Przedmiot: Systemy Wspomagania Decyzji	**Projekt końcowy**	**Wydział:** WEAIiIB **Semestr:** 5 **Grupa dziekańska:** 2
Imię i nazwisko: • Jan Gallina • Konrad Flis • Mateusz Gołąbek • Maria Jagintowicz	Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie	**Data wykonania ćwiczenia:** 04.01.2024 **Data sprawozdania:** 10.01.2024

Spis Treści

1. Cel projektu

Celem projektu było wybranie własnego problemu decyzyjnego oraz zbudowanie aplikacji, której użytkownik może ustawić konkretne parametry, takie jak wybór metody, kryteriów oraz metryk w celu porównania danych za pomocą algorytmów poznanych na laboratoriach. W tym celu stworzyliśmy program w języku Python, w którym zaimplementowaliśmy metody wspomagania decyzji Topsis, SP-CS oraz RSM, dodatkowo tworzą interfejs graficzny użytkownika.

2. Problem i zbiór danych

Jako problem w naszym projekcie wybraliśmy porównywanie ofert wynajmu mieszkań w różnych dzielnicach Warszawy, na podstawie ogłoszeń znalezionych na portalach internetowych. Dla każdej oferty wybraliśmy kryteria, które potencjalny wynajmujący mógł uznać za ważne. W naszym programie użyliśmy następujących kryteriów:

- Gęstość zaludnienia – gęstość zaludnienia dzielnicy, w której znajduję się mieszkanie
- Dostępność transportu publicznego – ocena gęstości połączeń i częstotliwości odjazdów oraz dostępność metra (w skali od 0 do 10, gdzie 10 to najlepsza komunikacja miejska)
- Bliskość natury – ocena odległości od najbliższego parku, lasu czy terenu zieleni (w skali od 0 do 10)
- Odległość od centrum – w skali od 0 do 10 (gdzie 0 to Śródmieście)

- Cena za metr kwadratowy – koszt wynajmu w przeliczeniu na metr kwadratowy
- Metraż – wielkość mieszkania
- Opłaty za media – dodatkowa wysokość opłaty za prąd, gaz, internet itd.
- Liczba pokoi

Z uwagi na charakter niektórych kryteriów, część z nich jest minimalizowana. Jest to istotne z uwagi na działanie algorytmów, co prezentuje dodatkowa kolumna bazy z informacją, które kryterium jest maksymalizowane. Są to:

- Dostępność transportu publicznego
- Bliskość natury
- Metraż
- Liczba pokoi

Pozostałe kryteria są minimalizowane. Nasza baza danych zawiera łącznie 26 ofert. Z każdej dzielnicy Warszawy w bazie znajduje się przynajmniej jedna oferta.

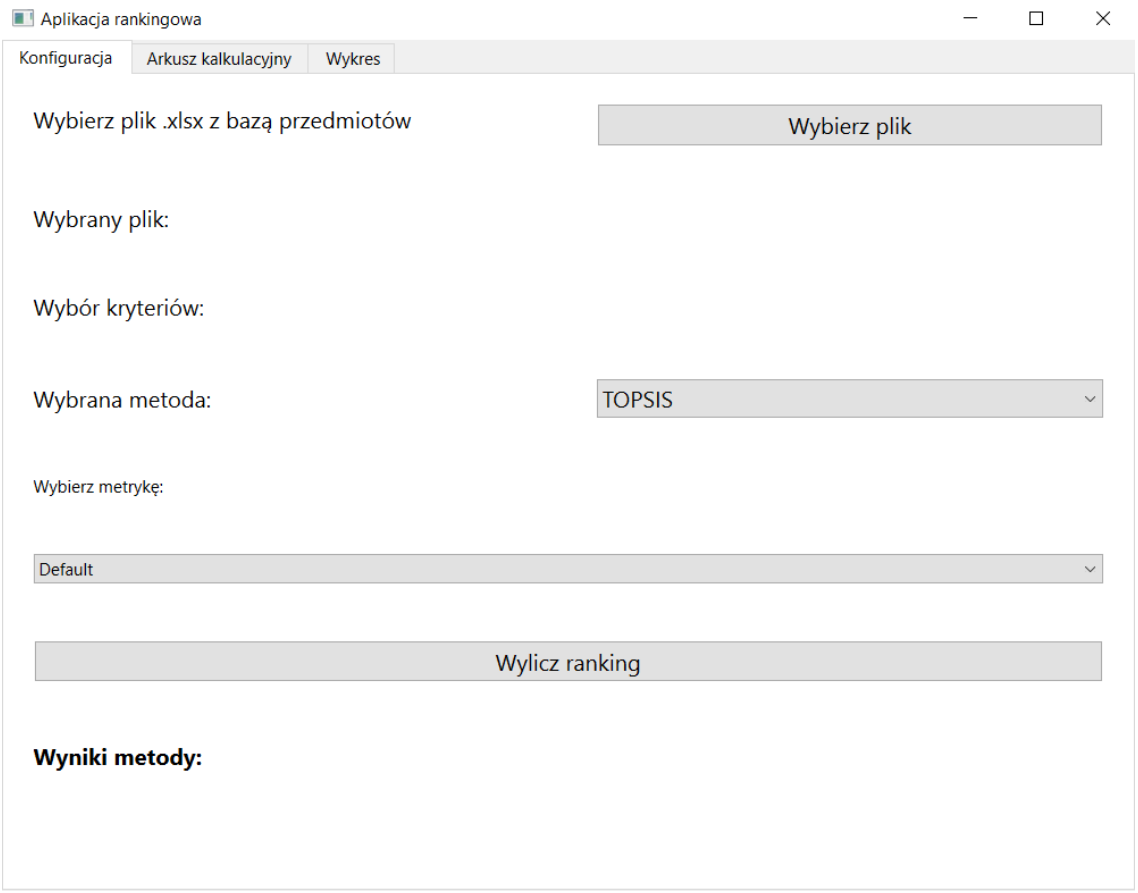
Lp.	Nazwa	Gęstość zaludni	Dostępność tra	Bliskość natury	Odległość od	Cena za metr kw	Metraż	Opłaty za med	Liczba pokoj	Wagi	Maksymalizacja
1.	Bemowo	5177	5	5	8	66	45	320	2	0,1	False
2.	Białołęka 1	2097	4	8	10	64	50	400	2	0,15	True
3.	Białołęka 2	2097	4	8	10	72	48	350	2	0,1	True
4.	Bielany	4127	7	3	8	89	39	350	2	0,15	False
5.	Mokotów 1	6378	8	2	2	76	59	450	3	0,2	False
6.	Mokotów 2	6378	8	2	2	86	46	400	2	0,1	True
7.	Mokotów 3	6378	8	2	2	102	39	380	2	0,1	False
8.	Ochota	8332	8	1	2	112	40	400	2	0,1	True
9.	Praga-Południe	8348	8	3	3	97	41	330	2		
10.	Praga-Północ	5381	8	4	3	79	53	450	3		
11.	Rembertów	1278	3	9	9	64	42	280	2		
12.	Śródmieście 1	6550	10	2	0	104	48	500	2		
13.	Śródmieście 2	6550	10	2	0	104	48	450	2		
14.	Targówek	5106	7	5	7	73	45	320	2		
15.	Ursus	7206	6	5	7	69	55	400	3		
16.	Ursynów 1	3458	8	6	6	73	52	410	3		
17.	Ursynów 2	3458	8	6	6	80	45	380	2		
18.	Wawer	1084	3	10	8	56	50	330	2		
19.	Wesoła	1150	2	7	8	41	62	350	3		
20.	Wilanów	1393	3	7	8	54	55	400	2		
21.	Włochy	1721	4	8	7	62	48	420	2		
22.	Wola 1	7848	9	3	2	88	42	450	2		
23.	Wola 2	7848	9	3	2	88	45	470	2		
24.	Wola 3	7848	9	3	2	92	55	510	3		
25.	Żoliborz 1	6922	8	1	2	92	50	480	3		
26.	Żoliborz 2	6922	8	1	2	90	52	500	3		

Zrzut ekranu 1 - baza danych do problemu

Dodatkową kolumną są domyślne wagi konkretnych kryteriów wykorzystywane w metodzie Topsis. W aplikacji użytkownik ma również możliwość zadania własnych wag. Ostatnie dwie kolumny dotyczą wartości dla kryteriów (tj. wagi oraz flaga maksymalizacji).

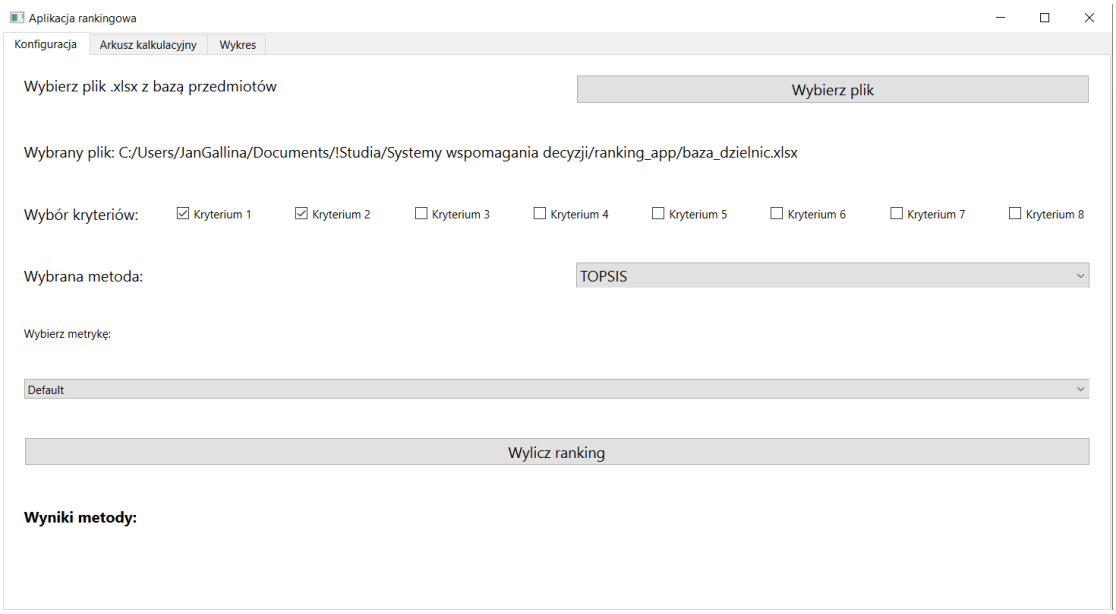
3. Wygląd i opis działania GUI

Nasz projekt został zaimplementowany w Pythonie, z wykorzystaniem biblioteki PyQt6, która służy do zbudowania aplikacji i interfejsu użytkownika, na podstawie napisanego kodu. Główne okno programu zawiera najważniejsze funkcjonalności, które w miarę działania programu rozszerzają się (np. wybór wag w kryteriach oraz tworzenie wykresu).



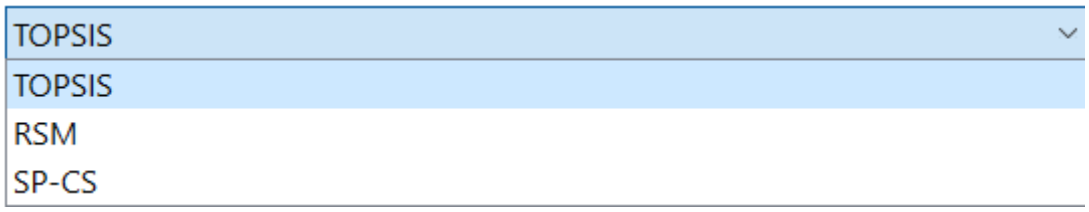
Zrzut ekranu 2 - główne okno programu po uruchomieniu aplikacji

Użytkownik, chcąc rozpocząć tworzenie rankingu, musi wybrać plik, na podstawie którego tworzone będą porównania. Aplikacja obsługuje pliki Excela z rozszerzeniem .xlsx i w odpowiednim formacie kolumn – pierwsza z nich to numer porządkowy, kolejna – nazwa, a w po serii kryteriów mogą znaleźć się dodatkowo wagi i flagi minimalizacji/maksymalizacji. Po wybraniu pliku odblokują się kolejne możliwości.



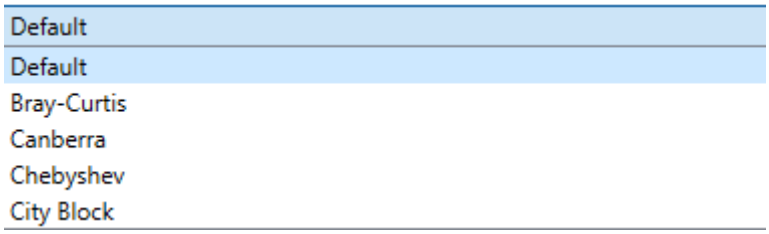
Zrzut ekranu 3 - główne okno po załadowaniu danych

Program odpowiednio rozpoznał występowaniu ośmiu kryteriów, z których pierwsze dwa są domyślnie wybrane. Z tego miejsca użytkownik może wybrać interesujące go kryteria oraz metodę, według której tworzony będzie ranking.



