Οδηγός χρήσης

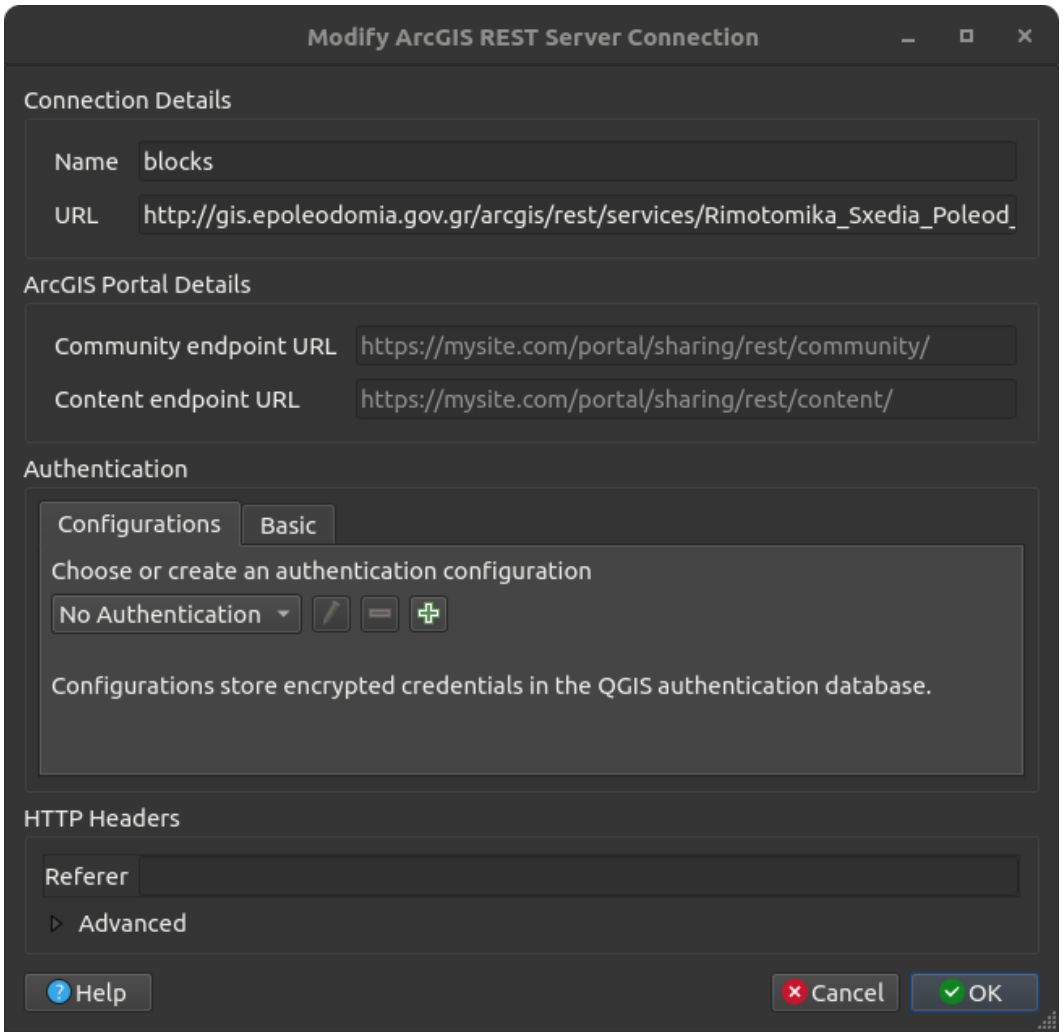
Οι αναγκαίες ψηφιοποιήσεις πραγματοποιήθηκαν με τη χρήση λογισμικών ανοιχτού κώδικα (open source) και της γλώσσας προγραμματισμού Python (3.9). Πιο συγκεκριμένα χρησιμοποιήθηκαν τα εξής:

- **Λειτουργικό σύστημα:** Ubuntu 22.04 LTS (Αν και δεν είναι απαραίτητο)
- **Λογισμικό GIS:** QGIS 3.26
- **Python IDE:** Anaconda Spyder (python 3.9) (Θα πρέπει να είναι εγκατεστημένο το module **geopandas**)
[1]

