Supplementary Material—Computed procedure for the Municipality

Theodoropoulos I. Andreas
Independent Researcher, 13451 Athens, Greece; theod.adreas@gmail.com

Supplementary File S1

```
from PIL import Image
import processing
import math
from pathlib import Path

# Classes Identification

def load_boundaries(filepath):
    with open(filepath, mode='r') as f:
csv_file = csv.reader(f, delimiter='=')
        bounds = {row[0]: row[1] for row in csv_file}
    return bounds

def create_id_field(name, vlayer=None):
    if vlayer is None:
        _vlayer = iface.activeLayer()
    else:
        _vlayer = iface.addVectorLayer(vlayer, "", "ogr")

    prov = layer.dataProvider()
field_names = [field.name() for field in prov.fields()]

    if not (name in field_names):
        _field = QgsField(name, QVariant.Int)
        _vlayer.startEditing()
        _vlayer.dataProvider().addAttributes([_field])
        _vlayer.updateFields()
        _id = _vlayer.dataProvider().fieldNameIndex(name)
        _vlayer.commitChanges()

        count = 1
        _vlayer.startEditing()
        for f in _vlayer.getFeatures():
attr_id = f.attributes()[_id]
rownum = count
            count += 1
            f[_id] = rownum
            _vlayer.updateFeature(f)

        _vlayer.commitChanges()

def clip_raster_by_vector(input_raster, input_vector, output_raster, overwrite=False):
```

```
    if overwrite:
        if os.path.isfile(output_raster):
os.remove(output_raster)

    if not os.path.isfile(input_raster):
        print ("File doesn't exists", input_raster)
        return None
    else:
        params = {'INPUT': input_raster,
                  'MASK': input_vector,
                  'NODATA': 255.0,
                  'ALPHA_BAND': False,
                  'CROP_TO_CUTLINE': False,
                  'KEEP_RESOLUTION': True,
                  'OPTIONS': 'COMPRESS=LZW',
                  'DATA_TYPE': 0,
                  'OUTPUT': output_raster,
                  }

        feedback = qgis.core.QgsProcessingFeedback()
alg_name = 'gdal:cliprasterbymasklayer'
        print(processing.algorithmHelp(alg_name))
        result = processing.run(alg_name, params, feedback=feedback)
        return result
```

 # DTM and DSM data Identification

```
params_dtm = {'COLUMN_PREFIX': 'dtm_',
              'INPUT':
"C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Building_Polygons.shp",
              'INPUT_RASTER':
'C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Municipality_DTM.tif',
              'OUTPUT':
'C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\zonal_buildings_test_1.shp',
              'RASTER_BAND': 1,
              'STATISTICS': [2]}

params_dsm = {'COLUMN_PREFIX': 'dsm_',
              'INPUT':
'C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\zonal_buildings_test_1.shp',
              'INPUT_RASTER':
'C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Municipality_DSM.tif',
              'OUTPUT':
'C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Municipality_Building_Stats.shp',
              'RASTER_BAND': 1,
              'STATISTICS': [6]}

processing.run('qgis:zonalstatisticsfb', params_dtm)
processing.run('qgis:zonalstatisticsfb', params_dsm)
```

 # Layers Identification

```
layer = QgsVectorLayer(

"C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Municipality_Building_Stats.shp")

layer_provider = layer.dataProvider()
layer_provider.addAttributes([QgsField('Height', QVariant.Int)])
layer_provider.addAttributes([QgsField("Floors", QVariant.Int)])
layer_provider.addAttributes([QgsField("Grey_L_P", QVariant.Double)])
layer_provider.addAttributes([QgsField("Grey_D_P", QVariant.Double)])
layer_provider.addAttributes([QgsField("Tiles_P", QVariant.Double)])
layer_provider.addAttributes([QgsField("Green_P", QVariant.Double)])
layer_provider.addAttributes([QgsField("BLD_TYPE", QVariant.String)])
layer_provider.addAttributes([QgsField("Area", QVariant.Double)])
layer_provider.addAttributes([QgsField("Grey_L_m2", QVariant.Double)])
layer_provider.addAttributes([QgsField("Grey_D_m2", QVariant.Double)])
layer_provider.addAttributes([QgsField("Tiles_m2", QVariant.Double)])
layer_provider.addAttributes([QgsField("Green_m2", QVariant.Double)])
layer_provider.addAttributes([QgsField("Orient", QVariant.String)])
layer_provider.addAttributes([QgsField("Old_kWh/yr", QVariant.Double)])
layer_provider.addAttributes([QgsField("Pd_kWh/m2", QVariant.Double)])
layer_provider.addAttributes([QgsField("E_Rate_old", QVariant.String)])
layer_provider.addAttributes([QgsField("E_Rate_New", QVariant.String)])
layer_provider.addAttributes([QgsField("Old_CO2_em", QVariant.Double)])
layer_provider.addAttributes([QgsField("New_CO2_em", QVariant.Double)])
layer_provider.addAttributes([QgsField("Savings", QVariant.Double)])
layer.updateFields()

features = layer.getFeatures()

prov = layer.dataProvider()
field_names = [field.name() for field in prov.fields()]

for r in range(len(field_names)):
   if field_names[r] == "Grey_L_P":
      q = r
eliffield_names[r] == "Grey_D_P":
      w = r
eliffield_names[r] == "Tiles_P":
      p = r
eliffield_names[r] == "Green_P":
xj = r
eliffield_names[r] == "BLD_TYPE":
      o = r
eliffield_names[r] == "Orient":
      xi = r
eliffield_names[r] == "Area":
xa = r
eliffield_names[r] == "Grey_L_m2":
```

```
xs = r
eliffield_names[r] == "Grey_D_m2":
xd = r
eliffield_names[r] == "Tiles_m2":
xg = r
eliffield_names[r] == "Green_m2":
xh = r
eliffield_names[r] == "Old_kWh/yr":
xk = r
eliffield_names[r] == "E_Rate_old":
xz = r
eliffield_names[r] == "Pd_kWh/m2":
      xl = r
eliffield_names[r] == "Savings":
      xc = r
eliffield_names[r] == "E_Rate_New":
      xv = r
eliffield_names[r] == "Old_CO2_em":
xb = r
eliffield_names[r] == "New_CO2_em":
xn = r

create_id_field('fid',

"C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Municipality_Building_Stats.shp")
```