Zrzut ekranu 4 - wybór metody

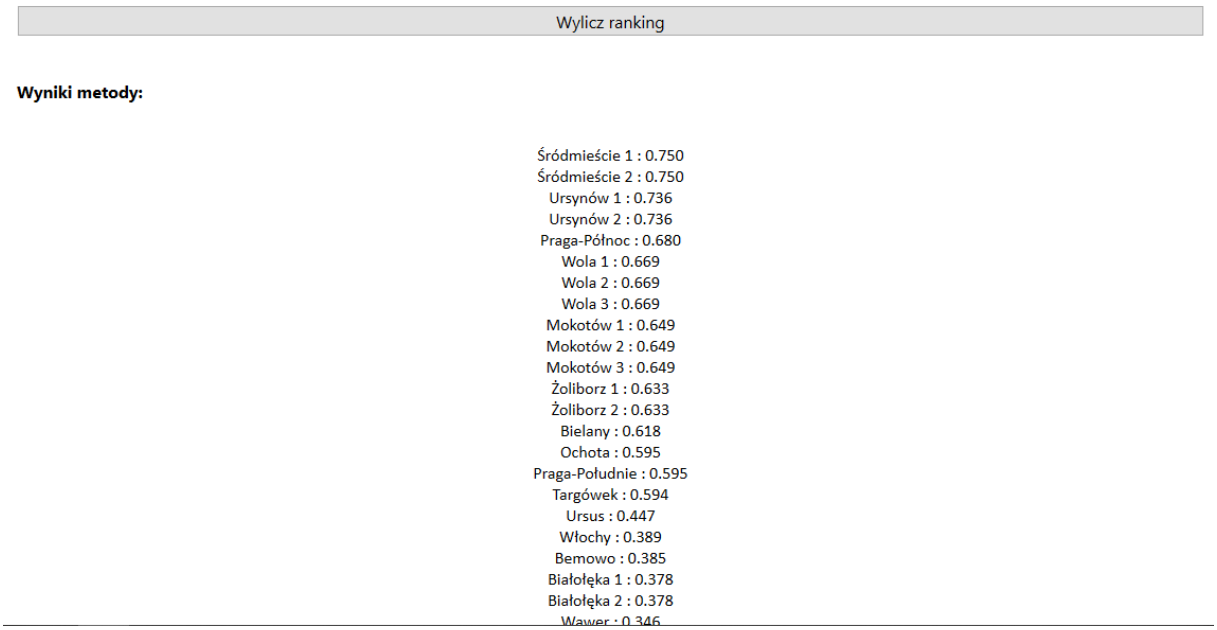
Dodatkowo użytkownik może wybrać jedną z pięciu zaimplementowanych metryk do obliczania odległości alternatyw od odpowiednich punktów, zgodnie z działaniem algorytmu.



Zrzut ekranu 5 - wybór metryki

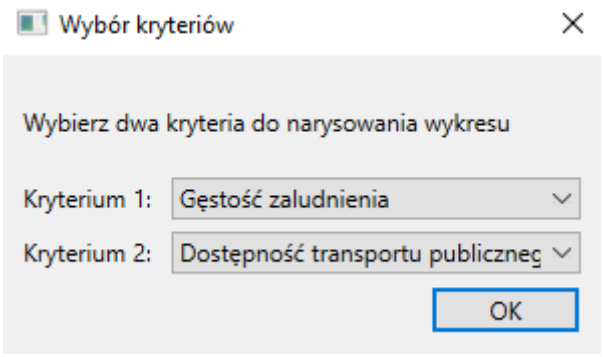
Każda z metryk charakteryzuje się nieco odmiennymi charakterystykami, z czego metryka wybrana domyślnie to klasyczna odległość dwóch punktów na płaszczyźnie kartezjańskiej.

Po wskazaniu wybranej metody, kryteriów oraz metryki, program jest gotowy do obliczenia wyników. Pokazują się one po naciśnięciu przycisku „Wylicz ranking”. Wyniki ukazują się w liście w kolejności zgodnej z obliczonym współczynnikiem scoringowym.

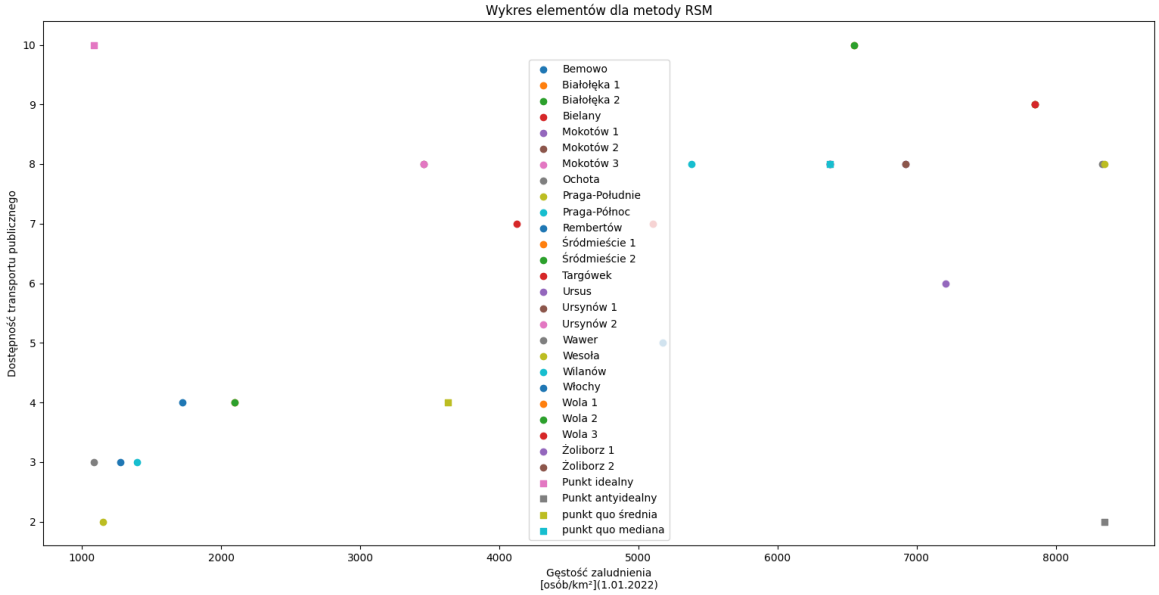


Zrzut ekranu 6 - przykładowe wyniki obliczone przez program

Kolejną funkcjonalnością jest możliwość tworzenia wykresów. Wykresy mogą powstawać tylko dla dwóch wybranych kryteriów, więc jeśli w pierwszym etapie ranking został utworzony dla więcej niż dwóch kryteriów, pojawi się dodatkowe okno z zapytaniem, dla których stworzyć wykres.



Zrzut ekranu 7 - wybór kryteriów do tworzenia wykresu



Zrzut ekranu 8 - przykładowy wykres dla metody RSM

Na wykresie, oprócz alternatyw, zaznaczone są specjalne punkty w zależności od metody. Te punkty są umieszczone na dole legendy, a na wykresie zaznaczone są kwadratami.

4. Pseudokody algorytmów

Poniżej zamieściliśmy schematy wszystkich trzech zrealizowanych algorytmów.

Algorytm Topsis:

1. Zaimportuj dane: mieszkania i ich własności, wektor wag i klasy ograniczeń
2. Znormalizuj macierz
3. Usuń alternatywy niemieszczące się w klasach odniesienia

3.1 Dla każdej alternatywy sprawdź, czy jakakolwiek jej cecha nie mieści się w klasach odniesienia
3.2 Jeśli tak – usuń alternatywę. Jeśli nie – dodaj ją do klasy
4. Wyznacz punkty idealny i antyidealny
5. Wyznacz odległości punktów od punktów idealnego i antyidealnego
6. Utwórz ranking na podstawie scoringów
7. Wyświetl wyniki

Algorytm SPCS

1. Inicjalizacja danych: mieszkania i ich własności
2. Usuń punkty zdominowane
3. Utwórz po 3 punkty status quo i aspiracji:
   3.1 Znajdź wartości najlepsze w zbiorze i dodaj je do zbioru punktów aspiracji
   3.2 Zaburz punkty aspiracji poprzez losową zmianę wartości
   3.3 Znajdź wartości średnią, medianę i losową wokół średniej, a następnie dodaj je do punktów status quo
4. Wyznacz losowe 3 odcinki między punktami status quo i aspiracji
5. Wyznacz odległości między punktami niezdominowanymi a odcinkami w wybranej metryce
6. Uwzględnij rzutowanie na odcinek i dodaj do odległości wynik rzutowania, tworząc scoring
   6.1 Znormalizuj scoring
7. Utwórz rankingi na podstawie scoringu
8. Wyświetl wyniki

Algorytm RSM

1. Inicjalizacja danych: mieszkania i ich własności
2. Usuń punkty zdominowane
3. Utwórz punkty status quo, aspiracji oraz antyidealny
4. Zbadaj odległości między punktami niezdominowanymi a granicami optymalności i punktami idealnymi
5. Dokonaj normalizacji
6. Utwórz ranking na podstawie odległości
7. Wyświetl wyniki

5. Struktura i kod programu

Nasz kod podzielony został na 4 pliki. Pierwsze trzy, są to pliki odpowiadające każdej z metod. W środku opisane są funkcje obliczające rankingi scoringowe odpowiednio dla każdej metody. Każda z nich posiada również, fragment odpowiadający za wybór odpowiedniej metryki wskazanej przez użytkownika w trakcie działania aplikacji.

Metoda Topsis

```python
from typing import List, Union, Optional, Tuple
from math import sqrt
import pandas as pd
import numpy as np
from scipy.spatial.distance import braycurtis, chebyshev, canberra, cityblock

Number = Union[float, int]


def euclid_norm(D: List[List], j: int) -> float:
    """
    Norma euklidesowa kolumny j z macierzy D
    :param D: (List[List[Number]]) : macierz decyzyjna
    :param j: (int) : indeks kolumny kryterium
    :return: pierwiastek sumy kwadratów
    """
    s = 0.0  # suma kwadratów elementów z kolumny
    for i in range(len(D)):
        s += D[j][i] ** 2
    return sqrt(s)


def topsis(D: List[List[Number]], W: List[Number], metric: str, W_max: Optional[List[bool]] = None) \
        -> Tuple[List[float], int, List[List[float]], List[float], List[float]]:
    """
    Metoda topsis tworząca ranking produktów
    :param D: (List[List[Number]]) : macierz decyzjna D[m x N]
    :param W: (List[Number]) : wektor wag
    :param W_max: (List[bool]) : wektor logiczny określający, które maksymalizujemy kryterium (domyślnie każde)
    :param metric: str : nazwa wykorzystywanej metryki
    :return: (Tuple[List[float], int, List[List[float]], List[float], List[float]]) : wektor współczynników skoringowych
    liczba kryetriów, macierz znormalizowana, punkty idealne, punkty antyidealne
    """
    print(D)
    m = len(D[0])  # liczba elementów
    n = len(D)  # liczba kryteriow
    N = [[0.0 for _ in range(m)] for _ in range(n)]  # macierz znormalizowana
    p_ideal = [0.0 for _ in range(n)]  # tablica punktów idealnych
    p_anti_ideal = [float('inf') for _ in range(n)]  # tablica punktów antyidealnych
    d_star = [0.0 for __ in range(m)]  # tablica odległości od punktu idealnego
    d_minus = [0.0 for __ in range(m)]  # tablica odległości od punktu nieidealnego
    c = [0.0 for __ in range(m)]  # współczynnik skoringowy

    if W_max is not None:  # minimalizacja czy maksymalizacja kryterium
        for i in range(len(W_max)):
            if not W_max[i] and i < n:
                p_ideal[i] = float('inf')
                p_anti_ideal[i] = 0
    else:
        W_max = [True for _ in range(n)]  # uzupełnienie parametru domyślnego


    for j in range(n):
        en = euclid_norm(D, j)
        for i in range(m):
            N[j][i] = W[j] * D[j][i] / en  # normalizacja macierzy
            if W_max[j] and p_ideal[j] < N[j][i]:  # znalezienie punktów idealnych
                p_ideal[j] = N[j][i]
            if not W_max[j] and p_ideal[j] > N[j][i]:
                p_ideal[j] = N[j][i]
            if W_max[j] and p_anti_ideal[j] > N[j][i]:
                p_anti_ideal[j] = N[j][i]
            if not W_max[j] and p_anti_ideal[j] < N[j][i]:
                p_anti_ideal[j] = N[j][i]

    if metric == "Default":
        for i in range(m):  # obliczenie odległości
            s_star = 0.0  # suma kwadratów róznicy punktu od punktu idealnego
            s_minus = 0.0  # suma kwadratów róznicy punktu od punktu antyidealnego
            for j in range(n):
                s_star += (N[j][i] - p_ideal[j]) ** 2
                s_minus += (N[j][i] - p_anti_ideal[j]) ** 2
            d_star[i] = sqrt(s_star)
            d_minus[i] = sqrt(s_minus)
            c[i] = d_minus[i] / (d_minus[i] + d_star[i])

    elif metric == "Bray-Curtis":
        N_as_array = np.asarray(N)
        p_ideal_as_vector = np.asarray(p_ideal)
        p_anti_ideal_as_vector = np.asarray(p_anti_ideal)
        for i in range(m):
            d_star[i] = braycurtis(N_as_array[:, i], p_ideal_as_vector)
            d_minus[i] = braycurtis(N_as_array[:, i], p_anti_ideal_as_vector)
            c[i] = d_minus[i] / (d_minus[i] + d_star[i])
```

```python
    elif metric == "Canberra":
        N_as_array = np.asarray(N)
        p_ideal_as_vector = np.asarray(p_ideal)
        p_anti_ideal_as_vector = np.asarray(p_anti_ideal)
        for i in range(m):
            d_star[i] = canberra(N_as_array[:, i], p_ideal_as_vector)
            d_minus[i] = canberra(N_as_array[:, i], p_anti_ideal_as_vector)
            c[i] = d_minus[i] / (d_minus[i] + d_star[i])

    elif metric == "Chebyshev":
        N_as_array = np.asarray(N)
        p_ideal_as_vector = np.asarray(p_ideal)
        p_anti_ideal_as_vector = np.asarray(p_anti_ideal)
        for i in range(m):
            d_star[i] = chebyshev(N_as_array[:, i], p_ideal_as_vector)
            d_minus[i] = chebyshev(N_as_array[:, i], p_anti_ideal_as_vector)
            c[i] = d_minus[i] / (d_minus[i] + d_star[i])

    elif metric == "City Block":
        N_as_array = np.asarray(N)
        p_ideal_as_vector = np.asarray(p_ideal)
        p_anti_ideal_as_vector = np.asarray(p_anti_ideal)
        for i in range(m):
            d_star[i] = cityblock(N_as_array[:, i], p_ideal_as_vector)
            d_minus[i] = cityblock(N_as_array[:, i], p_anti_ideal_as_vector)
            c[i] = d_minus[i] / (d_minus[i] + d_star[i])

    return c, n, N, p_ideal, p_anti_ideal


def compute_topsis(file_name: str, criteria: List[int], metric: str, weights: List[float]) -> Tuple[str, int, List[List[float]], List[float], List[float], List[str], List[str]]:
    """
    Funkcja wyliczająca z pliku ranking metodą topsis
    :param file_name: (str) : nazwa pliku
    :param criteria: (List[int]) : lista wybranych kryteriów
    :param metric: str : metryki
    :param weights: List[float] : lista wag podana przez użytkownika
    :return: (Tuple[str, int, List[List[float]], List[float], List[float]], str, str, List[str]) : wektor współczynników
    skoringowych jako str, liczba kryetriów, macierz znormalizowana, punkty idealne, punkty antyidealne,
    lista nazw kryetriów, lista nazw sprzętów
    """
    df = pd.read_excel(file_name)  # wczytanie excel z bazą słuchawek

    if not weights or weights is None:  # jeśli użytkownik nie podał wag (na razie się tak nie da) to wybierz je z pliku
        W = df['Wagi'].dropna().tolist()  # wektor wag
    else:
        W = weights

    W_max = df['Maksymalizacja'].dropna().tolist()  # wektor logiczny określający, które maksymalizujemy kryterium
    D = []  # macierz decyzyjna
    c_names = []  # wektor nazw kryteriów
    criteria = sorted(criteria)
    for j in df.columns:

        if j == 'Lp.' or j == 'Nazwa' or df.columns.get_loc(j) - 1 not in criteria:
            continue
        if j == 'Wagi':
            break
        D.append(df[j].tolist())
        c_names.append(j)

    c, n, N, p_ideal, p_anti_ideal = topsis(D, W, metric, W_max)  # tworzenie rankingu

    rank = []
    items_names = []
    for i in range(len(D[0])):
        rank.append((df['Nazwa'][i], c[i]))
        items_names.append(df['Nazwa'][i])

    rank.sort(key=lambda tup: tup[1], reverse=True)  # posortowanie rankingu

    rank_str = ''
    for name, score in rank:
        rank_str += name + ' : ' + '{0:1.3f}'.format(score) + '\n'  # zapis rankingu jako str

    return rank_str, n, N, p_ideal, p_anti_ideal, c_names, items_names