ΦΑΣΗ 1

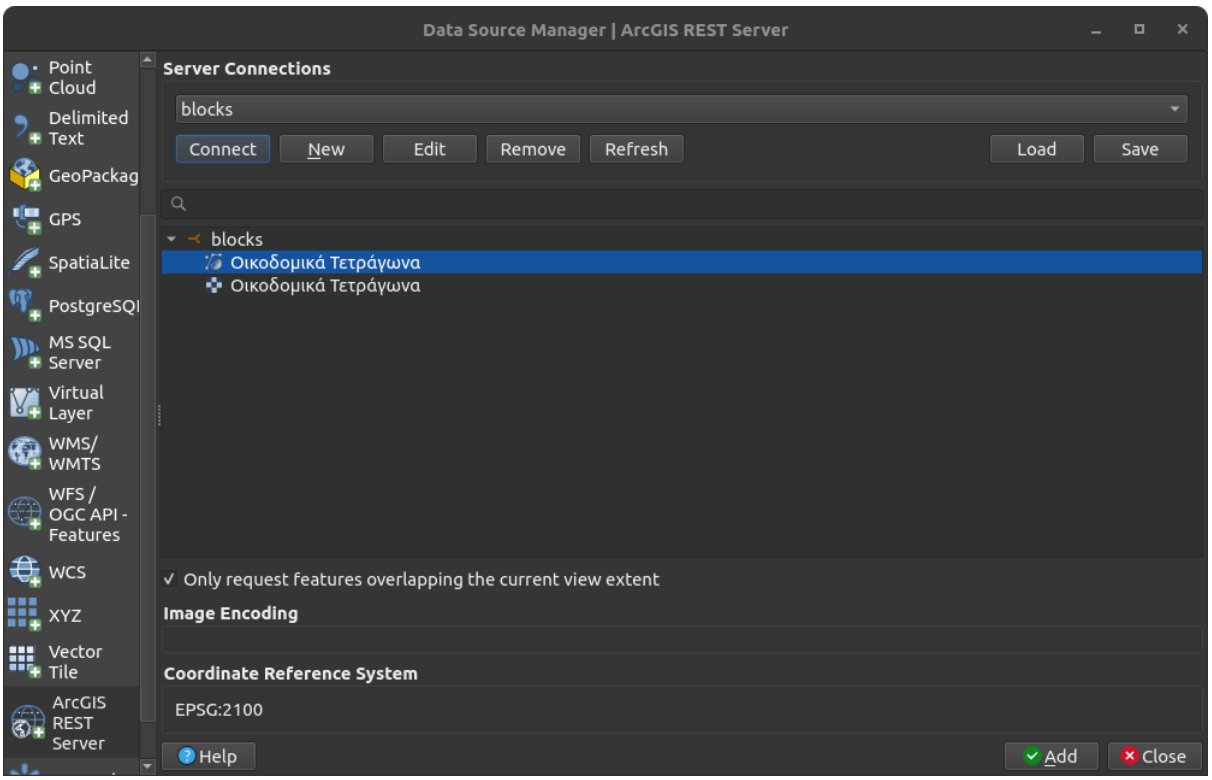
Κατεβάζουμε τα δεδομένα των οικοδομικών τετραγώνων από τον ιστότοπο e-poleodomia[(1)] και πιο συγκεκριμένα μέσω του υπερσυνδέσμου [(2)].

Μέσω QGIS εισάγουμε τον παραπάνω σύνδεσμο ως URL στην επιλογή Add ArcGIS REST Server Layer, από όπου και αντλούμε τα δεδομένα.



[1](1) http://gis.epoleodomia.gov.gr/v11/index.html#/
[(2)]http://gis.epoleodomia.gov.gr/arcgis/rest/services/Rimotomika_Sxedia_Poleod_Meletes/OikodomikaTetragona/MapServer

Στη συνέχεια πατάμε connect έτσι ώστε να συνδεθούμε με τον Server.



Έπειτα αποθηκεύουμε τα δεδομένα σε νέο shapefile με '**encoding: greek**' ως **blocks.shp**. Στη συνέχεια εισάγουμε το **blocks.shp** στο Anaconda Spyder και χρησιμοποιούμε τον αλγόριθμο του αρχείου python **esek_p1.py**. Ο αλγόριθμος του συγκεκριμένου αρχείου θα δημιουργήσει τις απαραίτητες στήλες που θα πρέπει να συμπληρωθούν έτσι ώστε να γίνει η επεξεργασία στη δεύτερη φάση.

