1313    #        'NODATA'          : None,
1314    #        'OPTIONS'         : '',
1315    #        'SET_RESOLUTION'  : False,
1316    #        'SOURCE_CRS'      : mCrs,
1317    #        'TARGET_CRS'      : mCrs,
1318    #        'X_RESOLUTION'    : None,
1319    #        'Y_RESOLUTION'    : None,
1320    #        'INPUT'           : layer,
1321    #        'OUTPUT           : '}
1322    #
1323    #
1324    #    'gdal:cliprasterbymasklayer'
1325
1326
1327
1328
1329        return layer
1330
1331    def get_ratio_swir_layer():
1332        entries = []
1333        temp_file = working_dir+'/S2/resampled/swir_ratio.tiff'
1334
1335
```

```
1336        AppendRaster2CalcEntry(
1337            entries    = entries,
1338            ref_name   = '11@1',
1339            raster     = S2_layers[8],
1340            bandnumber = 1)
1341
1342        AppendRaster2CalcEntry(
1343            entries    = entries,
1344            ref_name   = '12@1',
1345            raster     = S2_layers[9],
1346            bandnumber = 1)
1347
1348        formula = "('12@1'/'11@1')"
1349
1350        NDVI = QgsRasterCalculator(
1351            formulaString    = formula,
1352            outputFile       = temp_file,
1353            outputFormat     = 'GTiff',
1354            outputExtent     = S2_layers[2].extent(),
1355            nOutputColumns   = S2_layers[2].width(),
1356            nOutputRows      = S2_layers[2].height(),
1357            rasterEntries    = entries)
1358
1359        NDVI.processCalculation()
1360        layer =  QgsRasterLayer(temp_file, 'swir_ratio')
1361        return layer
1362
1363
1364
```

A.4. Análisis

A.4.1. Incertidumbre

A.4.2. Exactitud y sensibilidad

```python
import numpy as np
from PIL import Image

def contar_pixeles_tif(path, valores):
    # Cargar la imagen TIFF como matriz numpy
    matriz = np.array(Image.open(path))

    # Calcular el histograma de los valores en la matriz
    histograma = np.histogram(matriz, bins=np.concatenate((valores, [valores[-1] + 1])))

    # Crear un diccionario con los resultados
    resultado = dict(zip(valores, histograma[0]))

    return resultado

# Ruta del archivo TIFF
ruta_tif = r"C:\Users\71742480Y\Desktop\temporal\model_diff\dif_only_s2\8_dif.tif"

# Valores a contar (ordenados de manera ascendente)
valores_a_contar = np.sort(np.array([2, 1, 0, -1]))

# Contar los píxeles en el archivo TIFF
resultado = contar_pixeles_tif(ruta_tif, valores_a_contar)

# Obtener los valores necesarios para los cálculos
pixeles_clase_2 = resultado.get(2, 0)
pixeles_clase_0 = resultado.get(0, 0)
pixeles_clase_1 = resultado.get(1, 0)
pixeles_clase_menos_1 = resultado.get(-1, 0)

# Calcular la exactitud
suma_clases = pixeles_clase_2 + pixeles_clase_0 + pixeles_clase_1 + pixeles_clase_menos_1
exactitud = (pixeles_clase_2 + pixeles_clase_0) / suma_clases if suma_clases != 0 else 0.0

# Calcular la sensibilidad
sensibilidad = pixeles_clase_0 / (pixeles_clase_menos_1 + pixeles_clase_0) if (pixeles_clase_menos_1 + p

# Imprimir los resultados
print(f"Exactitud: {exactitud}")
print(f"Sensibilidad: {sensibilidad}")


```

A.4.3. Incertidumbre

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.lines import Line2D


dataframe = pd.read_csv(ruta_csv, sep=";")

# Convertir los valores de la columna 'color' a lista
colors = dataframe['color'].apply(lambda x: '#' + x).tolist()

# Figura
fig, ax = plt.subplots()

# Trazar los puntos A
ax.scatter(dataframe["Area"], dataframe["A_incer"], c=colors, marker='s', label="A_incer")

# Ajustar y trazar la línea de tendencia
coefs_a = np.polyfit(dataframe["Area"], dataframe["A_incer"], 1)
trend_a = np.polyval(coefs_a, dataframe["Area"])
ax.plot(dataframe["Area"], trend_a, c="blue", linestyle="-", label="Tendencia A_incer")

# Trazar los puntos de B
ax.scatter(dataframe["Area"], dataframe["B_incer"], c=colors, label="B_incer")

# Ajustar y trazar la línea de tendencia polinómica de grado 2 para B
coefs_b = np.polyfit(dataframe["Area"], dataframe["B_incer"], 1)
trend_b = np.polyval(coefs_b, dataframe["Area"])
ax.plot(dataframe["Area"], trend_b, c="red", linestyle="-", label="Tendencia B_incer")

# Etiquetas de los ejes
ax.set_xlabel("Superficie del habitat modelado (ha)")
ax.set_ylabel("Incertidumbre")

# Crear una lista de manejadores de leyenda personalizados y una lista de etiquetas para cada color y c
legend_elements = [Line2D([0], [0], marker='o', color='w', markerfacecolor=color, label=code, markersize
                   for color, code in zip(dataframe['color'].apply(lambda x: '#' + x).unique(), datafram
legend_elements.append(Line2D([0], [0], color='blue', linestyle='-', label='Jerarquico'))
legend_elements.append(Line2D([0], [0], color='red', linestyle='-', label='No jerarquico'))


ax.legend(handles=legend_elements, loc='upper left')
```

```
43
44
45  plt.show()
46
47
```

B. Salidas generadas por el complemento de QGIS

B.1. Archivos principales

- Clasification.tif – Mapa clasificado ordenado de 1 a N habitats ordenados alfabéticamente según su nombre.
- Incertidumbre.tif – Mapa de incertidumbre

B.2. Directorios

- Figures – almacena todos los gráficos generados
- Forest_mask: almacena la máscara que combina MCH y NDVI a la resolución del set de datos espectrales.
- GoodSampling: contiene en formato shape el muestreo seleccionado
- InputModelo: contiene las variables predictoras para cada modelo jerárquico (AOO-P, AOO-L y AO-RP)
- Max_Temperatures, Mean_Temperatures, Min_Temperatures y Pluviometry: almacenan las capas de clima recortadas al área de estudio.
- Models: contiene subdirectorios que almacenan los modelos individuales generados para cada hábitat.
- S2: contiene los predictores de satélite remuestreados.
- Sampling: almacena los 1260 archivos correspondientes a los 210 muestreos generados.
- Topo: contiene los predictores topográficos generados a su resolución original y los predictores topográficos remuestreados a la resolución de satélite.