 # Typology

```
Buffer_parameters = {'INPUT'
:'C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Municipality_Building_Stats.shp',
          'GEOMETRY' : 'geometry', 'DISTANCE' : 0.5, 'FIELD' : None, 'DISSOLVE' : False
, 'OUTPUT' :
'C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\zonal_bld_stats_BUFFERED.shp'
        }
import processing
processing.run("gdal:buffervectors", Buffer_parameters)
processing.run('qgis:polygonstolines', {'INPUT' :
'C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\zonal_bld_stats_BUFFERED.shp',
'OUTPUT' :
'C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\zonal_bld_stats_BUFFERED_lines.
shp'})

Intersection_params = {'INPUT'
:'C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\zonal_bld_stats_BUFFERED_lines
.shp',
          'OVERLAY' :
'C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Municipality_Building_Stats.shp',
'OUTPUT' :
'C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\zonal_bld_stats_INTERSECTED.sh
p'
```

```
        }
processing.run("qgis:intersection", Intersection_params)
layer = QgsVectorLayer(

"C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\zonal_bld_stats_INTERSECTED.s
hp")
typology = []
prov = layer.dataProvider()
field_names = [field.name() for field in prov.fields()]
features = layer.getFeatures()
layer.startEditing()
for f in features:
   if f['fid'] not in typology:
typology.append(f['fid'])

# Import data to SHP

layer =
QgsVectorLayer("C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Municipality_Buil
ding_Stats.shp")
prov = layer.dataProvider()
field_names = [field.name() for field in prov.fields()]
features = layer.getFeatures()
layer.startEditing()
layer_provider = layer.dataProvider()
for f in features:
    _id = f.id()
    if f['fid'] in typology:
       continuous = 'Continuous'
attr_value = {o: str(continuous)}
layer_provider.changeAttributeValues({_id: attr_value})
else :
       stand = 'Stand alone'
attr_value = {o: str(stand)}
layer_provider.changeAttributeValues({_id: attr_value})

layer.commitChanges()

# Create Vertices

_buildings = {
   'INPUT':
"C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Municipality_Building_Stats.shp",
   'OUTPUT':
"C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\build_outlines_VERTICES.shp"}
processing.run('qgis:extractvertices', _buildings)

# Import Layer

vlayer = iface.addVectorLayer(
```

```
    "C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\build_outlines_VERTICES.shp",
    "", "ogr")

# Add Geometry Attributes

_Vertice_layer = {
    'INPUT':
"C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\build_outlines_VERTICES.shp",
    'CALC_METHOD': 0,
    'OUTPUT':
"C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Buildings_Geometry.shp"}
processing.run("qgis:exportaddgeometrycolumns", _Vertice_layer)

vlayer = iface.addVectorLayer(
    "C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Buildings_Geometry.shp", "",
    "ogr")

input_raster =
"C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\New_city_block_modified.tif"
output_raster =
"C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Image_modified_clipped.png"
input_vector =
"C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Building_Polygons.shp"
result = clip_raster_by_vector(input_raster, input_vector, output_raster, overwrite=True)

# Image Import

png =
"C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Image_modified_clipped.png"

im = Image.open(png)
pix = im.load()

#  Distance between pixels

bound_file = "C:\\Users\\Adreas\\Desktop\\boundaries.txt"
bounds = load_boundaries(bound_file)

x_min_TIFF = float(bounds['xmin'])
y_min_TIFF = float(bounds['ymin'])
x_max_TIFF = float(bounds['xmax'])
y_max_TIFF = float(bounds['ymax'])

dx = (x_max_TIFF - x_min_TIFF) / im.size[0]
dy = (y_max_TIFF - y_min_TIFF) / im.size[1]

# Pixels' coordinates given

x = []
for i in range(im.size[0]):
```

```
x.append(x_min_TIFF + (dx * i))
y = []
for i in range(im.size[1]):
y.append(y_max_TIFF - (dy * i))
```

 # Vertices' Coordinates Lists for each building

```
layer = qgis.utils.iface.activeLayer()
prov = layer.dataProvider()
field_names = [field.name() for field in prov.fields()]
y_coordinates = []
x_coordinates = []
bld_id = []

for i in range(len(field_names)):
    if field_names[i] == 'fid':
        for feature in layer.getFeatures():
bld_id.append(feature[field_names[i]])
eliffield_names[i] == 'xcoord':
        for feature in layer.getFeatures():
x_coordinates.append(feature[field_names[i]])
eliffield_names[i] == 'ycoord':
        for feature in layer.getFeatures():
y_coordinates.append(feature[field_names[i]])

x_min = min(x_coordinates)
y_min = min(y_coordinates)
x_max = max(x_coordinates)
y_max = max(y_coordinates)
```