ΦΑΣΗ 2

Αφού συμπληρωθούν τα απαραίτητα πεδία του **blocks.shp**, το συγκεκριμένο αρχείο θα πρέπει να ξαναεισαχθεί στο Anaconda Spyder όπου θα χρησιμοποιηθεί ο αλγόριθμος που περιγράφεται στο αρχείο python **esek_p2.py**. Ο αλγόριθμος αυτός θα δημιουργήσει τα απαραίτητα παραδοτέα αρχεία σε μορφή shapefile. Πιο συγκεκριμένα αυτα είναι τα εξής:

- **All_blocks**: Το οποίο περιέχει όλα τα οικοδομικά τετράγωνα καθώς και κάποιες επιπλέον πληροφορίες για αυτά.

- **all_kx_kf.shp**: Το αρχείο αυτό περιέχει όλους τους κοινόχρηστους και κοινωφελείς χώρους της χωρικής ενότητας (ή του εκάστοτε Δήμου).

- **arsi_apallotriosis**: Το αρχείο αυτό περιέχει όλους τους κοινόχρηστους και κοινωφελείς χώρους για τους οποίους υπάρχει ενδεχόμενο άρσης της απαλλοτρίωσης τους. Επίσης, σε αυτούς περιγράφεται και ο βαθμός αναγκαιότητας τους

Στον πίνακα 1 περιγράφονται πιο αναλυτικά το περιεχόμενο όλων των στηλών που εμπεριέχονται στο αρχικό αλλά και στα παραδοτέα αρχεία shapefile.

Πίνακας 1: Περιγραφή στηλών των παραδοτέων αρχείων shapefile

Όνομα στηλης	Τύπος δεδομένων	Περιγραφή
id	Ακέραιος	Μοναδικός αριθμός ταυτότητας διανυσματικού πολυγώνου
kaek	Κείμενο	Αριθμός ΚΑΕΚ πολυγώνου
ot	Κείμενο	Όνομα οικοδομικού τετραγώνου
fek	Κείμενο	ΦΕΚ έγκρισης και τροποποιήσεων του σχεδίου πόλης
pol_en	Κείμενο	Ονομασία Πολεοδομικής Ενότητας / Γειτονιάς
pol_en_id	Ακέραιος	Μοναδικός αριθμός ο οποίος δίνεται από τον χρήστη. Κάθε Πολεοδομική Ενότητα θα πρέπει να έχει έναν μοναδικό αριθμό προσδιορισμού.
pol_tomeas	Κείμενο	Πολεοδομικός Τομέας
gen_use	Κείμενο	Γενική Πολεοδομική Χρήση
sp_use	Κείμενο	Ειδική Πολεοδομική Χρήση
kx_kf_cat	Ακέραιος	Αρίθμηση με βάση τις κατηγορίες χρήσης. Δείτε τον πίνακα 1.2.
kx_kf_sel	Ακέραιος	Περιγραφή κατάστασης πολυγώνου (1 αν είναι ΚΧ-ΚΦ, 0 αν δεν είναι)
category	Ακέραιος	Τιμές: 1, 2, 3, 4 (οπως αναφέρεται στην παρ. 2, άρθρ. 92 της εγκυκλίου)
cat_date	Κείμενο	Για τις περιπτώσεις της παρ. 2, άρθρ. 92 της εγκυκλίου
cat_notes	Κείμενο	Για όλες τις περιπτώσεις της παρ. 2, άρθρ. 92 της εγκυκλίου
enforce_cnt	Ακέραιος	Επανεπιβολή Ρυμοτομικής Απαλλοτρίωσης (1 αν ισχύει, 0 αν όχι)
apallotr	Ακέραιος	Κίνδυνος άρσης απαλλοτρίωσης
pop	Ακέραιος	Πλυθυσμός της πολεοδομικής ενότητας στην οποία βρίσκεται το πολύγωνο
ulop	Ακέραιος	Υλοποιημένος ή μη υλοποιημένος (1 αν είναι υλοποιημένος, 0 αν δεν είναι)
dom_adom	Ακέραιος	Δομημένος ή αδόμητος χώρος (1 αν είναι δομημένος, 0 αν δεν είναι)
kat_gps	Ακέραιος	Κατεύθυνση ΓΠΣ (1 αν υπάρχει, 0 αν δεν υπάρχει)
krisi_mel	Ακέραιος	Αναφέρεται στο ΓΠΣ - Ο βαθμός αναγκαιότητας συμπληρώνεται από τον μελετητή (Τιμές: 1 – 5). Αναφέρεται στην περίπτωση του δείκτη 2 και πιο συγκεκριμένα στις κατηγορίες 21, 22, 23 αν υπάρχει κάποια πρόβλεψη από Γ.Π.Σ. και στις κατηγορίες 4, 5, 6, 7 ανεξαρτήτου ύπαρξης Γ.Π.Σ. (Όπως αυτές αναφέρονται στον πίνακα 2)
index_1_d	Ακέραιος	Δείκτης πληθυσμιακής πυκνότητας
index_2_d	Ακέραιος	Δείκτης κατηγοριών χρήσης γης ΚΧ-ΚΦ
index_3_d	Ακέραιος	Δείκτης αποστάσεων ΚΧ-ΚΦ
index_4_d	Ακέραιος	Δείκτης σημερινής κατάστασης δόμησης
index_5_d	Ακέραιος	Δείκτης ύπαρξης ή όχι κατεύθυνσης από το Γ.Π.Σ.
necess_rate	Ακέραιος	Βαθμός αναγκαιότητας
necess_notes	Κείμενο	Σύντομη τεκμηρίωση

