

**Analyz Przestrzenne**

**Projekt 1**

**„Analiza obszaru pod inwestycję  
hotelową w gminie miejskiej”**

Krzysztof Ciszek

Semestr zimowy 2022/2023

## **Spis Treści**

1. Wstęp	3
2. Szczegółowy opis wykonania ćwiczenia i realizacji skryptu ipynb	4
3. Raport z testu na danych testowych z innego obszaru	43
4. Podsumowanie i wnioski	46

## **1. Wstęp**

Wybierając lokalizację nowego hotelu należy wziąć pod uwagę wiele różnych czynników, bardziej lub mniej mierzalnych. Warto pamiętać, że dobra lokalizacja sama w sobie nie zagwarantuje sukcesu obiektu hotelarskiego, jednak będzie sporym atutem. Wymienię teraz niektóre czynniki do uwzględnienia przy wyborze lokalizacji hotelu: walory wypoczynkowe (jakość powietrza, poziom hałasu, aspekty wizualne otoczenia, obiekty sąsiadujące, odległość od okolicznych atrakcji lub miejsc szczególnych), koszty budowy i aspekty techniczno-prawne (pokrycie terenu, miejscowe plany zagospodarowania przestrzennego, gazociąg, woda, prąd, internet, budowa drogi dojazdowej, użytkowanie terenu), inne (odległość: od skupisk ludzkich, od konkurencji, od restauracji, od sklepów, od budynków mieszkalnych, od istniejących dróg... nachylenie stoków, dostęp do światła słonecznego, kształt działki pod budowę). W tym projekcie obszarem branym pod uwagę przy wyborze optymalnej lokalizacji jest gmina miejska Świeradów-Zdrój, znajdująca się w górach izerskich oraz w powiecie lubańskim w woj. dolnośląskim. Gmina ta sąsiaduje z gminami Mirsk i Leśna oraz z Czechami. Gospodarka gminy jest oparta głównie na turystyce i działalności uzdrowiskowej, zimą do dyspozycji turystów jest sześć wyciągów narciarskich, a wiosną i latem wiele ścieżek i tras rowerowych. Przez gminę przechodzi także kilka pieszych szlaków turystycznych.

## 2. Szczegółowy opis wykonania ćwiczenia i realizacji skryptu ipynb

Cały projekt wykonałem w programie ArcGIS Pro z wykorzystaniem wbudowanego systemu wsparcia dla pythonowych notebooków (ipynb). Napisałem uniwersalny (działający dla poprawnych danych wejściowych dla innego obszaru) skrypt, wykorzystujący przede wszystkim bibliotekę arcpy.

Większość danych do tego projektu to były dane znajdujące się w obrębie gminy Świeradów-Zdrój, jednak prawidłowe wykonanie analizy wymagałoby również (w tym przypadku) pobrania danych dla sąsiadujących gmin. Więcej o tym i o możliwościach ulepszenia skryptu napiszę w sekcji z wnioskami.

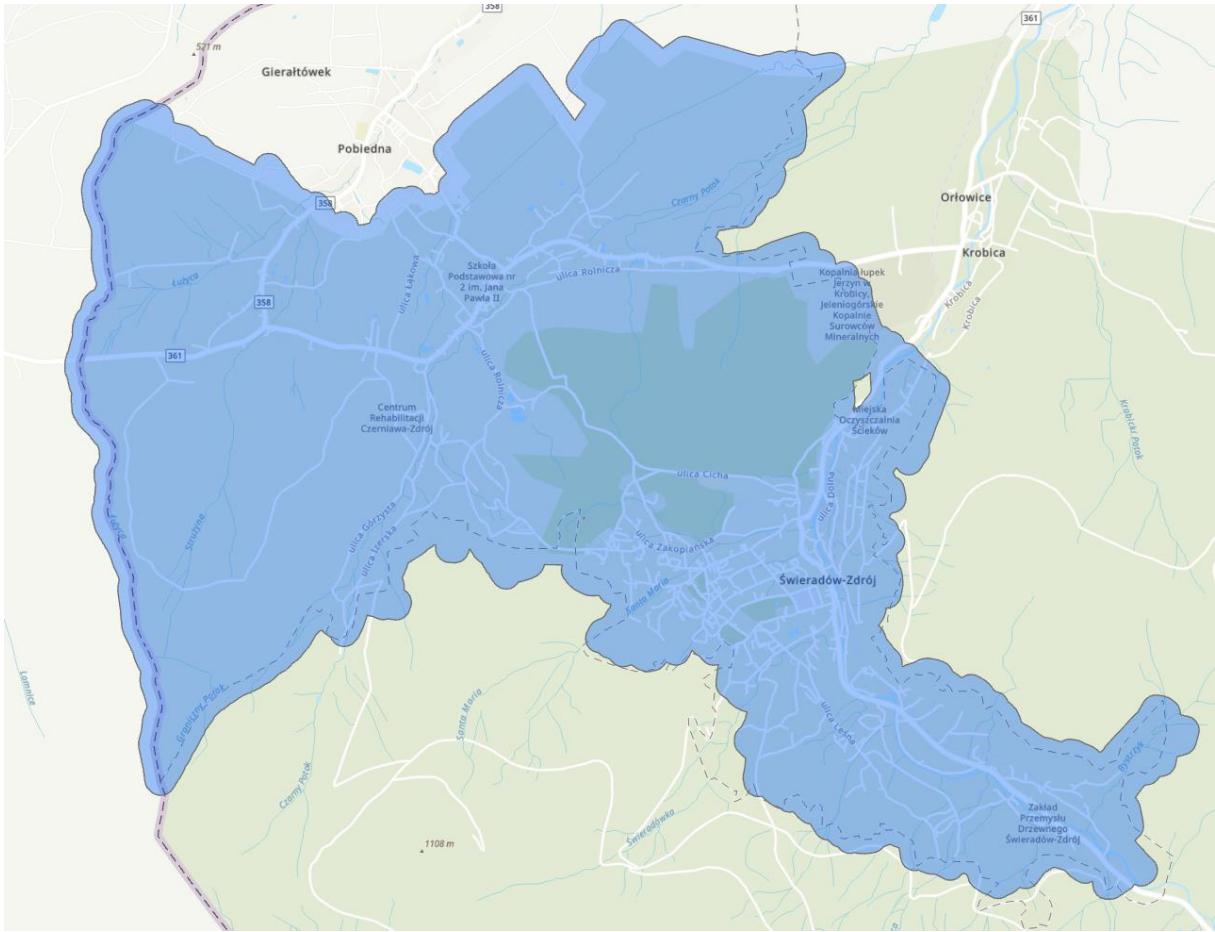
W tej sekcji opiszę chronologicznie (w domyślnej kolejności wykonania) poszczególne kroki analizy i odpowiadające im komórki z kodem z notebooka ipynb. Opiszę na czym polegały poszczególne kryteria wyboru określonego terenu, kod pozwalający na ich realizację oraz uzyskane wyniki. Będę też dodawał wizualizacje istotnych danych i wyników. Znajdą się tutaj screenshoty wyników działania algorytmów, ale nie dla wszystkich etapów i podetapów, te pomniejsze pominę. Dodatkowo w załącznikach załączę kod programu wraz z danymi wejściowymi i (być może) uproszczony data flow diagram.

Dane wejściowe: BDOT dla powiatu (folder z danymi w formacie shp), NMT dla obszaru opracowania (sklejony z kilku kafelków, jeden plik tif), gazociąg (warstwa liniowa), gleby (warstwa shp), granice gminy (poligon shp), działki ewidencyjne (warstwa shp z EGiB).

Po importie potrzebnych bibliotek (arcpy, sys, os, math) i sprawdzeniu dostępności rozszerzeń Spatial i 3D dla tej wersji licencji ArcGIS Pro przystąpiłem do ustawienia ścieżek do plików wejściowych i ustawienia niektórych zmiennych środowiskowych arcpy env (niektóre ustawienia nie działały, więcej o tym we wnioskach). Dla uniwersalności i łatwości zmiany danych przyjąłem konwencję podawania ścieżek do danych wejściowych, ale wyniki algorytmów zapisuję już w geodatabase (do której użytkownik podaje ścieżkę). Jest to jedyne miejsce w kodzie, gdzie użytkownik musi coś ustawić, wszystko inne powinno się wykonać automatycznie bez błędów (przy założeniu, że dane wejściowe nie zawierają błędów i mają poprawne atrybuty).

```
#-----ŚWIERADÓW-----#
workspace_path = r"C:\SEMS\AP\p1\analiza_hotele\analiza_hotele.gdb" # gdb
folder_bdot = r"C:\SEMS\AP\p1\DANE\bdot_caly"
path_nmt = r"C:\SEMS\AP\p1\DANE\nmt_gminy\nmt_swieradow.tif"
path_gazociag = r"C:\SEMS\AP\p1\DANE\gazociag\gazociag.shp"
path_gleby = r"C:\SEMS\AP\p1\DANE\gleby\gleby_gm_swieradow.shp"
path_granice_gminy = r"C:\SEMS\AP\p1\DANE\granice_gminy\gmina_swieradow.gpkg\main.gmina_swieradow"
path_dzialki = r"C:\SEMS\AP\p1\DANE\dzialki_gminy_egib\dzialki_egib.shp" # r"C:\SEMS\AP\p1\DANE\dzialki_gminy_egib\dzialki_egib.gpkg"
bufor_gminy = arcpy.analysis.Buffer(path_granice_gminy, "bufor_gminy", 100)
```

Po ustawnieniu ścieżek, zrobiłem stumetrowy bufor dla gminy, na podstawie którego powstał prostokąt ograniczający (extent), w którego obrębie działały algorytmy przetwarzania danych. Bufor gminy prezentuje się następująco.



Następna komórka skryptu pozbywa się kropek z nazw plików w folderze z danymi z BDOT (było to potrzebne, ponieważ ArcGIS Pro mia problem z kropkami w nazwach plików) i tworzy słownik (python dictionary) bdot, zawierający ścieżki do potrzebnych warstw, zapisane w postaci par klucz : wartość (np. 'budynki' : 'C:\ (...) \PLPZGiK3370210\_\_OT\_BUBD\_A.shp')

```
# Pozbycie się kropek z nazw plików w folderze bdot_caly

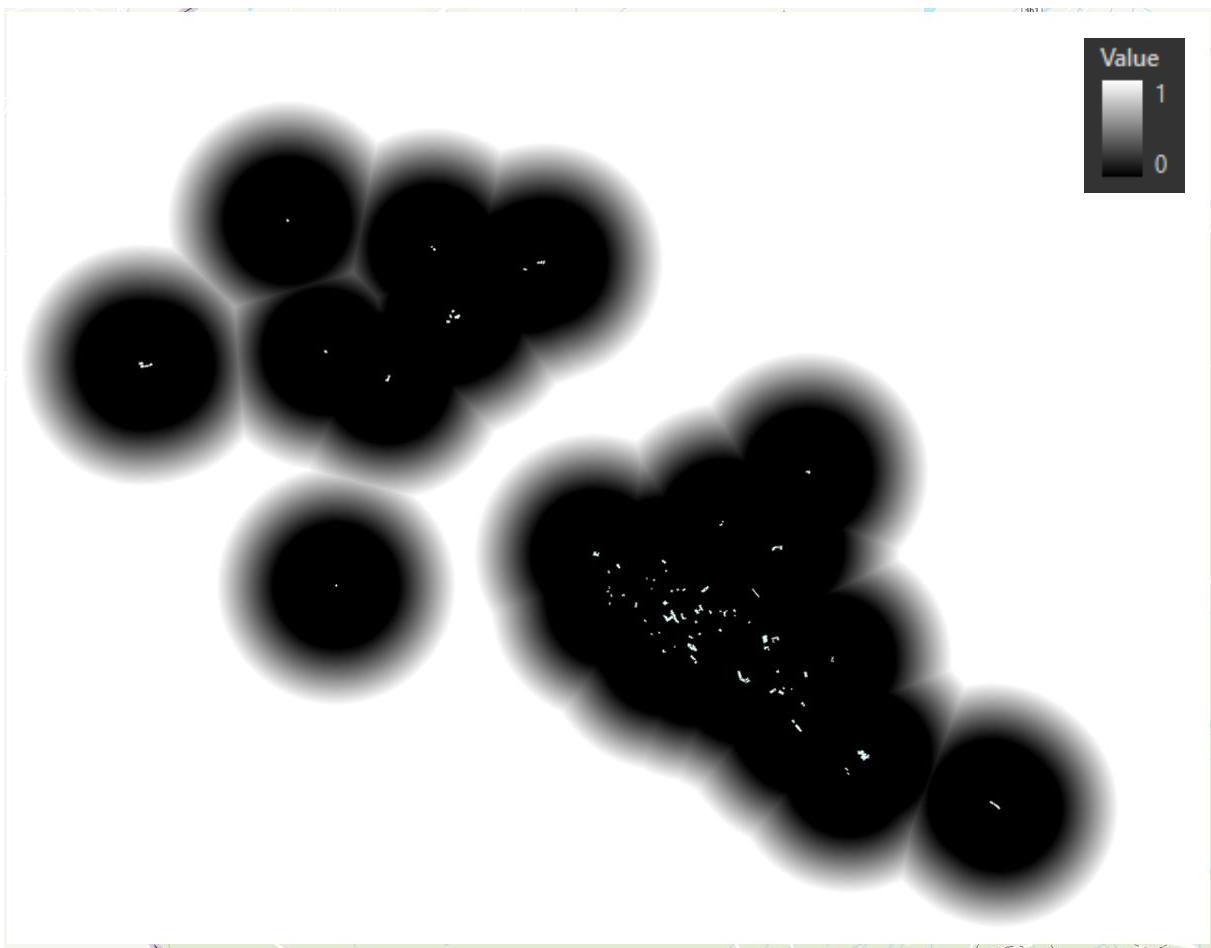
bdot = dict()
names = {'budynki':'bubd_a', 'drogi':'skdr_l', 'rzeki':'swrs_l', 'jeziora':'ptwp_a', 'lasy':'ptlz_a'}
extensions = ('.shp', '.prj', '.xlsx', '.dbf', '.shx', '.cpg')
for filename in os.listdir(folder_bdot):
    if os.path.isfile(os.path.join(folder_bdot, filename)):
        src = rf"{folder_bdot}\{filename}"
        if src.endswith(extensions):
            if src.count('.') == 4:
                for k, v in names.items():
                    if v in filename.lower():
                        dst = src.replace('.', '', 3)
                        os.rename(src, dst)
                        if src.endswith('.shp'):
                            bdot[k] = dst
            if 'PT' in src:
                dst = src.replace('.', '', 3)
                try:
                    os.rename(src, dst)
                except FileNotFoundError:
                    pass
```

## Kryterium 1

### Kryterium 1 - odległość od konkurencji minimum 400m

```
query = "FUNKCJA LIKE '%hotele%' OR FUNKCJA LIKE '%motel%' OR FUNKCJA LIKE '%pensjonat%' OR FUNKCJA LIKE '%domWypoczynkowy%'"
hotele = arcpy.analysis.Select(bdot['budynki'], 'hotele', query)
ed = arcpy.sa.EucDistance(hotele)
minimum = 400 # 400
maximum = 800 # 800
k1 = arcpy.sa.FuzzyMembership(ed, arcpy.sa.FuzzyLinear(minimum, maximum))
```

Kryterium 1 polegało na wyborze terenów oddalonych od konkurencji o minimum 400 metrów (400 to wybrana przez nas lub eksperta wartość, więcej o doborze wartości napiszę we wnioskach, w tej sekcji skupię się na opisie kryteriów z ustalonimi przykładowymi wartościami). Sens tego kryterium jest taki, żeby sobie „nie przeszkadzać” z konkurencją i nie zabierać klientów. Generalnie im dalej od konkurencji, tym lepiej, dlatego w tym kryterium użyłem narzędzi wykorzystujących logikę rozmytą (fuzzy logic). Pozwoliło mi to uzyskać wartości opisujące przydatności terenu zmieniającą się w sposób płynny, stopniowy (0 oznacza tereny nieprzydatne, a 1 tereny „najbardziej przydatne”). Ma to taką zaletę w porównaniu do logiki ostrej (boolowskie 0 i 1), że pozwala stwierdzić np., że tereny w odległości 1000 metrów od konkurencji są lepsze od tych bliskich na 500 metrów. W logice ostrej uzyskalibyśmy taką samą wartość (1) w obydwu przypadkach. W celu realizacji kryterium w kodzie, wybrałem te budynki z warstwy BUBDA z BDOT, których atrybut określający funkcję szczegółową był jednym z: hotel, motel, pensjonat, domWypoczynkowy. Wybrane budynki posłużyły za argument funkcji EucDistance, która zwróciła raster z odległościami euklidesowymi od najbliższego wybranego budynku. Następnie użyłem funkcji FuzzyMembership, aby dostać zreklasyfikowany raster z wartościami oznaczającymi przydatność gruntu w skali 0-1. Wartości zmieniają się liniowo od 400m do 800m (tak uznałem, że po przekroczeniu odległości od konkurencji równej 800m już nie ma dla nas dużego znaczenia jak daleko jesteśmy i wszystko ponad 800m nadaje się tak samo dobrze). Przed 400 i po 800 wartości wynoszą odpowiednio 0 i 1. Wyniki prezentują się następująco.



Wyniki są zgodne z oczekiwaniami. Wartości rastra mieszczą się w zakresie 0-1. Widać odcień szarości powstałe pośrednio przez użycie fuzzy logic. W części południowo-wschodniej gminy widać duże skupisko budynków, całkowicie wykluczające dość duży obszar gminy (pod względem tego kryterium). Można zauważyć, że w części północno-zachodniej jest stosunkowo niewiele obiektów hotelarskich, są one jednak „rozrzucone” w sporej odległości od siebie, przez co też wykluczają dużą powierzchnię. Skłania to do przemyśleń, czy nie lepiej byłoby wymyślić reklasyfikującą funkcję biorącą pod uwagę nie tylko odległość od najbliższego obiektu, ale też ich poziom „skupienia” w jednym miejscu. Wtedy tereny północno-zachodnie (mówiąc w ogólności) byłyby lepsze od tych południowo-wschodnich. Jak bardzo lepsze i w ogóle jak dobre, to zależałoby od implementacji takiej funkcji.

## Kryterium 2

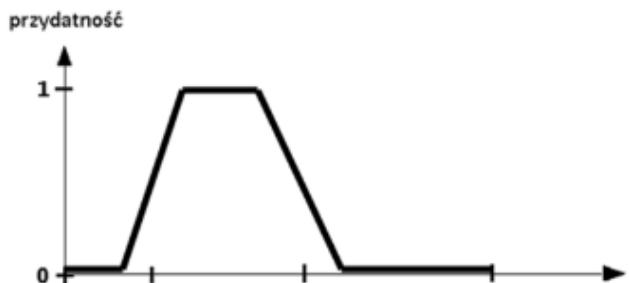
### Kryterium 2 - odległość od budynków mieszkalnych od 25 do 150m

```
query = "FUNOGBUD LIKE '%miesz%' OR FUNOGBUD LIKE '%Miesz%'" # 1036
mieszkalne = arcpy.analysis.Select(bdot['budynki'], "mieszkalne", query)
ed = arcpy.sa.EucDistance(mieszkalne)
minimum = 25 # 25
maximum = 50 # 50
fuzzy1 = arcpy.sa.FuzzyMembership(ed, arcpy.sa.FuzzyLinear(minimum, maximum))
minimum = 150 # 150
maximum = 125 # 125
fuzzy2 = arcpy.sa.FuzzyMembership(ed, arcpy.sa.FuzzyLinear(minimum, maximum))

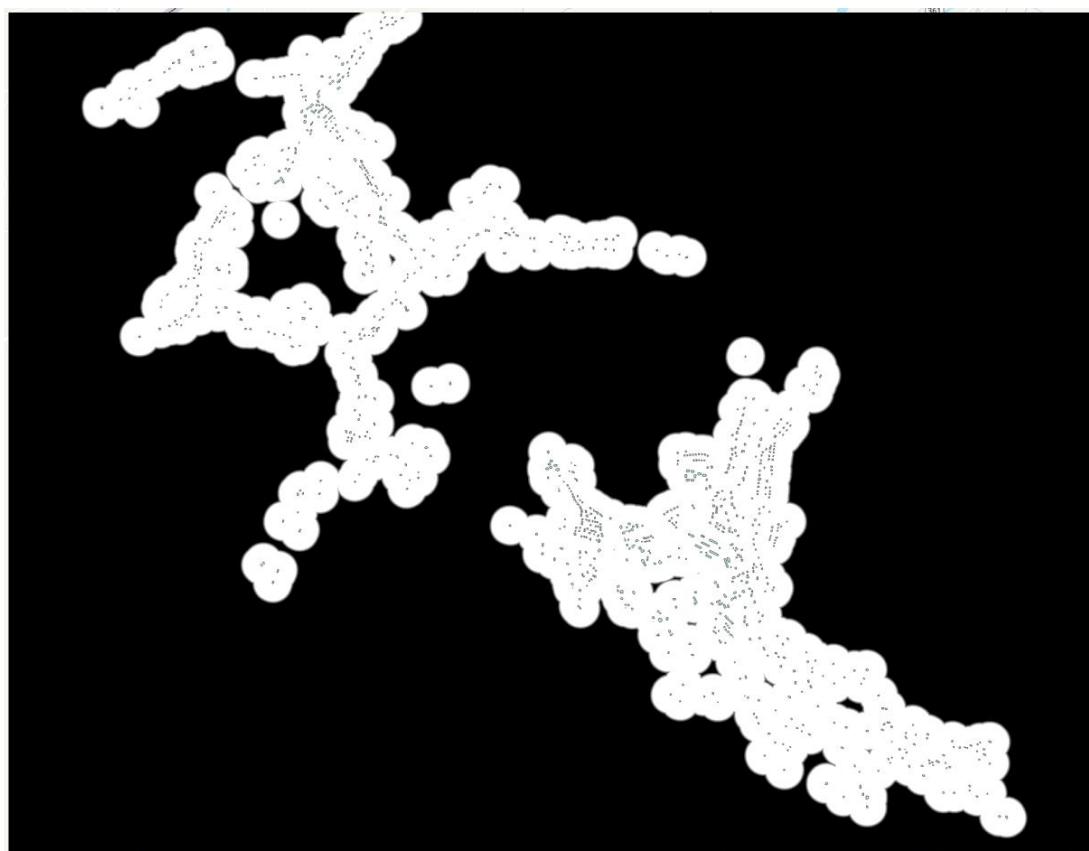
k2 = arcpy.sa.Times(fuzzy1, fuzzy2)
```

Kryterium 2 polegało na wyborze terenów położonych w określonej odległości od budynków mieszkalnych. Nie za blisko (by mieć trochę przestrzeni wokół hotelu) i nie za daleko (m.in. ze względu na to, że tam, gdzie są budynki mieszkalne jest też zwykle cała infrastruktura - sklepy, drogi, itd.). Wartości odległości „sprzyjające” (ustalone przykładowe wartości) mieścią się w zakresie od 25 do 150 metrów od budynków mieszkalnych. Ponownie użyłem narzędzi logiki rozmytej. Przyjąłem, że odległości od 50 do 125 są tak samo dobre (i przyjmują najwyższą wartość: 1), a od 25 do 50 i od 125 do 150 metrów funkcja przydatności odpowiednio: liniowo rośnie i liniowo maleje. Odległości poza przedziałem 25-150 metrów przyjmują wartości przydatności 0. Potrzebne budynki mieszkalne wyodrębniłem z warstwy BUBDA, używając atrybutu odpowiadającego funkcji szczegółowej budynku. Po uzyskaniu rastra odległości euklidesowych od budynków, użyłem funkcji FuzzyMembership, z tym że musiałem rozbić funkcję przydatności na dwie części (a raczej złożyć całą funkcję przydatności z dwóch osobnych wyników funkcji FuzzyMembership), ponieważ ArcGIS Pro (w przeciwieństwie do QGISA) niestety nie zawiera algorytmów tworzących funkcję o potrzebnym kształcie zamkniętych w jednej funkcji. Do połączenia dwóch wyników FuzzyMembership użyłem funkcji Times, która mnoży wartości odpowiadających sobie pikseli z obydwu rastrow. Uzyskanie w ten sposób dobrego (oczekiwanej) wyniku było możliwe m.in. dlatego, że wartości obydwu funkcji są znormalizowane (są w zakresie 0-1) i kształty funkcji są odpowiednie (przedziały, gdzie funkcje rosną i maleją nie nachodzą na siebie oraz istnieje przedział, gdzie obydwie funkcje przyjmują wartość 1). Na rysunku obok