 # Pixel and color Identification for buildings' polygons

```
coordinates_id = []
for i in range(len(bld_id)):
    if bld_id[i] not in coordinates_id:
coordinates_id.append(bld_id[i])

for i in range(len(coordinates_id)):
x_min_bld = x_max
y_min_bld = y_max
x_max_bld = x_min
y_max_bld = y_min
    pixels = []
    grey1 = 0
    grey2 = 0
    grey3 = 0
    tiles = 0
    green = 0
    black = 0
```

```
    for j in range(len(bld_id)):
        if bld_id[j] == coordinates_id[i]:
            if x_coordinates[j] <= x_min_bld:
x_min_bld = x_coordinates[j]
            if x_coordinates[j] >= x_max_bld:
x_max_bld = x_coordinates[j]
            if y_coordinates[j] <= y_min_bld:
y_min_bld = y_coordinates[j]
            if y_coordinates[j] >= y_max_bld:
y_max_bld = y_coordinates[j]
    for a in range(im.size[0]):
        if x[a] >= x_min_bld and x[a] <= x_max_bld:
            for b in range(im.size[1]):
                if y[b] >= y_min_bld and y[b] <= y_max_bld:
pixels.append([a, b])
                    if pix[a, b][0] <= 255 and pix[a, b][0] >= 50 and pix[a, b][
                        1] >= \
pix[a, b][0] - 100 and pix[a, b][1] <= pix[a, b][
                        0] - 5 and \
pix[a, b][1] <= 210 and pix[a, b][2] <= pix[a, b][
                        1] - 10 and \
pix[a, b][2] <= pix[a, b][0] - 5 and pix[a, b][
                        2] <= 150 and \
pix[a, b][2] >= pix[a, b][0] - 120 and pix[a, b][
                        2] >= 5:
                        tiles = tiles + 1
elifpix[a, b][0] >= 75 and pix[a, b][0] <= 255 and \
pix[a, b][1] >= 75 and pix[a, b][1] <= 235 and \
pix[a, b][2] >= 75 and pix[a, b][2] <= 235:
                        grey1 = grey1 + 1
elifpix[a, b][0] >= 60 and pix[a, b][0] <= 150 and \
pix[a, b][1] >= 70 and pix[a, b][1] <= 150 and \
pix[a, b][2] >= 70 and pix[a, b][2] <= 150:
                        grey2 = grey2 + 1
elifpix[a, b][0] <= 110 and pix[a, b][1] >= 53 and pix[a, b][2] <= 110:
                        green = green + 1
elifpix[a, b][0] < 70 and pix[a, b][1] <= 70 and pix[a, b][2] <= 70:
                        black = black + 1.
```

 # Normalization ofcolor Percentages

```
    Percentage_Grey1 = 100 * grey1 / (grey1 + grey2 + grey3 + tiles + green + black)
    Percentage_Grey2 = 100 * grey2 / (grey1 + grey2 + grey3 + tiles + green + black)
Percentage_Tiles = 100 * tiles / (grey1 + grey2 + grey3 + tiles + green + black)
Percentage_Green = 100 * green / (grey1 + grey2 + grey3 + tiles + green + black)
```

 # Orientation Identification

```
    orientation = False
```

```
   if Percentage_Tiles>= 60:
x_coord_tile = []
y_coord_tile = []
for k in range(len(bld_id)):
        if bld_id[k] == coordinates_id[i]:
x_coord_tile.append(x_coordinates[k])
y_coord_tile.append(y_coordinates[k])
for k in range(len(x_coord_tile)):
        if x_coord_tile[k] == min(x_coord_tile):
            x1 = x_coord_tile[k]
            y1 = y_coord_tile[k]
for k in range(len(y_coord_tile)):
        if y_coord_tile[k] == max(y_coord_tile):
            x2 = x_coord_tile[k]
            y2 = y_coord_tile[k]
for k in range(len(y_coord_tile)):
        if x_coord_tile[k] == max(x_coord_tile):
            x3 = x_coord_tile[k]
            y3 = y_coord_tile[k]

      a0 = math.atan((x2 - x1) / (y2 - y1))

      if x2 - x1 > 0 and y2 - y1 > 0:
         a1 = a0
elif x2 - x1 > 0 and y2 - y1 < 0:
        a1 = math.pi + a0
elif x2 - x1 < 0 and y2 - y1 < 0:
        a1 = math.pi + a0
elif x2 - x1 < 0 and y2 - y1 > 0:
        a1 = 2 * math.pi - a0
elif x2 == x1 and y2 - y1 > 0:
        a1 = 0
elif x2 == x1 and y2 - y1 < 0:
        a1 = math.pi
elif y2 == y1 and x2 - x1 > 0:
        a1 = math.pi / 2
elif y2 == y1 and x2 - x1 < 0:
        a1 = 3 * math.pi / 2

      a1 = a1 * 200 / math.pi

      dist_1 = math.sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1))

      a02 = math.atan((x3 - x2) / (y3 - y2))

      if x3 - x2 > 0 and y3 - y2 > 0:
         a2 = a02
elif x3 - x2 > 0 and y3 - y2 < 0:
        a2 = math.pi + a02
elif x3 - x2 < 0 and y3 - y2 < 0:
```

```
            a2 = math.pi + a02
elif x3 - x2 < 0 and y3 - y2 > 0:
            a2 = 2 * math.pi - a02
elif x3 == x2 and y3 - y2 > 0:
            a2 = 0
elif x3 == x2 and y3 - y2 < 0:
            a2 = math.pi
elif y3 == y2 and x3 - x2 > 0:
            a2 = math.pi / 2
elif y3 == y2 and x3 - x2 < 0:
            a2 = 3 * math.pi / 2

        a2 = a2 * 200 / math.pi

        dist_2 = math.sqrt((x3 - x2) * (x3 - x2) + (y3 - y2) * (y3 - y2))

        if dist_1 > dist_2 :
            if a1 >= 0 and a1 <= 75 :
                orientation = True
elif a1 >= 125 and a1 <= 275:
                orientation = True
elif a1 >= 325 and a1 <= 425:
                orientation = True

elif dist_1 < dist_2 :

            if a2 >= 0 and a2 <= 75 :
                orientation = True
elif a2 >= 125 and a1 <= 275:
                orientation = True
elif a2 >= 325 and a2 <= 425:
                orientation = True
```