Πίνακας 2: Κωδικοί συμπλήρωσης ανά χρήση γης για τη στήλη kx_kf_cat

Κωδικός	Χρήση γης
1	Κοινόχρηστος χώρος
21	Νηπιαγωγείο
22	Δημοτικό
23	Γυμνάσιο - Λύκειο
3	Αθλητικές εγκαταστάσεις
4	Υγεία
5	Διοίκηση
6	Πολιτιστικές χρήσεις
7	Θρησκευτικός χώρος

Αρχείο esek_p1 (Πηγαίος κώδικας Python)

```
# -*- coding: utf-8 -*-
"""
Created on Sat Oct 15 18:27:20 2022

@author: SUPCO
"""
import geopandas as gpd

# Διαβάζεται το αρχειο shapefile

gdf = gpd.read_file("final/blocks.shp", encoding='ISO-8859-7')

gdf2 = gpd.GeoDataFrame()

#Δημιουργούνται οι απαραίτητες στήλες

gdf2['id'] = int()

gdf2['geometry'] = gdf['geometry']

gdf2['kaek'] = str()

gdf2['ot'] = gdf['ARITHMOS_O']

gdf2['fek'] = gdf['FEK']

gdf2['pol_en'] = str()

gdf2['pol_en_id'] = int()

gdf2['pol_tomeas'] = str()

gdf2['gen_use'] = str()

gdf2['sp_use'] = str()

gdf2['kx_kf_cat'] = int()

gdf2['kx_kf_sel'] = int()

gdf2['category'] = int()

gdf2['cat_date'] = str()

gdf2['cat_notes'] = str()

gdf2['enforce_cnt'] = int()
```

```
gdf2['apallotr'] = int()

gdf2['pop'] = int()

gdf2['necess_rate'] = int()

gdf2['necess_notes'] = str()

gdf2['ulop'] = int()

gdf2['dom_adom'] = int()

gdf2['kat_gps'] = int()

gdf2['krisi_mel'] = int()


#Εξάγεται το αρχείο shapefile
gdf2.to_file("final/blocks_p1.shp", encoding='ISO-8859-7')
```

Αρχείο esek_p2 (Πηγαίος κώδικας Python)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 11 17:10:57 2022

@author: SUPCO
"""

import math
import decimal
import pandas as pd
import geopandas as gpd

# Διαβάζεται το αρχειο shapefile
gdf = gpd.read_file("final/blocks_p1.shp", encoding='ISO-8859-7')

# Δημιουργούνται τα κεντροειδή των πολυγώνων
gdf["x"] = gdf.centroid.x
gdf["y"] = gdf.centroid.y

# Υπολογίζεται η έκταση των πολυγώνων
gdf["pol_area"] = gdf.area


# Αντικαθίστανται οι κενές τιμές όπου χρειάζεται για τους υπολογισμούς
gdf["dom_adom"].fillna(0, inplace=True)

gdf["pop"].fillna(0, inplace=True)

gdf["kat_gps"].fillna(0, inplace=True)

gdf["kx_kf_cat"].fillna(0, inplace=True)