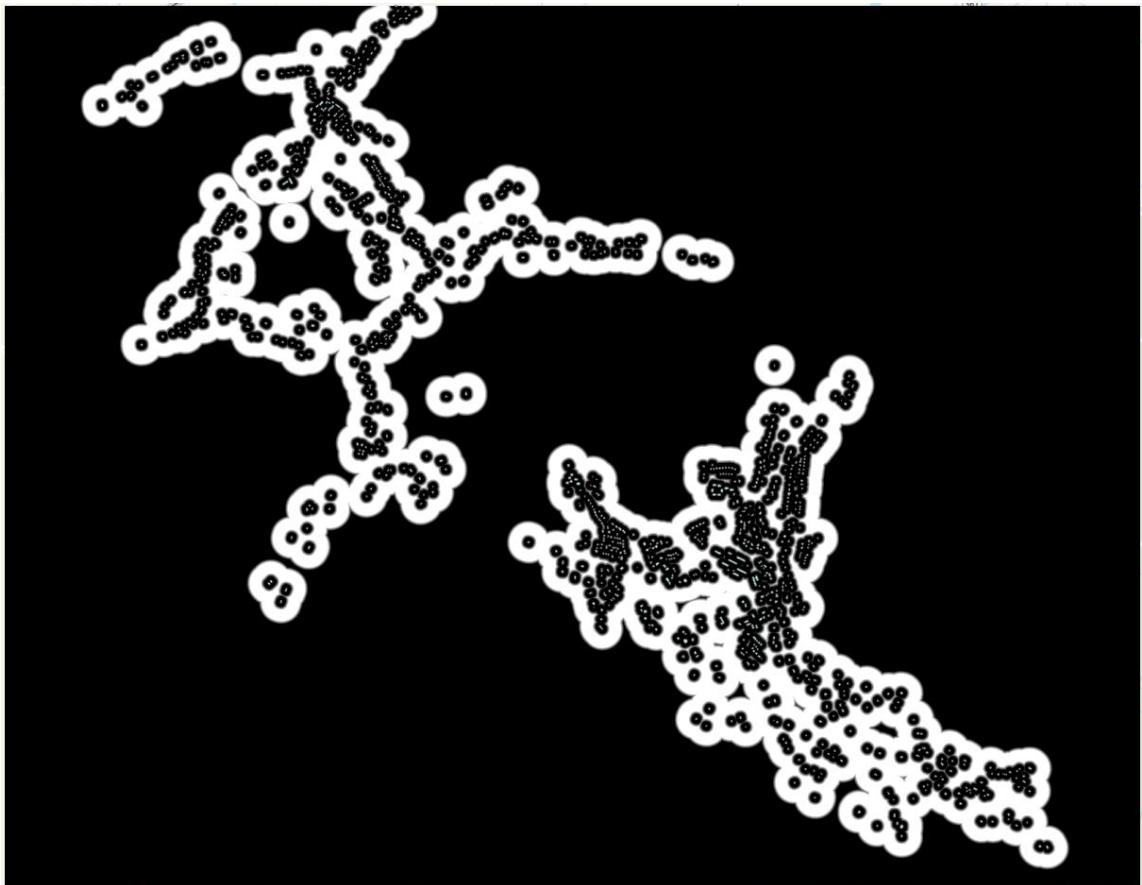
przedstawiam dla zobrazowania kształt powstałej funkcji (wartości odległości na osi x na rysunku nie mają znaczenia w celu zwizualizowania kształtu funkcji dla lepszego zrozumienia kryterium).



Wyniki dwóch cząstkowych zastosowań FuzzyMembership (czarny=0, biały=1; taka reprezentacja wartości kolorami nie zmienia się w kolejnych czarno-białych screenshotach prezentujących wyniki końcowe danego kryterium).



Wynik połączenia obydwu warstw.



### Kryterium 3

#### Kryterium 3 - odległość od istniejących dróg od 15 do 100m

```
drogi = bdot['drogi']
ed = arcpy.sa.EucDistance(drogi)
minimum = 15 # 15
maximum = 20 # 20
fuzzy1 = arcpy.sa.FuzzyMembership(ed, arcpy.sa.FuzzyLinear(minimum, maximum))
minimum = 200 # 200
maximum = 100 # 100
fuzzy2 = arcpy.sa.FuzzyMembership(ed, arcpy.sa.FuzzyLinear(minimum, maximum))

k3 = arcpy.sa.Times(fuzzy1, fuzzy2)
```

Kryterium 3 polegało na ocenieniu jakości terenu pod względem odległości od istniejących dróg. Nie chcemy mieć hotelu tuż przy drodze, ale też nie chcemy mieć go zbyt daleko, bo wiązałoby się to z kosztami budowy drogi dojazdowej. Ponownie użyłem tutaj logiki rozmytej. Kształt funkcji określającej przydatność jest podobny do tego z kryterium 2 („trapez”). Uznałem, że tereny najbardziej przydatne pod względem kryterium 3 mieszczą się w zakresie odległości 20-100 metrów. Od 100 do 200 funkcja liniowo maleje, od 15 do 20 liniowo rośnie. Wszędzie indziej przyjmuje wartość zero, oznaczającą całkowitą nieprzydatność terenu pod względem uwzględnianego kryterium. Dane o geometrii liniowej reprezentujące drogi wziąłem z warstwy SKDR z BDOT. Po uzyskaniu rastra z odległościami euklidesowymi, użyłem – podobnie jak przy kryterium 2 – dwóch wyników funkcji FuzzyMembership z odpowiednio podanymi argumentami (o wartościach 15 i 20 oraz 200 i 100). Następnie połączylem je, używając do tego funkcji Times (mnożenie rastrow). Wynikowy raster prezentuje się następująco.

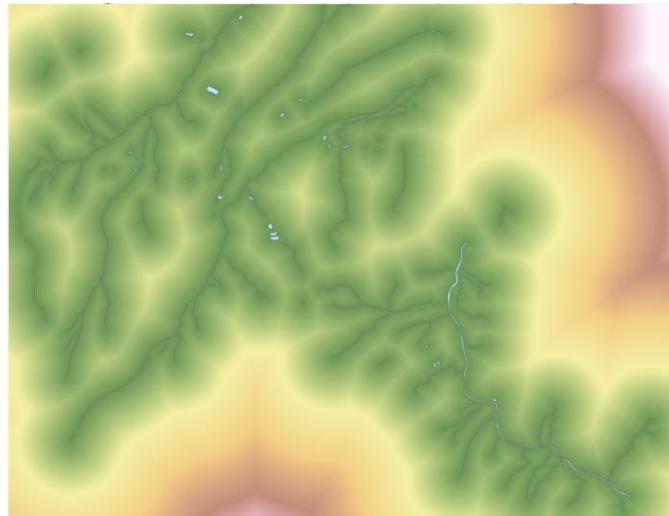


## Kryterium 4

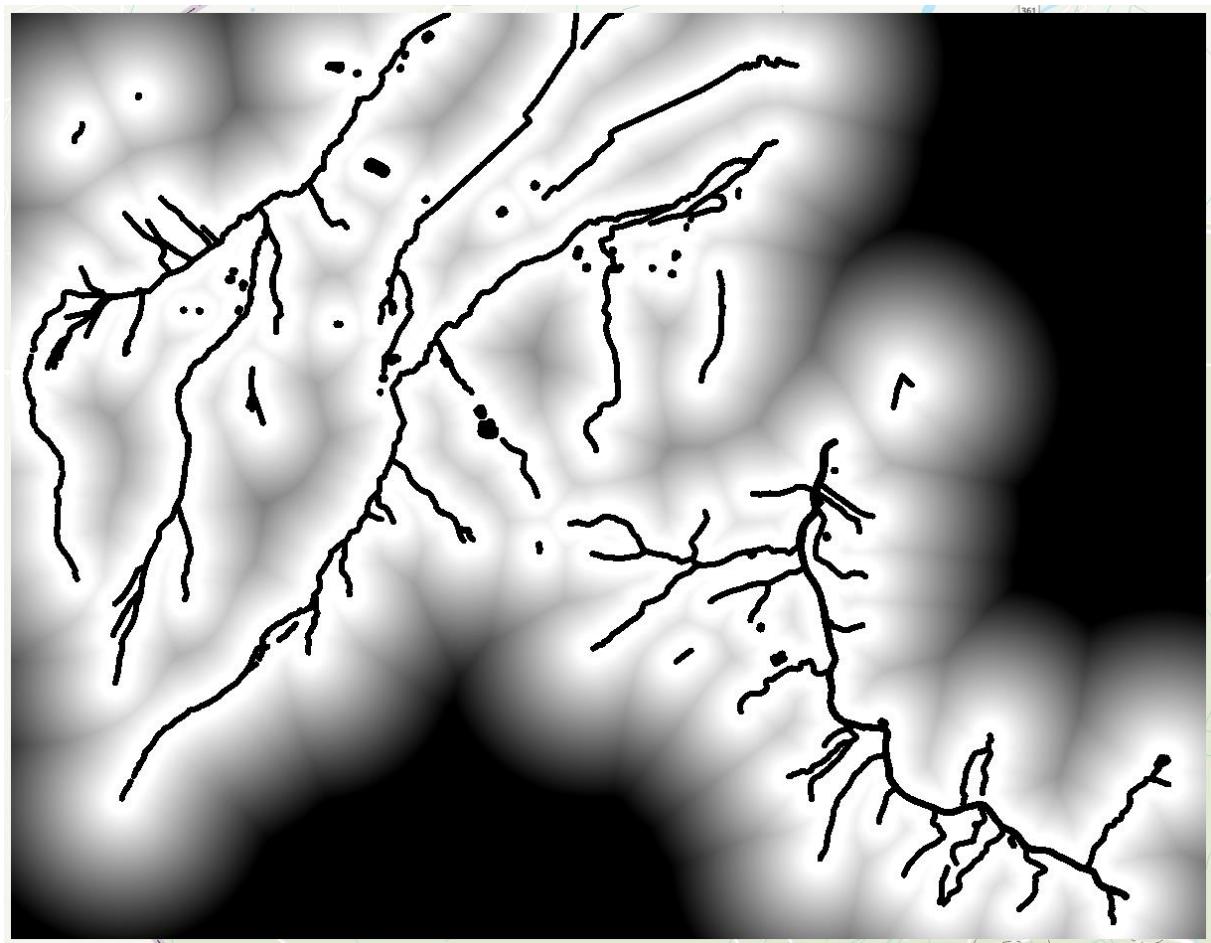
### Kryterium 4 - odległość od rzek i zbiorników wodnych nieprzekraczalna strefa ochronna poniżej 20m

```
query = "X_KOD='SWRS01' OR X_KOD='SWRS02'"  
rzeki = arcpy.analysis.Select(bdot['rzeki'], 'rzeki', query)  
bufor_rzeki = arcpy.analysis.Buffer(rzeki, 'bufor_rzeki', 0.5)  
jeziora = bdot['jeziora']  
wody = arcpy.management.Merge([bufor_rzeki, jeziora], 'wody')  
ed = arcpy.sa.EucDistance(wody)  
minimum = 20 # 20  
maximum = 20.001 # 25 20.001  
fuzzy1 = arcpy.sa.FuzzyMembership(ed, arcpy.sa.FuzzyLinear(minimum, maximum))  
minimum = 1000 # 1000  
maximum = 100 # 100  
fuzzy2 = arcpy.sa.FuzzyMembership(ed, arcpy.sa.FuzzyLinear(minimum, maximum))  
  
k4 = arcpy.sa.Times(fuzzy1, fuzzy2)
```

Kryterium 4 polegało na wyznaczeniu terenów blisko rzek i zbiorników wodnych, jednak z zachowaniem dwudziestometrowej strefy ochronnej. W celu uzyskania geometrii rzek, wybrałem odpowiednie obiekty po atrybutie X\_KOD z warstwy liniowej SWRS z BDOT. Jeziora i zbiorniki wodne wziąłem z warstwy PTWP z BDOT. Następnie utworzyłem półmetrowy bufor dla rzek, żeby obydwie warstwy miały ten sam typ geometrii. Przed przystąpieniem do użycia narzędzi logiki rozmytej, połączylem obydwie warstwy funkcją Merge. Uznałem, że im bliżej wody tym lepiej, a odległości w przedziale 20-100 metrów są tak samo dobrymi. Od 100 metrów do 1 kilometra funkcja przydatności liniowo maleje i po 1000 metrach przyjmuje wartości zerowe. Wartości zerowe przyjmuje też dla strefy ochronnej (0-20 metrów). Można tutaj było zrobić raster zero-jedynkowy dla strefy ochronnej i połączyć go poprzez mnożenie z wynikiem FuzzyMembership dla wartości 1000 i 100, jednak zamiast tego użyłem pewnego „tricku”, a mianowicie; tak jak wcześniej zrobiłem dwa rastery FuzzyMembership, z tym że wartości minimum i maksimum podawane do jednej z nich są bardzo blisko siebie (20 i 20.001) i wystarczająco dobrze aproksymują linię pionową, która powstałaby przy użyciu sposobu z rastrem boolowskim. Łączenie wyników wykonałem analogicznie jak przy wcześniej opisanych użyciach funkcji FuzzyMembership i Times. Połączone warstwy z wodami i odległości euklidesowe:



Wynik końcowy kryterium 4.

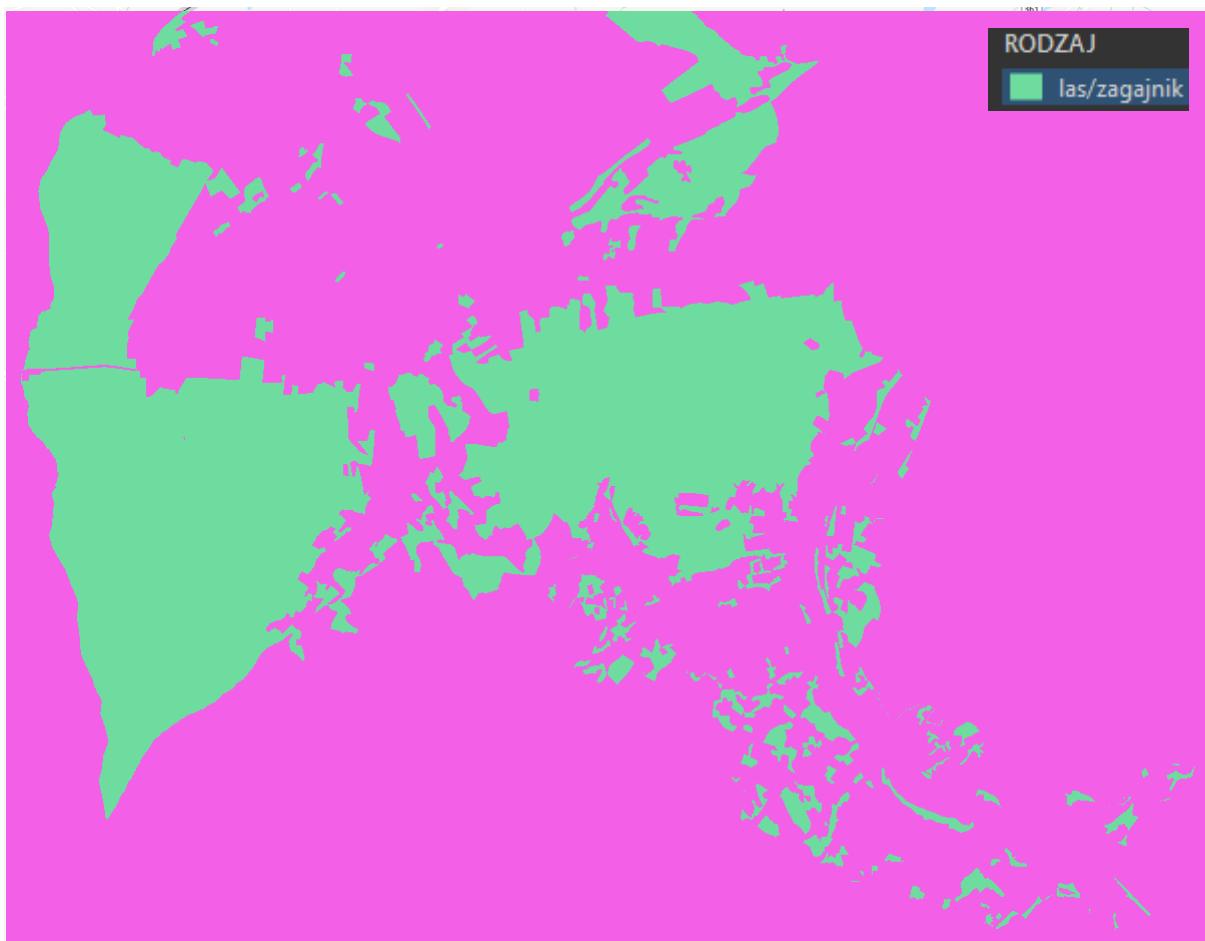


## Kryterium 5

### Kryterium 5 - pokrycie terenu nie w lesie

```
lasy_raster = arcpy.conversion.FeatureToRaster(bdot['lasy'], "RODZAJ", 'lasy_raster')
k5 = arcpy.sa.Reclassify(lasy_raster, "RODZAJ", arcpy.sa.RemapValue([["las", 0], ["zadrzewienie", 1], ["zagajnik", 0], ["NODATA", 1]]))
```

Kryterium 5 polegało na wybraniu terenów nie znajdujących się w lesie – ze względów prawnych nie można budować w lesie (lub też nie ma sensu przechodzić przez wszystkie etapy pozyskania działki leśnej i potem jej zmiany na budowlaną, kiedy mamy możliwości zakupu działek budowlanych bez lasu na ich obszarze). Zastosowałem tutaj logikę ostrą; tereny znajdujące się na obszarze lasów i zagajników otrzymały wartości 0 oznaczające nieprzydatność terenu, a wszystkie inne tereny oznaczyłem wartością 1, oznaczającą całkowitą przydatność. Warstwę z lasami otrzymałem z BDOT, była to warstwa poligonalna o nazwie PTLZ. Przerobiłem wektor na warstwę rastrową z zachowaniem atrybutu RO DZAJ (funkcja FeatureToRaster), na podstawie którego dokonałem reklasyfikacji wartości rastra z użyciem funkcji Reclassify. Wynik prezentuje się następująco.



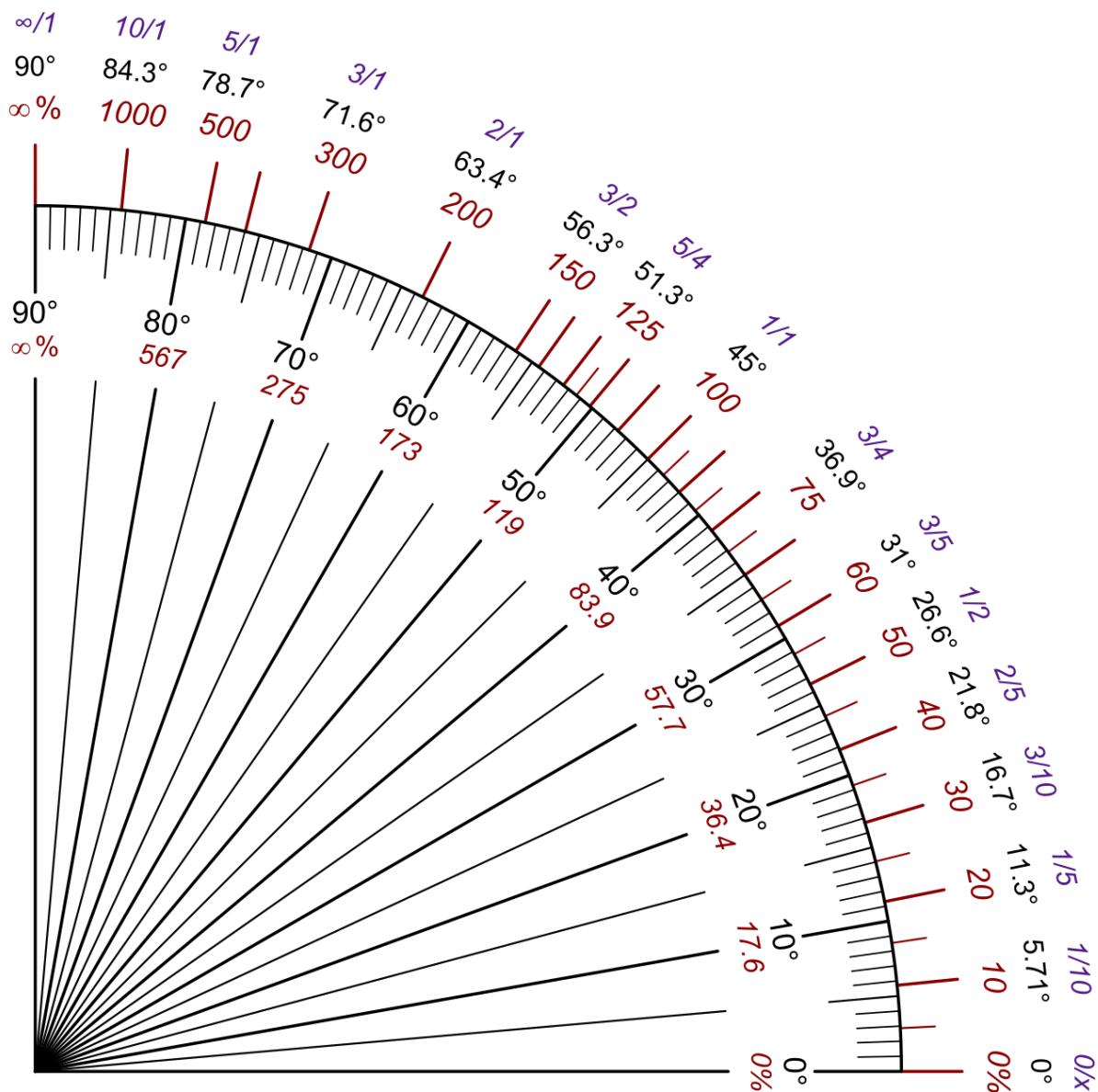
## Kryterium 6

### Kryterium 6 - nachylenie stoków maksymalnie 20%

```
slope_perc = arcpy.sa.Slope(path_nmt, "PERCENT_RISE", 1, "PLANAR", "METER")
minimum = 20 # 20
maximum = 5 # 5

k6 = arcpy.sa.FuzzyMembership(slope_perc, arcpy.sa.FuzzyLinear(minimum, maximum))
```

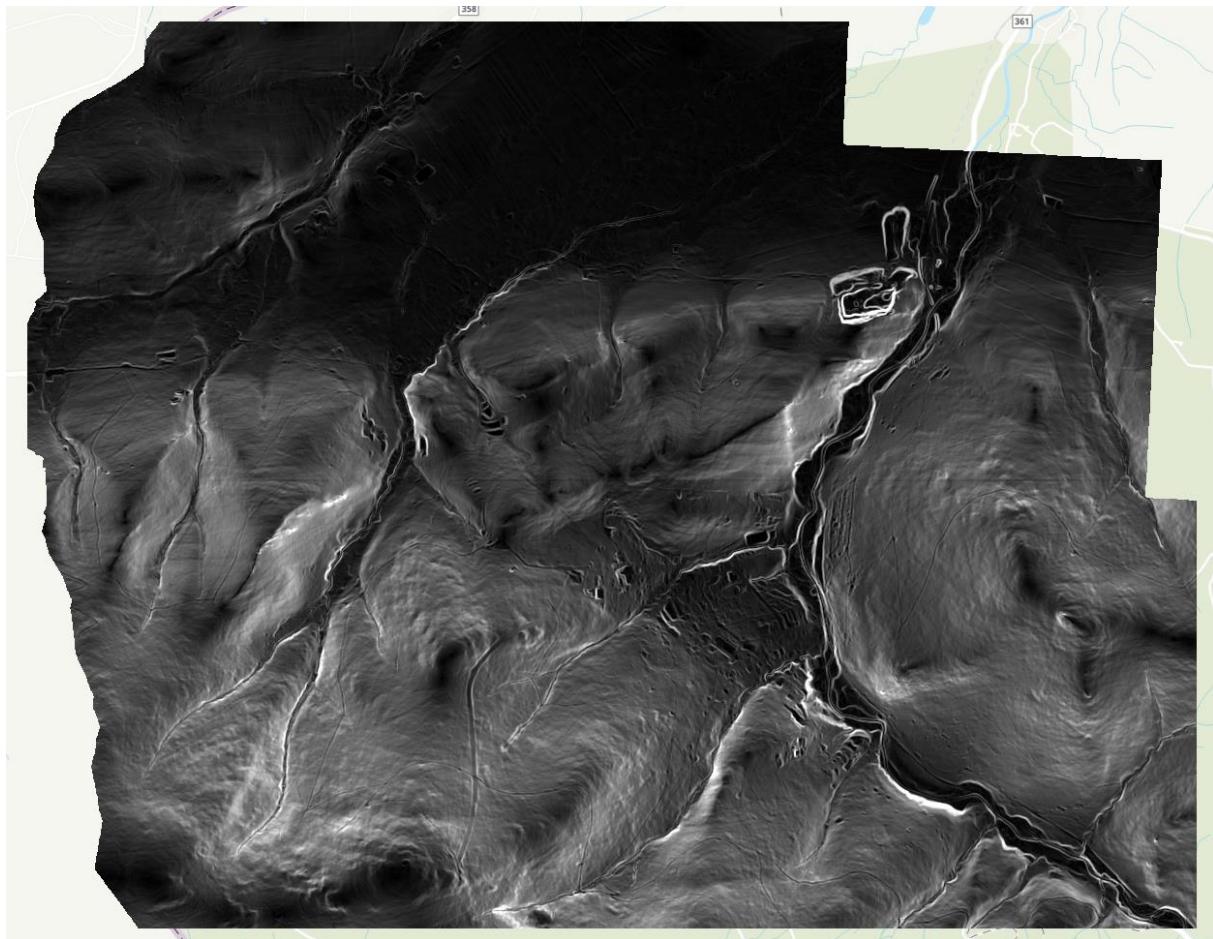
Kryterium 6 polegało na wybraniu terenów położonych na nie zbyt stromych stokach – nachylenie stoków maksymalnie 20%, co oznacza 2 metry przewyższenia na każde 10 metrów odległości poziomej. Załączam rysunek.