 # Import color results to SHP file

```
    layer = QgsVectorLayer(

"C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Municipality_Building_Stats.shp")
layer_provider = layer.dataProvider()
field_names = [field.name() for field in prov.fields()]
    features = layer.getFeatures()
layer.startEditing()
    for f in features:
        if f['fid'] == coordinates_id[i]:
            _id = f.id()
attr_value = {q: (Percentage_Grey1),
                      w: (Percentage_Grey2),
                      p: (Percentage_Tiles), xj: (Percentage_Green)}
layer_provider.changeAttributeValues({_id: attr_value})
```

 # Import orientation results to SHP file

```
            if Percentage_Tiles>= 60 :
                if orientation == True :
                    _orientation = 'Both sides'
                    _id = f.id()
attr_value = {xi: str(_orientation)}
layer_provider.changeAttributeValues({_id: attr_value})
elif orientation == False :
                    _orientation = 'One side'
                    _id = f.id()
attr_value = {xi: str(_orientation)}
layer_provider.changeAttributeValues({_id: attr_value})
elifPercentage_Tiles<60 :
                _orientation = 'Flat roof'
                _id = f.id()
attr_value = {xi: str(_orientation)}
layer_provider.changeAttributeValues({_id: attr_value})
layer.commitChanges()
```

 # Building's Height and Floors Identification

```
floor_thresh = 3

layer = QgsVectorLayer(

"C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Municipality_Building_Stats.shp")
features = layer.getFeatures()
prov = layer.dataProvider()
field_names = [field.name() for field in prov.fields()]
for i in range(len(field_names)):
   if field_names[i] == "dtm_mean":
x_floor = i
eliffield_names[i] == "dsm_max":
y_floor = i
eliffield_names[i] == "Height":
       z = i
eliffield_names[i] == "Floors":
       h = i

layer_provider = layer.dataProvider()
layer.startEditing()
for f in features:
   _id = f.id()
   _dtm = f.attributes()[x_floor]
   _dsm = f.attributes()[y_floor]
   _height = _dsm - _dtm
   _floors = _height / floor_thresh
attr_value = {z: int(_height), h: int(_floors)}
layer_provider.changeAttributeValues({_id: attr_value})
```

```
layer.commitChanges()

# Area Identification of Buildings

_Bld_area = {
    'INPUT':
"C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Municipality_Building_Stats.shp",
    'CALC_METHOD': 0,
    'OUTPUT':
"C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Buildings_AREA.shp"}
processing.run("qgis:exportaddgeometrycolumns", _Bld_area)

vlayer = QgsVectorLayer(
    "C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Buildings_AREA.shp")

layer = vlayer
prov = layer.dataProvider()
field_names = [field.name() for field in prov.fields()]
bld_area_id = []
bld_area = []
GREY_LIGHT = []
GREY_DARK = []
Tiles_perc = []
Green_perc = []
category_count = []
floors_count = []
orient_tiles = []

for i in range(len(field_names)):
    if field_names[i] == 'fid':
        for feature in layer.getFeatures():
bld_area_id.append(feature[field_names[i]])
eliffield_names[i] == 'area_2':
        for feature in layer.getFeatures():
bld_area.append(feature[field_names[i]])
eliffield_names[i] == 'Grey_L_P':
        for feature in layer.getFeatures():
GREY_LIGHT.append(feature[field_names[i]])
eliffield_names[i] == 'Grey_D_P':
         for feature in layer.getFeatures():
GREY_DARK.append(feature[field_names[i]])
eliffield_names[i] == 'Tiles_P':
        for feature in layer.getFeatures():
Tiles_perc.append(feature[field_names[i]])
eliffield_names[i] == 'Green_P':
        for feature in layer.getFeatures():
Green_perc.append(feature[field_names[i]])
eliffield_names[i] == 'Floors':
        for feature in layer.getFeatures():
floors_count.append(feature[field_names[i]])
```

```
eliffield_names[i] == 'Category':
        for feature in layer.getFeatures():
category_count.append(feature[field_names[i]])
eliffield_names[i] == 'Orient':
        for feature in layer.getFeatures():
orient_tiles.append(feature[field_names[i]])

# Import data to SHP - calculation of energy data
electricity_production_flat = 1530.42
electricity_production_gable_one_side = 1460.27
electricity_production_gable_two_sides = 1348.24
total_electricity_consumption = 53.6
primary_electricity_consumption = total_electricity_consumption * 2.9
space_float_1kWh = 15
space_gable_1kWh = 10
ec_electricity_percentage = 0.173
G_Limit = 2.74
F_Limit = 2.73
E_Limit = 2.27
D_Limit = 1.82
C_Limit = 1.41
B_Limit = 1
B_PLUS_Limit = 0.75
A_Limit = 0.50
A_PLUS_Limit = 0.33
Primary_Thermal_Cons = 99.22
layer =
QgsVectorLayer("C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Municipality_Buil
ding_Stats.shp")
prov = layer.dataProvider()
field_names = [field.name() for field in prov.fields()]
features = layer.getFeatures()
layer.startEditing()
layer_provider = layer.dataProvider()

for i in range(len(field_names)):
   if field_names[i] == 'fid':
      for f in features:
         _id = f.id()
         Rating = 0
for k in range(len(bld_area_id)):
            if bld_area_id[k] == f[field_names[i]]:
               area = bld_area[k]
area_grey_light = GREY_LIGHT[k] * area / 100
area_grey_dark = GREY_DARK[k] * area / 100
area_tiles = Tiles_perc[k] * area / 100
               tiles = Tiles_perc[k]
area_green = Green_perc[k] * area / 100