# Μετατρέπονται οι στήλες του DataFrame σε λίστες
uid = gdf["id"].to_list()

pop = gdf["pop"].to_list()

pol_en= gdf["pol_en"].to_list()

pol_en_id = gdf["pol_en_id"].to_list()

pol_area = gdf["pol_area"].to_list()
```

```
x = gdf["x"].to_list()

y = gdf["y"].to_list()

kx_kf_sel = gdf["kx_kf_sel"].to_list()

kx_kf_cat = gdf["kx_kf_cat"].to_list()

category = gdf["category"].to_list()

ulop = gdf["ulop"].to_list()

dom_adom = gdf["dom_adom"].to_list()

kat_gps = gdf["kat_gps"].to_list()

krisi_mel = gdf["krisi_mel"].to_list()

necess_rate = gdf["necess_rat"].to_list()


#Δημιουργείται λιστα με τις μοναδικές τιμές των Πολεοδομικών ενοτήτων
pol_len_id_un = list(set(pol_en_id))


# ------------------------------------------------#
# 1. Πληθυσμιακη πυκνότητα
# ------------------------------------------------#


def index_1_method(
    uid=uid,
    gdf=gdf,
    pol_en_id_un=pol_len_id_un,
    pol_en_id=pol_en_id,
    pol_area=pol_area,
    pop=pop):


    index_1 = ['NULL' for x in range(len(uid))]

    index_1_d = ['NULL' for x in range(len(uid))]

    # Iterate

    for it in pol_len_id_un:

        total_area = []

        total_pop_DE = 0

        i = 0

        while i < len(uid):

            if pol_en_id[i] == it:

                total_area.append(pol_area[i])

                if pop[i] > 0:

                    total_pop_DE = pop[i]

                else:

                    pass
```

```
            i += 1

        foo = round((total_pop_DE / ((sum(total_area) / 10_000) * 1.35)), 2)

        k = 0

        while k < len(uid):

            if pol_en_id[k] == it:

                index_1[k] = foo

            k += 1

    #Δημιουργείται η αναγκαιότητα

    i = 0

    while i < len(index_1):

        if index_1[i] < 100:

            index_1_d[i] = 1

        elif (index_1[i] >= 100) and (index_1[i] < 400):

            index_1_d[i] = 3

        else:

            index_1_d[i] = 5

        i += 1

    gdf["index_1"] = index_1  # add list to dataframe

    gdf["index_1_d"] = index_1_d  # add list to dataframe


# ------------------------------------------------#
# 2. Χρήσεις γης ΚΧ-ΚΦ
# ------------------------------------------------#

def index_2_method(uid=uid,
                   gdf=gdf,
                   pol_en_id_un=pol_len_id_un,
                   pol_en_id=pol_en_id,
                   pol_area=pol_area,
                   pop=pop):


    index_2 = ['NULL' for x in range(len(uid))]

    index_2_d = ['NULL' for x in range(len(uid))]


    #Υπολογισμός των κατηγοριών 1 και 3

    def cat_1_3(use_cat, pol_en_id_un=pol_len_id_un,
                kx_kf_cat=kx_kf_cat, ulop=ulop):

        for it in pol_len_id_un:

            total_area = []
```

```
            total_pop_DE = 0
            i = 0
            while i < len(uid):
                  if (pol_en_id[i] == it) and (kx_kf_cat[i] == use_cat) and (ulop[i] ==
1):
                    total_area.append(pol_area[i])
                    total_pop_DE = pop[i]
                i += 1

            if (sum(total_area) == 0) or (total_pop_DE == 0):
                foo = 0
            else:
                foo = round(sum(total_area) / total_pop_DE, 2)


            k = 0
            while k < len(uid):
                if (pol_en_id[k] == it) and (kx_kf_cat[k] == use_cat):
                    index_2[k] = foo
                    if use_cat == 1:
                        if foo < 100:
                            index_2_d[k] = 1
                        elif (foo >= 100) and (foo < 400):
                            index_2_d[k] = 3
                        else:
                            index_2_d[k] = 5
                    else:
                        if foo < 5.5:
                            index_2_d[k] = 4
                        else:
                            index_2_d[k] = 2
                k += 1

    #Υπολογισμός της κατηγορίας 2: Σχολικές μονάδες

    def education(cat, perc, thres, pol_en_id_un=pol_len_id_un, pol_area=pol_area,
                  pop=pop, pol_en_id=pol_en_id,kx_kf_cat=kx_kf_cat, ulop=ulop,
                  index_2=index_2, index_2_d=index_2_d):
```