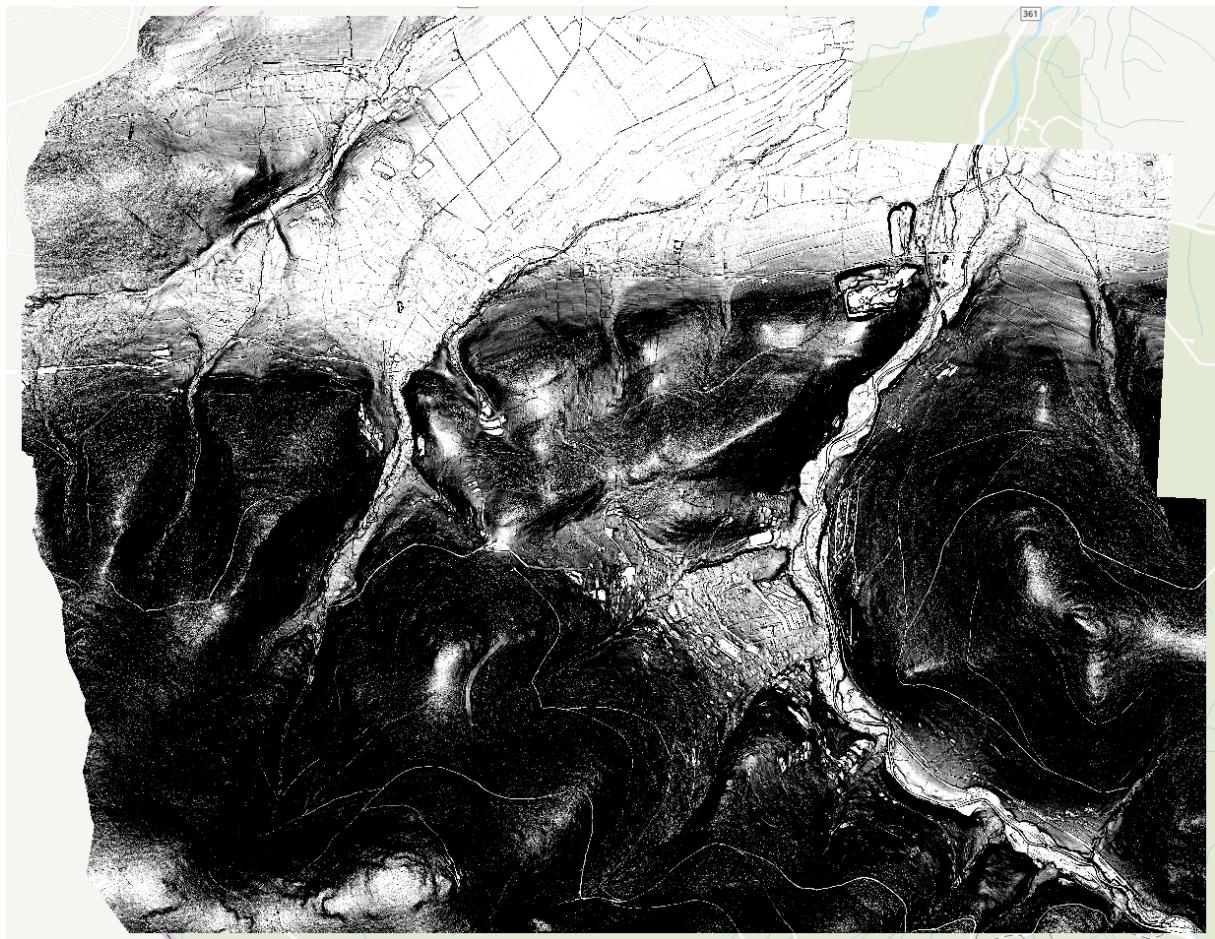
W celu wygenerowania rastra z wartościami nachylenia skorzystałem z funkcji Slope, a następnie w celu reklasyfikacji wartości do pożądanego przedziału 0-1,

z funkcji FuzzyMembership, gdzie wybrałem wartości 20 i 5 jako graniczne. Wszystko poniżej nachylenia równego 5% jest traktowane jako tereny tak samo przydatne, funkcja przydatności liniowo rośnie od wartości nachylenia 5 do 20. Odrzuciłem tereny powyżej 20% nachylenia – uznałem je za nieprzydatne. Daną wejściową do funkcji Slope był raster z numerycznym modelem terenu dla odpowiedniego obszaru. Chyba nie muszę opisywać sensu tego kryterium... Wiadomo, że nie będziemy budować hotelu na stromym zboczu, bo albo się nie da, albo koszty wyrównania terenu byłyby zbyt wysokie, poza tym otoczenie nadal byłoby strome i dotarcie do naszego hotelu byłoby utrudnione.

Wynik funkcji Slope.



Wynik końcowy kryterium 6.



## Kryterium 7

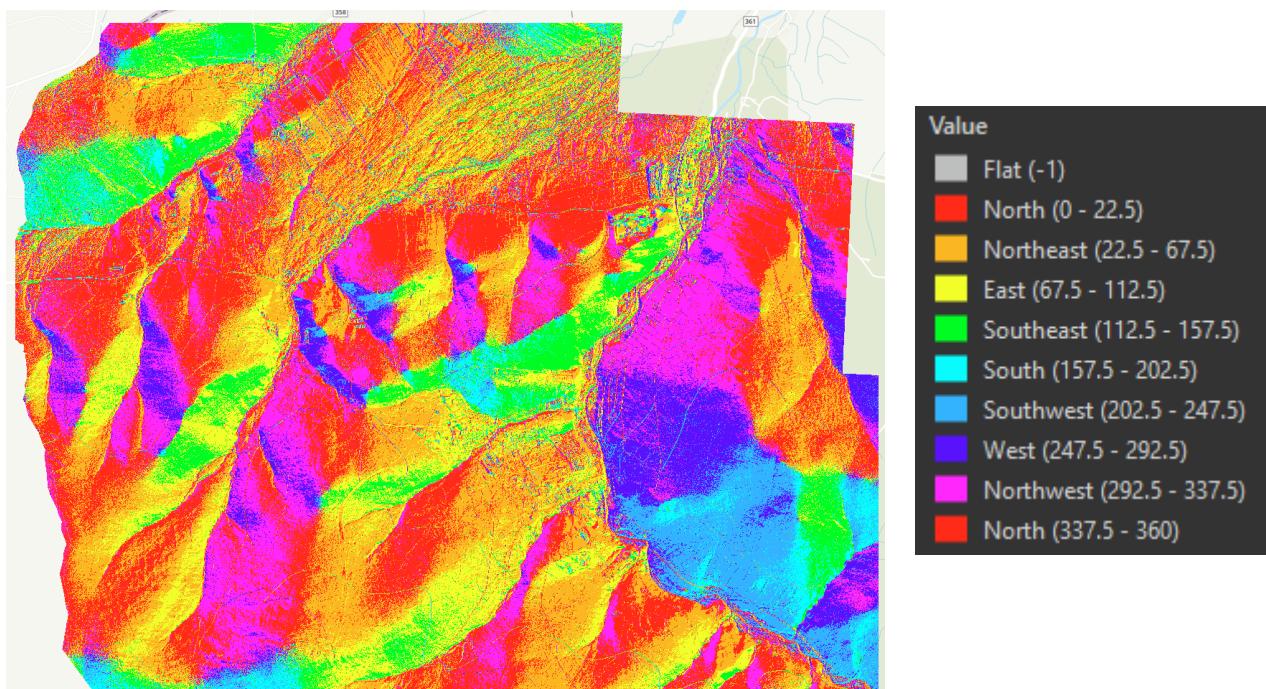
### Kryterium 7 - dostęp światła słonecznego stoki południowe (SW-SE)

```
aspect = arcpy.sa.Aspect(path_nmt)
minimum = 110 # 110
maximum = 155 # 155
fuzzy1 = arcpy.sa.FuzzyMembership(aspect, arcpy.sa.FuzzyLinear(minimum, maximum))
minimum = 250 # 250
maximum = 205 # 205
fuzzy2 = arcpy.sa.FuzzyMembership(aspect, arcpy.sa.FuzzyLinear(minimum, maximum))

k7_1 = arcpy.sa.Times(fuzzy1, fuzzy2)
k7 = arcpy.sa.Con(aspect, 1, k7_1, "VALUE = -1")
```

Kryterium 7 polegało na wybraniu terenów z dostępem do światła słonecznego, czyli głównie stoków południowych (ponieważ obszar analizy znajduje się na północnej półkuli). Dane wejściowe to model NMT (ten sam, co w poprzednim kryterium). Do wygenerowania rastra z kierunkiem nachylenia stoku użyłem funkcji Aspect. Zastosowano tu pewne mapowanie wartości - stoki zwrócone „twarzą” ku północy (stoki północne) przyjmują wartość zero (i wartości około zera na kole azymutów). Azymut/wartość równa 180 oznacza stok zwrocony idealnie w stronę południową itd.... Ponownie użyłem narzędzi fuzzy logic. Przyjąłem, że wartości odpowiadające azymutom w przedziale 155-205 stopni (a więc stoki południowo-wschodnie, południowe i południowo-zachodnie) są tak samo dobre i najlepsze. Zwracając się „ku północy” w zakresie 45 stopni od każdej z tych wartości, funkcja przydatności liniowo maleje, wszędzie indziej osiąga wartości równe zero. Po złączeniu rastrów funkcją Times, dołączylem wartości -1 z funkcji Aspect odpowiadające terenom idealnie płaskim.

Wynik funkcji Aspect.



## Wynik końcowy kryterium 7.



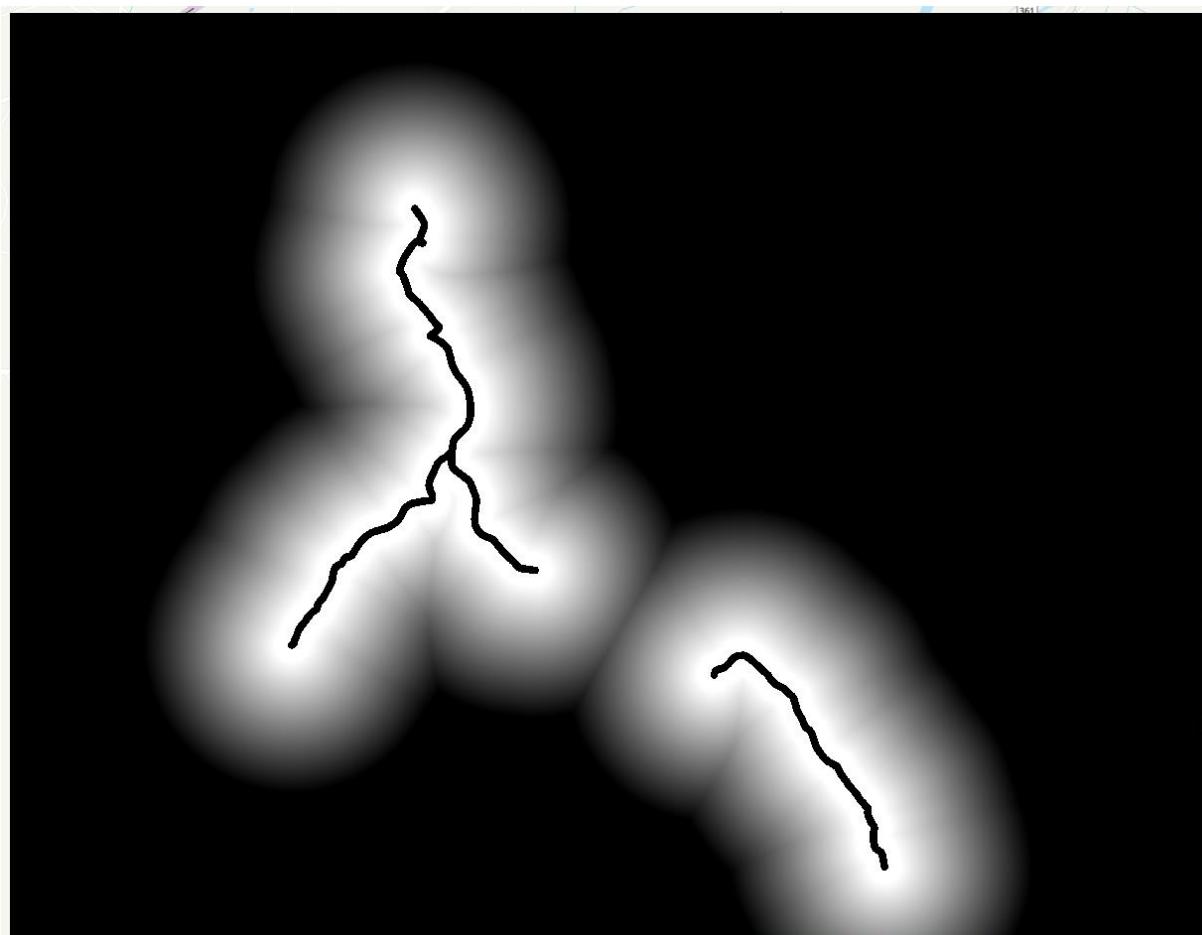
## Kryterium 8

### Kryterium 8 - poza strefą ochronną gazociągu minimum 25m

```
ed = arcpy.sa.EucDistance(path_gazociag)
minimum = 25 # 25
maximum = 25.001 # 26 25.001
fuzzy1 = arcpy.sa.FuzzyMembership(ed, arcpy.sa.FuzzyLinear(minimum, maximum))
minimum = 1000 # 1000
maximum = 100 # 100
fuzzy2 = arcpy.sa.FuzzyMembership(ed, arcpy.sa.FuzzyLinear(minimum, maximum))

k8 = arcpy.sa.Times(fuzzy1, fuzzy2)
```

Kryterium 8 polegało na wyznaczeniu terenów wystarczająco blisko gazociągu (wymyślony gazociąg, ponieważ dane prawdziwe nie są udostępniane jako wektor) tak, by nie płacić zbyt dużo za budowę dołącza, ale z zachowaniem strefy ochronnej wynoszącej 25 metrów. Zrobiłem to analogicznie jak w kryterium 4 ze zbiornikami wodnymi. Przyjąłem wartości odległości pomiędzy 25 i 100 metrów jako optymalne, powyżej 1000m jako nieprzydatne, a pomiędzy 100 a 1000 – liniowo zmieniające się od 1 do 0. Wynik końcowy prezentuje się następująco.



## Kryterium 9

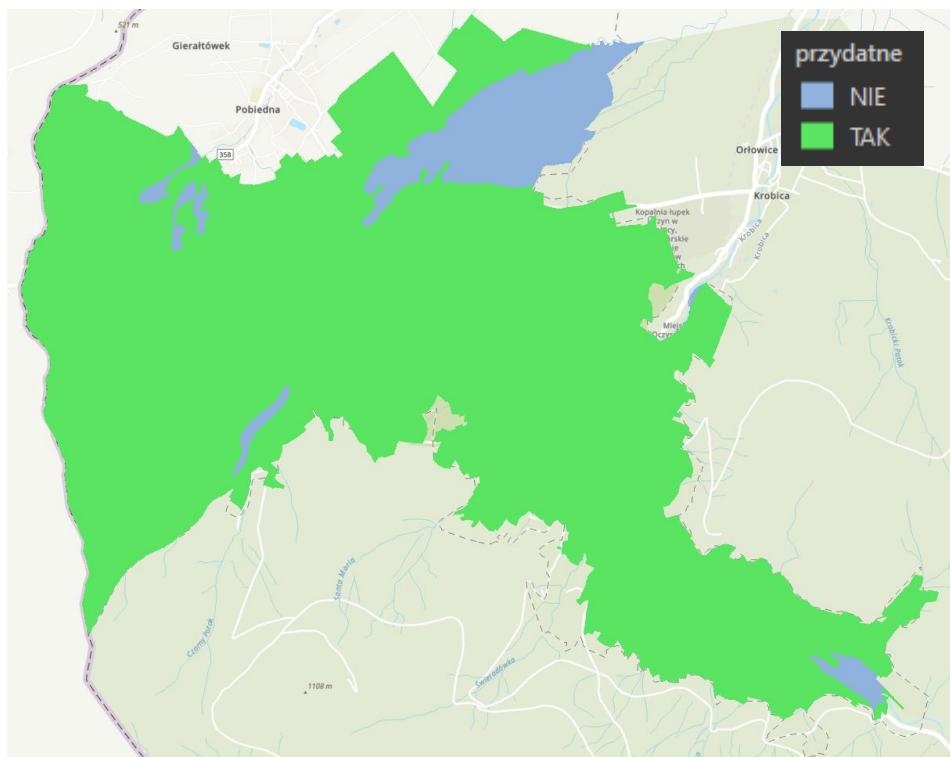
## Kryterium 9 - użytkowanie terenu - przydatne gleby

```
codeblock = """
def is_useful(string):
    if string in ['G', 'E', 'M', 'T', 'F', 'Fb', 'Fc', 'FG']:
        return "NIE" # 0
    return "TAK" # 1
"""

expression = "is_useful(!TYP!)"
field_name = "przydatne"
gleby = arcpy.management.CalculateField(path_gleby, field_name, expression,"PYTHON3", codeblock)

k9 = arcpy.conversion.FeatureToRaster(gleby, "przydatne", "k9")
k9 = arcpy.sa.Reclassify(k9, "przydatne", arcpy.sa.RemapValue([["NIE", 0], ["TAK", 1]]))
```

Kryterium 9 polegało na wyznaczeniu terenów przydatnych pod względem typu gleby. Dane pochodziły z bazy danych glebowo-rolniczych. Uznaлиsmy, że gleby przydatne pod inwestycję hotelową to wszystkie inne niż: glejowe, mułowo-torfowe i torfowo-mułowe, murszowo-mineralne i murszowate, torfowe i murszowo-torfowe, mady, mady brunatne, mady czarnoziemne, aluwialne glejowe. W warstwie z danymi był atrybut TYP, na podstawie którego stworzyłem nowy atrybut określający przydatność gleby, z użyciem funkcji CalculateField. Następnie przekonwertowałem wektor na raster oraz zrobiłem reklasyfikację względem nowo utworzonego atrybutu. W wyniku tego otrzymałem raster boolewski określający przydatność terenu pod względem typu gleby. Wynik końcowy przedstawia się następująco.