               category = category_count[k]
               floors = floors_count[k]
```

```
orient_find = orient_tiles[k]
                if category == 1:
Rating_old = "G"
                    Rating = Rating + 1
elif category == 2:
Rating_old = "D"
                    Rating = Rating + 2
elif category == 3:
Rating_old = "C"
                    Rating = Rating + 3
elif category == 4:
Rating_old = "B"
                    Rating = Rating + 4
elif category == 0:
Rating_old = "NONE"
attr_value = {xs: (area_grey_light),xd: (area_grey_dark), xg: (area_tiles), xh:
(area_green),xz: (Rating_old)}
layer_provider.changeAttributeValues({_id: attr_value})

                if tiles < 60:
                    consumption = total_electricity_consumption * area * floors
                    if area_grey_light>area_grey_dark and area_green<area_grey_light:
kWp = area_grey_light / space_float_1kWh
                        if kWp<= 10:
                            _production = ((kWp * electricity_production_flat) / area) / floors

savings_start = abs((100 * (total_electricity_consumption - _production) /
total_electricity_consumption))
                            if (total_electricity_consumption - _production) < 0:
                                savings = 100 + savings_start
elif (total_electricity_consumption - _production) > 0:
                                savings = savings_start
ec = primary_electricity_consumption * ec_electricity_percentage
pv_ec_production = ec_electricity_percentage * _production * 2.9
new_ec_consumption_start = ec - (pv_ec_production)
                            if new_ec_consumption_start>= 0:
new_ec_consumption = new_ec_consumption_start
elifnew_ec_consumption_start<= 0:
new_ec_consumption = 0
                            Old_e_CO2 = primary_electricity_consumption * area * floors * 0.989
                            New_e_CO2 = ((primary_electricity_consumption - (_production * 2.9))
* 0.989) * area * floors
                            if New_e_CO2 < 0:
                                New_e_CO2 = 0

                            if Rating == 1:
                                Reference = (Primary_Thermal_Cons + ec) / G_Limit
                                new = (Primary_Thermal_Cons + (new_ec_consumption)) / Reference
                                if new <G_Limit and new >= E_Limit:
rating_new = "F"
```

```
attr_value = {xa: (area), xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
elif new <= F_Limit and new >= D_Limit:
rating_new = "E"
attr_value = {xa: (area), xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
elif new <E_Limit and new >C_Limit:
rating_new = "D"
attr_value = {xa: (area), xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
elif new <D_Limit and new >B_Limit:
rating_new = "C"
attr_value = {xa: (area), xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
                        if Rating == 2:
                            Reference = (Primary_Thermal_Cons + ec) / D_Limit
                            new = (Primary_Thermal_Cons + (new_ec_consumption)) / Reference
                            if new <D_Limit and new >= B_Limit:
rating_new = "C"
attr_value = {xa: (area), xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
elif new <= C_Limit and new >= B_PLUS_Limit:
rating_new = "B"
attr_value = {xa: (area), xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
elif new <B_Limit and new >A_Limit:
rating_new = "B_PLUS"
attr_value = {xa: (area), xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
                        if Rating == 3:
                            Reference = (Primary_Thermal_Cons + ec) / C_Limit
                            new = (Primary_Thermal_Cons + (new_ec_consumption)) / Reference
                            if new <C_Limit and new >= B_PLUS_Limit:
rating_new = "B"
attr_value = {xa: (area), xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
elif new <= B_Limit and new >= A_Limit:
rating_new = "B_PLUS"
attr_value = {xa: (area), xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
                        if Rating == 4:
```

```
                        Reference = (Primary_Thermal_Cons + ec) / B_Limit
                        new = (Primary_Thermal_Cons + (new_ec_consumption)) / Reference
                        if new <B_Limit and new >= A_Limit:
rating_new = "B_PLUS"
attr_value = {xa: (area), xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})

elifkWp> 10:
                    production = ((10 * electricity_production_flat) / area) / floors

savings_start = abs((100 * (total_electricity_consumption - _production) /
total_electricity_consumption))
                    if (total_electricity_consumption - _production) < 0:
                        savings = 100 + savings_start
elif (total_electricity_consumption - _production) > 0:
                        savings = savings_start
ec = primary_electricity_consumption * ec_electricity_percentage
pv_ec_production = ec_electricity_percentage * _production * 2.9
new_ec_consumption_start = ec - (pv_ec_production)
                    if new_ec_consumption_start>= 0:
new_ec_consumption = new_ec_consumption_start
elifnew_ec_consumption_start<= 0:
new_ec_consumption = 0
                    Old_e_CO2 = primary_electricity_consumption * area * floors * 0.989