```
        for it in pol_len_id_un:
            total_area = []
            total_pop_DE = 0
            i = 0
            while i < len(uid):
                if (pol_en_id[i] == it) and (kx_kf_cat[i] == cat) and (ulop[i] == 1):
                    total_area.append(pol_area[i])
                    total_pop_DE = pop[i]
                i += 1
            users = perc * total_pop_DE
            k = 0
            while k < len(uid):
                if (krisi_mel[k] > 0):
                    index_2[k] = krisi_mel[k]
                    index_2_d[k] = krisi_mel[k]
                elif (pol_en_id[k] == it) and (kx_kf_cat[k] == cat):
                    if sum(total_area) == 0:
                        index_2[k] = 0
                        index_2_d[k] = 5
                    else:
                        foo = round(sum(total_area) / users, 2)
                        index_2[k] = foo
                        if foo < thres:
                            index_2_d[k] = 4
                        else:
                            index_2_d[k] = 2

                k += 1
    #Υπολογισμός για τις υπόλοιπες κατηγορίες
                  def     rest_categories(uid=uid,     kx_kf_cat=kx_kf_cat,index_2=index_2,
index_2_d=index_2_d):
        i = 0
        while i < len(uid):
            if (kx_kf_cat[i] > 3) and (kx_kf_cat[i] < 20) and (krisi_mel[i] > 0):
```

```
                index_2[i] = krisi_mel[i]

                index_2_d[i] = krisi_mel[i]

            i += 1


    cat_1_3(1)
    cat_1_3(3)

    education(21, 0.02, 15)
    education(22, 0.1, 8)
    education(23, 0.09, 8)

    rest_categories()


    #Προσθέτουμε τις λίστες στο DataFrame
    gdf["index_2"] = index_2
    gdf["index_2_d"] = index_2_d

# ------------------------------------------------#
# 3. Απόσταση ΚΧ-ΚΦ από άλλους υλοποιημένους ΚΧ-ΚΦ
# ------------------------------------------------#

def index_3_method(uid=uid, gdf=gdf, kx_kf_cat=kx_kf_cat, x=x, y=y):

    index_3_KX_KX = ['NULL' for x in range(len(uid))]

    index_3_KX_KF = ['NULL' for x in range(len(uid))]

    index_3_KF_KF = ['NULL' for x in range(len(uid))]

    index_3_KX_KX_d = ['NULL' for x in range(len(uid))]

    index_3_KX_KF_d = ['NULL' for x in range(len(uid))]

    index_3_KF_KF_d = ['NULL' for x in range(len(uid))]


    index_3_d = ['NULL' for x in range(len(uid))]


    #Υπολογισμός της ευκλίδειας απόστασης μεταξύ δύο σημείων

    def eucl_dist(x1, y1, x2, y2):

        return math.sqrt(((x2 - x1) ** 2) + ((y2 - y1) ** 2))


    #Καταχώριση όλων των ΚΧ-ΚΦ

    kx_id = []

    kx_coords = []

    kf_id = []

    kf_coords = []

    i = 0

    while i < len(uid):

        if kx_kf_cat[i] == 1:

            kx_id.append(uid[i])
```

```
            kx_coords.append([x[i], y[i]])
        elif kx_kf_cat[i] > 1:
            kf_id.append(uid[i])
            kf_coords.append([x[i], y[i]])
        i += 1
#Καταχώριση όλων των υλοποιημένων ΚΧ-ΚΦ
kx_id_u = []
kx_coords_u = []
kf_id_u = []
kf_coords_u = []
i = 0
while i < len(uid):
    if (kx_kf_cat[i] == 1) and (ulop[i] == 1):
        kx_id_u.append(uid[i])
        kx_coords_u.append([x[i], y[i]])
    elif (kx_kf_cat[i] > 1) and (ulop[i] == 1):
        kf_id_u.append(uid[i])
        kf_coords_u.append([x[i], y[i]])
    i += 1
#Υπολογισμός της ελάχιστης απόστασης από ΚΧ σε ΚΧ
if len(kx_id) == 0:
    pass
elif len(kx_id) == 1:
    uid_index = uid.index(kx_id[0])
    index_3_KX_KX[uid_index] = 0
else:
    i = 0
    while i < len(kx_id):
        dis = []
        base_point = kx_coords[i]
        base_point_index = kx_id[i]
        k = 0
        while k < len(kx_id_u):
```