```

Zrzut ekranu 9 - kod metody Topsis

Metoda Topsis polega na znormalizowaniu macierzy decyzyjnej oraz wyznaczeniu punktu idealnego oraz punktu antyidealnego z wartości dostępnych ze wszystkich kryteriów. Następnie według wybranej metryki oblicza się odległość każdej z alternatyw od wyznaczonych punktów. Na tej podstawie wyznacza się ranking scoringowy.

Metoda SP-CS

```python
from typing import List, Tuple, Optional, Union
import random
import pandas as pd
from math import sqrt
import numpy as np
from scipy.spatial.distance import braycurtis, chebyshev, canberra, cityblock

Number = Union[float, int]


def sp_cs(D: List[List[Number]], W_max: Optional[List[bool]], metric: str) -> Tuple[List[float], List[Number], List[Number],
                                                                                    List[float], List[Number], List[float],
                                                                                    List[float], List[float], List[float]]:
    """
    Funkcja wyliczająca ranking metodą SP-CS
    :param D: (List[List[Number) : macierz elementów
    :param W_max: (List[bool]) : wektor maksymalizacji kryteriów
    :param metric: (str) : nazwa wykorzystywanej metryki do obliczania odległości
    :return: (Tuple[str, int, List[Number], List[Number], List[float], List[Number], List[float], List[float],
     List[float], List[float], List[str], List[str]]) : wektor współczynników skoringowych,
     punkty elementów x, punkty elementów y, punkty quo, punkty aspiracji
    """
    m = len(D[0])  # liczba elementów
    n = len(D)  # liczba kryteriow

    aspiration_idx_set = set()  # zbiór indeksów punktu aspiracji
    aspiration_value = []  # wartości punktu aspiracji
    quo_point_mean = []  # punkt quo średnia
    quo_point_median = []  # punkt quo mediana
    quo_point_random = []  # punkt quo losowo
    for j in range(n):  # dla każdego kryterium
        best_idxs = []
        if W_max[j]:  # posorotwanie od wartości
            elements_sorted = [(idx, v) for idx, v in enumerate(D[j])]
            elements_sorted.sort(key=lambda tup: tup[1], reverse=True)
        else:
            elements_sorted = [(idx, v) for idx, v in enumerate(D[j])]
            elements_sorted.sort(key=lambda tup: tup[1])
        best_value = elements_sorted[0][1]
        worst_value = elements_sorted[-1][1]
        aspiration_value.append(best_value)
        quo_point_mean.append(abs(best_value - worst_value) / 2)
        quo_point_median.append(elements_sorted[m // 2][1])
        quo_point_random.append(abs(best_value - worst_value) * random.random() + worst_value)
        best_idxs.append(elements_sorted[0][0])
        k = 1
        while best_value == elements_sorted[k][1]:
            best_idxs.append(elements_sorted[k][0])  # zebranie punktów o najlepszych wartościach
            k += 1
        for idx in best_idxs:
            aspiration_idx_set.add(idx)

    aspiration_idx = list(aspiration_idx_set)

    threshold_value = []  # wyznaczenie granicy do znalezienia punktów zdominowanych
    for j in range(n):
        if W_max[j]:
            worst_value = float('inf')
            for i in aspiration_idx:
                if D[j][i] < worst_value:
                    worst_value = D[j][i]
        else:
            worst_value = 0
            for i in aspiration_idx:
                if D[j][i] > worst_value:
                    worst_value = D[j][i]
        threshold_value.append(worst_value)

    not_dominated_idx = []  # wyznaczenie punktów niezdominowanych
    for i in range(m):
        dominated = True
        for j in range(n):
            if W_max[j]:
                if D[j][i] >= threshold_value[j]:
                    dominated = False
                    break
            else:
                if D[j][i] <= threshold_value[j]:
                    dominated = False
                    break
        if not dominated:
            not_dominated_idx.append(i)

    disrupted_aspiration_point1 = [coord * (0.9 + random.random() * 0.2) for coord in aspiration_value]
```

```
    disrupted_aspiration_point2 = [coord * (0.85 + random.random() * 0.3) for coord in aspiration_value]
    disrupted_aspiration_point3 = [coord * (0.8 + random.random() * 0.4) for coord in aspiration_value]

    data_0 = []
    data_1 = []
    for idx in not_dominated_idx:
        data_0.append(D[0][idx])
        data_1.append(D[1][idx])

    score_sum = [0. for _ in range(len(data_0))]
    for quo_point, aspiration_point in [(quo_point_mean, disrupted_aspiration_point1),
                                        (quo_point_median, disrupted_aspiration_point2),
                                        (quo_point_random, disrupted_aspiration_point3)]:
        a = (quo_point[1] - aspiration_point[1]) / (quo_point[0] - aspiration_point[0])
        b = quo_point[1] - a * quo_point[0]
        d = sqrt((quo_point[0] - aspiration_point[0]) ** 2 + (quo_point[1] - aspiration_point[1]) ** 2)
        score1 = []  # odległość znormalizowana rzutu między punktem quo a aspiracji
        score2 = []  # odległość nieznormalizowana od prostej między quo a aspiracji
        for point_idx in range(len(data_0)):
            a_p = -1 / a
            b_p = data_1[point_idx] - a_p * data_0[point_idx]
            x = (b_p - b) / (a - a_p)
            y = a * x + b
            d1 = sqrt((quo_point[0] - x) ** 2 + (quo_point[1] - y) ** 2)
            d2 = sqrt((x - aspiration_point[0]) ** 2 + (y - aspiration_point[1]) ** 2)
            if 0.99 * d < d1 + d2 < 1.01 * d:
                score1.append(d1 / d)
            elif d1 > d2:
                score1.append(1 + d2 / d)
            elif d2 > d1:
                score1.append(-d1 / d)
            if metric == "Default":
                h = sqrt((x - data_0[point_idx]) ** 2 + (y - data_1[point_idx]) ** 2)
                score2.append(h)
            else:
                projection_point_as_vector = np.asarray([x,y])
                data_point_as_vector = np.asarray([data_0[point_idx],data_1[point_idx]])
                if metric == "Bray-Curtis":
                    h = braycurtis(data_point_as_vector,projection_point_as_vector)
                    score2.append(h)
                elif metric == "Canberra":
                    h = canberra(data_point_as_vector,projection_point_as_vector)
                    score2.append(h)
                elif metric == "Chebyshev":
                    h = chebyshev(data_point_as_vector,projection_point_as_vector)
                    score2.append(h)
                elif metric == "City Block":
                    h = cityblock(data_point_as_vector,projection_point_as_vector)
                    score2.append(h)

        score2 = [-el / max(score2) for el in score2]  # normalizacja score2
        for i in range(len(score_sum)):
            score_sum[i] += score1[i] + score2[i]

    score = [el / 3 for el in score_sum]

    for idx in range(m):
        if idx not in not_dominated_idx:
            score.insert(idx, -float('inf'))

    return score, data_0, data_1, quo_point_mean, quo_point_median, quo_point_random, disrupted_aspiration_point1, \
           disrupted_aspiration_point2, disrupted_aspiration_point3

def compute_sp_cs(file_name: str, criteria: List[int], metric: str) -> Tuple[str, int, List[Number], List[Number], List[float], List[Number], List[float],
                                                                             List[float], List[float], List[float], List[str], List[str]]:
    """
    Funkcja wyliczająca z pliku ranking metodą sp-cs
    :param file_name: (str) : nazwa pliku
    :param criteria: (List[int]) : lista wybranych kryteriów
    :param metric: (str) : nazwa wykorzystywanej metryki
    :return: (Tuple[str, int, List[Number], List[Number], List[float], List[Number], List[float], List[float],
     List[float], List[float], List[str], List[str]]) : wektor współczynników skoringowych jako str, liczba kryetriów,
     punkty elementów x, punkty elementów y, punkty quo, punkty aspiracji, lista nazw kryteriów i lista nazw elementów
    """
    df = pd.read_excel(file_name)  # wczytanie excel z bazą słuchawek
    W_max = df['Maksymalizacja'].dropna().tolist()  # wektor logiczny określający, które maksymalizujemy kryterium
    D = []  # macierz decyzyjna
    c_names = []  # wektor nazw kryteriów
    criteria = sorted(criteria)
    for j in df.columns:
        if j == 'Lp.' or j == 'Nazwa' or df.columns.get_loc(j) - 1 not in criteria:
            continue
        if j == 'Wagi':
            break
        D.append(df[j].tolist())
        c_names.append(j)
    (variable) items_names: list
    items_names = []
    for i in range(len(D[0])):
        items_names.append(df['Nazwa'][i])

    score, data_0, data_1, quo_point_mean, quo_point_median, quo_point_random, disrupted_aspiration_point1, \
    disrupted_aspiration_point2, disrupted_aspiration_point3 = sp_cs(D, W_max, metric)  # tworzenie rankingu

    rank = []
    for i in range(len(D[0])):
        rank.append((items_names[i], score[i]))

    rank.sort(key=lambda tup: tup[1], reverse=True)  # posortowanie rankingu

    rank_str = ''
    for name, score in rank:
        rank_str += name + ' : ' + '{0:1.3f}'.format(score) + '\n'  # zapis rankingu jako str

    return rank_str, n, data_0, data_1, quo_point_mean, quo_point_median, quo_point_random, \
           disrupted_aspiration_point1, disrupted_aspiration_point2, disrupted_aspiration_point3, c_names, items_names
```