                    New_e_CO2 = ((primary_electricity_consumption - (_production * 2.9))
* 0.989) * area * floors
                    if New_e_CO2 < 0:
                        New_e_CO2 = 0

                    if Rating == 1:
                        Reference = (Primary_Thermal_Cons + ec) / G_Limit
                        new = (Primary_Thermal_Cons + (new_ec_consumption)) / Reference
                        if new <G_Limit and new >= E_Limit:
rating_new = "F"
attr_value = {xa: (area), xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
elif new <= F_Limit and new >= D_Limit:
rating_new = "E"
attr_value = {xa: (area), xl: (_production), xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
elif new <E_Limit and new >C_Limit:
rating_new = "D"
attr_value = {xa: (area), xl: (_production), xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
elif new <D_Limit and new >B_Limit:
rating_new = "C"
```

```
attr_value = {xa: (area), xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
                        if Rating == 2:
                            Reference = (Primary_Thermal_Cons + ec) / D_Limit
                            new = (Primary_Thermal_Cons + (new_ec_consumption)) / Reference
                            if new <D_Limit and new >= B_Limit:
rating_new = "C"
attr_value = {xa: (area), xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
elif new <= C_Limit and new >= B_PLUS_Limit:
rating_new = "B"
attr_value = {xa: (area), xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues(
                                    {_id: attr_value})
elif new <B_Limit and new >A_Limit:
rating_new = "B_PLUS"
attr_value = {xa: (area), xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),n: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
                        if Rating == 3:
                            Reference = (Primary_Thermal_Cons + ec) / C_Limit
                            new = (Primary_Thermal_Cons + (new_ec_consumption)) / Reference
                            if new <C_Limit and new >= B_PLUS_Limit:
rating_new = "B"
attr_value = {xa: (area), xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues( {_id: attr_value})
elif new <= B_Limit and new >= A_Limit:
rating_new = "B_PLUS"
attr_value = {xa: (area), xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
                        if Rating == 4:
                            Reference = (Primary_Thermal_Cons + ec) / B_Limit
                            new = (Primary_Thermal_Cons + (new_ec_consumption)) / Reference
                            if new <B_Limit and new >= A_Limit:
rating_new = "B_PLUS"
attr_value = {xa: (area), xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
elif tiles >= 60:
area_tiles_corrected = (area / 2) / 0.9397
area_total = area_tiles_corrected * 2
                    if orient_find == "Both sides":
area_final = area_tiles_corrected * 2
electricity_production_gable = electricity_production_gable_two_sides
eliforient_find == "One sides":
```

```
area_final = area_tiles_corrected
electricity_production_gable = electricity_production_gable_one_side


                consumption = total_electricity_consumption * area * floors
                if area_grey_light>area_grey_dark and area_green<area_grey_light:
kWp = (area_final) * space_gable_1kWh
                    if kWp<= 10:
                        _production = ((kWp * electricity_production_gable) / area) / floors

savings_start = abs((100 * (total_electricity_consumption - _production) /
total_electricity_consumption))
                        if (total_electricity_consumption - _production) < 0:
                            savings = 100 + savings_start
elif (total_electricity_consumption - _production) > 0:
                            savings = savings_start
ec = primary_electricity_consumption * ec_electricity_percentage
pv_ec_production = ec_electricity_percentage * _production * 2.9
new_ec_consumption_start = ec - (pv_ec_production)
                        if new_ec_consumption_start>= 0:
new_ec_consumption = new_ec_consumption_start
elifnew_ec_consumption_start<= 0:
new_ec_consumption = 0
                        Old_e_CO2 = primary_electricity_consumption * area * floors * 0.989
                        New_e_CO2 = ((primary_electricity_consumption - (_production * 2.9))
* 0.989) * area * floors
                        if New_e_CO2 < 0:
                            New_e_CO2 = 0

                        if Rating == 1:
                            Reference = (Primary_Thermal_Cons + ec) / G_Limit
                            new = (Primary_Thermal_Cons + (new_ec_consumption)) / Reference
                            if new <G_Limit and new >= E_Limit:
rating_new = "F"
attr_value = {xa: (area_total),xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
elif new <= F_Limit and new >= D_Limit:
rating_new = "E"
attr_value = {xa: (area_total) ,xl: (_production), xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
elif new <E_Limit and new >C_Limit:
rating_new = "D"
attr_value = {xa: (area_total) ,xl: (_production), xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
elif new <D_Limit and new >B_Limit:
rating_new = "C"
```

```
attr_value = {xa: (area_total) ,xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
                        if Rating == 2:
                            Reference = (Primary_Thermal_Cons + ec) / D_Limit
                            new = (Primary_Thermal_Cons + (new_ec_consumption)) / Reference
                            if new <D_Limit and new >= B_Limit:
rating_new = "C"
attr_value = {xa: (area_total) ,xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
elif new <= C_Limit and new >= B_PLUS_Limit:
rating_new = "B"
attr_value = {xa: (area_total) ,xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
elif new <B_Limit and new >A_Limit:
rating_new = "B_PLUS"
attr_value = {xa: (area_total) ,xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),n: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
                        if Rating == 3:
                            Reference = (Primary_Thermal_Cons + ec) / C_Limit
                            new = (Primary_Thermal_Cons + (new_ec_consumption)) / Reference
                            if new <C_Limit and new >= B_PLUS_Limit:
rating_new = "B"
attr_value = {xa: (area_total) ,xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues( {_id: attr_value})
elif new <= B_Limit and new >= A_Limit:
rating_new = "B_PLUS"
attr_value = {xa: (area_total) ,xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
                        if Rating == 4:
                            Reference = (Primary_Thermal_Cons + ec) / B_Limit
                            new = (Primary_Thermal_Cons + (new_ec_consumption)) / Reference
                            if new <B_Limit and new >= A_Limit:
rating_new = "B_PLUS"
attr_value = {xa: (area_total) ,xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
elifkWp> 10:
                        _production = ((10 * electricity_production_flat) / area) / floors

savings_start = abs((100 * (total_electricity_consumption - _production) /
total_electricity_consumption))
                        if (total_electricity_consumption - _production) < 0:
                            savings = 100 + savings_start
elif (total_electricity_consumption - _production) > 0:
```

```
                        savings = savings_start
ec = primary_electricity_consumption * ec_electricity_percentage
pv_ec_production = ec_electricity_percentage * _production * 2.9
new_ec_consumption_start = ec - (pv_ec_production)
                    if new_ec_consumption_start>= 0:
new_ec_consumption = new_ec_consumption_start
elifnew_ec_consumption_start<= 0:
new_ec_consumption = 0
                    Old_e_CO2 = primary_electricity_consumption * area * floors * 0.989
                    New_e_CO2 = ((primary_electricity_consumption - (_production * 2.9))
* 0.989) * area * floors
                    if New_e_CO2 < 0:
                        New_e_CO2 = 0

                    if Rating == 1:
                        Reference = (Primary_Thermal_Cons + ec) / G_Limit
                        new = (Primary_Thermal_Cons + (new_ec_consumption)) / Reference
                        if new <G_Limit and new >= E_Limit:
rating_new = "F"
attr_value = {xa: (area_total),xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
elif new <= F_Limit and new >= D_Limit:
rating_new = "E"
attr_value = {xa: (area_total),xl: (_production), xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
elif new <E_Limit and new >C_Limit:
rating_new = "D"
attr_value = {xa: (area_total),xl: (_production), xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
elif new <D_Limit and new >B_Limit:
rating_new = "C"
attr_value = {xa: (area_total),xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
                    if Rating == 2:
                        Reference = (Primary_Thermal_Cons + ec) / D_Limit
                        new = (Primary_Thermal_Cons + (new_ec_consumption)) / Reference
                        if new <D_Limit and new >= B_Limit:
rating_new = "C"
attr_value = {xa: (area_total),xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
elif new <= C_Limit and new >= B_PLUS_Limit:
rating_new = "B"
attr_value = {xa: (area_total),xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
```

```
elif new <B_Limit and new >A_Limit:
rating_new = "B_PLUS"
attr_value = {xa: (area_total),xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),n: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
                        if Rating == 3:
                            Reference = (Primary_Thermal_Cons + ec) / C_Limit
                            new = (Primary_Thermal_Cons + (new_ec_consumption)) / Reference
                            if new <C_Limit and new >= B_PLUS_Limit:
rating_new = "B"
attr_value = {xa: (area_total),xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues( {_id: attr_value})
elif new <= B_Limit and new >= A_Limit:
rating_new = "B_PLUS"
attr_value = {xa: (area_total),xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
                        if Rating == 4:
                            Reference = (Primary_Thermal_Cons + ec) / B_Limit
                            new = (Primary_Thermal_Cons + (new_ec_consumption)) / Reference
                            if new <B_Limit and new >= A_Limit:
rating_new = "B_PLUS"
attr_value = {xa: (area_total),xl: (_production),xk: (consumption),xc: (savings),xv:
(rating_new),xb: (Old_e_CO2),xn: (New_e_CO2)}
layer_provider.changeAttributeValues({_id: attr_value})
layer.commitChanges()
```