## Łączenie kryteriów

```
Łączenie kryteriów

Zrobienie kryteriów ostrzych ze stref ochronnych wodociągu i wód

# 2 + 2 = 4 (5) ostrzych (las, gleby i dodatkowo strefy ochronne wód i gazociągu)
# Weighted sum do fuzzy 7/8 k1 k2 k3 k4 k6 k7 k8
# Fuzzy overlay do ostrzych 4/5 k5 k9 k4_ostre k8_ostre
# i potem mnożymy funkcją Times
# Lub Raster Calculator
# dwa scenariusze - jednakowe wagie i wagie eksperckie

ed = arcpy.sa.EucDistance(wody)
k4_ostre = arcpy.sa.Con(ed, 1, 0, "VALUE > 20")

ed = arcpy.sa.EucDistance(path_gazociag)
k8_ostre = arcpy.sa.Con(ed, 1, 0, "VALUE > 25")

Scenariusz 1: wagie jednakowe

wstable_1 = arcpy.sa.WSTable([[k1,"VALUE",0.1428],[k2,"VALUE",0.1428],[k3,"VALUE",0.1428],[k4,"VALUE",0.1428],[k6,"VALUE",0.1428],[k7,"VALUE",0.1428],[k8,"VALUE",0.1432]])
joined_fuzzy_1 = arcpy.sa.WeightedSum(wstable_1)

fo_ostre = arcpy.sa.FuzzyOverlay([k5, k9, k4_ostre, k8_ostre], "AND") # "AND"

final_fuzzy_1 = arcpy.sa.Times(joined_fuzzy_1, fo_ostre)

ff1_reclass = arcpy.sa.Con(final_fuzzy_1, 1, 0, "VALUE > 0.5") # VALUE > 0.5

Scenariusz 2: wagie eksperckie

• Kryterium 1 - 0.2 - odległość od konkurencji minimum 400m
• Kryterium 2 - 0.05 - odległość od budynków mieszkalnych od 25 do 150m
• Kryterium 3 - 0.05 - odległość od istniejących dróg od 15 do 100m
• Kryterium 4 - 0.2 - odległość od rzek i zbiorników wodnych
• Kryterium 6 - 0.15 - nachylenie stoków maksymalnie 20%
• Kryterium 7 - 0.3 - dostęp światła słonecznego stoki południowe (SW-SE)
• Kryterium 8 - 0.05 - odległość od gazociągu

wstable_2 = arcpy.sa.WSTable([[k1,"VALUE",0.2],[k2,"VALUE",0.05],[k3,"VALUE",0.05],[k4,"VALUE",0.2],[k6,"VALUE",0.15],[k7,"VALUE",0.3],[k8,"VALUE",0.05]])
joined_fuzzy_2 = arcpy.sa.WeightedSum(wstable_2)

fo_ostre = arcpy.sa.FuzzyOverlay([k5, k9, k4_ostre, k8_ostre], "AND") # "AND"

final_fuzzy_2 = arcpy.sa.Times(joined_fuzzy_2, fo_ostre)

ff2_reclass = arcpy.sa.Con(final_fuzzy_2, 1, 0, "VALUE > 0.5") # VALUE > 0.5
```

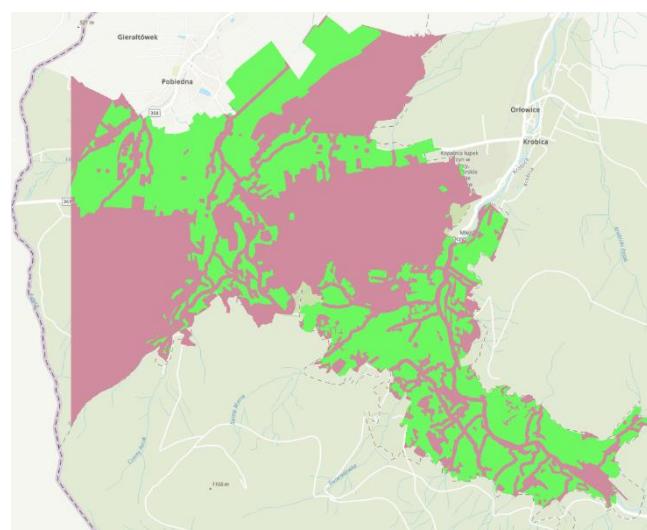
Po uzyskaniu wyników wszystkich opisanych do tej pory kryteriów, trzeba było je jakoś ze sobą połączyć. Wszystkie rastery boolowskie będące wynikami kryteriów można pomnożyć, uzyskując swego rodzaju maskę przydatności terenu. Służy do tego funkcja FuzzyOverlay. Przy łączaniu kryteriów ostrzych trzeba było pamiętać o strefach ochronnych wodociągu i wód, do zrobienia ich użyłem funkcji EucDistance i Con. Natomiast dla warstw rozmytych, sposób łączenia jest inny. Możemy każdej warstwie przypisać jakąś wagę, która będzie określała jak bardzo dane kryterium jest ważne w porównaniu do innych. Trzeba pamiętać, by wszystkie wagie sumowały się do liczby 1. Wartości na rastrach są przemnażane przez daną wagę i następnie sumowane. W wyniku tego procesu powstaje jedna warstwa zawierająca w sobie część informacji z każdej warstwy rozmytej. Służy do tego funkcja WeightedSum. Wyniki funkcji FuzzyOverlay i WeightedSum należy połączyć poprzez mnożenie w celu uzyskania finalnej warstwy rastrowej określającej dla każdego piksela przydatność terenu pod inwestycję hotelarską. Na koniec można ewentualnie wyodrębnić część finalnej warstwy przydatności, poprzez reklasyfikację wartości, np. wybrać tereny przydatne jako te, gdzie wartości na finalnej warstwie są powyżej jakiejś ustalonej wartości progowej. U mnie ta wartość wynosi 0.5, ponieważ dla

wyższych wartości uzyskiwałem stosunkowo mało przydatnych terenów. Można też zrobić analizę wszystkich pikseli rastra i uszeregować wartości rosnąco, po czym przyjąć top 20% pikseli jako przydatne. Wymagałoby to napisania dodatkowego kodu.

Jeśli chodzi o wagę dla kryteriów rozmytych, to od tego momentu skrypt rozwidla się na dwie ścieżki. Pierwszy scenariusz to wagи jednakowe (lub prawie jednakowe – sześć wag równych 0.1428 i jedna równa 0.1432). Drugi scenariusz, to tzw. „wagi eksperckie”, ustalone według mojego uznania.

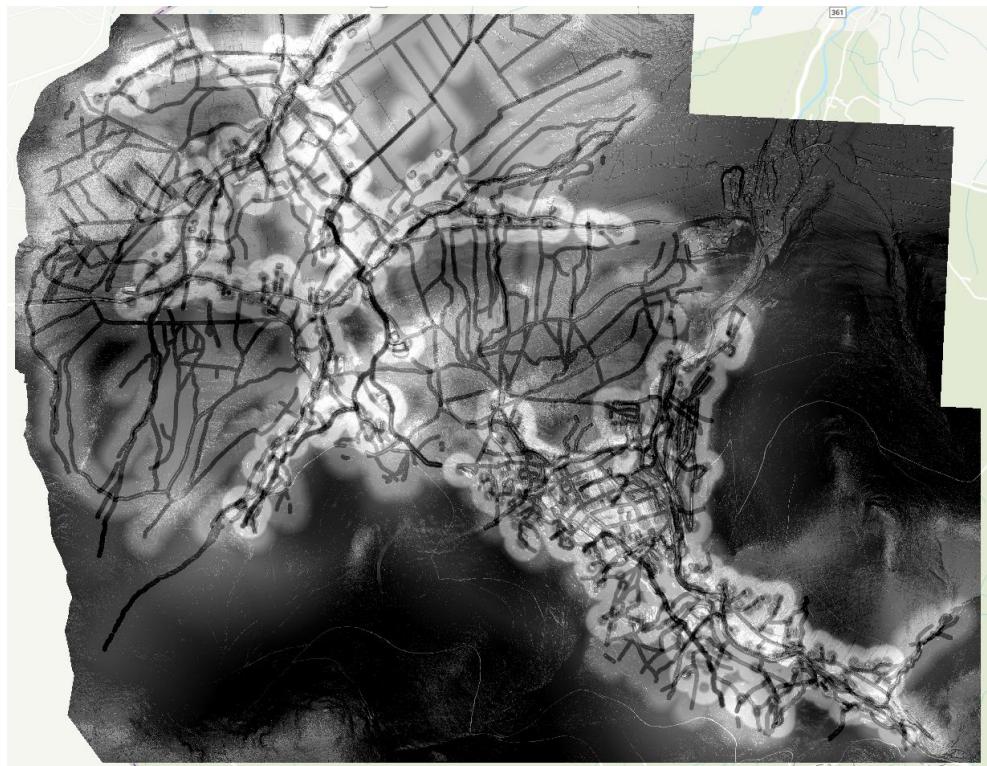
## **Scenariusz 2: wagi eksperckie**

- Kryterium 1 - 0.2 - odległość od konkurencji minimum 400m
  - Kryterium 2 - 0.05 - odległość od budynków mieszkalnych od 25 do 150m
  - Kryterium 3 - 0.05 - odległość od istniejących dróg od 15 do 100m
  - Kryterium 4 - 0.2 - odległość od rzek i zbiorników wodnych
  - Kryterium 6 - 0.15 - nachylenie stoków maksymalnie 20%
  - Kryterium 7 - 0.3 - dostęp światła słonecznego stoki południowe (SW-SE)
  - Kryterium 8 - 0.05 - odległość od gazociągu

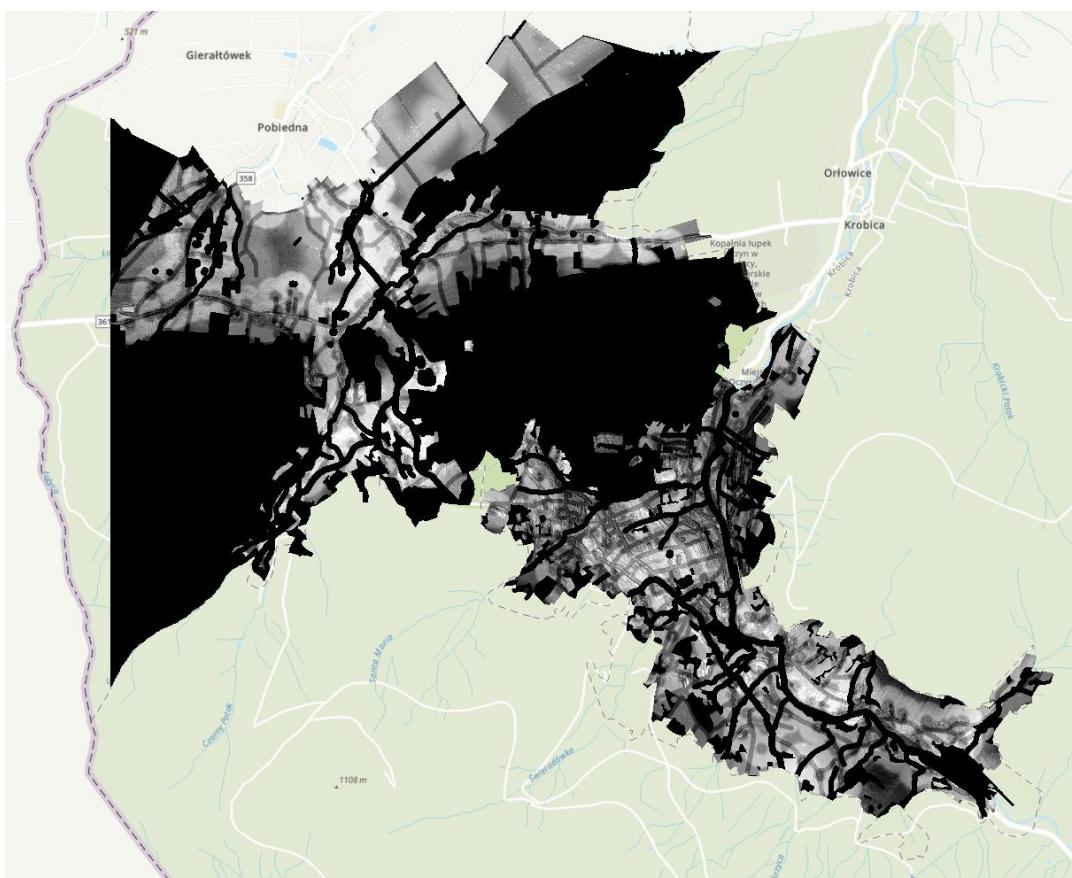


## Łączenie kryteriów – Scenariusz 1: wagi jednakowe

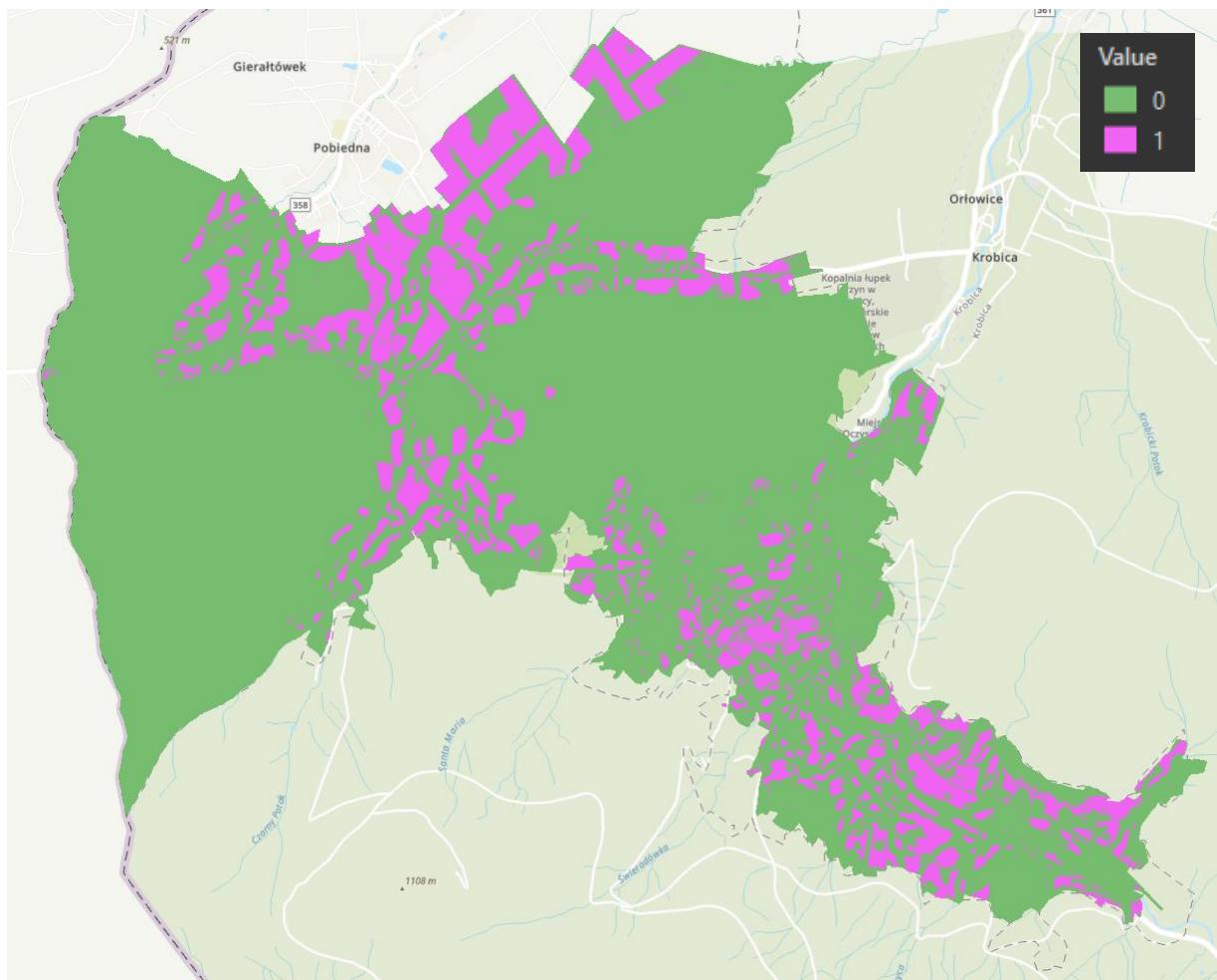
Wynik połączenia kryteriów rozmytych.



Wynik połączenia kryteriów rozmytych i ostrych.

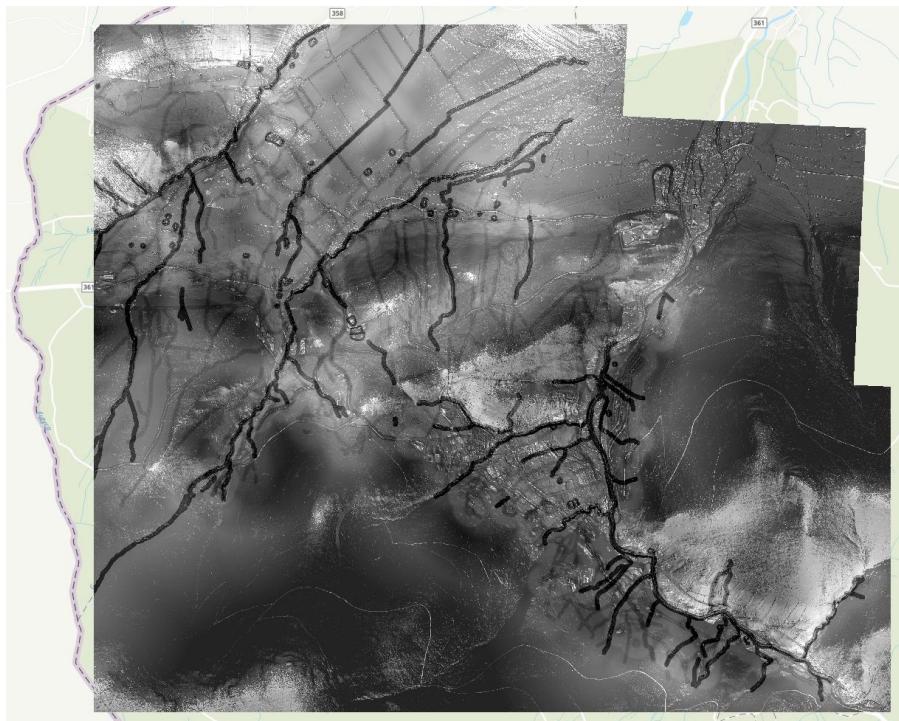


Wynik wybrania terenów powyżej progowej wartości przydatności (0.5).

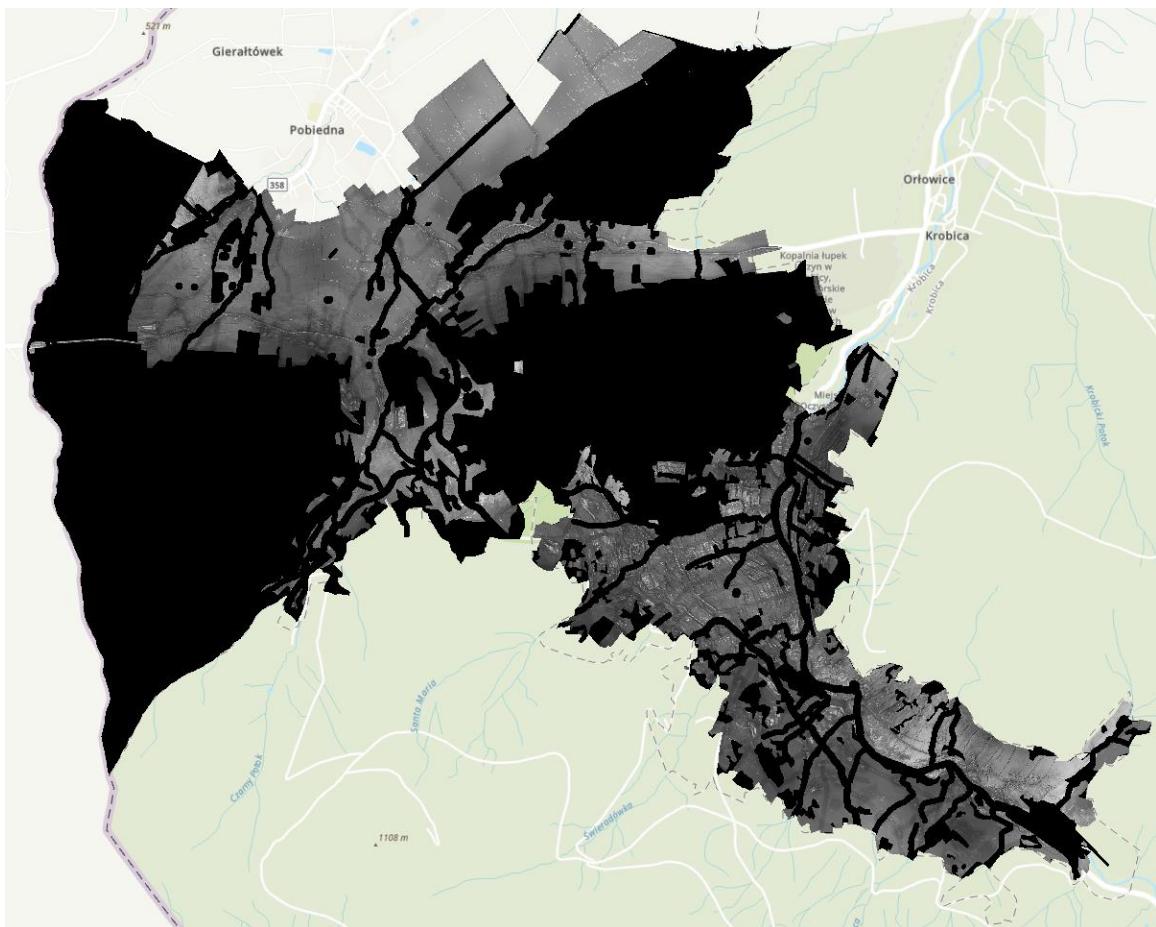


## Łączenie kryteriów – Scenariusz 2: wagi eksperckie

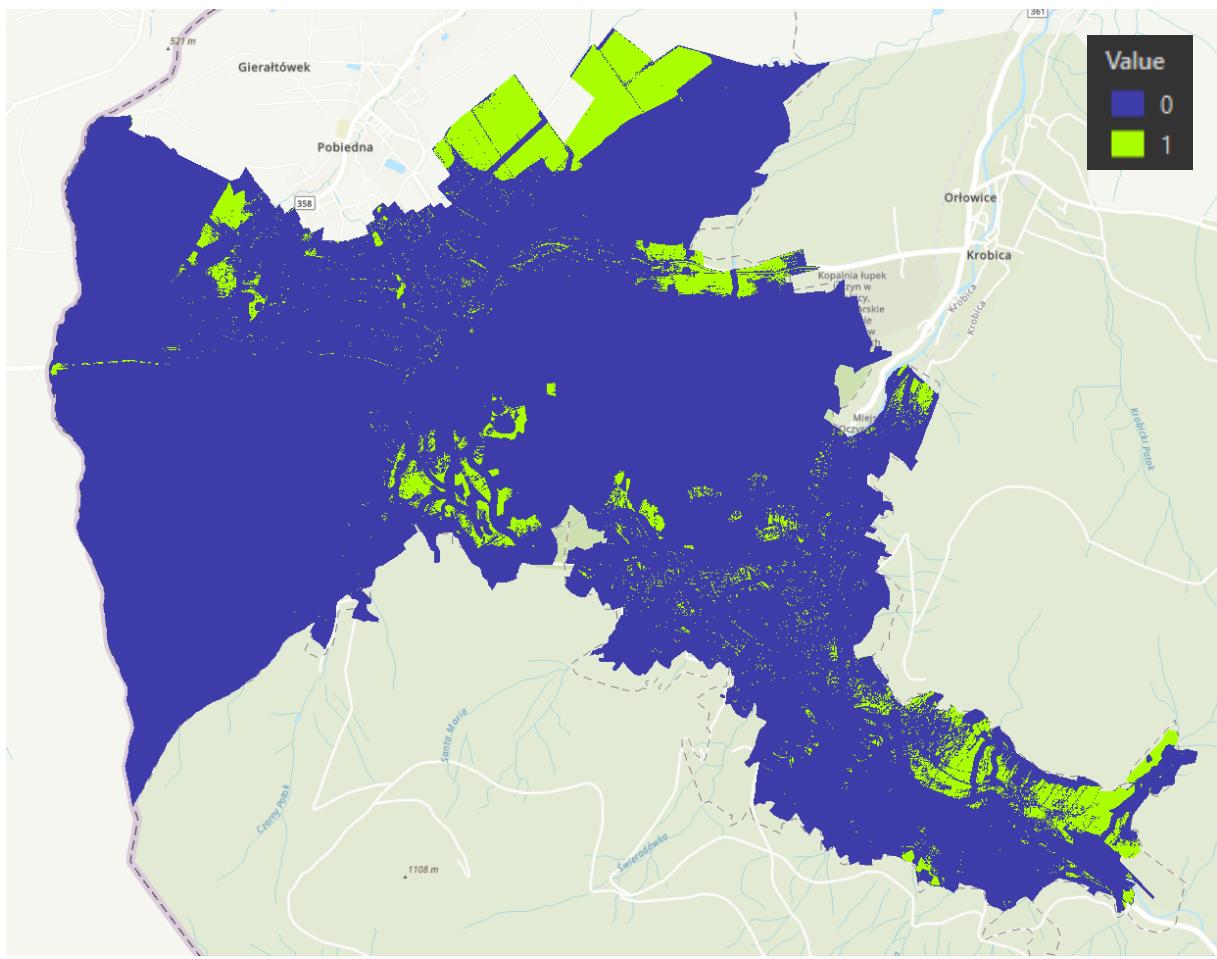
Wynik połączenia kryteriów rozmytych.



Wynik połączenia kryteriów rozmytych i ostrych.



Wynik wybrania terenów powyżej progowej wartości przydatności (0.5).