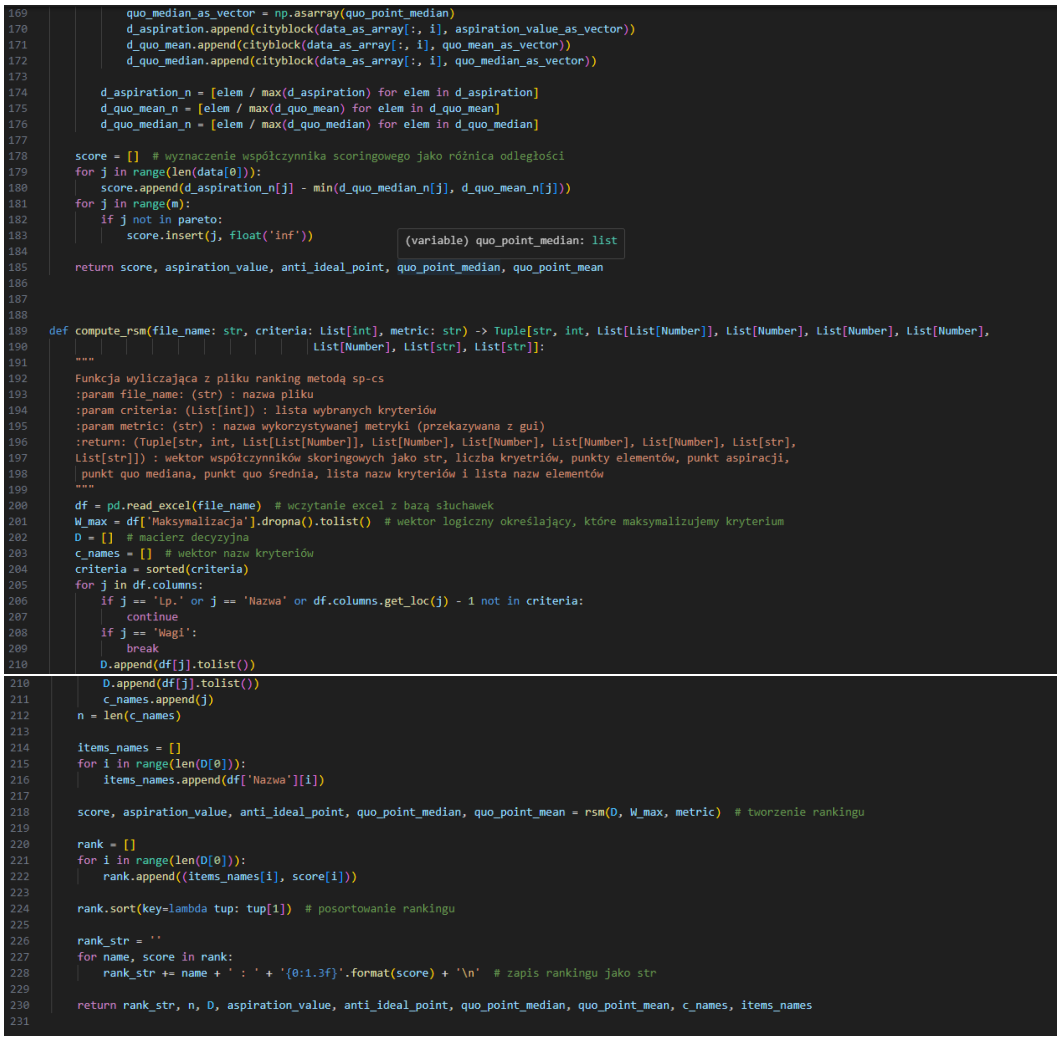
Zrzut ekranu 10 - kod metody SP-CS

Metoda SP-CS jest zaimplementowana dla dwóch kryteriów. Na podstawie macierzy decyzyjnej oblicza się punkty stałe konieczne do obliczenia rankingu. Wyznaczane są trzy zestawy punktów wyznaczających odcinek. Punkty aspiracji z niewielkim zakłóceniem oraz trzy punkty status quo, punkt średni, punkt mediany oraz punkt losowy. Na ich podstawie oblicza się odległość alternatyw od wyznaczanych przez te punkty odcinków zgodnie z wybraną metryką. Następnie, po wyznaczeniu rzutu punktu na odcinek, oblicza się, w którym miejscu tego odcinka umieszczony jest rzut. Z tych par wyników wyznacza się średnią odległość oraz średnią pozycję rzutu, co pozwala na obliczenie współczynnika scoringowego.

Metoda RSM

```python
from typing import List, Tuple, Optional, Union

import numpy as np
import pandas as pd
from math import sqrt
from scipy.spatial.distance import braycurtis, chebyshev, canberra, cityblock

Number = Union[float, int]


def rsm(D: List[List[Number]], W_max: Optional[List[bool]], metric: str) -> Tuple[List[float], List[Number], List[Number],
                                                                                  List[Number], List[Number]]:
    """
    Funkcja wyliczająca ranking metodą SP-CS
    :param D: (List[List[Number]) : macierz elementów
    :param W_max: (List[bool]) : wektor maksymalizacji kryteriów
    :param metric: (str) : nazwa wykorzystywanej metryki
    :return: (Tuple[str, int, List[Number], List[Number], List[Number], List[Number]) : wektor współczynników
    skoringowych, punkt aspiracji, punkt antyidealny, punkt quo mediana, punkt quo średnia
    """
    m = len(D[0])  # liczba elementów
    n = len(D)  # liczba kryteriow

    aspiration_idx_set = set()  # zbiór indeksów punktu aspiracji
    aspiration_value = []  # wartości punktu aspiracji
    anti_ideal_point = []  # punkt antyidealny
    opt_threshold = []  # punkt graniczny
    quo_point_mean = []  # punkt quo średnia
    quo_point_median = []  # punkt quo mediana
    for j in range(n):  # dla każdego kryterium
        best_idxs = []
        if W_max[j]:  # posorotwanie od wartości
            elements_sorted = [(idx, v) for idx, v in enumerate(D[j])]
            elements_sorted.sort(key=lambda tup: tup[1], reverse=True)
        else:
            elements_sorted = [(idx, v) for idx, v in enumerate(D[j])]
            elements_sorted.sort(key=lambda tup: tup[1])
        best_value = elements_sorted[0][1]
        worst_value = elements_sorted[-1][1]
        if W_max[j]:
            opt_threshold.append(abs(best_value - worst_value) * 0.25 + worst_value)
        else:
            opt_threshold.append(abs(best_value - worst_value) * 0.25 + best_value)
        anti_ideal_point.append(worst_value)
        aspiration_value.append(best_value)
        quo_point_mean.append(abs(best_value - worst_value) / 2)
        quo_point_median.append(elements_sorted[m // 2][1])
        best_idxs.append(elements_sorted[0][0])
        k = 1
        while best_value == elements_sorted[k][1]:
            best_idxs.append(elements_sorted[k][0])  # zebranie punktów o najlepszych wartościach
            k += 1
        for idx in best_idxs:
            aspiration_idx_set.add(idx)

    pareto = []  # wyznaczenie punktów niezdominowanych
    for i in range(m):
        dominated = True
        for j in range(n):
            if W_max[j]:
                if D[j][i] >= opt_threshold[j]:
                    dominated = False
                    break
            else:
                if D[j][i] <= opt_threshold[j]:
                    dominated = False
                    break
        if not dominated:
            pareto.append(i)

    """
    # sprawodzenie czy punkty quo nie są zdominowane
    median_is_greater = []
    for i in range(n):
        median_is_greater.append(quo_point_median[i] > quo_point_mean[i])
    if len(set(median_is_greater)) == 1:
        raise ValueError("Punkty quo zdominowane")
    """

    data = []  # wyznaczenie współrzędnych punktów niezdominowanych
    for i in range(n):
        criterion = []
        for idx in pareto:
            criterion.append(D[i][idx])
```

```
        data.append(criterion)

    if metric == "Default":
        d_square_aspiration = [0. for _ in range(len(data[0]))]  # wyznaczenie odległości punktów od punktu aspiracji
        for i in range(n):
            for j in range(len(data[0])):
                d_square_aspiration[j] += (data[i][j] - aspiration_value[i]) ** 2
        d_aspiration = [sqrt(elem) for elem in d_square_aspiration]
        d_aspiration_n = [elem / max(d_aspiration) for elem in d_aspiration]

        d_square_quo_mean = [0. for _ in range(len(data[0]))]  # wyznaczenie odległości punktów od punktu quo średniej
        for i in range(n):
            for j in range(len(data[0])):
                d_square_quo_mean[j] += (data[i][j] - quo_point_mean[i]) ** 2
        d_quo_mean = [sqrt(elem) for elem in d_square_quo_mean]
        d_quo_mean_n = [elem / max(d_quo_mean) for elem in d_quo_mean]

        d_square_quo_median = [0. for _ in range(len(data[0]))]  # wyznaczenie odległości punktów od punktu quo mediana
        for i in range(n):
            for j in range(len(data[0])):
                d_square_quo_median[j] += (data[i][j] - quo_point_median[i]) ** 2
        d_quo_median = [sqrt(elem) for elem in d_square_quo_median]
        d_quo_median_n = [elem / max(d_quo_median) for elem in d_quo_median]

    elif metric == "Bray-Curtis":
        d_aspiration = []
        d_quo_mean = []
        d_quo_median = []
        data_as_array = np.asarray(data)
        for i in range(len(data[0])):
            aspiration_value_as_vector = np.asarray(aspiration_value)
            quo_mean_as_vector = np.asarray(quo_point_mean)
            quo_median_as_vector = np.asarray(quo_point_median)
            d_aspiration.append(braycurtis(data_as_array[:, i], aspiration_value_as_vector))
            d_quo_mean.append(braycurtis(data_as_array[:, i], quo_mean_as_vector))
            d_quo_median.append(braycurtis(data_as_array[:, i], quo_median_as_vector))

        d_aspiration_n = [elem / max(d_aspiration) for elem in d_aspiration]    # normalizacja
        d_quo_mean_n = [elem / max(d_quo_mean) for elem in d_quo_mean]
        d_quo_median_n = [elem / max(d_quo_median) for elem in d_quo_median]

    elif metric == "Canberra":
        d_aspiration = []
        d_quo_mean = []
        d_quo_median = []
        data_as_array = np.asarray(data)
        for i in range(len(data[0])):
            aspiration_value_as_vector = np.asarray(aspiration_value)
            quo_mean_as_vector = np.asarray(quo_point_mean)
            quo_median_as_vector = np.asarray(quo_point_median)
            d_aspiration.append(canberra(data_as_array[:, i], aspiration_value_as_vector))
            d_quo_mean.append(canberra(data_as_array[:, i], quo_mean_as_vector))
            d_quo_median.append(canberra(data_as_array[:, i], quo_median_as_vector))

        d_aspiration_n = [elem / max(d_aspiration) for elem in d_aspiration]
        d_quo_mean_n = [elem / max(d_quo_mean) for elem in d_quo_mean]
        d_quo_median_n = [elem / max(d_quo_median) for elem in d_quo_median]

    elif metric == "Chebyshev":
        d_aspiration = []
        d_quo_mean = []
        d_quo_median = []
        data_as_array = np.asarray(data)
        for i in range(len(data[0])):
            aspiration_value_as_vector = np.asarray(aspiration_value)
            quo_mean_as_vector = np.asarray(quo_point_mean)
            quo_median_as_vector = np.asarray(quo_point_median)
            d_aspiration.append(chebyshev(data_as_array[:, i], aspiration_value_as_vector))
            d_quo_mean.append(chebyshev(data_as_array[:, i], quo_mean_as_vector))
            d_quo_median.append(chebyshev(data_as_array[:, i], quo_median_as_vector))

        d_aspiration_n = [elem / max(d_aspiration) for elem in d_aspiration]
        d_quo_mean_n = [elem / max(d_quo_mean) for elem in d_quo_mean]
        d_quo_median_n = [elem / max(d_quo_median) for elem in d_quo_median]

    elif metric == "City Block":

        d_aspiration = []
        d_quo_mean = []
        d_quo_median = []
        data_as_array = np.asarray(data)
        for i in range(len(data[0])):
            aspiration_value_as_vector = np.asarray(aspiration_value)
            quo_mean_as_vector = np.asarray(quo_point_mean)
```



```
            quo_median_as_vector = np.asarray(quo_point_median)
            d_aspiration.append(cityblock(data_as_array[:, i], aspiration_value_as_vector))
            d_quo_mean.append(cityblock(data_as_array[:, i], quo_mean_as_vector))
            d_quo_median.append(cityblock(data_as_array[:, i], quo_median_as_vector))

        d_aspiration_n = [elem / max(d_aspiration) for elem in d_aspiration]
        d_quo_mean_n = [elem / max(d_quo_mean) for elem in d_quo_mean]
        d_quo_median_n = [elem / max(d_quo_median) for elem in d_quo_median]

    score = []  # wyznaczenie współczynnika scoringowego jako różnica odległości
    for j in range(len(data[0])):
        score.append(d_aspiration_n[j] - min(d_quo_median_n[j], d_quo_mean_n[j]))
    for j in range(m):
        if j not in pareto:
            score.insert(j, float('inf'))

    return score, aspiration_value, anti_ideal_point, quo_point_median, quo_point_mean


def compute_rsm(file_name: str, criteria: List[int], metric: str) -> Tuple[str, int, List[List[Number]], List[Number], List[Number], List[Number],
                                                                           List[Number], List[str], List[str]]:
    """
    Funkcja wyliczająca z pliku ranking metodą sp-cs
    :param file_name: (str) : nazwa pliku
    :param criteria: (List[int]) : lista wybranych kryteriów
    :param metric: (str) : nazwa wykorzystywanej metryki (przekazywana z gui)
    :return: (Tuple[str, int, List[List[Number]], List[Number], List[Number], List[Number], List[Number], List[str],
    List[str]]) : wektor współczynników skoringowych jako str, liczba kryetriów, punkty elementów, punkt aspiracji,
     punkt quo mediana, punkt quo średnia, lista nazw kryteriów i lista nazw elementów
    """
    df = pd.read_excel(file_name)  # wczytanie excel z bazą słuchawek
    W_max = df['Maksymalizacja'].dropna().tolist()  # wektor logiczny określający, które maksymalizujemy kryterium
    D = []  # macierz decyzyjna
    c_names = []  # wektor nazw kryteriów
    criteria = sorted(criteria)
    for j in df.columns:
        if j == 'Lp.' or j == 'Nazwa' or df.columns.get_loc(j) - 1 not in criteria:
            continue
        if j == 'Wagi':
            break
        D.append(df[j].tolist())
        D.append(df[j].tolist())
        c_names.append(j)
    n = len(c_names)

    items_names = []
    for i in range(len(D[0])):
        items_names.append(df['Nazwa'][i])

    score, aspiration_value, anti_ideal_point, quo_point_median, quo_point_mean = rsm(D, W_max, metric)  # tworzenie rankingu

    rank = []
    for i in range(len(D[0])):
        rank.append((items_names[i], score[i]))

    rank.sort(key=lambda tup: tup[1])  # posortowanie rankingu

    rank_str = ''
    for name, score in rank:
        rank_str += name + ' : ' + '{0:1.3f}'.format(score) + '\n'  # zapis rankingu jako str

    return rank_str, n, D, aspiration_value, anti_ideal_point, quo_point_median, quo_point_mean, c_names, items_names
```

(variable) quo_point_median: list

Zrzut ekranu 11 - kod metody RSM

Metoda RSM również rozpoczyna działanie od obliczenia punktów stałych tj. punktu aspiracji, punktu antyidealnego, punktu granicznego, punktów status quo. Punkty status quo obliczane są jako punkty średniej i mediany. Następnie ze zbioru alternatyw wybiera się alternatywy niezdominowane, dla których kontynuuje się działania algorytmu. Współczynnik rankingowy obliczony jest jako różnica odległości od punktu aspiracji i minimum z odległości od punktów status quo. Odległość jest obliczana według odpowiedniej metryki.