```
            if kx_id_u[k] == base_point_index:

                pass

            else:

                foo = eucl_dist(
                    base_point[0],
                    base_point[1],
                    kx_coords_u[k][0],
                    kx_coords_u[k][1],
                )

                dis.append(foo)

            k += 1

        uid_index = uid.index(base_point_index)

        index_3_KX_KX[uid_index] = round(min(dis), 2)

        i += 1

#Υπολογισμός της ελάχιστης απόστασης από ΚΧ σε ΚΦ

if len(kx_id) == 0:

    pass

elif len(kx_id) == 1:

    uid_index = uid.index(kx_id[0])

    index_3_KX_KF[uid_index] = 0

else:

    i = 0

    while i < len(kx_id):

        dis = []

        base_point = kx_coords[i]

        base_point_index = kx_id[i]

        k = 0

        while k < len(kf_id_u):

            if kf_id_u[k] == base_point_index:

                pass

            else:

                foo = eucl_dist(
                    base_point[0],
                    base_point[1],
                    kf_coords_u[k][0],
                    kf_coords_u[k][1],
                )

                dis.append(foo)

            k += 1
```

```
            uid_index = uid.index(base_point_index)
            index_3_KX_KF[uid_index] = round(min(dis), 2)
            i += 1
    #Υπολογισμός της ελάχιστης απόστασης από ΚΦ σε ΚΦ
    if len(kf_id) == 0:
        pass
    elif len(kf_id) == 1:
        uid_index = uid.index(kf_id[0])
        index_3_KF_KF[uid_index] = 0
    else:
        i = 0
        while i < len(kf_id):
            dis = []
            base_point = kf_coords[i]
            base_point_index = kf_id[i]
            k = 0
            while k < len(kf_id_u):
                if kf_id_u[k] == base_point_index:
                    pass
                else:
                    foo = eucl_dist(
                        base_point[0],
                        base_point[1],
                        kf_coords_u[k][0],
                        kf_coords_u[k][1],
                    )
                    dis.append(foo)
                k += 1
            uid_index = uid.index(base_point_index)
            index_3_KF_KF[uid_index] = round(min(dis), 2)
            i += 1
        #Δημιουργείται η αναγκαιότητα για ΚΧ - ΚΧ
        i = 0
        while i < len(index_3_KX_KX):
            if index_3_KX_KX[i] == "NULL":
                pass
```

```
    elif index_3_KX_KX[i] == 0:
        index_3_KX_KX_d[i] = 5
    else:
        if index_3_KX_KX[i] < 500:
            index_3_KX_KX_d[i] = 1
        elif (index_3_KX_KX[i] >= 500) and (index_3_KX_KX[i] < 1500):
            index_3_KX_KX_d[i] = 3
        else:
            index_3_KX_KX_d[i] = 5
    i += 1
#Δημιουργείται η αναγκαιότητα για ΚΧ - ΚΦ
i = 0
while i < len(index_3_KX_KF):
    if index_3_KX_KF[i] == "NULL":
        pass
    elif index_3_KX_KF[i] == 0:
        index_3_KX_KF_d[i] = 5
    else:
        if index_3_KX_KF[i] < 500:
            index_3_KX_KF_d[i] = 5
        elif (index_3_KX_KF[i] >= 500) and (index_3_KX_KF[i] < 1500):
            index_3_KX_KF_d[i] = 3
        else:
            index_3_KX_KF_d[i] = 1
    i += 1

#Δημιουργείται η αναγκαιότητα για ΚΦ - ΚΦ
i = 0
while i < len(index_3_KF_KF):
    if index_3_KF_KF[i] == "NULL":
        pass
    elif index_3_KF_KF[i] == 0:
        index_3_KF_KF_d[i] = 5
    else:
```

```
                if index_3_KF_KF[i] < 500:
                    index_3_KF_KF_d[i] = 1
                elif (index_3_KF_KF[i] >= 500) and (index_3_KF_KF[i] < 1500):
                    index_3_KF_KF_d[i] = 3
                else:
                    index_3_KF_KF_d[i] = 5
            i += 1
    #Υπολογίζεται η τελική αναγκαιότητα
    i = 0
    while i < len(index_3_d):
        if (index_3_KX_KX_d[i] == "NULL") and (index_3_KX_KF_d[i] == "NULL"):
            pass
        else:
            index_3_d[i] = math.ceil((index_3_KX_KX_d[i] + index_3_KX_KF_d[i]) / 2)
        if (index_3_KF_KF_d[i] == "NULL"):
            pass
        else:
            index_3_d[i] = index_3_KF_KF_d[i]
        i += 1

    gdf["index_3_KX_KX"] = index_3_KX_KX
    gdf["index_3_KX_KX_d"] = index_3_KX_KX_d
    gdf["index_3_KX_KF"] = index_3_KX_KF
    gdf["index_3_KX_KF_d"] = index_3_KX_KF_d
    gdf["index_3_KF_KF"] = index_3_KF_KF
    gdf["index_3_KF_KF_d"] = index_3_KF_KF_d
    gdf["index_3_d"] = index_3_d