## Kryterium 10 – Scenariusz 1: wagi jednakowe

```
Scenariusz 1: wagi jednakowe

ff1_polygons = arcpy.conversion.RasterToPolygon("ff1_reclass", "ff1_polygons", "NO_SIMPLIFY", "Value", "SINGLE_OUTER_PART", None)

query = "gridcode = 1"
wybrane = arcpy.analysis.Select(ff1_polygons, "wybrane", query)

ff1_stats_table = arcpy.analysis.TabulateIntersection(path_dzialki, "ID_DZIAŁKI", wybrane, "ff1_stats_table", None, None, None, "UNKNOWN") # path_table

ff1_dzialki = arcpy.management.AddJoin(path_dzialki, "ID_DZIAŁKI", ff1_stats_table, "ID_DZIAŁKI", "KEEP_COMMON", "NO_INDEX_JOIN_FIELDS")

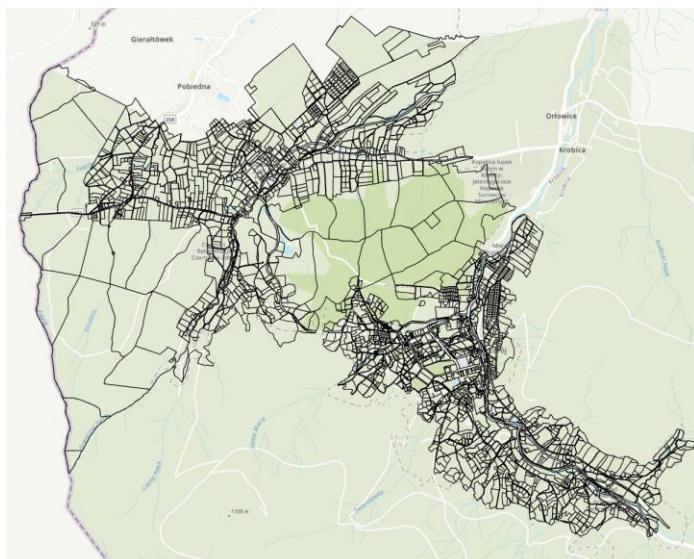
query = "PERCENTAGE > 70" # >70
ff1_dzialki_wybrane = arcpy.analysis.Select(ff1_dzialki, "ff1_dzialki_wybrane", query)

ff1_wybrane_dissolved = arcpy.management.Dissolve(ff1_dzialki_wybrane, "ff1_wybrane_dissolved", None, None, "SINGLE_PART", "DISSOLVE_LINES", '')

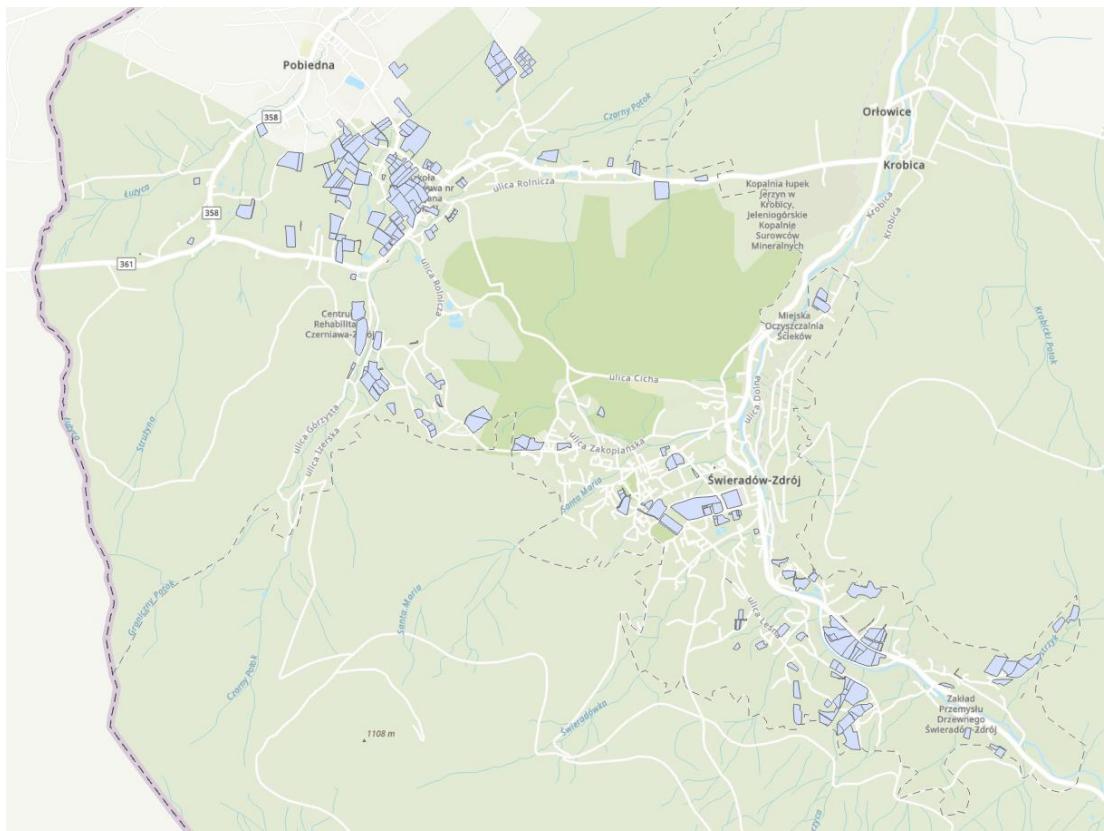
query = "Shape_Area > 10000" # powierzchnia > 1ha
ff1_dissolved_duze = arcpy.analysis.Select(ff1_wybrane_dissolved, "ff1_dissolved_duze", query)
```

Kryterium 10 polegało na wyborze przydatnych działek (lub obszarów złożonych z kilku przylegających działek), czyli obszarów o powierzchni większej niż 1 hektar. Dane potrzebne do tego etapu to warstwa poligonalna z działkami (dane pochodzą z EGiB) oraz rastrowe wyniki z poprzedniego etapu. Na początku zamieniłem rastrowe wyniki przydatności terenu na poligony funkcją RasterToPolygon, następnie wybrałem tylko przydatne poligony (odpowiadające wartości 1 na rastrze) funkcją Select. Następnie użyłem narzędzia TabulateIntersection, by uzyskać tabelę zawierającą id działki oraz powierzchnię zajętą i procentowe pokrycie tej działki przez poligony przydatne z drugiej warstwy. W kolejnym kroku dołączyciem atrybuty z tej tabeli do warstwy z działkami (funkcja AddJoin). Następnie wybrałem tylko te działki, które były pokryte w co najmniej 70% (taką wartość wybrałem) przez tereny przydatne (funkcja Select) oraz „rozpuściłem” granice między sąsiadującymi działkami funkcją Dissolve uzyskując poligony. Ostatnim krokiem w tym etapie było wybranie poligonów o powierzchni większej od 1ha za pomocą funkcji Select.

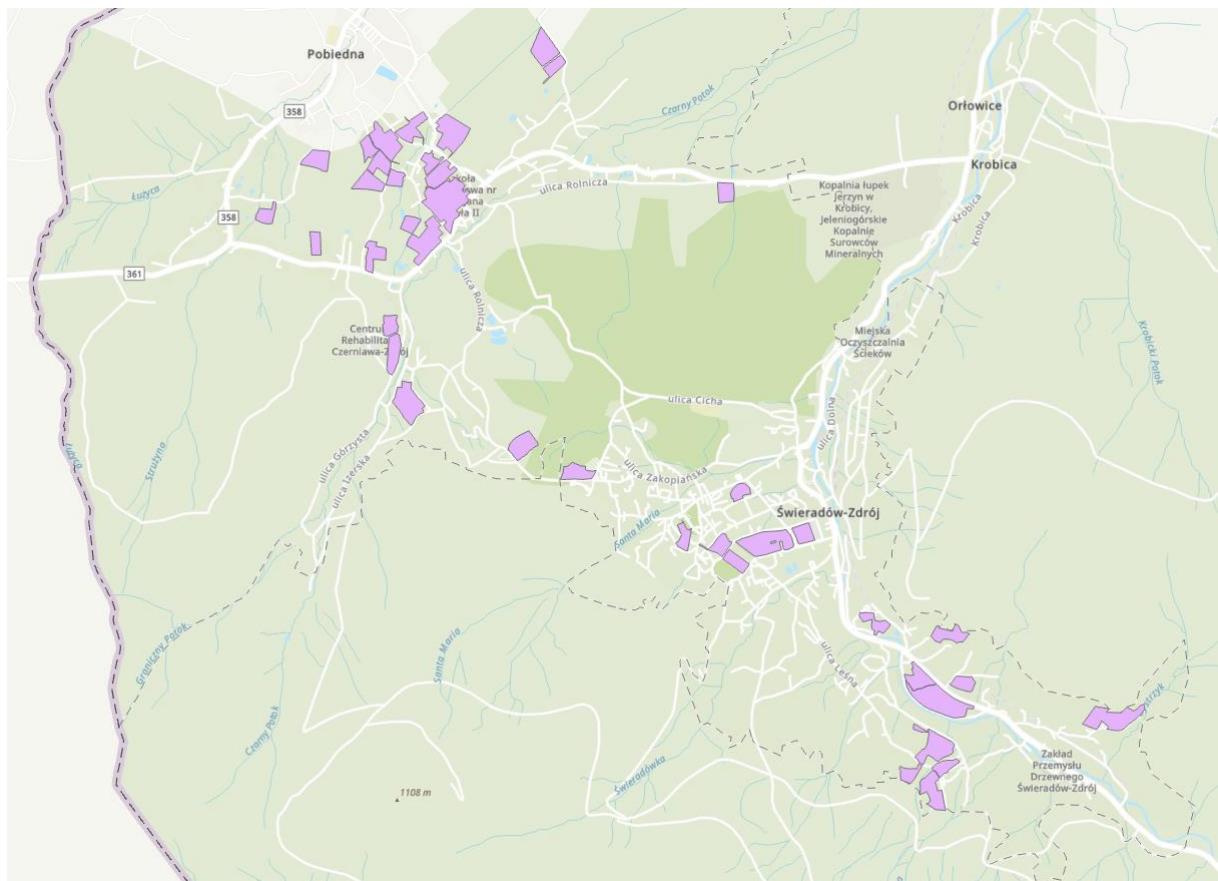
Działki ewidencyjne.



Działki pokryte terenem przydatnym w co najmniej 70%.



Obszary powstałe przez sklejenie przydatnych działek o dużej powierzchni.



## Kryterium 10 – Scenariusz 2: wagi eksperckie

```
Scenariusz 2: wagi eksperckie

ff2_polygons = arcpy.conversion.RasterToPolygon("ff2_reclass", "ff2_polygons", "NO_SIMPLIFY", "Value", "SINGLE_OUTER_PART", None)

query = "gridcode = 1"
wybrane = arcpy.analysis.Select(ff2_polygons, "wybrane", query)

ff2_stats_table = arcpy.analysis.TabulateIntersection(path_dzialki, "ID_DZIAŁKI", wybrane, "ff2_stats_table", None, None, None, "UNKNOWN") # pat

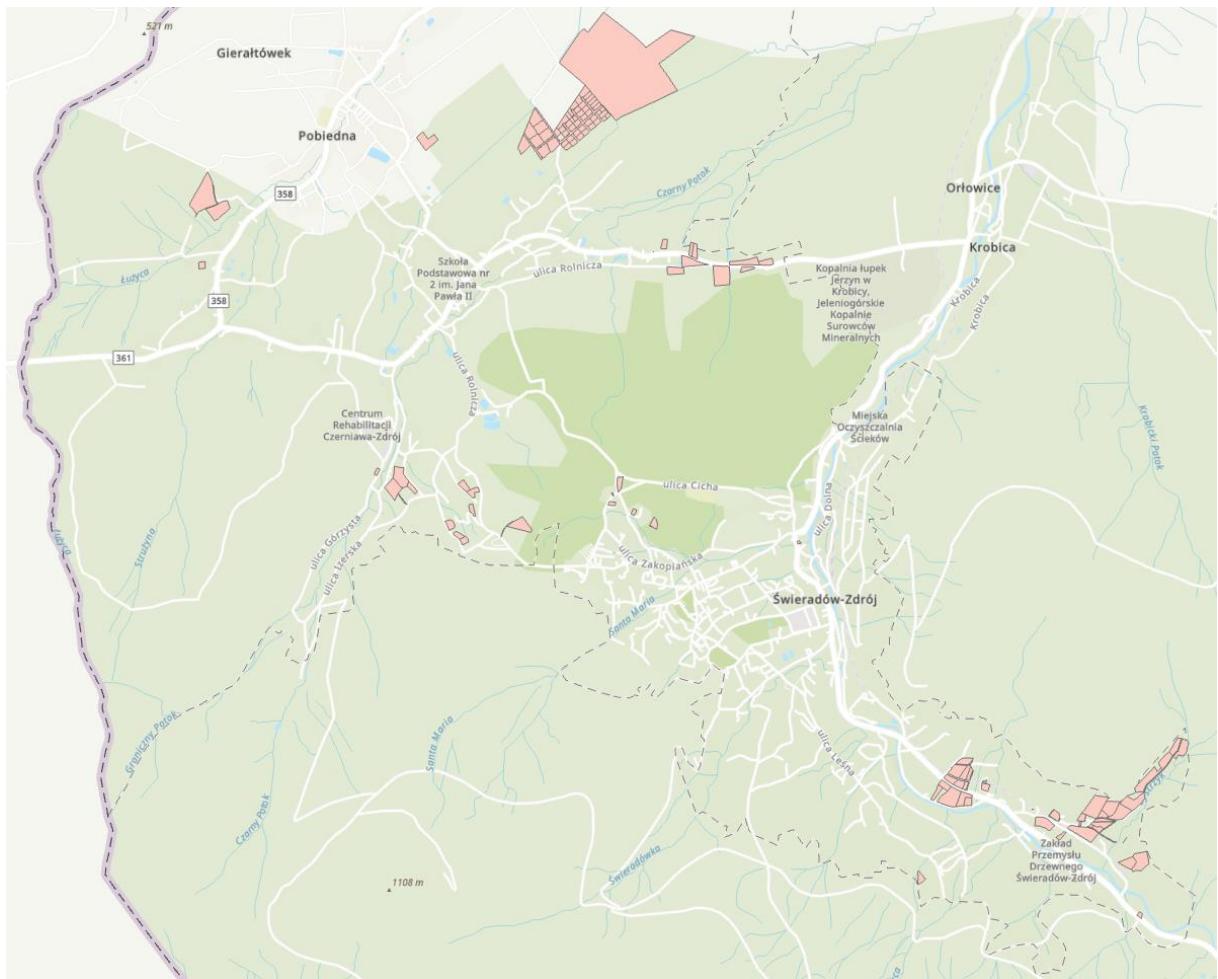
ff2_dzialki = arcpy.management.AddJoin(path_dzialki, "ID_DZIAŁKI", ff2_stats_table, "ID_DZIAŁKI", "KEEP_COMMON", "NO_INDEX_JOIN_FIELDS")

query = "PERCENTAGE > 70" # >70
ff2_dzialki_wybrane = arcpy.analysis.Select(ff2_dzialki, "ff2_dzialki_wybrane", query)

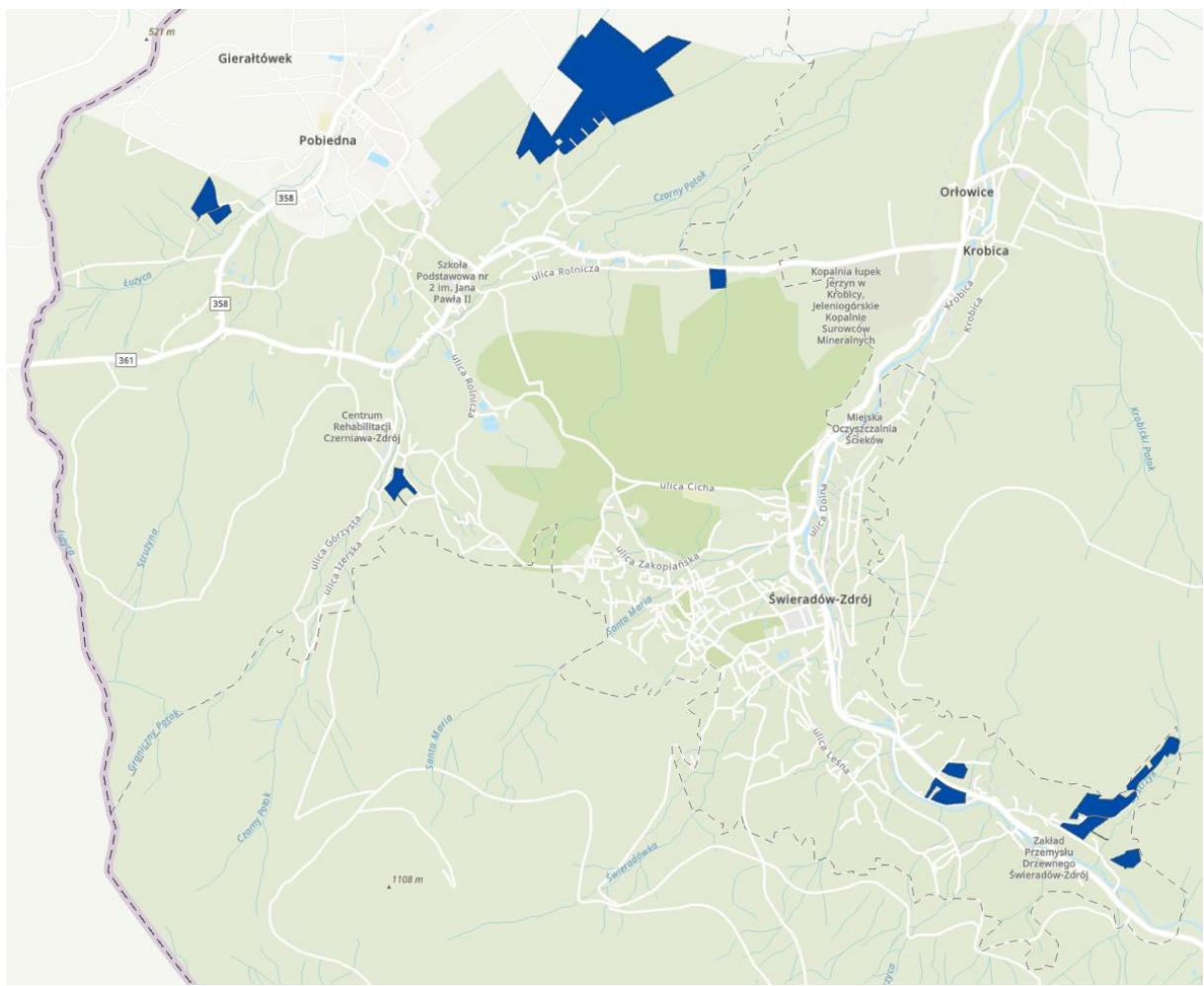
ff2_wybrane_dissolved = arcpy.management.Dissolve(ff2_dzialki_wybrane, "ff2_wybrane_dissolved", None, None, "SINGLE_PART", "DISSOLVE_LINES", '')

query = "Shape_Area > 10000" # powierzchnia > 1ha
ff2_dissolved_duze = arcpy.analysis.Select(ff2_wybrane_dissolved, "ff2_dissolved_duze", query)
```

Działki pokryte terenem przydatnym w co najmniej 70%.



Obszary powstałe przez sklejenie przydatnych działek o dużej powierzchni.



## Kryterium 11

### Kryterium 11 - ostateczne v1 - kształt obszaru jak najbardziej zwarty

#### Scenariusz 1: wagi jednakowe

C (compactness) = pierwiastek z pola obiektu dzielonego przez pole koła. Promień do wzoru na pole koła należy wziąć z obwodu koła, które jest równe obwodowi obiektu.

```
codeblock = """
def compactness(pole_obiektu, obwod_obiektu):
    # pole = pi * r**2
    # obwod = 2*pi*r
    r = obwod_obiektu / (2*math.pi)
    pole_kola = math.pi * r**2
    c = math.sqrt(pole_obiektu / pole_kola)
    return c
"""

expression = "compactness(!Shape_Area!, !Shape_Length!)"
field_name = "compactness"
ff1_compactness = arcpy.management.CalculateField(ff1_dissolved_duze, field_name, expression, "PYTHON3", codeblock)

# znalezienie najbardziej zwartego obszaru
compactness = {key:value for (key, value) in arcpy.da.SearchCursor(ff1_compactness,['OID@', 'compactness'])}
max_oid, max_compactness = max(compactness.items(), key=lambda krotka: krotka[1])
print(max_oid, max_compactness)

24 0,893322519992807

query = f"OBJECTID IN ({max_oid})"
ff1_best = arcpy.analysis.Select(ff1_compactness, "ff1_best", query)
```

Kryterium 11 polegało na wybraniu jak najbardziej zwartego obszaru spośród poligonów uzyskanych jako wynik kryterium 10 i uznaniu tego za ostateczny wynik analizy – za optymalną działkę dla inwestycji hotelarskiej. W celu obliczenia zwartości działki (ang. compactness), zastosowałem współczynnik zwartości.

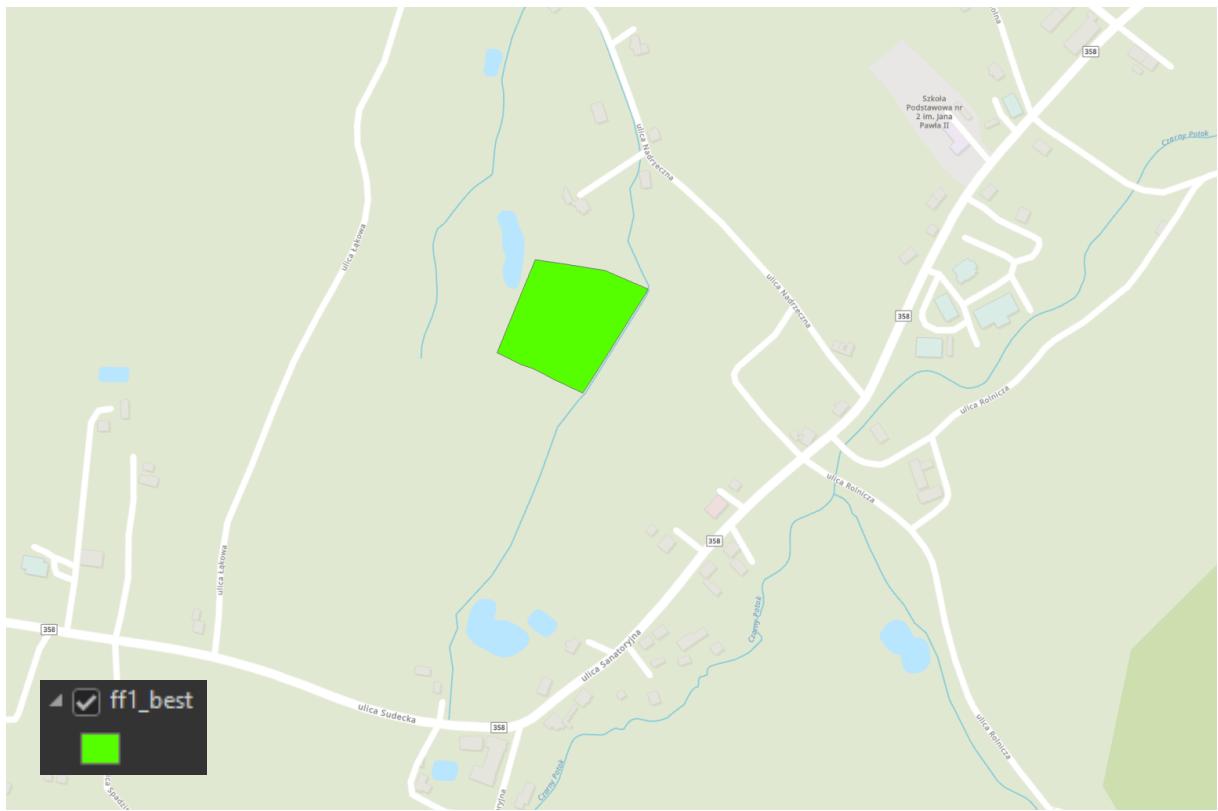
współczynnik zwartości kształtu C wg formuły:

$$C = \sqrt{\frac{Pow_{obiektu}}{Pow_{koła}}}$$

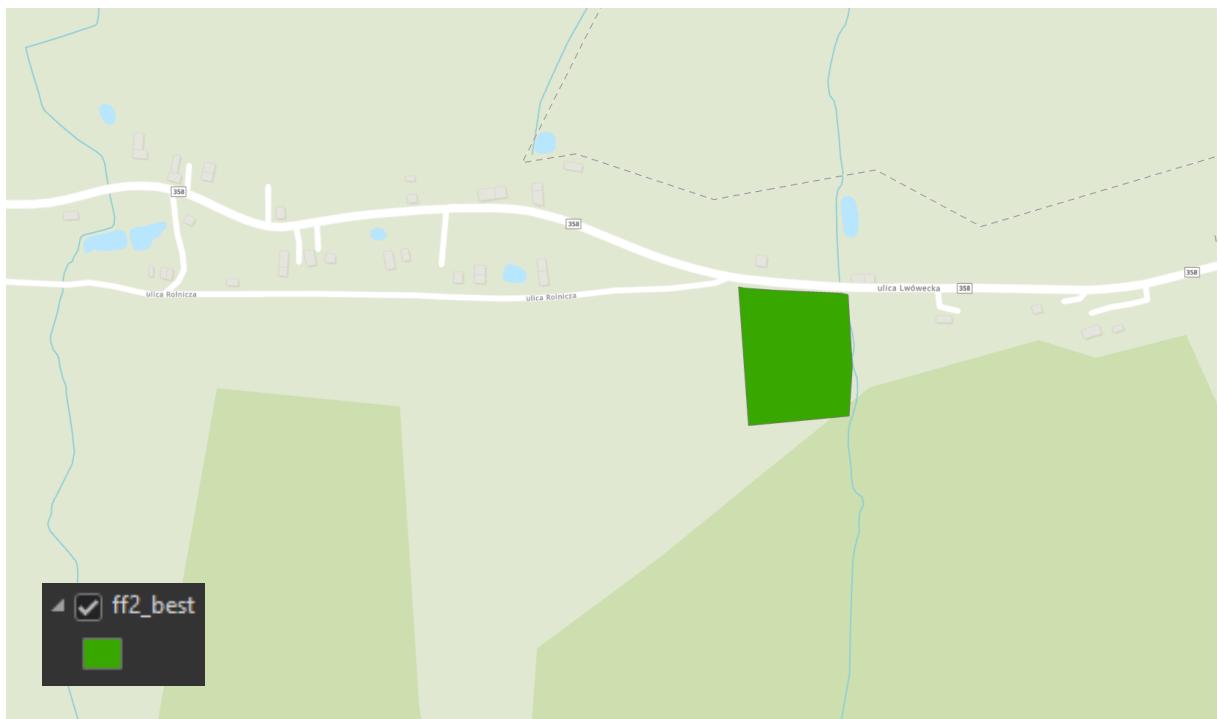
gdzie  $Pow_{koła}$  jest powierzchnią koła o obwodzie równym obwodowi obiektu.