Plik main.py

```python
import sys
from typing import List
from PyQt6.QtWidgets import QApplication, QMainWindow, QWidget, QVBoxLayout, QHBoxLayout, QMessageBox, \
    QFileDialog, QComboBox, QTableWidget, QTableWidgetItem, QTabWidget, QLabel, QPushButton, QDialog, QDialogButtonBox,\
    QCheckBox, QDoubleSpinBox
from PyQt6.QtGui import QFont
from PyQt6.QtCore import Qt, pyqtSlot, QEventLoop, pyqtSignal
import pandas as pd
from topsis import compute_topsis
from sp_cs import compute_sp_cs
from rsm import compute_rsm
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg
from matplotlib.backends.backend_qt5agg import NavigationToolbar2QT
import matplotlib.pyplot as plt
import matplotlib

matplotlib.use('TkAgg')

### Okno Główne ###

class MainWindow(QMainWindow):

    def __init__(self):
        """
        Okno główne, wyświetlające aplikacje
        """
        super(MainWindow, self).__init__()

        ### Właściwości bazy danych ###

        self.file_name = None
        self.data_0 = []
        self.data_1 = []
        self.dap1 = []
        self.dap2 = []
        self.dap3 = []
        self.quo_point_mean = []
        self.quo_point_median = []
        self.quo_point_random = []
        self.method = "TOPSIS"
        self.n = 0
        self.N = []
        self.p_ideal = []
        self.p_anti_ideal = []
        self.criteria = []
        self.items_names = []
        self.crit_numbers = [] #lista zaznaczonych kryteriów (checkboxów)
        self.crits_in_orig_file = 0
        self.checkboxes = []

        self.chosen_criteria = []
        self.chosen_metric = "Default"

        self.weights = []   # lista z wagami
        self.data_from_dialog = []

        ### Ustawienia okna ###

        self.resize(800, 600)  # rozmiar
        self.setWindowTitle('Aplikacja rankingowa')  # nazwa okna

        tabs = QTabWidget()  # zakładki Konfiguracja, Arkusz, Wykres
        tabs.setTabPosition(QTabWidget.TabPosition.North)  # pozycja zakładek
        tabs.setMovable(False)  # przemieszczanie zakładek

        tabs.addTab(Config(self), 'Konfiguracja')  # dodanie zakładki Konfiguracja
        tabs.addTab(Sheet(self), 'Arkusz kalkulacyjny')  # dodanie zakładki Arkusz
        tabs.addTab(Chart(self), 'Wykres')  # dodanie zakładek Wykres

        self.setCentralWidget(tabs)  # umieszczenie zakładek w oknie


### Zakładki ###


class Config(QWidget):

    def __init__(self, parent: MainWindow):
        """
        Zakładka z konfiguracją danych do obliczeń
        :param parent: (MainWindow) : okno rodzic
        """
        super(Config, self).__init__()

        self.parent = parent  # wskaźnik na rodzica
        self.parent.method = "TOPSIS"  # nazwa metody

        layout = QVBoxLayout()  # układ główny
        layout_config = QVBoxLayout()  # rozmieszczenie konfiguracji
        layout_choose_file = QHBoxLayout()  # rozmieszczenie układu z wyborem pliku
        layout_choose_method = QHBoxLayout()  # rozmieszczenie układu z wyborem metody
        self.layout_choose_categories = QHBoxLayout() #rozmieszczenie wyboru kryteriów z listy

        layout.setContentsMargins(20, 20, 20, 20)  # wielkość ramki
        layout.setSpacing(40)  # odległości między widżetami

        ### Układ konfiguracji ###

        label_choose_file = QLabel("Wybierz plik .xlsx z bazą przedmiotów")  # etykieta z poleceniem
        font_choose_file = label_choose_file.font()
        font_choose_file.setPointSize(12)
        label_choose_file.setFont(font_choose_file)  # ustawienie wielkości czcionki
        label_choose_file.setAlignment(Qt.AlignmentFlag.AlignLeft | Qt.AlignmentFlag.AlignTop)  # rozmieszczenie
        layout_choose_file.addWidget(label_choose_file)  # dodanie widżetu do układu
```

```
        button_choose_file = QPushButton(self)  # przycisk otwierający dialog wyboru
        button_choose_file.setText("Wybierz plik")  # nazwa przycisku
        font_button_choose_file = button_choose_file.font()
        font_button_choose_file.setPointSize(12)
        button_choose_file.setFont(font_button_choose_file)
        button_choose_file.clicked.connect(self.choose_file)  # przypisanie akcji
        layout_choose_file.addWidget(button_choose_file)  # dodanie widżetu do układu

        layout_config.addLayout(layout_choose_file)  # dodanie układu wyboru pliku do układu konfiguracji

        self.label_file_name = QLabel("Wybrany plik: ")  # etykieta z nazwą wybranego pliku
        font_file_name = self.label_file_name.font()
        font_file_name.setPointSize(12)
        self.label_file_name.setFont(font_file_name)  # ustawienie wielkości czcionki
        self.label_file_name.setAlignment(Qt.AlignmentFlag.AlignLeft | Qt.AlignmentFlag.AlignTop)  # rozmieszczenie
        layout_config.addWidget(self.label_file_name)  # dodanie widżetu do układu

        label_crits = QLabel("Wybór kryteriów:")  # etykieta
        font_crits = label_crits.font()
        font_crits.setPointSize(12)
        label_crits.setFont(font_crits)  # ustawienie wielkości czcionki
        self.layout_choose_categories.addWidget(label_crits)  # dodanie widżetu do układu

        layout_config.addLayout(self.layout_choose_categories)

        label_method_name = QLabel("Wybrana metoda:")  # etykieta z nazwą wybranej metody
        font_method_name = label_method_name.font()
        font_method_name.setPointSize(12)
        label_method_name.setFont(font_method_name)  # ustawienie wielkości czcionki
        layout_choose_method.addWidget(label_method_name)  # dodanie widżetu do układu

        combo_method = QComboBox()  # lista wyboru metod
        combo_method.addItems(["TOPSIS", "RSM", "SP-CS"])  # dostępne metody
        font_combo_method = combo_method.font()
        font_combo_method.setPointSize(12)
        combo_method.setFont(font_combo_method)
        combo_method.currentTextChanged.connect(self.choose_method)  # przypisanie akcji

        layout_choose_method.addWidget(combo_method)  # dodanie widżetu do układu

        # metryki (set visibility)
        #frame_metric = QFrame()
        #frame_metric.hide()
        layout_metric = QVBoxLayout()
        #frame_metric.setLayout(layout_metric)

        label_metric = QLabel("Wybierz metrykę: ")
        layout_metric.addWidget(label_metric)

        combo_metric = QComboBox()
        combo_metric.addItems(["Default", "Bray-Curtis", "Canberra", "Chebyshev", "City Block"])
        combo_metric.currentTextChanged.connect(self.choose_metric)
        layout_metric.addWidget(combo_metric)

        #combo_method.currentTextChanged.connect(lambda state, combobox=combo_method, frame=frame_metric:
        #                                        self.set_frame_visibility(combobox, frame))     # po zmianie na topsis ramka nie znika

        layout_config.addLayout(layout_choose_method)

        # layout schowany do frame z wyborem wag (bo qdialog chyba nie może wysyłać danych do rodzica)

        layout_config.addLayout(layout_metric)

        button_compute = QPushButton(self)  # przycisk wyliczający ranking
        button_compute.setText("Wylicz ranking")  # nazwa przycisku
        font_compute = button_compute.font()
        font_compute.setPointSize(12)
        button_compute.setFont(font_compute)
        button_compute.clicked.connect(self.compute)  # przypisanie akcji
        layout_config.addWidget(button_compute)

        label_results = QLabel("Wyniki metody:")  # etykieta z poleceniem
        font_results = label_results.font()
        font_results.setPointSize(12)
        font_results.setBold(True)
        label_results.setFont(font_results)  # ustawienie wielkości czcionki
        label_results.setAlignment(Qt.AlignmentFlag.AlignLeft | Qt.AlignmentFlag.AlignBottom)  # rozmieszczenie
        layout_config.addWidget(label_results)  # dodanie widżetu do układu

        layout.addLayout(layout_config)  # dodanie układu konfiguracji do głównego układu

        ### Układ główny ###

        self.results = QLabel("")  # etykieta z poleceniem
        self.results.setFont(QFont('Calibri', 12))  # ustawienie wielkości czcionki
        self.results.setAlignment(Qt.AlignmentFlag.AlignVCenter | Qt.AlignmentFlag.AlignHCenter)  # rozmieszczenie
        layout_config.addWidget(self.results)  # dodanie widżetu do układu

        self.setLayout(layout)  # ustanowienie układu

    ### Akcje ###

    def choose_file(self) -> None:
        """
        Wybranie pliku z danymi
        :return: None
        """
        self.clear_layout()
        self.parent.file_name = QFileDialog.getOpenFileName(self, filter="*.xlsx")[0]  # nazwa pliku
        self.label_file_name.setText("Wybrany plik: " + self.parent.file_name)  # aktualizacja etykiety
        self.parent.crits_in_orig_file = self.create_temporary_df()
        self.parent.checkboxes = [QCheckBox(f'Kryterium {i + 1}') for i in range(self.parent.crits_in_orig_file)]
        for checkbox in self.parent.checkboxes:
            self.layout_choose_categories.addWidget(checkbox)
            if self.parent.checkboxes.index(checkbox) in [0, 1]:
                checkbox.setChecked(True)
                self.parent.crit_numbers.append(self.parent.checkboxes.index(checkbox) + 1)
            checkbox.clicked.connect(self.on_checkbox_clicked)

    def create_temporary_df(self) -> int:
        df = pd.read_excel(self.parent.file_name)  # wczytanie excel z bazą słuchawek
        D = []  # macierz decyzyjna
        c_names = []  # wektor nazw kryteriów
        for j in df.columns:
```