# ------------------------------------------------#
# 4. Σημερινή κατάσταση (Δομημένο - Αδόμητο)
# ------------------------------------------------#

def index_4_method(uid=uid, dom_adom=dom_adom, kx_kf_sel=kx_kf_sel):
    index_4_d = ['NULL' for x in range(len(uid))]
    i = 0
```

```
    while i < len(index_4_d):

        if (dom_adom[i] == 0) and (kx_kf_sel[i] == 1):

            index_4_d[i] = 5

        elif (dom_adom[i] == 1) and (kx_kf_sel[i] == 1):

            index_4_d[i] = 1

        i += 1

    gdf["index_4_d"] = index_4_d  # add list to dataframe



# ------------------------------------------------#
# 5. Κατεύθυνση απο ΓΠΣ
# ------------------------------------------------#


def index_5_method(uid=uid, kat_gps=kat_gps, kx_kf_sel=kx_kf_sel):

    index_5_d = ['NULL' for x in range(len(uid))]


    i = 0

    while i < len(index_5_d):

        if (kat_gps[i] == 0) and (kx_kf_sel[i] == 1):

            index_5_d[i] = 1

        elif (kat_gps[i] == 1) and (kx_kf_sel[i] == 1):

            index_5_d[i] = 5

        i += 1

    gdf["index_5_d"] = index_5_d  # add list to dataframe


# Call funtions
index_1_method()

index_2_method()

index_3_method()

index_4_method()

index_5_method()

#Δημιουργία τελικής στήλης αναγκαιότητας

def final_calc(gdf=gdf, uid=uid):

    index_1_d = gdf["index_1_d"].to_list()

    index_2_d = gdf["index_2_d"].to_list()

    index_3_d = gdf["index_3_d"].to_list()

    index_4_d = gdf["index_4_d"].to_list()
```

```
    index_5_d = gdf["index_5_d"].to_list()


    i = 0

    while i<len(uid):

        total = []

        def calc(index, c):

            if type(index[c]) == int:

                if index[c] > 0:

                    total.append(index[c] * 0.2)

        calc(index_1_d, i)

        calc(index_2_d, i)

        calc(index_3_d, i)

        calc(index_4_d, i)

        calc(index_5_d, i)

        if (sum(total) % 2) > 0.5:

            necess_rate[i] = math.ceil(sum(total))

        else:

            necess_rate[i] = math.floor(sum(total))


        i+=1

    gdf.drop(columns = ["necess_rat"], inplace=True)

    gdf["necess_rate"] = necess_rate


final_calc()



#Εξάγονται τα παραδοτέα αρχεία shapefiles


#All_blocks
all_blocks  =  gdf[['id',  'geometry',  'kaek',  'ot',  'fek',  'pol_en',  'pol_tomeas',
'gen_use', 'sp_use', 'kx_kf_cat', 'category',
                  'cat_date', 'cat_notes', 'enforce_cn']].copy()

all_blocks.to_file("final/all_blocks.shp", encoding='ISO-8859-7')



#all_kx_kf
all_kx_kf = gdf.loc[gdf['kx_kf_sel'] == 1].copy()

all_kx_kf  =  all_kx_kf[['id',  'geometry',  'kaek',  'ot',  'fek',  'pol_en',  'pol_tomeas',
'gen_use', 'sp_use', 'kx_kf_cat', 'category',
                  'cat_date', 'cat_notes', 'enforce_cn']].copy()

all_kx_kf.to_file("final/all_kx_kf.shp", encoding='ISO-8859-7')
```

```
#arsi_apallotriosis
arsi_apallotriosis = gdf.loc[gdf['apallotr'] == 1].copy()

arsi_apallotriosis   =   arsi_apallotriosis[['id',   'geometry',   'kaek',   'ot',   'fek',
'pol_en','pol_tomeas',      'gen_use',      'sp_use',      'kx_kf_cat',      'category',
'cat_date', 'cat_notes', 'enforce_cn', 'index_1_d', 'index_2_d',
                                                'index_3_d', 'index_4_d', 'index_5_d',
'necess_rate', 'necess_not']].copy()

arsi_apallotriosis.to_file("final/arsi_apallotriosis.shp", encoding='ISO-8859-7')
```