Napisałem funkcję w pythonie, której użyłem jako argumentu w arcgisowej funkcji CalculateField. Udało mi się w ten sposób stworzyć nowy atrybut reprezentujący zwartość poligona. Następnie z użyciem funkcji SearchCursor oraz pythonowych struktur danych uzyskałem ID oraz wartość compactness poligona o największej zwartości. Uzyskane w ten sposób ID użyłem do wybrania tegoż poligona (funkcja Select).

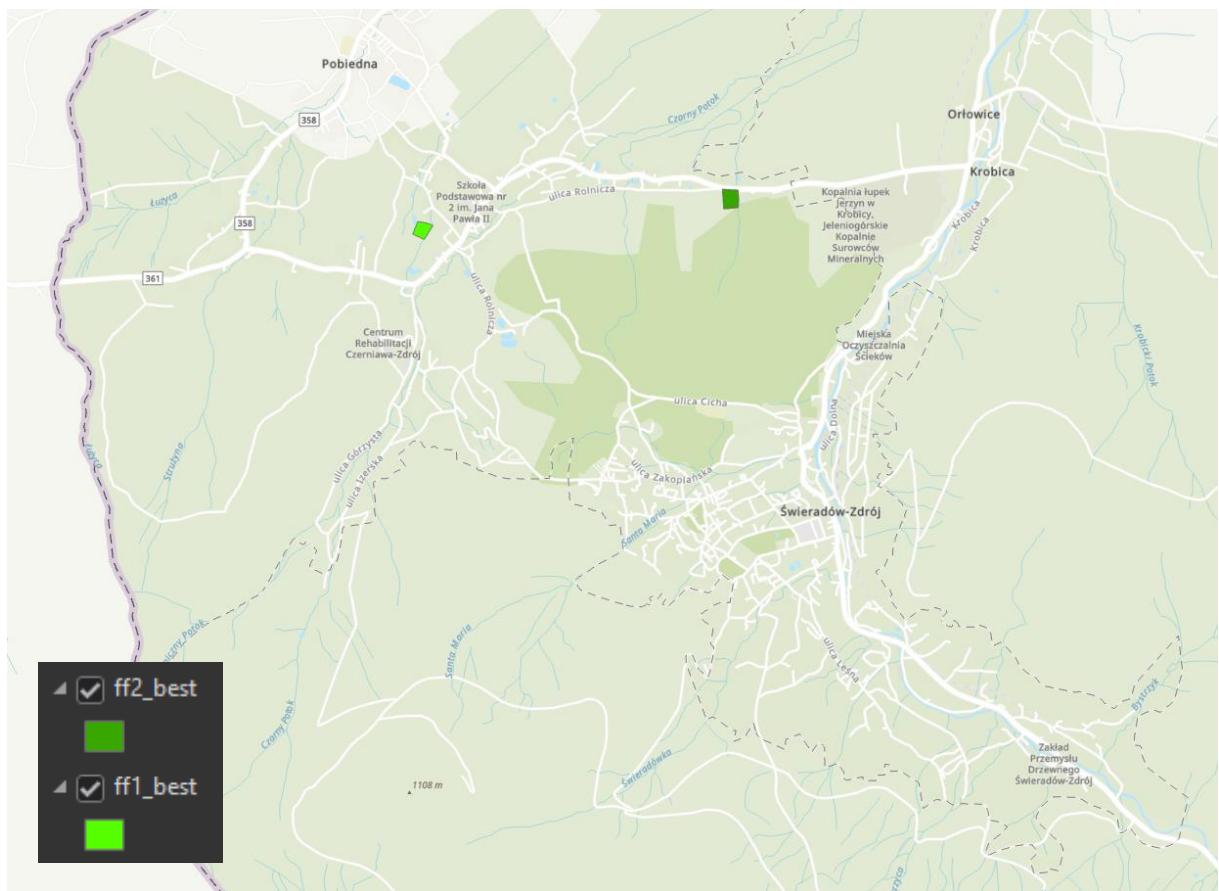
## Kryterium 11 – ostateczne v1 – Scenariusz 1: wagi jednakowe



## Kryterium 11 – ostateczne v1 – Scenariusz 2: wagi eksperckie



## Najbardziej zwarte działki na tle gminy.



## Kryterium 12

### Kryterium 12 - ostateczne v2 - wskazanie optymalnej lokalizacji z wykorzystaniem map kosztów

#### Stworzenie mapy kosztów

```
layers_to_union = list()
for filename in os.listdir(folder_bdot):
    if os.path.isfile(os.path.join(folder_bdot, filename)):
        src = rf'{folder_bdot}\{filename}'
        if 'PT' in src and src.endswith('.shp'):
            layers_to_union.append(src)
#
# print(src)

PT_union = arcpy.management.Merge(layers_to_union, "PT_union", None, "NO_SOURCE_INFO")

arcpy.management.AddField(PT_union, "Value_v2", "SHORT", None, None, None, '', "NULLABLE", "NON_REQUIRED", '')
```

```
codeblock = """
def xkod_to_cost(x_kod):
    costs_dict = {
        "PTWP01": 0,
        "PTWP02": 200,
        "PTWP03": 0,
        "PTZB01": 200,
        "PTZB02": 100,
        "PTZB03": 200,
        "PTZB04": 200,
        "PTZB05": 50,
        "PTLZ01": 100,
        "PTLZ02": 50,
        "PTLZ03": 50,
        "PTRK": 15,
        "PTRK02": 15,
        "PTUT01": 0,
        "PTUT02": 90,
        "PTUT03": 100,
        "PTUT04": 20,
        "PTUT05": 20,
        "PTTR01": 20,
        "PTTR02": 1,
        "PTKM01": 100,
        "PTKM02": 200,
        "PTKM03": 170,
        "PTKM04": 200,
        "PTGN01": 1,
        "PTGN02": 1,
        "PTGN03": 1,
        "PTGN04": 1,
        "PTPL01": 50,
        "PTSO01": 0,
        "PTSO02": 0,
        "PTWZ01": 0,
        "PTWZ02": 0,
        "PTNZ01": 150,
        "PTNZ02": 150
    }
    cost = costs_dict[x_kod]
    return cost
"""

expression = "xkod_to_cost(!X_KOD!)"
field_name = "Value_v2"
PT_union = arcpy.management.CalculateField(PT_union, field_name, expression, "PYTHON3", codeblock)

PT_union_raster = arcpy.conversion.FeatureToRaster(PT_union, "Value_v2", "PT_union_raster", 1)

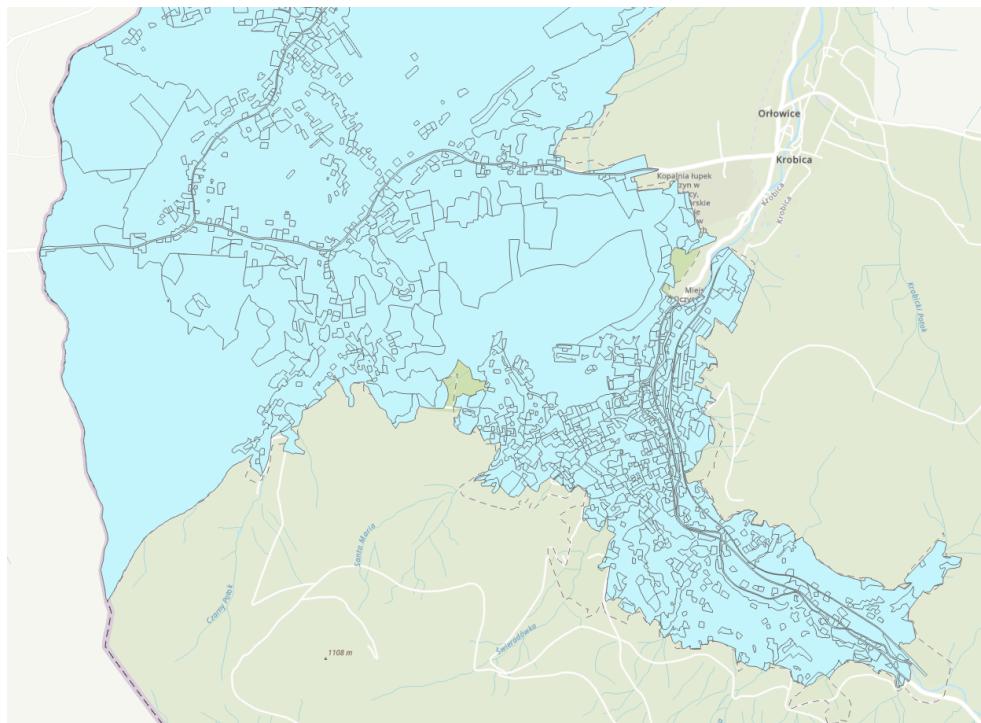
mapa_kosztow = arcpy.sa.SetNull(PT_union_raster, PT_union_raster, "Value = 0")
```

Kryterium 12 polegało na wskazaniu optymalnej lokalizacji ze względu na najtańsze przyłączenie gazociągu. W tym celu trzeba było stworzyć mapę kosztów, czyli raster z wartościami kosztu budowy gazociągu o długości odpowiadającej długości piksela. Wartości uzależniliśmy od typu pokrycia terenu w danym miejscu, np. koszt budowy gazociągu na terenie kolejowym będzie o wiele większy niż koszt budowy gazociągu o takiej samej długości na terenie rolniczym. Dane dotyczące pokrycia terenu uzyskałem łącząc kilka różnych warstw z BDOT (seria warstw PT – Pokrycie Terenu), skorzystałem z funkcji Merge. Następnie na podstawie atrybutu X\_KOD, określającego szczegółowo typ pokrycia terenu dodałem nowy atrybut określający koszt budowy gazociągu. Wartości kosztów dla par (typ pokrycia: koszt) trzeba było ustalić samodzielnie, biorąc pod uwagę bariery bezwzględne (tereny, przez które bezwzględnie nie możemy prowadzić gazociągu) i bariery względne (takie tereny, przez które poprowadzenie gazociągu jest trudne i wiąże się z dużymi kosztami). Po prawej stronie zamieszczam słownik z wartościami. Takie słowniki robi się zazwyczaj tak, że wartość 1 przypisuje się do terenu o najmniejszym koszcie, a reszta wartości to wielokrotności kosztu budowy na najtańszym terenie (np. jeśli koszt budowy 1 metra gazociągu na najtańszym terenie wynosi 200 złotych, a na innym terenie 1000 złotych, to słownikowe wartości określające koszt budowy wynoszą odpowiednio 1 i 5). Zera w słowniku oznaczają brak danych ("NoData") lub bariery bezwzględne. Po utworzeniu i wypełnieniu pola z kosztem budowy (funkcja

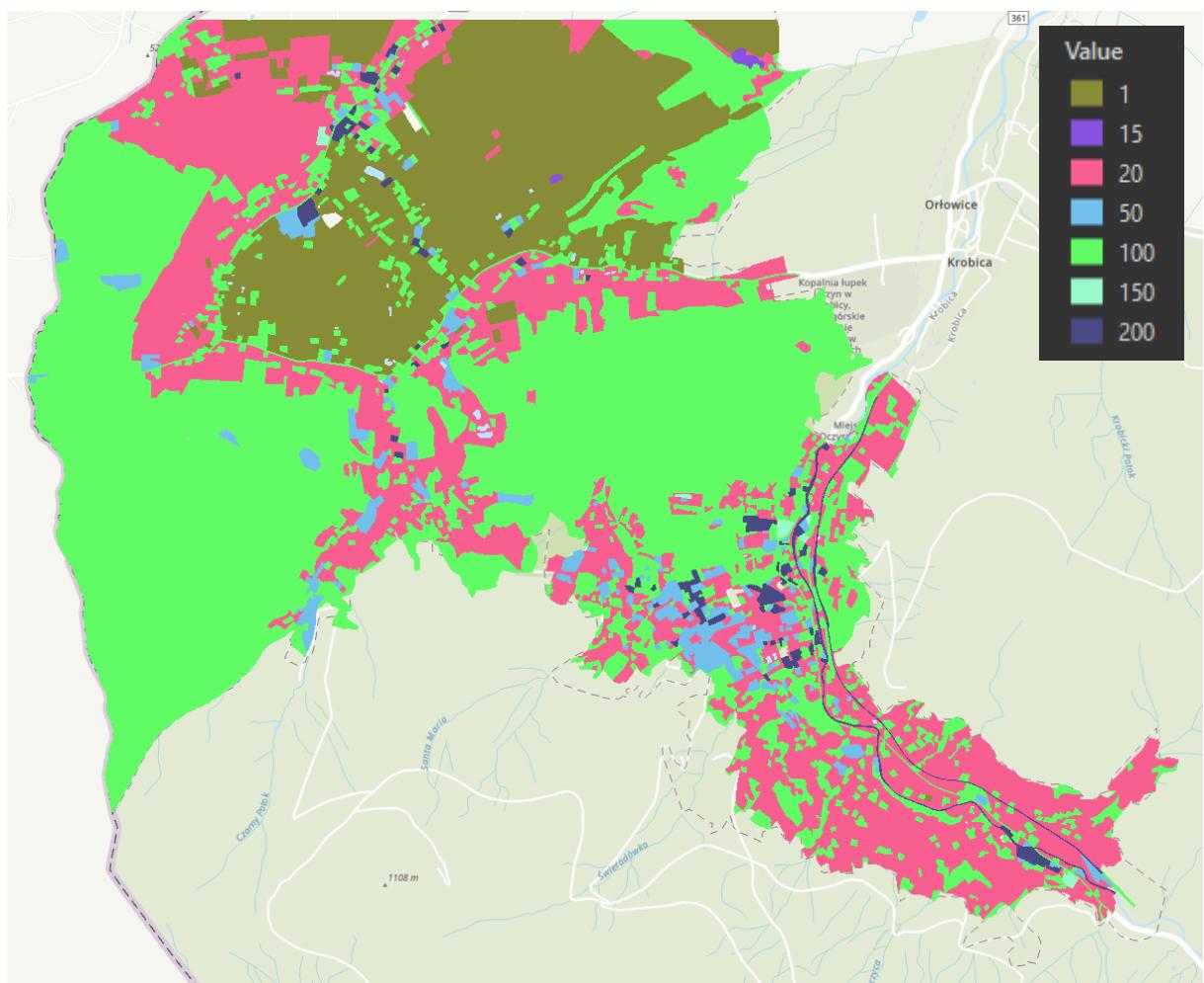
CalculateField), zamieniłem wektor na raster (FeatureToRaster) oraz użyłem funkcji SetNull, by wymazać z rastra całkowicie obszary o koszcie 0 ("NoData"). Powstała w ten sposób mapa kosztów względnych. Następnie z użyciem funkcji CostDistance (funkcja ta wykorzystuje mapę kosztów względnych) zrobiłem mapę kosztów skumulowanych reprezentującą dla każdego piksela najmniejszy koszt dotarcia do dowolnej części obiektu źródłowego, w tym przypadku działek (obszarów) o powierzchni większej niż 1 ha. Ostatnim krokiem było użycie funkcji CostPath, by dostać ścieżkę o najmniejszym koszcie od gazociągu do dowolnej dużej działki. Funkcja ta przyjmuje jako argumenty m.in. obiekty destynacyjne (gazociąg) i mapę kosztów skumulowanych.

```
costs_dict = {
    "PTWP01": 0,
    "PTWP02": 200,
    "PTWP03": 0,
    "PTZB01": 200,
    "PTZB02": 100,
    "PTZB03": 200,
    "PTZB04": 200,
    "PTZB05": 50,
    "PTLZ01": 100,
    "PTLZ02": 50,
    "PTLZ03": 50,
    "PTRK": 15,
    "PTRK02": 15,
    "PTUT01": 0,
    "PTUT02": 90,
    "PTUT03": 100,
    "PTUT04": 20,
    "PTUT05": 20,
    "PTTR01": 20,
    "PTTR02": 1,
    "PTKM01": 100,
    "PTKM02": 200,
    "PTKM03": 170,
    "PTKM04": 200,
    "PTGN01": 1,
    "PTGN02": 1,
    "PTGN03": 1,
    "PTGN04": 1,
    "PTPL01": 50,
    "PTSO01": 0,
    "PTSO02": 0,
    "PTWZ01": 0,
    "PTWZ02": 0,
    "PTNZ01": 150,
    "PTNZ02": 150}
```

Złączone warstwy pokrycia terenu.

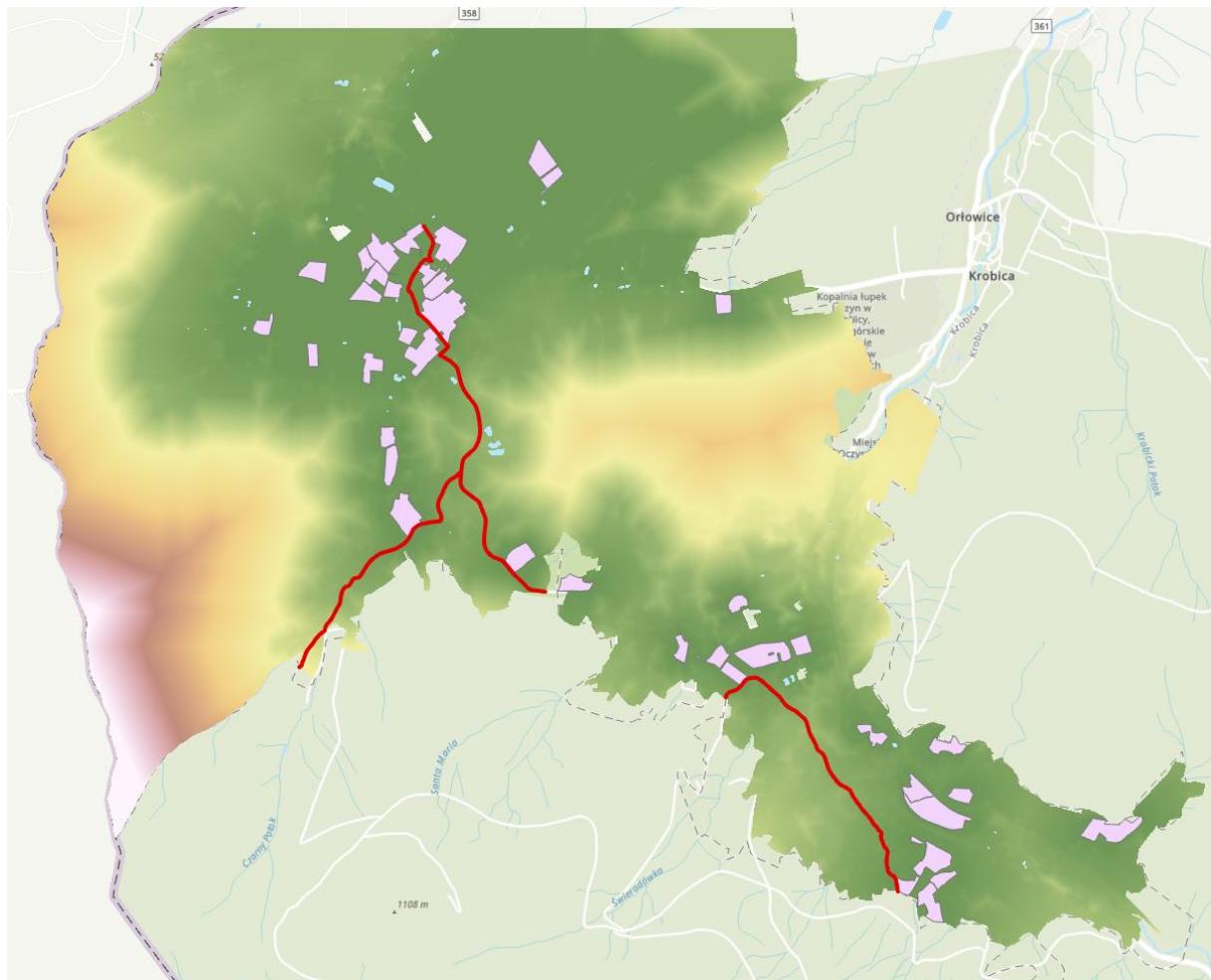


Mapa kosztów względnych.

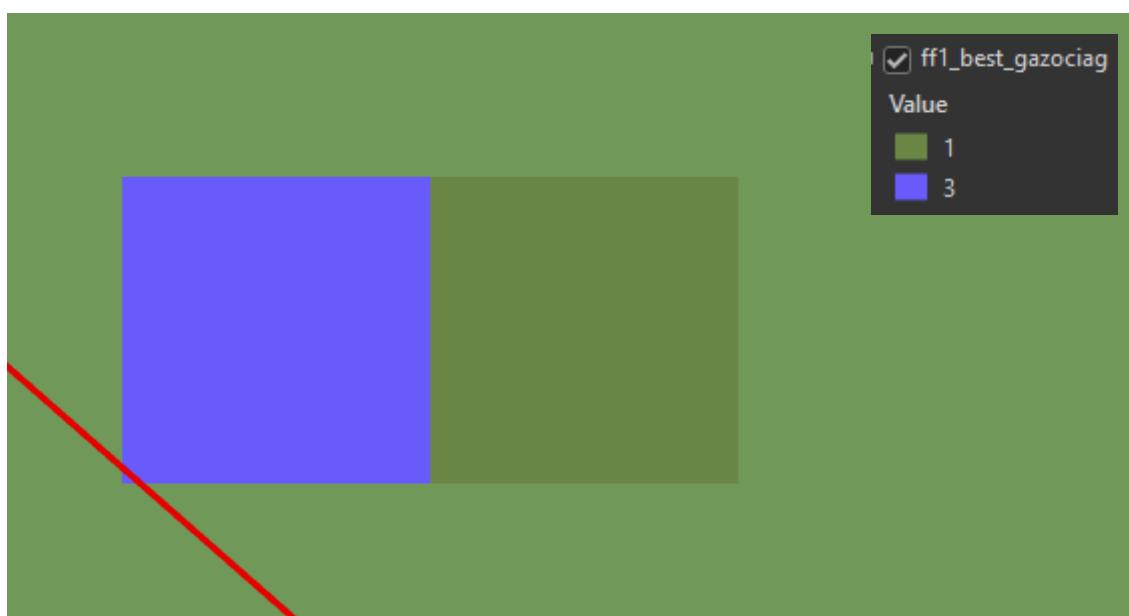


## Kryterium 12 – ostateczne v2 – Scenariusz 1: wagi jednakowe

Mapa kosztów skumulowanych, przydatne obszary większe niż 1ha i gazociąg (lokalizacja gazociągu nie ma znaczenia przy uzyskaniu tej mapy kosztów).