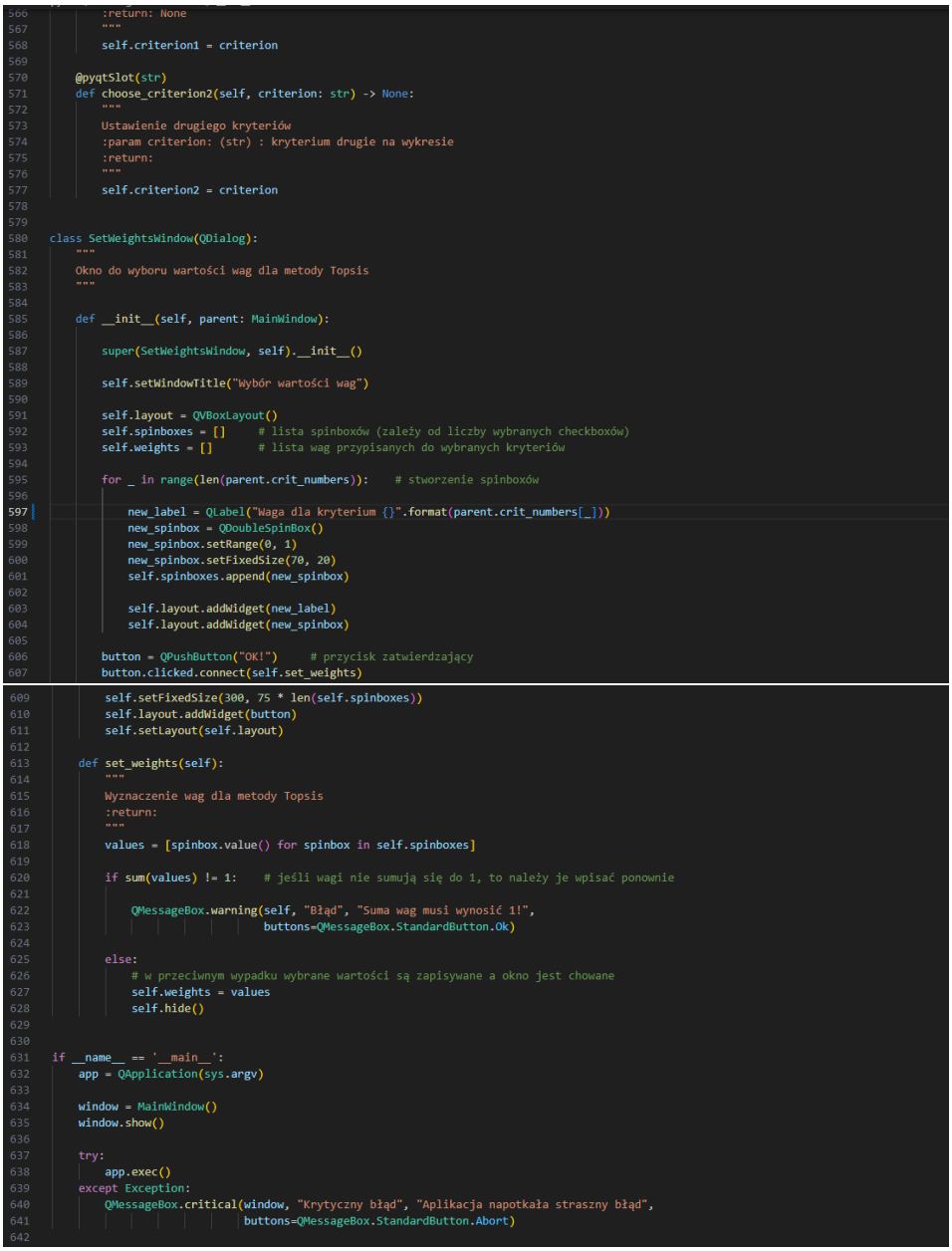
```
            if j == 'Lp.' or j == 'Nazwa':
                continue
            if j == 'Wagi':
                break
            D.append(df[j].tolist())
            c_names.append(j)
        return len(c_names)

    def clear_layout(self) -> None:
        while self.layout_choose_categories.count() != 1:
            item = self.layout_choose_categories.itemAt(self.layout_choose_categories.count() - 1)
            widget = item.widget()
            if widget and isinstance(widget, QCheckBox):
                widget.setParent(None)
                widget.deleteLater()

    @pyqtSlot(str)
    def choose_method(self, method: str) -> None:
        """
        Wybranie metody
        :param method: (str) : metoda z ComboBox
        :return: None
        """
        self.parent.method = method  # nazwa metody

    def on_checkbox_clicked(self):
        """
        Wybór kryteriów - kiedy przycisk wciśnięty, numer kryterium (nie indeks!) dodaje się do listy
        :return: None
        """
        sender_checkbox = self.sender()
        checkbox_text = sender_checkbox.text()
        checkbox_state = True if sender_checkbox.isChecked() else False
        if checkbox_state:
            self.parent.crit_numbers.append(int(checkbox_text[-1]))
        else:
            self.parent.crit_numbers.remove(int(checkbox_text[-1]))

    @pyqtSlot()
    def compute(self) -> None:
        """
        Wyliczenie rankingu
        :return: None
        """
        if self.parent.file_name is not None:

            if len(self.parent.crit_numbers) < 2:
                QMessageBox.warning(self, "Nieprawidłowe dane", "Wybierz co najmniej 2 kryteria",
                                    buttons=QMessageBox.StandardButton.Ok)

            elif self.parent.method == "TOPSIS":    # jeśli wybrano metodę topsis

                test_window = SetWeightsWindow(self.parent)
                test_window.exec()  # wyświetl okno pytające o wagi

                if test_window.isHidden():  # jeślin okno zostanie schowane (automatycznie po zatwierdzeniu wag)

                    self.parent.weights = test_window.weights   # przekaż te wagi rodzicowi
                    # wykonaj metodę
                    rank, self.parent.n, self.parent.N, self.parent.p_ideal, self.parent.p_anti_ideal, \
                    self.parent.criteria, self.parent.items_names = \
                        compute_topsis(self.parent.file_name, self.parent.crit_numbers, self.parent.chosen_metric,
                                       self.parent.weights)

            elif self.parent.method == "RSM":

                rank, self.parent.n, self.parent.N, self.parent.p_ideal, self.parent.p_anti_ideal, \
                    self.parent.quo_point_median, self.parent.quo_point_mean, \
                    self.parent.criteria, self.parent.items_names = \
                    compute_rsm(self.parent.file_name, self.parent.crit_numbers, self.parent.chosen_metric)

            elif self.parent.method == "SP-CS":

                if len(self.parent.crit_numbers) == 2:
                    rank, self.parent.n, self.parent.data_0, self.parent.data_1, self.parent.quo_point_mean, \
                        self.parent.quo_point_median, self.parent.quo_point_random, self.parent.dap1, self.parent.dap2, \
                        self.parent.dap3, self.parent.criteria, self.parent.items_names = \
                        compute_sp_cs(self.parent.file_name, self.parent.crit_numbers, self.parent.chosen_metric)
                else:
                    QMessageBox.warning(self, "Nieprawidłowe dane", "Metoda SP-CS działa tylko dla 2 kryteriów",
                                        buttons=QMessageBox.StandardButton.Ok)

            else:
                rank, self.parent.n, self.parent.N, self.parent.p_ideal, self.parent.p_anti_ideal, \
                    self.parent.criteria, self.parent.items_names = compute_topsis(self.parent.file_name)
                self.results.setText(rank)
            else:
                QMessageBox.warning(self, "Brak danych", "Najpierw załaduj dane w oknie Konfiguracja",
                                    buttons=QMessageBox.StandardButton.Ok)

    def continue_after_weights_set(self, test_window):

        rank, self.parent.n, self.parent.N, self.parent.p_ideal, self.parent.p_anti_ideal, \
        self.parent.criteria, self.parent.items_names = \
            compute_topsis(self.parent.file_name, self.parent.crit_numbers, self.parent.chosen_metric)

    def choose_metric(self, value_from_combobox):

        self.parent.chosen_metric = value_from_combobox


class Sheet(QWidget):

    def __init__(self, parent: MainWindow):
        """
        Zakładka z arkuszem danych
        :param parent: (MainWindow) : okno rodzic
        """
        super(Sheet, self).__init__()

        self.parent = parent  # wskaźnik na rodzica

        layout = QVBoxLayout()  # układ
        self.setLayout(layout)
```

```
        self.table = QTableWidget()  # widżet tabela
        layout.addWidget(self.table)

        self.button = QPushButton("Załaduj arkusz")  # przycisk na załadowanie arkusza
        self.button.clicked.connect(self.load_excel_data)  # przypisanie akcji
        layout.addWidget(self.button)

    @pyqtSlot()
    def load_excel_data(self) -> None:
        """
        Załadowanie danych z arkusza Excel
        :return: None
        """
        if self.parent.file_name is not None:  # gdy jest ścieżka
            df = pd.read_excel(self.parent.file_name)  # załadowanie danych

            df.fillna(" ", inplace=True)  # zastąpienie NaN pustym str
            self.table.setRowCount(df.shape[0])
            self.table.setColumnCount(df.shape[1])
            self.table.setHorizontalHeaderLabels(df.columns)

            for row in df.iterrows():  # wypełnianie danych
                values = row[1]
                for col_idx, value in enumerate(values):
                    table_item = QTableWidgetItem(str(value))
                    self.table.setItem(row[0], col_idx, table_item)

            self.table.setColumnWidth(2, 300)  # szerokość kolumn
        else:
            QMessageBox.warning(self, "Brak danych", "Najpierw załaduj dane w oknie Konfiguracja",
                                buttons=QMessageBox.StandardButton.Ok)  # ostrzeżenie


class Chart(QWidget):

    def __init__(self, parent: MainWindow):
        """
        Zakładka z wykresem
        :param parent: (MainWindow) : okno rodzic
        """
        super(Chart, self).__init__()
```

```
        self.parent = parent  # wskaźnik na rodzica

        self.figure = plt.figure()  # wykres
        self.canvas = FigureCanvasQTAgg(self.figure)
        self.toolbar = NavigationToolbar2QT(self.canvas, self)

        self.button = QPushButton("Narysuj wykres")  # przycisk na rysowanie wykresu
        self.button.clicked.connect(self.plot_graph)

        layout = QVBoxLayout()  # układ
        layout.addWidget(self.toolbar)
        layout.addWidget(self.canvas)
        layout.addWidget(self.button)
        self.setLayout(layout)

    @pyqtSlot()
    def plot_graph(self) -> None:
        """
        Rysowanie wykresu
        :return: None
        """
        if self.parent.file_name is not None and self.parent.n != 0:
            if self.parent.method == "TOPSIS" and self.parent.n == 2:  # rysowanie wykresu 2 zmiennych
                self.figure.clear()
                ax = self.figure.add_subplot()
                ax.clear()
                for i in range(len(self.parent.items_names)):
                    ax.scatter(self.parent.N[0][i], self.parent.N[1][i], label=self.parent.items_names[i])
                ax.scatter(self.parent.p_ideal[0], self.parent.p_ideal[1], marker="s", label="Punkt idealny")
                ax.scatter(self.parent.p_anti_ideal[0], self.parent.p_anti_ideal[1], marker="s",
                           label="Punkt antyidealny")
                ax.set(xlabel=self.parent.criteria[0], ylabel=self.parent.criteria[1],
                       title="Parametry mieszkań na tle punktów idealnych metody TOPSIS")
                ax.legend()
                self.canvas.draw()
            elif self.parent.method == "TOPSIS" and self.parent.n != 2:
                criterion_choice = CriterionChoiceDialog(self, self.parent.criteria)  # wybór kryteriów do wyrysowania
                criterion_choice.exec()
                for idx, name in enumerate(self.parent.criteria):
                    if criterion_choice.criterion1 == name:
                        idx1 = idx
                    if criterion_choice.criterion2 == name:
```

```
                        idx2 = idx
                self.figure.clear()
                ax = self.figure.add_subplot()
                ax.clear()
                for i in range(len(self.parent.items_names)):  # rysowanie wykresu dla wybranych kryteriów
                    ax.scatter(self.parent.N[idx1][i], self.parent.N[idx2][i], label=self.parent.items_names[i])
                ax.scatter(self.parent.p_ideal[idx1], self.parent.p_ideal[idx2], marker="s", label="Punkt idealny")
                ax.scatter(self.parent.p_anti_ideal[idx1], self.parent.p_anti_ideal[idx2], marker="s",
                           label="Punkt antyidealny")
                ax.set(xlabel=criterion_choice.criterion1, ylabel=criterion_choice.criterion2,
                       title="Parametry mieszkań na tle punktów idealnych metody TOPSIS")
                ax.legend()
                self.canvas.draw()
            elif self.parent.method == "SP-CS":
                self.figure.clear()
                ax = self.figure.add_subplot()
                ax.clear()
                ax.plot([self.parent.quo_point_mean[0], self.parent.dap1[0]],
                        [self.parent.quo_point_mean[1], self.parent.dap1[1]])
                ax.plot([self.parent.quo_point_median[0], self.parent.dap2[0]],
                        [self.parent.quo_point_median[1], self.parent.dap2[1]])
                ax.plot([self.parent.quo_point_random[0], self.parent.dap3[0]],
                        [self.parent.quo_point_random[1], self.parent.dap3[1]])
                ax.scatter(self.parent.quo_point_mean[0], self.parent.quo_point_mean[1], label="punkt quo średnia", marker="s")
                ax.scatter(self.parent.quo_point_median[0], self.parent.quo_point_median[1], label="punkt quo mediana", marker="s")
                ax.scatter(self.parent.quo_point_random[0], self.parent.quo_point_random[1], label="punkt quo losowy", marker="s")
                ax.scatter(
                    [self.parent.dap1[0], self.parent.dap2[0], self.parent.dap3[0]],
                    [self.parent.dap1[1], self.parent.dap2[1], self.parent.dap3[1]],
                    label="punkt aspiracji", marker="s")