 # City block Difference (clip buildings)

```
import processing

Difference_parameters = {'INPUT'
:"C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\City_Block.shp",
        'OVERLAY':
"C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Building_Polygons.shp",
        'OUTPUT': 'C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\City_Block
without BLDS.shp'
        }

processing.run("qgis:difference", Difference_parameters)
```

 # Clip city block from image

```
input_raster =
"C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\New_city_block_modified.tif"
output_raster =
"C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\City_block_image.png"
input_vector = "C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\City_Block without
BLDS.shp"
```

```
result = clip_raster_by_vector(input_raster, input_vector, output_raster, overwrite = True)

# City block's AreaIdentification

_City_block_No_blds = {
   'INPUT': "C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\City_Block without
BLDS.shp",
   'CALC_METHOD': 0,
   'OUTPUT':
"C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Area_City_Block_No_Blds.shp"}

processing.run("qgis:exportaddgeometrycolumns", _City_block_No_blds)

_City_block = {'INPUT':
"C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\City_Block.shp",
'CALC_METHOD': 0, 'OUTPUT':
"C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Area_City_Block_With_Blds.shp"}
processing.run("qgis:exportaddgeometrycolumns", _City_block)

layer =
QgsVectorLayer("C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Area_City_Block
_No_Blds.shp")
prov = layer.dataProvider()
field_names = [field.name() for field in prov.fields()]
for i in range(len(field_names)):
   if 'area' in field_names[i].lower():
      for feature in layer.getFeatures():
         _area_no_blds = feature[field_names[i]]

layer= QgsVectorLayer(

"C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\Area_City_Block_With_Blds.shp")
prov = layer.dataProvider()
field_names = [field.name() for field in prov.fields()]

for i in range(len(field_names)):
   if 'area' in field_names[i].lower():
      for feature in layer.getFeatures():
         _area_city_block = feature[field_names[i]]

# Image import

from PIL import Image
im = Image.open(
   "C:\\Users\\Adreas\\Desktop\\Municipality_DTM_DSM\\City_block_image.png")
pix = im.load()
soil = 0
Green_land = 0
O_T_pixels = []
gr1 = 0
```

```
gr2 = 0
gr3 = 0
bl = 0

# City block color Identification

for a in range(im.size[0]):
    for b in range(im.size[1]):
O_T_pixels.append([a,b])
        if pix[a, b][0] <= 255 and pix[a, b][0] >= 50 and pix[a, b][1] >= pix[a, b][0] - 100 and
pix[a, b][1] <= pix[a, b][0] - 15 and pix[a, b][1] <= 160 and pix[a, b][2] <= pix[a, b][1] - 5
and pix[a, b][2] <= pix[a, b][0] - 5 and pix[a, b][2] <= 130 and pix[a, b][2] >=pix[a, b][0] -
110 and pix[a, b][2] >= 30:
            soil = soil + 1
elifpix[a, b][0] >= 180 and pix[a, b][0] <= 255 and pix[a, b][1] >= 145 and pix[a, b][1] <=
235 and pix[a, b][2] >= 145 and pix[a, b][2] <= 235:
            gr1= gr1 + 1
elifpix[a, b][0] >= 80 and pix[a, b][0] <= 200 and pix[a, b][1] >= 130 and pix[a, b][1] <= 190
and pix[a, b][2] >= 130 and pix[a, b][2] <= 190:
            gr2= gr2 + 1
elifpix[a, b][0] >= 75 and pix[a, b][0] <= 150 and pix[a, b][1] >= 90 and pix[a, b][1] <= 150
and pix[a, b][2] >= 90 and pix[a, b][2] <= 150:
            gr3= gr3 + 1
elifpix[a, b][0] <= 110 and pix[a, b][1] >= 53 and pix[a, b][2] <= 110:
Green_land = Green_land + 1
elifpix[a, b][0] < 70 and pix[a, b][1] <= 70 and pix[a, b][2] <= 70:
            bl = bl + 1

Percentage_Soil = 100 * soil / (gr1 + gr2 + gr3 + soil + Green_land + bl)
Percentage_Green_land = 100 * Green_land / (gr1 + gr2 + gr3 + soil + Green_land + bl)

Norm_Green_land = (Percentage_Green_land / 100) * (_area_no_blds / _area_city_block)
Norm_Soil = (Percentage_Soil / 100) * (_area_no_blds / _area_city_block)
N_soil = Norm_Soil * 100
N_green = Norm_Green_land * 100

# Area Identification for City Block

_area_soil = _area_city_block * Norm_Soil
_area_greenland = _area_city_block * Norm_Green_land

# Import Data to SHP

layer =
QgsVectorLayer("C:\\Users\Adreas\\Desktop\\Municipality_DTM_DSM\\City_Block.shp")
layer_provider = layer.dataProvider()
layer_provider.addAttributes([QgsField("Soil_perc", QVariant.Double)])
layer_provider.addAttributes([QgsField("Green_perc", QVariant.Double)])
layer_provider.addAttributes([QgsField("Area", QVariant.Double)])
layer_provider.addAttributes([QgsField("Soil_area", QVariant.Double)])
```

```
layer_provider.addAttributes([QgsField("Green_area", QVariant.Double)])
layer.updateFields()

layer.startEditing()

features = layer.getFeatures()
layer.startEditing()
for f in features:
    _id = f.id()
attr_value = {1:(N_soil), 2:(N_green), 3: (_area_city_block), 4:(_area_soil),
5:(_area_greenland)}
layer_provider.changeAttributeValues({_id: attr_value})
layer.commitChanges()
```

 # Sustainability Factor identification

```
sustain_factor =100 * ((_area_soil + _area_greenland) /_area_city_block)
```

 # Import Data to SHP

```
layer =
QgsVectorLayer("C:\\Users\Adreas\\Desktop\\Municipality_DTM_DSM\\City_Block.shp")
layer_provider = layer.dataProvider()
layer_provider.addAttributes([QgsField("Sustain_Ft", QVariant.Double)])

layer.updateFields()
layer.startEditing()
features = layer.getFeatures()
layer.startEditing()
for f in features:
    _id = f.id()
attr_value = {6:(sustain_factor)}
layer_provider.changeAttributeValues({_id: attr_value})
layer.commitChanges()
```