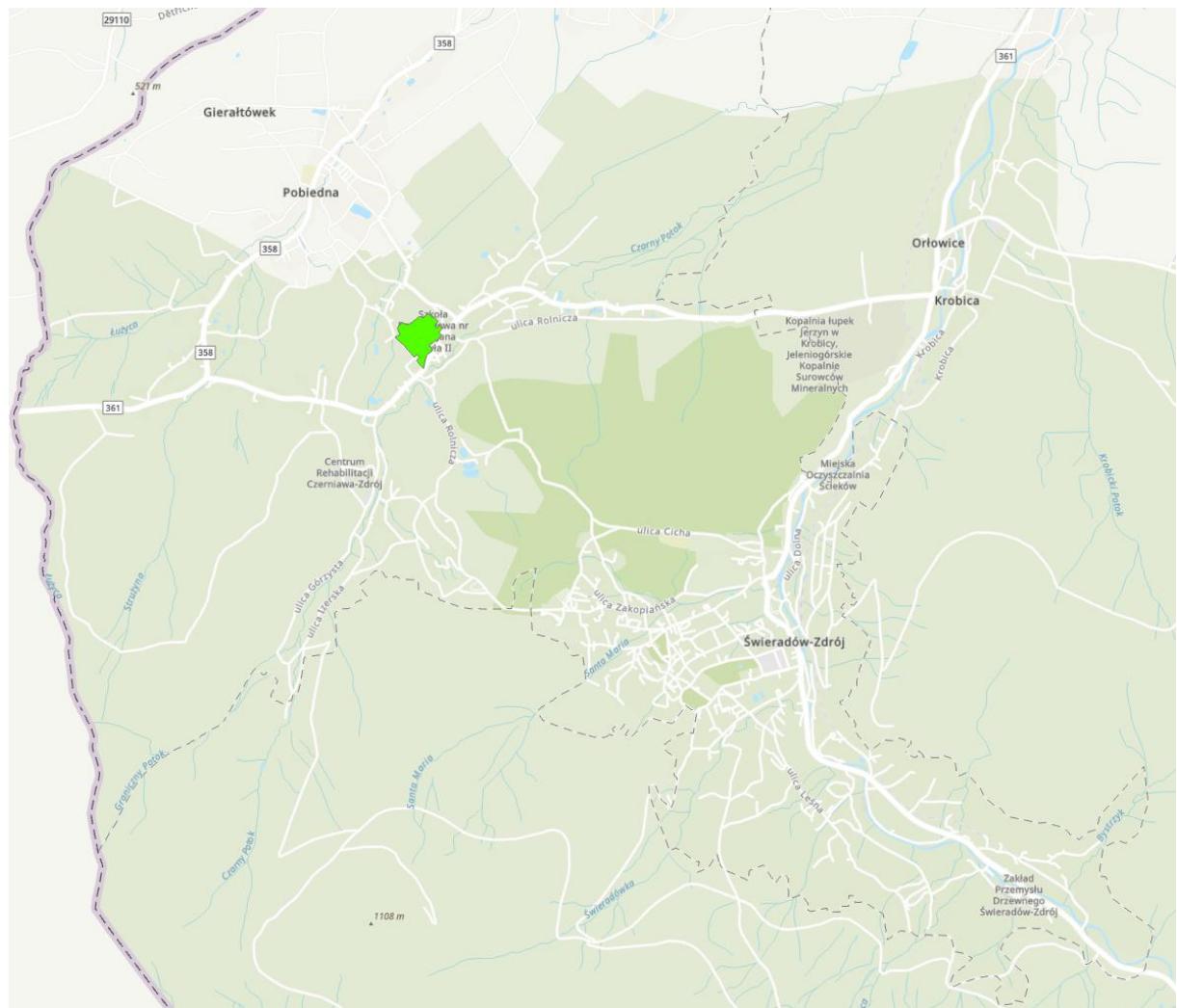
Ścieżka o najmniejszym koszcie (przebieg gazociągu).



Działka (obszar) o najmniejszym koszcie przyłączenia gazociągu.

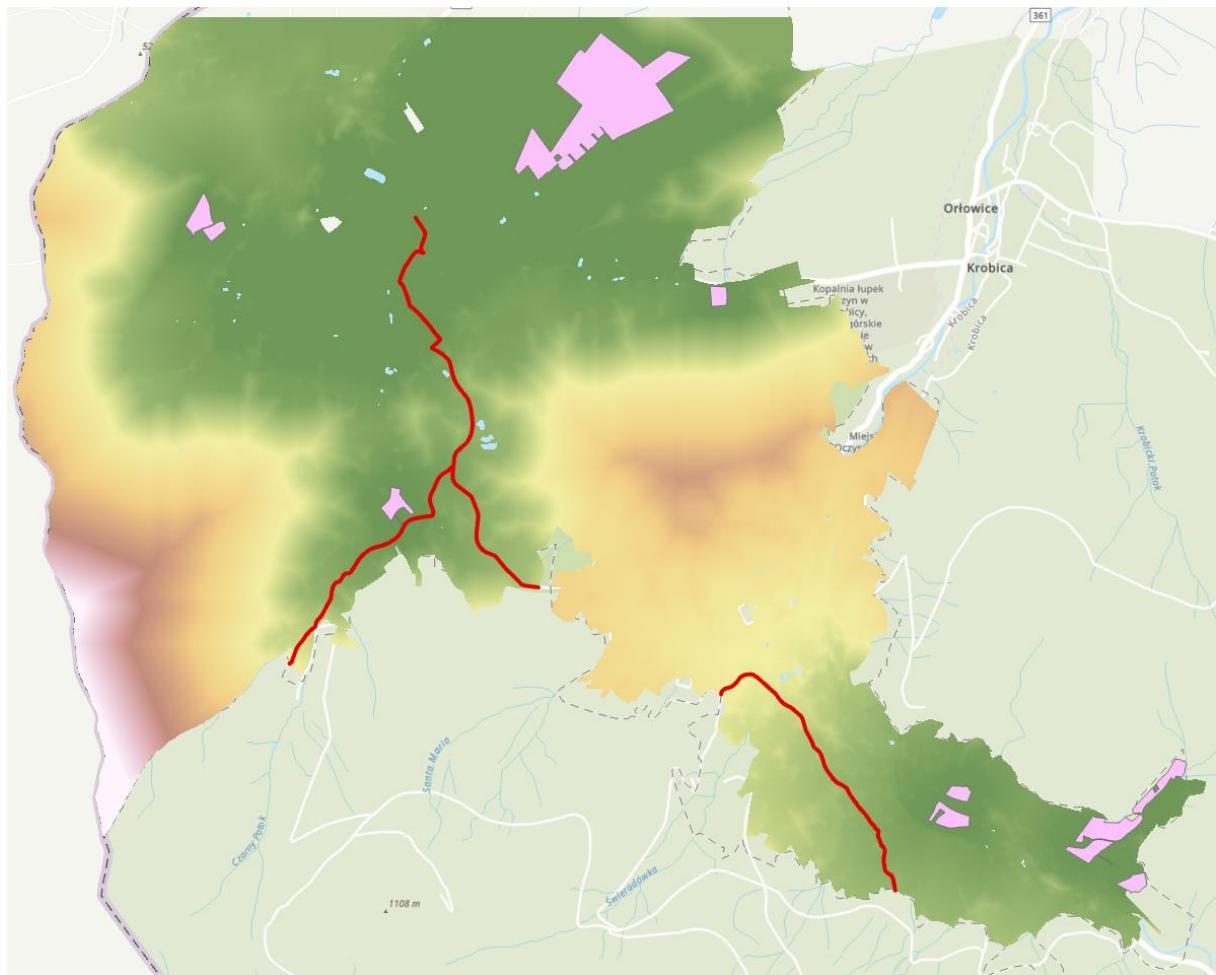


Położenie działki na tle gminy.

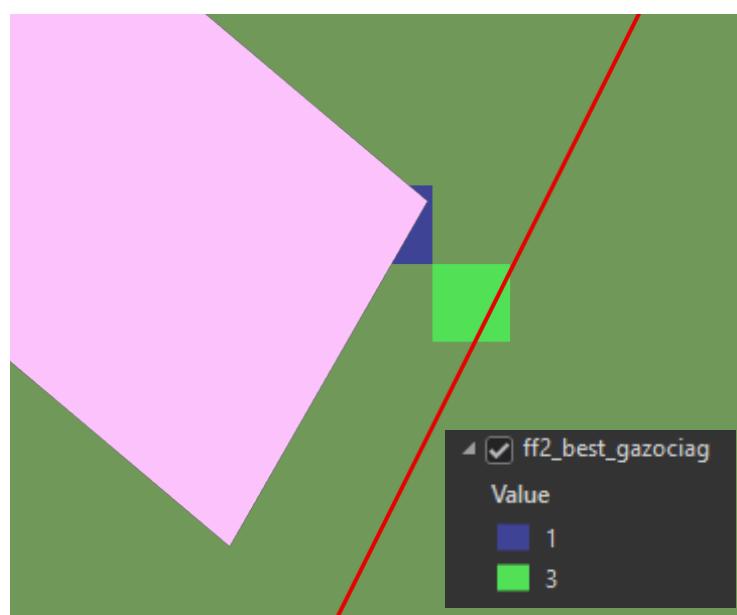


## Kryterium 12 – ostateczne v2 – Scenariusz 2: wagi eksperckie

Mapa kosztów skumulowanych, przydatne obszary większe niż 1ha i gazociąg (lokalizacja gazociągu nie ma znaczenia przy uzyskaniu tej mapy kosztów).



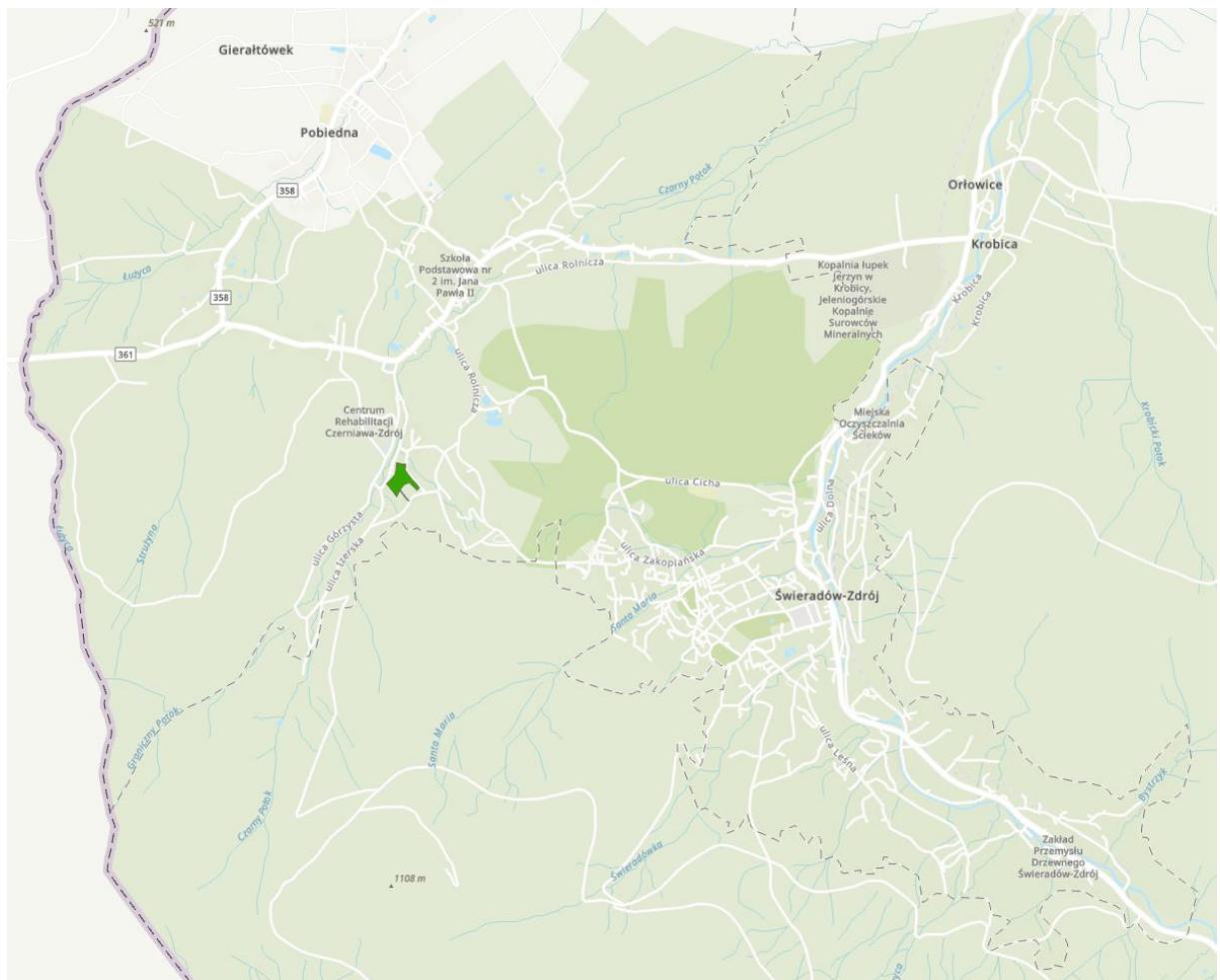
Ścieżka o najmniejszym koszcie (przebieg gazociągu).



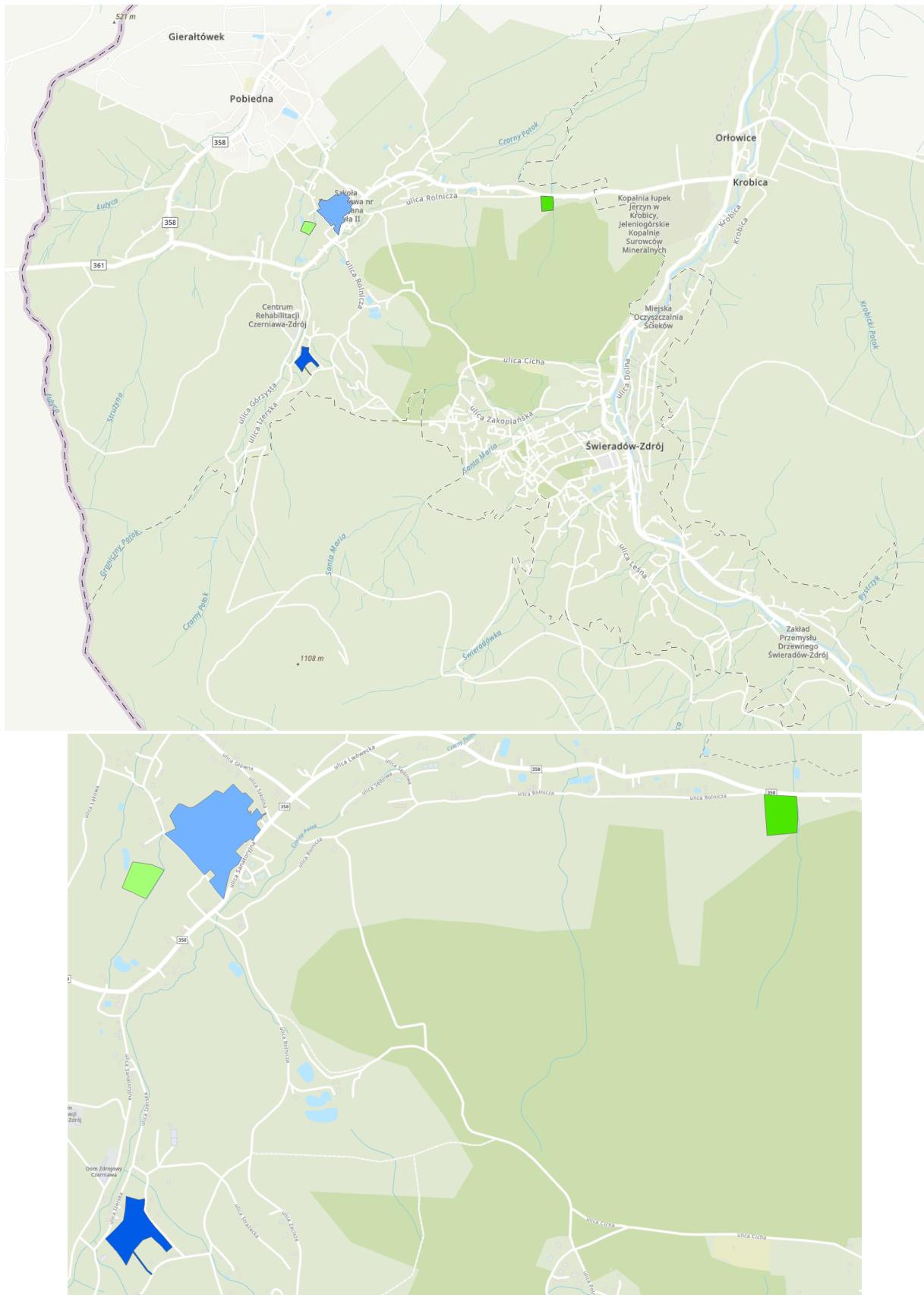
Działka (obszar) o najmniejszym koszcie przyłączenia gazociągu.



Położenie działki na tle gminy.



Na zielono – obszary wybrane ze względu na zwartość. Na niebiesko – obszary wybrane ze względu na koszt dołączenia gazociągu. Jaśniejsze odcienie – wagi jednakowe. Ciemniejsze odcienie – wagi eksperckie.



### 3. Raport z testu na danych z innego obszaru

Tak jak wspomniałem wcześniej, kod przeze mnie przygotowany jest dość uniwersalny w takim sensie, że jeśli dane są poprawne i mają taką samą strukturę, typ i atrybuty jak dane dla Świeradów-Zdrój, to wszystkie komórki notebooka powinny przejść po kolej bez żadnych problemów. Dane testowe dostaliśmy od prowadzącej przedmiot, dotyczyły one gminy Pleśna w powiecie tarnowskim. W danych były: BDOT, działki ewidencyjne, gazociąg, gleby, NMT. W moim skrypcie jako daną wejściową przyjmuję też granice gminy, także to musiałem pozyskać samemu z PRG. Po zapisaniu danych na dysku, zmieniłem ścieżki do danych w czwartej komórce z kodem i odpaliłem skrypt (Cell: Run All). Pierwszych kilka komórek przeszło bez problemu, jednak już przy kryterium 1 pojawił się błąd.

```
▼ Kryterium 1 - odległość od konkurencji minimum 400m

In [9]: query = "FUNSZCZ LIKE '%hotel%' OR FUNSZCZ LIKE '%motel%' OR FUNSZCZ LIKE '%pensjonat%' OR FUNSZCZ LIKE '%domhypocynkowy%'"
        hotele = arcpy.analysis.Select(bdot['budynki'], "hotele", query)
        ed = arcpy.sa.EucDistance(hotele)
        minimum = 400 # 400
        maximum = 800 # 800
        k1 = arcpy.sa.FuzzyMembership(ed, arcpy.sa.FuzzyLinear(minimum, maximum))

ExecuteError
Traceback (most recent call last)
In [9]:
Line 2:     hotele = arcpy.analysis.Select(bdot['budynki'], "hotele", query)
File C:\ALL\ArcGISPro\ArcGIS_Pro_3_0\Resources\ArcPy\arcpy\analysis.py, in Select
Line 138:     raise e

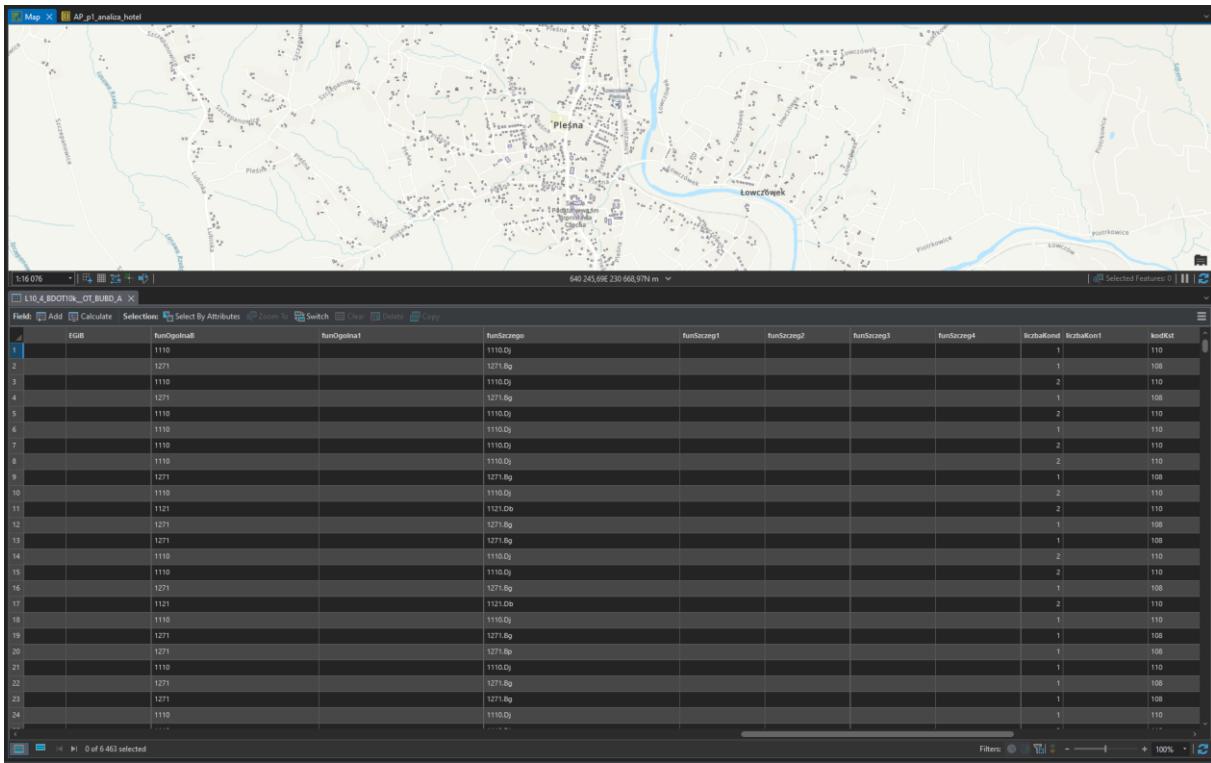
File C:\ALL\ArcGISPro\ArcGIS_Pro_3_0\Resources\ArcPy\arcpy\analysis.py, in select
Line 135:     retval = convertArcObjectToPyObject(gp.Select_analysis(*gp_fixargs((in_features, out_feature_class, where_clause), True)))

File C:\ALL\ArcGISPro\ArcGIS_Pro_3_0\Resources\ArcPy\arcpy\geoprocessing_base.py, in <lambda>
Line 512:     return lambda *args: val(*gp_fixargs(args, True))

ExecuteError: ERROR 000358: Invalid expression FUNSZCZ LIKE '%hotel%' OR FUNSZCZ LIKE '%motel%' OR FUNSZCZ LIKE '%pensjonat%' OR FUNSZCZ LIKE '%domhypocynkowy%'
Failed to execute (Select).
```