```

```
                for idx in range(len(self.parent.data_0)):
                    ax.scatter(self.parent.data_0[idx], self.parent.data_1[idx], label=self.parent.items_names[idx])
                ax.set(xlabel=self.parent.criteria[0], ylabel=self.parent.criteria[1],
                       title="Krzywa szkieletowa dla metody SP-CS")
                ax.legend()
                self.canvas.draw()
            elif self.parent.method == "RSM" and self.parent.n == 2:
                self.figure.clear()
                ax = self.figure.add_subplot()
                ax.clear()
                d = [0. for _ in range(2)]
                for i in range(len(self.parent.N[0])):
                    for j in range(2):
                        d[j] = self.parent.N[j][i]
                    ax.scatter(d[0], d[1], label=self.parent.items_names[i])
                ax.scatter(self.parent.p_ideal[0], self.parent.p_ideal[1],
                           marker="s", label="Punkt idealny")
                ax.scatter(self.parent.p_anti_ideal[0], self.parent.p_anti_ideal[1],
                           marker="s", label="Punkt antyidealny")
                ax.scatter(self.parent.quo_point_mean[0], self.parent.quo_point_mean[1],
                           label="punkt quo średnia", marker="s")
                ax.scatter(self.parent.quo_point_median[0], self.parent.quo_point_median[1],
                           label="punkt quo mediana", marker="s")
                ax.set(xlabel=self.parent.criteria[0], ylabel=self.parent.criteria[1],
                       title="Wykres elementów dla metody RSM")
                ax.legend()
                self.canvas.draw()
            elif self.parent.method == "RSM" and self.parent.n == 3:
                self.figure.clear()
                ax = self.figure.add_subplot(111, projection='3d')
                ax.clear()
                d = [0. for _ in range(3)]
                for i in range(len(self.parent.N[0])):
                    for j in range(3):
                        d[j] = self.parent.N[j][i]
                    ax.scatter(d[0], d[1], d[2], label=self.parent.items_names[i])
                ax.scatter(self.parent.p_ideal[0], self.parent.p_ideal[1], self.parent.p_ideal[2],
                           marker="s", label="Punkt idealny")
                ax.scatter(self.parent.p_anti_ideal[0], self.parent.p_anti_ideal[1], self.parent.p_anti_ideal[2],
                           marker="s", label="Punkt antyidealny")
                ax.scatter(self.parent.quo_point_mean[0], self.parent.quo_point_mean[1], self.parent.quo_point_mean[2],
                           label="punkt quo średnia", marker="s")
                ax.scatter(self.parent.quo_point_median[0], self.parent.quo_point_median[1],
                           self.parent.quo_point_median[2], label="punkt quo mediana", marker="s")
                ax.set(xlabel=self.parent.criteria[0], ylabel=self.parent.criteria[1], zlabel=self.parent.criteria[2],
                       title="Wykres elementów dla metody RSM")
                ax.legend(prop={'size': 5})
                self.canvas.draw()
        else:
            QMessageBox.warning(self, "Brak danych", "Najpierw załaduj i wylicz dane w oknie Konfiguracja",
                                buttons=QMessageBox.StandardButton.Ok)  # ostrzeżenie


class CriterionChoiceDialog(QDialog):

    def __init__(self, parent: MainWindow, criteria: List[str]):
        """
        Okno dialogowe, które pyta o kryteria do wyrysowania
        :param parent: (MainWindow) : okno rodzic
        :param criteria: (List[str]) : lista kryteriów dostępnych do wyboru
        """
        super().__init__(parent)

        self.setModal(False)
        self.setFixedSize(300, 150)
        self.setWindowTitle("Wybór kryteriów")

        self.criteria = criteria
        self.criterion1 = self.criteria[0]
        self.criterion2 = self.criteria[0]

        QBtn = QDialogButtonBox.StandardButton.Ok

        self.buttonBox = QDialogButtonBox(QBtn)
        self.buttonBox.accepted.connect(self.accept)

        self.layout = QVBoxLayout()
        label = QLabel("Wybierz dwa kryteria do narysowania wykresu")
        self.layout.addWidget(label)

        self.layout_choice1 = QHBoxLayout()
        self.layout_choice2 = QHBoxLayout()

        criterion1 = QLabel("Kryterium 1: ")
        combo_criterion1 = QComboBox()  # lista wyboru metod
        combo_criterion1.addItems(self.criteria)  # dostępne metody
        combo_criterion1.currentTextChanged.connect(self.choose_criterion1)  # przypisanie akcji

        self.layout_choice1.addWidget(criterion1)
        self.layout_choice1.addWidget(combo_criterion1)

        criterion2 = QLabel("Kryterium 2: ")
        combo_criterion2 = QComboBox()  # lista wyboru metod
        combo_criterion2.addItems(self.criteria)  # dostępne metody
        combo_criterion2.currentTextChanged.connect(self.choose_criterion2)  # przypisanie akcji

        self.layout_choice2.addWidget(criterion2)
        self.layout_choice2.addWidget(combo_criterion2)

        self.layout.addLayout(self.layout_choice1)
        self.layout.addLayout(self.layout_choice2)

        self.
            (method) def setLayout(a0: QLayout | None) -> None

        self.setLayout(self.layout)

        self.show()

    ### Akcje ###

    @pyqtSlot(str)
    def choose_criterion1(self, criterion: str) -> None:
        """
        Ustawienie pierwszego kryteriów
        :param criterion: (str) : kryterium pierwsze na wykresie
```



```
        :return: None
        """
        self.criterion1 = criterion

    @pyqtSlot(str)
    def choose_criterion2(self, criterion: str) -> None:
        """
        Ustawienie drugiego kryteriów
        :param criterion: (str) : kryterium drugie na wykresie
        :return:
        """
        self.criterion2 = criterion


class SetWeightsWindow(QDialog):
    """
    Okno do wyboru wartości wag dla metody Topsis
    """

    def __init__(self, parent: MainWindow):

        super(SetWeightsWindow, self).__init__()

        self.setWindowTitle("Wybór wartości wag")

        self.layout = QVBoxLayout()
        self.spinboxes = []     # lista spinboxów (zależy od liczby wybranych checkboxów)
        self.weights = []       # lista wag przypisanych do wybranych kryteriów

        for _ in range(len(parent.crit_numbers)):    # stworzenie spinboxów

            new_label = QLabel("Waga dla kryterium {}".format(parent.crit_numbers[_]))
            new_spinbox = QDoubleSpinBox()
            new_spinbox.setRange(0, 1)
            new_spinbox.setFixedSize(70, 20)
            self.spinboxes.append(new_spinbox)

            self.layout.addWidget(new_label)
            self.layout.addWidget(new_spinbox)

        button = QPushButton("OK!")     # przycisk zatwierdzający
        button.clicked.connect(self.set_weights)
        self.setFixedSize(300, 75 * len(self.spinboxes))
        self.layout.addWidget(button)
        self.setLayout(self.layout)

    def set_weights(self):
        """
        Wyznaczenie wag dla metody Topsis
        :return:
        """
        values = [spinbox.value() for spinbox in self.spinboxes]

        if sum(values) != 1:    # jeśli wagi nie sumują się do 1, to należy je wpisać ponownie

            QMessageBox.warning(self, "Błąd", "Suma wag musi wynosić 1!",
                                buttons=QMessageBox.StandardButton.Ok)

        else:
            # w przeciwnym wypadku wybrane wartości są zapisywane a okno jest chowane
            self.weights = values
            self.hide()


if __name__ == '__main__':
    app = QApplication(sys.argv)

    window = MainWindow()
    window.show()

    try:
        app.exec()
    except Exception:
        QMessageBox.critical(window, "Krytyczny błąd", "Aplikacja napotkała straszny błąd",
                             buttons=QMessageBox.StandardButton.Abort)

```

Zrzut ekranu 12 - kod pliku main.py

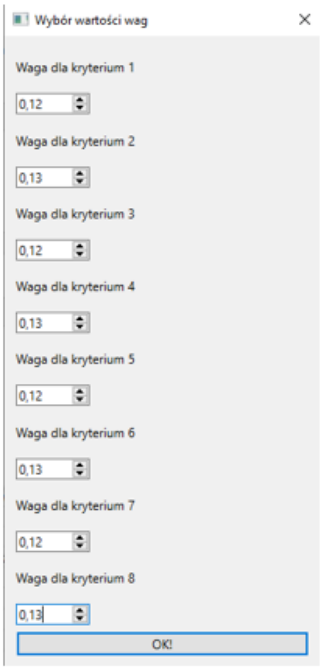
Plik main łaczy ze sobą wszystkie 3 metody. Odpowiada też za stworzenie interfejsu użytkownika i budowę całej aplikacji.

6. Uzyskane wyniki

Ilość możliwych do wybrania metod, kryteriów oraz metryk sprawia, że nie jest możliwe porównanie wszystkich kombinacji, dlatego poniżej zaprezentujemy kilka porównań dla zaimplementowanych metod.

Wpływ wag na metodę Topsis

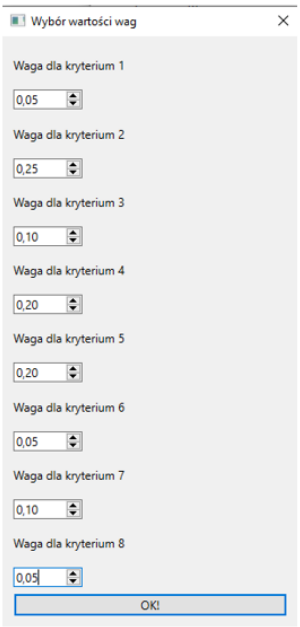
Porównanie zmiany wag na wyniki otrzymane metodą Topsis. Test przeprowadzono na wybranych wszystkich kryteriach.



Wawer : 0.576
Ursynów 1 : 0.566
Włochy : 0.549
Ursynów 2 : 0.535
Rembertów : 0.533
Wesoła : 0.529
Praga-Północ : 0.518
Wilanów : 0.507
Białołęka 1 : 0.491
Białołęka 2 : 0.490
Śródmieście 2 : 0.490
Śródmieście 1 : 0.486
Mokotów 1 : 0.472
Wola 3 : 0.471
Wola 2 : 0.453
Wola 1 : 0.453
Targówek : 0.445
Mokotów 2 : 0.443
Mokotów 3 : 0.431
Żoliborz 2 : 0.426
Żoliborz 1 : 0.425
Praga-Południe : 0.418

Zrzut ekranu 13 - wybrane wagi oraz uzyskane wyniki

W tej iteracji wagi wybrano w miarę równe i uniwersalne.



Śródmieście 2 : 0.643
Śródmieście 1 : 0.640
Wola 1 : 0.628
Wola 2 : 0.627
Wola 3 : 0.621
Praga-Północ : 0.615
Mokotów 1 : 0.607
Mokotów 2 : 0.593
Mokotów 3 : 0.571
Żoliborz 2 : 0.567
Praga-Południe : 0.566
Żoliborz 1 : 0.565
Ursynów 1 : 0.561
Ursynów 2 : 0.550
Ochota : 0.539
Targówek : 0.472
Ursus : 0.430
Włochy : 0.422
Wawer : 0.417
Bielany : 0.394
Wesoła : 0.376
Rembertów : 0.373

Zrzut ekranu 14 - wybrane wagi oraz uzyskane wyniki

Wagi wybrano z perspektywy studenta, któremu zależy na niskiej cenie, lokalizacji i transporcie.

Wnioski:

W tym porównaniu widać znaczący wpływ wyboru wag na wyniki. Zwycięzca z pierwszej iteracji jest jednym z ostatnich pozycji w drugiej iteracji. W obydwóch rankingach na wysokiej pozycji utrzymuj się Praga-Północ, która pozostała na stabilnej pozycji. Z uwagi na ilość danych odczytywanie z wykresu byłoby mocno utrudnione.

Wpływ wybranej metryki na metodę SP-CS

Przetestowano jak zmiana wybranej metryki wpływa na wyniki na podstawie algorytmu SP-CS. W tym teście wybrano losowe kryteria (Cena za metr kwadratowy, opłata za media)

Żoliborz 2 : 0.331	Wola 3 : 0.446	Żoliborz 2 : 0.314
Wola 3 : 0.321	Żoliborz 2 : 0.437	Wola 3 : 0.313
Mokotów 1 : 0.284	Mokotów 1 : 0.401	Wola 2 : 0.241
Żoliborz 1 : 0.266	Praga-Północ : 0.371	Żoliborz 1 : 0.240
Wola 2 : 0.259	Wola 2 : 0.369	Mokotów 1 : 0.222
Praga-Północ : 0.256	Żoliborz 1 : 0.353	Praga-Północ : 0.193
Włochy : 0.201	Włochy : 0.291	Śródmieście 1 : 0.167
Wola 1 : 0.170	Wola 1 : 0.283	Włochy : 0.161
Śródmieście 1 : 0.140	Śródmieście 1 : 0.281	Wola 1 : 0.108
Ursynów 1 : 0.112	Ursynów 1 : 0.184	Ursynów 1 : -0.018
Wilanów : 0.027	Wilanów : 0.131	Śródmieście 2 : -0.033
Białołęka 1 : 0.026	Śródmieście 2 : 0.121	Wilanów : -0.039
Ursus : 0.025	Białołęka 1 : 0.113	Białołęka 1 : -0.084
Śródmieście 2 : 0.019	Ursus : 0.103	Ursus : -0.108
Mokotów 2 : -0.033	Mokotów 2 : 0.019	Mokotów 2 : -0.220
Ursynów 2 : -0.151	Ursynów 2 : -0.096	Ursynów 2 : -0.385
Mokotów 3 : -0.274	Ochota : -0.248	Ochota : -0.456
Ochota : -0.280	Mokotów 3 : -0.265	Mokotów 3 : -0.513
Wesoła : -0.410	Wesoła : -0.301	Wesoła : -0.547
Białołęka 2 : -0.413	Białołęka 2 : -0.358	Białołęka 2 : -0.680
Bielany : -0.415	Bielany : -0.388	Bielany : -0.755

Zrzut ekranu 15 - Wyniki uzyskane algorytmem SP-CS, dla metryki a) domyślnej, b)Bary-Curtis c)Canberra

Wnioski:

Dla metryki domyślnej a metryki Barya-Curtisa wyniki są niemal identyczne. Konkretne alternatywy różnią się maksymalnie kilka pozycji między sobą. Metryka Canberra daje bardziej odmienne wyniki, ale ogólny trend jest zachowany. Wybrana metryka ma wpływ na wyniki, choć ogólny trend jest zachowany, wyniki różnią się kolejnością pozycji.

Wpływ wybranych kryteriów na metodę RSM.

Dla metody RSM przetestowano wpływ wybranych kryteriów na uzyskane wyniki. W dwóch iteracjach wybrano po cztery losowe kryteria.

Zrzut ekranu 16 - wybrane kryteria w pierwszej iteracji

Zrzut ekranu 17 - wybrane kryteria w drugiej iteracji

Wawer : -0.532
Wesoła : -0.513
Rembertów : -0.472
Wilanów : -0.431
Włochy : -0.317
Białołęka 2 : -0.186
Białołęka 1 : -0.186
Ursynów 2 : 0.288
Ursynów 1 : 0.288
Bielany : 0.313
Targówek : 0.313
Bemowo : 0.337
Praga-Północ : 0.403
Praga-Południe : 0.628
Ochota : 0.629
Wola 3 : 0.653
Wola 1 : 0.653
Wola 2 : 0.653
Ursus : 0.686
Żoliborz 1 : 0.701
Żoliborz 2 : 0.701

Wola 3 : -0.872
Żoliborz 2 : -0.768
Śródmieście 1 : -0.766
Żoliborz 1 : -0.527
Wola 2 : -0.403
Praga-Północ : -0.152
Wola 1 : -0.151
Mokotów 1 : -0.151
Śródmieście 2 : -0.151
Włochy : 0.217
Wawer : 0.238
Praga-Południe : 0.239
Wesoła : 0.276
Białołęka 2 : 0.277
Bielany : 0.281
Bemowo : 0.308
Targówek : 0.308
Ursynów 1 : 0.349
Ursynów 2 : 0.398
Mokotów 3 : 0.400
Białołeka 1 : 0.425

Zrzut ekranu 18 uzyskane wyniki w pierwszej i drugiej iteracji

Wnioski:

Wpływ wybranych kryteriów jest bardzo duży. Otrzymane wyniki są zupełnie inne. Niektóre alternatywy są bardzo wysoko według różnych kryteriów, co sprawia, że gdy to kryterium jest wybrane ich współczynnik jest bardzo wysoki w porównaniu z innymi alternatywami. Gdy sprzyjające w ten sposób kryteria zostaną ominięte, najwyższe wyniki mogą okazać się jednymi z najgorszych w innym zestawieniu.

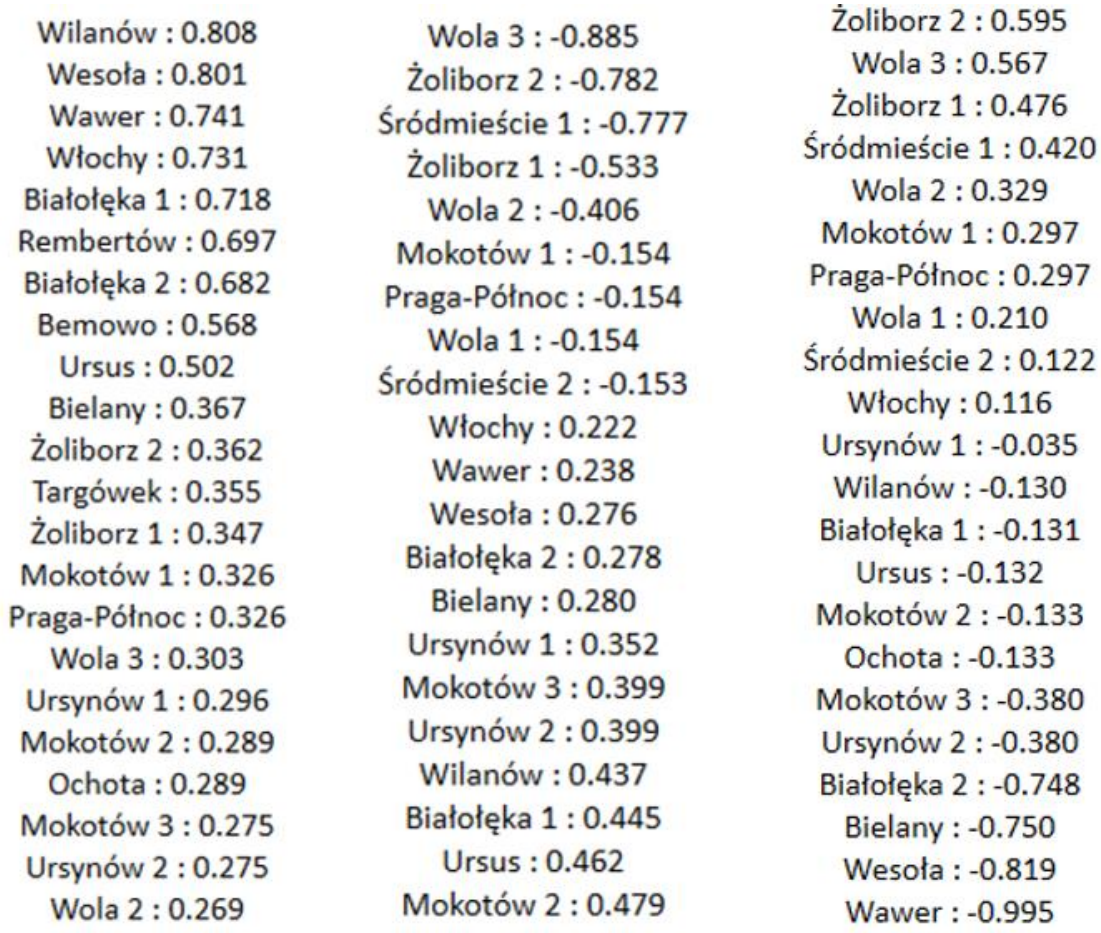
Schemat porównań

Kolejno porównywać będziemy identyczne ustawienia dla różnych metod, aby zaobserwować różnice w wynikach. Z uwagi na metodę SP-CS wybrane będą 2 kryteria (ta metoda działa tylko dla dwóch kryteriów), a wagi w metodzie Topsis dobrane na podobnym poziomie z małymi odchyleniami

Porównanie 1

Parametry porównania:

- Kryteria: Dostępność transportu, Opłaty za media
- Wagi metody Topsis: 0.5, 0.5
- Metryka: Domyślna



Zrzut ekranu 19 - Wyniki uzyskane dla metryki domyślnej, algorytmami a) Topsis b) RSM c) SP-CS

Wnioski:

Pomimo ustawienia równych wag dla obydwu kryteriów wyniki otrzymane metodą Topsis różnią się od wyników z pozostałych metod. Algorytm SP-CS i RSM dały niemal takie same wyniki. Ponownie jedyne różnice to zamienione pozycje obok siebie, różnice nie przewyższają kilku pozycji.

Porównanie 2

Parametry porównania:

- Kryteria: Bliskość natury, Metraż
- Wagi metody Topsis: 0.3, 0.7
- Metryka: Chebyshev

Mokotów 1 : 0.889	Wesoła : -0.739	Mokotów 1 : 0.702
Wola 3 : 0.753	Mokotów 1 : -0.655	Wola 3 : 0.593
Żoliborz 2 : 0.733	Ursus : -0.196	Żoliborz 2 : 0.424
Żoliborz 1 : 0.696	Wilanów : -0.196	Praga-Północ : 0.412
Praga-Północ : 0.667	Wola 3 : -0.196	Ursus : 0.327
Śródmieście 1 : 0.636	Praga-Północ : 0.034	Żoliborz 1 : 0.298
Śródmieście 2 : 0.636	Wawer : 0.093	Śródmieście 1 : 0.235
Mokotów 2 : 0.604	Ursynów 1 : 0.149	Śródmieście 2 : 0.235
Ursus : 0.567	Żoliborz 2 : 0.149	Ursynów 1 : 0.121
Wola 2 : 0.557	Białołęka 1 : 0.236	Mokotów 2 : 0.108
Wesoła : 0.556	Żoliborz 1 : 0.307	Wilanów : 0.062
Ochota : 0.556	Białołęka 2 : 0.323	Wola 2 : 0.056
Wola 1 : 0.517	Włochy : 0.323	Wesoła : 0.045
Mokotów 3 : 0.515	Ochota : 0.392	Bemowo : -0.036
Praga-Południe : 0.505	Praga-Południe : 0.413	Targówek : -0.036
Bielany : 0.482	Wola 1 : 0.441	Wola 1 : -0.118
Bemowo : 0.473	Bielany : 0.455	Ursynów 2 : -0.161
Targówek : 0.473	Mokotów 3 : 0.455	Praga-Południe : -0.176
Wilanów : 0.466	Śródmieście 1 : 0.466	Białołęka 1 : -0.209
Ursynów 1 : 0.460	Śródmieście 2 : 0.466	Białołęka 2 : -0.290
Ursynów 2 : 0.418	Bemowo : 0.525	Włochy : -0.290
Białołęka 1 : 0.340	Targówek : 0.525	Bielany : -0.292

Zrzut ekranu 20 - Wyniki uzyskane dla metryki domyślnej, algorytmami a) Topsis b) RSM c) SP-CS

Wnioski:

Dla tego porównania i wybranych wag najbliżej siebie są wyniki otrzymane z metody Topsis i SP-CS, mimo, że różnią się trochę od siebie, to trend jest zachowany. Metoda RSM daje trochę inne wyniki, wciąż zgadza się mieszkanie z Mokotowa1, ale pierwsze miejsce dla tej metody, zajmuje dosyć niskie pozycje w pozostałych dwóch metodach.

Porównanie 3

Parametry porównania:

- Kryteria: Odległość od centrum, Liczba pokoi
- Wagi metody Topsis: 0.4, 0,6
- Metryka: Canberra

Żoliborz 1 : 1.000	Żoliborz 1 : -1.000	Żoliborz 1 : 0.713
Żoliborz 2 : 1.000	Żoliborz 2 : -1.000	Żoliborz 2 : 0.713
Ochota : 0.804	Ochota : -0.483	Mokotów 1 : 0.540
Mokotów 1 : 0.722	Mokotów 1 : -0.222	Ochota : 0.352
Wola 3 : 0.596	Wola 3 : 0.238	Mokotów 2 : 0.300
Mokotów 2 : 0.556	Mokotów 2 : 0.294	Mokotów 3 : 0.300
Mokotów 3 : 0.556	Mokotów 3 : 0.294	Śródmieście 1 : 0.300
Śródmieście 1 : 0.556	Śródmieście 1 : 0.294	Śródmieście 2 : 0.300
Śródmieście 2 : 0.556	Śródmieście 2 : 0.294	Wola 3 : 0.291
Praga-Północ : 0.512	Wesoła : 0.409	Bielany : 0.155
Ursus : 0.444	Ursynów 1 : 0.452	Praga-Południe : 0.155
Bielany : 0.435	Ursus : 0.500	Wola 1 : 0.155
Praga-Południe : 0.435	Praga-Północ : 0.550	Wola 2 : 0.155
Wola 1 : 0.435	Bielany : 0.755	Praga-Północ : 0.041
Wola 2 : 0.435	Praga-Południe : 0.755	Ursus : -0.216
Ursynów 1 : 0.387	Wola 1 : 0.755	Bemowo : -0.355
Wesoła : 0.334	Wola 2 : 0.755	Targówek : -0.355
Bemowo : 0.278	Bemowo : inf	Ursynów 1 : -0.487
Targówek : 0.278	Białołęka 1 : inf	Ursynów 2 : -0.680
Ursynów 2 : 0.215	Białołęka 2 : inf	Wesoła : -0.778
Wilanów : 0.157	Rembertów : inf	Wilanów : -1.010
Białołęka 1 : 0.102	Targówek : inf	Białołęka 1 : -1.272

Zrzut ekranu 21- Wyniki uzyskane dla metryki domyślnej, algorytmami a) Topsis b) RSM c) SP-CS

Wnioski:

W tym porównaniu wyniki dla wszystkich trzech metod są bardzo podobne. Wszystkie wyniki są na bliskich pozycjach. Może to wynikać z podobnych wag wybranych kryteriów. Wartości *inf* oznacza, że dana alternatywa jest zdominowana.

7. Wnioski i dyskusja wyników

Poszczególne eksperymenty i wynikające z nich wnioski zostały podsumowane w punkcie szóstym. Całościowo warto zwrócić uwagę, że dla poszczególnych wywołań, z różnymi parametrami, możemy zaobserwować zwracanie podobnych rezultatów na zbliżonych pozycjach. Oczywiście nie pokrywają się one w pełni, ale wynika to z trzech różnych podejść do wyznaczenia rankingu i z elementu losowego, który jest uwzględniany przy SPCS i RSM. Wpływ na różnice ma przede wszystkim sposób wyznaczania odległości – w każdej metodzie korzystamy z różnych punktów odniesienia, odrzucamy też niektóre rozwiązania na podstawie innych kryteriów (w Topsis – klas odniesienia, a SPCS i RSM – podziału na punkty zdominowane i niezdominowane).

Wprowadzone przez nas modyfikacje do algorytmów, tj. różne metryki i sposoby wyboru kryteriów, mają wpływ na wyniki końcowe. O ile metryki są często małym składnikiem powodującym zmiany i zachowują się dosyć przewidywalnie, tak kryteria mają kluczowy wpływ na ostateczny ranking. Jest to

konsekwencja priorytetyzacji pewnych własności mieszkań przez użytkownika aplikacji – ranking promuje rozwiązania, które najbardziej odpowiadają jego oczekiwaniom.

Wszystkie z trzech zaimplementowanych metod mają pewne zalety i wady. Każda z nich jednak jest przydatnym narzędziem, które można wykorzystać w problemach rzeczywistych (po przyjęciu odpowiednich założeń i uproszczeń), aby stworzyć własne narzędzie wspomagania decyzji. Kiedy zależy nam na prostocie i dosyć dużej elastyczności, warto rozważyć algorytm Topsis, którego implementacja nie sprawia większych kłopotów przez wzgląd na swoją intuicyjność, a możliwość modyfikacji kryteriów i wag pomaga potencjalnemu użytkownikowi na bardziej szczegółowe porównania. Z drugiej zaś strony, metody SPCS i RSM uwzględniają więcej punktów odniesienia, przez co mogą wskazywać lepiej dopasowane rozwiązania.

8. Podział zadań

Imię i nazwisko	Zadanie
Konrad Flis	Koncepcja: dobór kryteriów do problemu
	Dane: zebranie danych i utworzenie pliku Excel
	Modyfikacja GUI: możliwość wyboru kryteriów na podstawie ich liczby wczytanej z pliku
	Modyfikacja GUI: kryteria domyślne
	Kod: adaptacja algorytmów do obsługi dynamicznie wybieranych kryteriów i ich zmian w trakcie działania algorytmu
	Kod: testowanie
	Sprawozdanie: przebiegi algorytmów
	Sprawozdanie: częściowe testowanie i porównanie wyników algorytmów
	Sprawozdanie: korekta
	Prezentacja: utworzenie szablonu
Jan Gallina	Prezentacja: opis problemu i danych
	Kod: Dodanie metryk dla metody SP-CS
	Kod: Poprawa wyboru metryk zgodnie z zasadą DRY
	Sprawozdanie: Szkielet sprawozdania
	Sprawozdanie: Cel projektu
	Sprawozdanie: opis problemu i zbioru danych
	Sprawozdanie: Opis działania GUI
	Sprawozdanie: Opis działania algorytmów
	Sprawozdanie: badanie wpływu zmiany parametrów na wyniki
	Sprawozdanie: porównywanie metod
Mateusz Gołąbek	Kod: metoda TOPSIS w wersji pierwszej
	Kod: metoda SP-CS w wersji pierwszej
	Kod: metoda RSM w wersji pierwszej
	Kod: GUI w wersji pierwszej
	Kod: okna ostrzeżeń
	Koncepcja: pomysł i dobór kryteriów do problemu
	Prezentacja: GUI

Maria Jagintowicz	Kod: wprowadzenie możliwości wyboru metryk do liczenia odległości w metodzie TOPSIS, RSM
	Modyfikacja GUI: wprowadzanie możliwości wyboru metryki przez użytkownika
	Modyfikacja GUI: Wprowadzenie możliwości wprowadzenia wartości wag przez użytkownika, jeśli została wybrana metoda TOPSIS
	Kod: aktualizacja metody TOPSIS, aby funkcja pobierała wartości wag wprowadzane przez użytkownika
	Prezentacja: metryki, możliwość wyboru wag w GUI, wyniki, porównanie metod
	Koncepcja: pomysł i dobór kryteriów do problemu