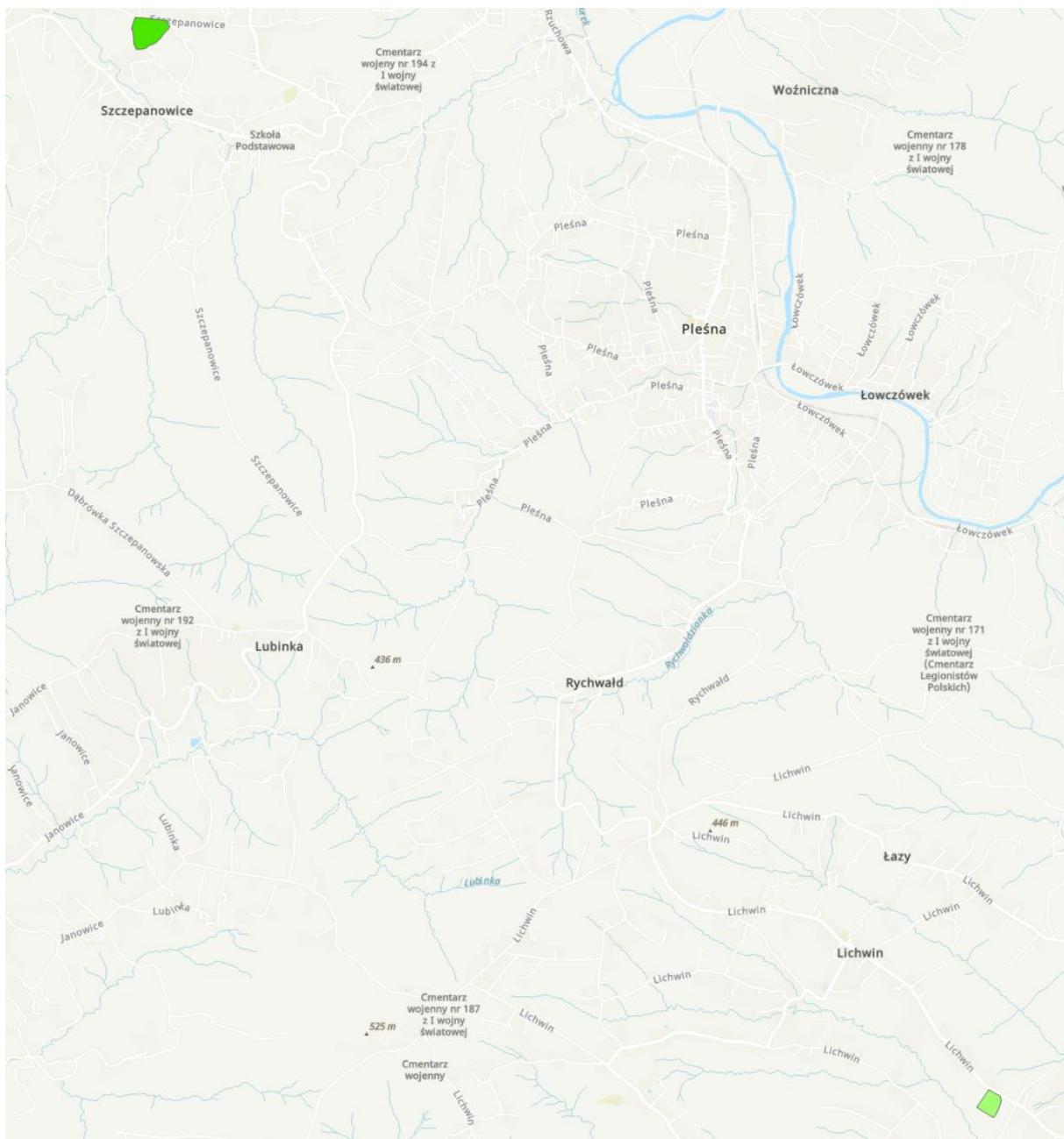
Po przeczytaniu komunikatu, przystąpiłem do sprawdzenia atrybutów warstwy BUBD. Okazało się, że warstwa nie ma atrybutu o nazwie FUNSZCZ, za to jest atrybut o nazwie funSzczego, jednak same wartości tego atrybutu są inne niż z FUNSZCZ z warstwy budynków ze Świeradowa.

funOgólna1	funSzczego	funSzczeg1	funSzczeg2
	1110.Dj		
	1271.Bg		
	1110.Dj		
	1271.Bg		
	1110.Dj		



Postanowiłem samemu pobrać (przez geoportaal) paczkę danych powiatowych BDOT dla tego obszaru, żeby sprawdzić czy to jakaś niespójność w bocie, bo spodziewałem się, że atrybuty dla warstw tak samo nazwanych w różnych rejonach kraju będą spójne (tak samo nazwane i o takiej samej strukturze). Okazało się, że w aktualnych danych, które pobrałem, nazwa i wartości atrybutów zgadzają się z tymi danymi, na których pracowałem dla gminy Świeradów-Zdrój. Nie zastanawiałem się wtedy nad przyczyną różnic w atrybutach, tylko po prostu zamieniłem dane BDOT od prowadzącej na pobrane przeze mnie aktualne dane BDOT. Myślę, że różnice mogły wynikać z tego, że dane od prowadzącej były nieaktualne (możliwe, że w przeszłości atrybuty warstw były inne) lub pochodziły z innego źródła niż dane na których pracowaliśmy dla Świeradowa. Możliwe też, że dane pobrane dla obszaru innego niż powiat mają inne atrybuty, ale bardzo bym się zdziwił, gdyby to była prawda, tak być nie powinno. Podejrzewam jeszcze kilka innych przyczyn takiego stanu rzeczy, ale nie będę tutaj ich wypisywał, bo to są tylko przypuszczenia i dla tego ćwiczenia nie ma to w zasadzie znaczenia, skoro pisałem skrypt dla danych o określonej strukturze i takie dane uzyskałem pierwszym sposobem, o którym pomyślałem. Przy pisaniu tego sprawozdania zauważałem, że obydwie warstwy (aktualna i stara(?)) mają atrybut X\_KOD, który zdaje się odpowiadać funkcji szczegółowej budynku. Jeśli to prawda (należaałoby to sprawdzić w rozporządzeniu), to można by zmienić skrypt w miejscach dotyczących budynków tak, by wyodrębniał budynki mieszkalne na podstawie tego atrybutu (więcej o pomysłach na ulepszenie kodu napiszę we

wnioskach). Jednak wcale nie wiadomo, czy kod by przeszedł bez błędów, bo różnice w strukturze danych mogą (i pewnie tak jest) dotyczyć też pozostałych warstw. Najważniejsze jest, że dla „normalnych” aktualnych danych, kod przeszedł do końca bez żadnego problemu po drodze i uzyskałem finalne wyniki. Przedstawię teraz tylko finalne wyniki dla gminy Pleśna, bez wyników poszczególnych kryteriów i etapów. Ważne jest to, że kod zadziałał bez problemu i dostaliśmy końcowe wyniki. Gazociąg przechodził bezpośrednio przez kilka różnych obszarów, co nasuwa kolejny pomysł jak można ulepszyć kod (jeśli jest taki przypadek jak tutaj, to wybrać kilka „najlepszych” działek pod względem kosztu budowy gazociągu).



#### **4. Podsumowanie i wnioski**

W tej ostatniej już części sprawozdania opiszę napotkane podczas realizacji projektu problemy, wymienię kilka pomysłów na ulepszenie skryptu ipynb i całego procesu analizy oraz podsumuję krótko projekt i napiszę kilka wniosków. Będzie to mało usystematyzowane, tę sekcję można potraktować jako swego rodzaju zbiór różnych myśli.

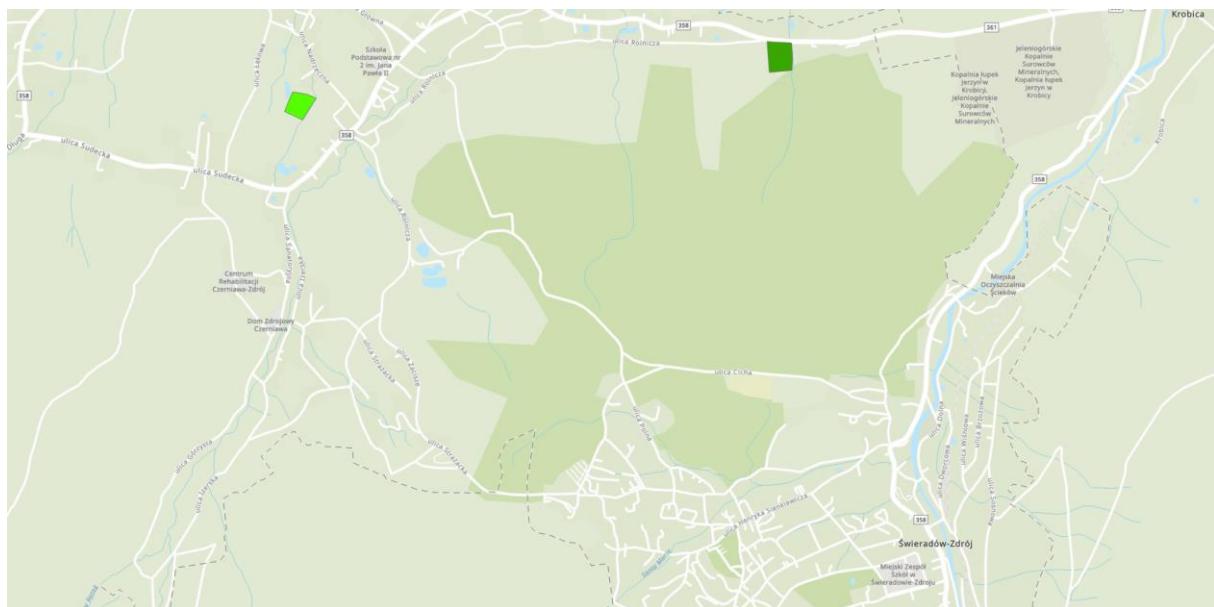
Przy wykonywaniu tego projektu dość często pojawiały się przeróżne problemy związane z programem ArcGIS Pro. Nie rozumiem jak – będąc właściwie liderem na rynku płatnego oprogramowania GIS – można dopuścić do tego, by program do obsługi danych przestrzennych nie chciał współpracować z dość popularnym formatem geopackage (.gpkg). Wyświetlanie dużych warstw rastrowych często nie działało w ogóle, albo działało z bardzo dużym opóźnieniem. Bardzo często w notebooku ipynb z jakiegoś powodu przestawały działać skróty klawiszowe, takie jak ctrl+v. Niektóre ustawienia środowiska z pakietu arcpy.env nie działały, w szczególności overwriteOutput. Wiele ustawień jak i funkcji działa w nieintuicyjny i nieprzewidywalny sposób, dokumentacja arcpy mogłaby być o wiele lepsza. Dużo czasu musiałem poświęcić na rozmyślania nad tym jak mam coś zrobić, a nie co mam zrobić. Czasami pojawiały się losowe błędy, gdzie rozwiązaniem było wyłączenie i ponowne włączenie programu, a „ERROR 999999: Something unexpected caused the tool to fail. Contact Esri Technical Support”, to już w ogóle hit... Ogólnie program działał zbyt wolno na dość dobrym komputerze (przy działaniu wbudowanych algorytmów, wyświetlaniu warstw, aktualizowaniu interfejsu) – ArcGIS powinien lepiej wykorzystać zasoby obliczeniowe komputera. Samo środowisko i sposób zwracania wyników, sposób ustawiania różnych parametrów – to wszystko jest bardzo specyficzne i nieintuicyjne i wymaga poświęcenia czasu na zapoznanie się z tym jak działa ArcGIS; pisanie skryptów z użyciem arcpy przyda się na pewno tym osobom (i będzie dla nich przyjemniejsze), które dobrze znają już ArcGISa.

Dobór wartości minimum i maksimum dla FuzzyMembership, jak i sam kształt funkcji przydatności to kwestia dyskusyjna i pewnie można by na ten temat zrobić rozległe badania i eksperymenty. Pozornie niezbyt znacząca zmiana konwencji doboru tych wartości może mieć znaczący wpływ na wynik końcowy analizy przestrzennej, a samych konwencji/systemów można stworzyć mnóstwo; nie ma tutaj jedynego słusznego wyboru i często wynik analizy będzie zależał od subiektywnego doboru takich wartości przez osobę lub grupę osób przeprowadzającą analizy. W przypadku gminy Świeradów-Zdrój, powinniśmy również pozyskać dane z graniczących gmin (powiatów), a nawet dane z

sąsiedniego kraju (gmina ta graniczy z innymi powiatami i z Czechami). Jeśli obszarem analizy byłoby województwo, to (jeśli miałoby to znaczenie dla poprawności i prawdziwości wyników algorytmów przetwarzających dane przestrzenne) należałoby pozyskać dane z sąsiednich województw (by mieć te dane w strefie buforowej). W przypadku gminy Pleśna nie ma tego problemu (a raczej zagadnienia), bo gmina ta leży wewnątrz powiatu; jest otoczona przez inne gminy z tego samego powiatu, więc dane np. z paczki powiatowej BDOTu będą dostępne na całym obszarze buforowym. Odnosząc się do danych testowych – jeśli rzeczywiście „w obiegu” są dwie wersje BDOTu, to można by wykrywać (oczywiście automatycznie w kodzie) z jaką wersją bdotu mamy do czynienia i tej informacji użyć do wybrania ścieżki przeprowadzenia programu. Wymagałoby to jednak około 2 razy więcej kodu niż jest teraz. Lokalizacje budynków hoteli można wziąć też z OSM (Open Street Map) lub Google Maps. Można też zrobić web scraping adresów/lokalizacji hoteli z różnych serwisów internetowych (takich jak booking.com). Dane takie byłyby zapewne o geometrii punktowej, a w bdocie mamy przecież poligony. Można by wtedy zamienić poligony na punkty (np. poprzez centroidy) i zrobić jedną warstwę z hotelami przez łączenie lokalizacji z kilku różnych źródeł. Problemem wymagającym przemyślenia jak to rozwiązać byłoby wtedy duplikowanie się punktów lub posiadanie czterech różnych punktów reprezentujących ten sam hotel. Jednak jeśli zastosowanie takiej warstwy byłoby takie jak w tym projekcie, to nie miałoby to za bardzo znaczenia, a nawet można by wytoczyć argument, że to nawet lepiej, że mamy więcej danych na podstawie których powstanie raster odległości. Przykładem kolejnego, nieuwzględnionego wcześniej kryterium przydatności terenu pod inwestycję hotelową może być kryterium bliskości szlaków turystycznych; im mniejsza odległość hotelu od szlaku, tym lepiej. Kryterium to miałoby zastosowanie (i można by mu nadać większą wagę) przede wszystkim w gminach położonych w górach – dla Świeradowa pasuje. Dane dotyczące szlaków pieszych/górskich/turystycznych można by pozyskać z OSM (<https://wiki.openstreetmap.org/wiki/Hiking>) (<https://wiki.openstreetmap.org/wiki/Tag:route%23Hiking>) np. przez overpass turbo lub pobierając paczkę danych z geofabrik.de i wybór po odpowiednich tagach. Tak uzyskaną warstwę liniową ze szlakami można by potraktować w taki sam sposób, w jaki potraktowałem np. drogi lub wody (tylko bez strefy ochronnej). Nie będę dokładnie opisywał jak to zrobić, bo opisałem już to dla wymienionych warstw. Generalnie można by użyć narzędzi logiki rozmytej, ustawić jakieś wartości np. maksimum na 50, minimum na 500 (oznaczałoby to, że przydatność terenu w odległości od szlaku w przedziale 0-50 metrów byłaby tak samo dobra (i najlepsza, wartość 1), od 50 do 500 metrów liniowo by malała, a ponad 500 metrów uznalibyśmy tereny za nieprzydatne pod względem

tego kryterium). Wynikiem końcowym byłby oczywiście raster z wartościami rozmytymi z przedziału 0-1. Ewentualnie można by też przypisać głównym szlakom większą wagę, np. dzieląc warstwę ze szlakami na kilka mniejszych (względem typu lub kategorii szlaku) i dla każdej warstwy z osobna zrobić raster przydatności (wartości min max mogłyby być takie same wszędzie), a na końcu przypisać wagi każdej z warstw i je połączyć w jedną, tak jak w tym projekcie robiłem przy łączeniu kryteriów rozmytych.

Działki (obszary) uzyskane jako finalne wyniki były różne zarówno przy kryterium zwartości jak i przy kryterium najtańszego przyłącza gazociągu. W obydwu przypadkach dostałem dwa różne wyniki – jeden dla wag jednakowych i jeden dla wag eksperckich. Jeśli chodzi o kryterium gazociągowe, to akurat tak się złożyło, że dla obydwu wariantów wag, były tam działki tuż przy gazociągu, dlatego najtańsza ścieżka budowy nie była dłuża i skomplikowana, tylko była po prostu najkrótszą odlegością od gazociągu do działki. Jeśli chodzi o kryterium zwartości, to dla obydwu wariantów wag uzyskane działki znajdowały się w północnej części gminy, nie w głównej części miasta. Rzeczywiście, działki te są dość zwarte, kształtem przypominają kwadrat.



Obydwie działki znajdują się nad „rzekami” (są to raczej małe cieki wodne). Nasuwa to na myśl kolejny pomysł ulepszenia skryptu – moglibyśmy w jakiś sposób (np. po atrybutach, choć pewnie trzeba by było coś do tego dodać, żeby działało zgodnie z oczekiwaniami) wybrać tylko istotne dla nas cieki wodne i na nich używać algorytmów przetwarzania danych.

„Rzeka” przy najbardziej zwartej działce i wagach eksperckich.



Obydwie działki są rzeczywistymi, pojedynczymi działkami ewidencyjnymi, a nie obszarami powstałymi przez łączenie sąsiadujących działek.

Najbardziej zwarta działka z wag eksperckich

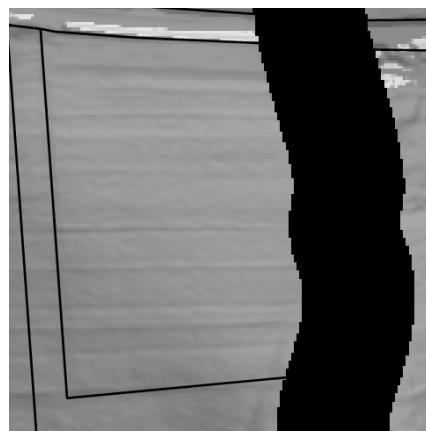




Najbardziej zwarta działka uzyskana z wariantu wag eksperckich znajduje się tuż przy drodze, wygląda dobrze, jednak według mnie jest położona zbyt daleko od większych pagórków i gór, które położone bardziej na południe. Myślę, że mogłoby to być kolejnym kryterium w przypadku gmin górskich (odległość od gór).

Według mnie, wskazanie najbardziej zwartego obszaru jako najlepszego wcale nie jest dobrym pomysłem. Lepiej byłoby uwzględnić compactness jako ważoną część i wybrać działkę gdzie ogólny wysoki procent przydatności miałby decydujące znaczenie, bo w tym co teraz mamy to nie ma różnicy czy działka (grupa działek) jest przydatna w 70% czy w 100%. Myślę, że w tym konkretnym przypadku gminy Świeradów-Zdrój (gdzie wyników jest mało) najlepiej byłoby wyjść od obrazka z przydatnością, ew. z nałożonymi działkami i samemu (lub wraz z grupą "ekspertów" – ludzi zainteresowanych/decydentów) ocenić przydatność i wybrać najlepszą lokalizację (lub kilka najlepszych).

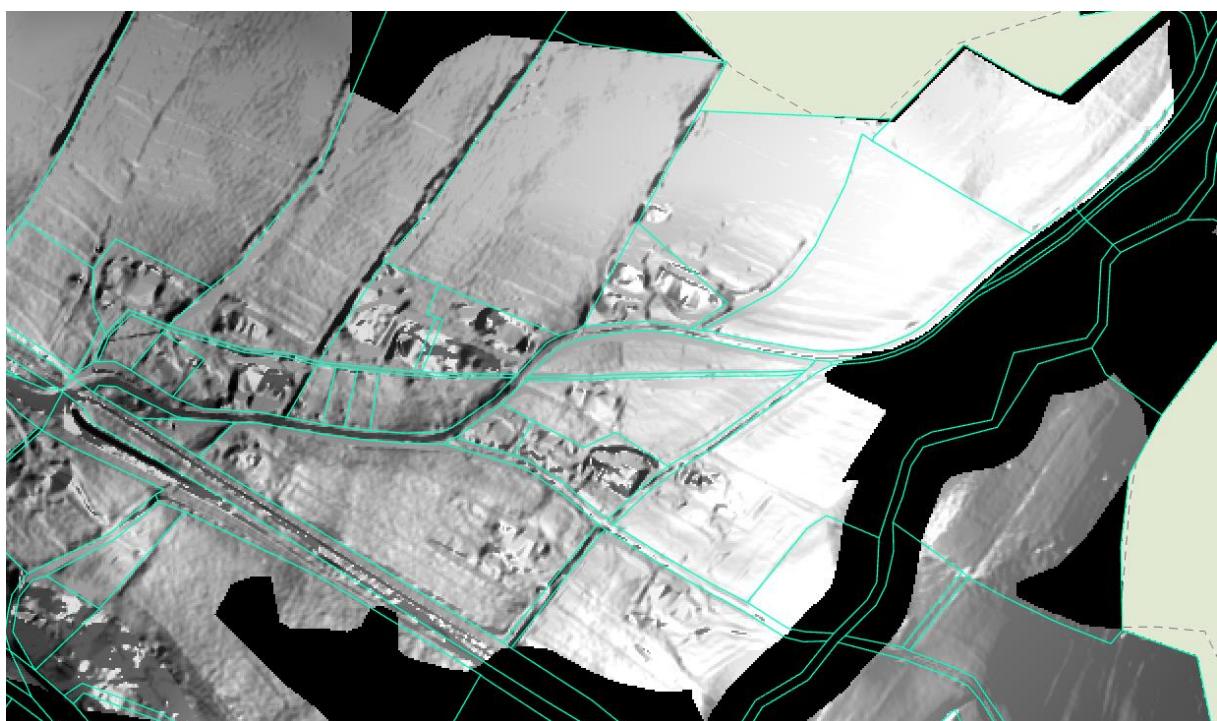
Najbardziej zwarta działka zahacza o strefę ochronną wód



Obszar o dużej przydatności w południowej części miasta



Rejony przy ulicy Myśliwskiej



## Rejony przy ulicy Myśliwskiej



Na koniec kilka subiektywnych wniosków, odnoszących się bardziej do samego projektu, niż do technicznej części jego wykonania.

Uważam, że pisanie tego raportu to była strata mojego czasu, ponieważ:

- a) niczego nowego się nie dowiedziałem pisząc go
- b) nikomu ten raport nie posłuży/nie przyda się w żaden sposób
- c) według mnie do zaliczenia ćwiczenia zamiast sprawozdania w zupełności wystarczyłby uniwersalny, dobrze opisany lub pogrupowany kod z notatnika ipynb (którego przygotowanie i tak zajęło sporo czasu)
- d) zbyt długo to zajęło (jak i samo pisanie kodu i użeranie się z ArcGISem)

Uważam, że ćwiczenie z analizą lokalizacji hoteli było przydatne, ponieważ:

- a) dowiedziałem się o możliwościach interakcji z ArcGIS Pro poprzez kod (biblioteka arcpy)
- b) poznalem pewne sposoby analiz przestrzennych (fuzzy logic, wagi itp.)
- c) poznalem kilka funkcji ArcGISa, których wcześniej nie znałem