

```
In [ ]: #excel fileből kiveszem a szükséges adatokat

import googlemaps
import pandas as pd
import numpy as np
from tqdm import tqdm # progress bar
import googlemaps
gmaps = googlemaps.Client(key = 'AIzaSyB2CKH1EvRhGai4s9d9fRDti7SfPYtx0d4')
data = pd.read_excel(r'/Users/Kende/Cloud_Documents/Gyerekek/Kende/Egyetem/BME/MSc/HUHA2 projekt/Számítások/Szallitasi optimalizacio.xlsx')
df = pd.DataFrame(data, columns = ['ID', 'Szelessegi fok', 'Hosszusagi fok', 'TSZH termeles [kg/het]']) #definiáloma tszh-t is majd későbbre
```

```
In [ ]: df = np.asarray(df)
df = df[~np.isnan(df).any(axis = 1)]
print(df)
```

```
In [ ]: routes = []

for kiindulo_pontok in df:
    kiindulo_ID = kiindulo_pontok[0]
    for vegpontok in df:
        vegpont_ID = vegpontok[0]
        if kiindulo_ID != vegpont_ID:
            routes.append([kiindulo_ID, vegpont_ID, kiindulo_pontok[1], kiindulo_pontok[2], vegpontok[1], vegpontok[2]])

routes = np.asarray(routes)
print(routes)
```

```
In [ ]: import csv

data_2 = pd.read_csv(r'/Users/Kende/Cloud_Documents/Gyerekek/Kende/Egyetem/BME/MSc/HUHA2 projekt/Python_kodok/Kicsit lopott utvonalak.csv')
df_2 = pd.DataFrame(data_2, columns = ["Kezdo ID", "Veg ID", "Kezdo szelesseg", "Kezdo hosszusag", "Vegso szelesseg", "Vegso hosszusag", "Tavolsag", "Ido"]) #definiáloma
df_2 = np.asarray(df_2)

with open("/Users/Kende/Cloud_Documents/Gyerekek/Kende/Egyetem/BME/MSc/HUHA2 projekt/Python_kodok/Utvonal_adatok.csv", 'w') as csvfile:
    csvwriter = csv.writer(csvfile)
    csvwriter.writerow(["Kezdo ID", "Veg ID", "Kezdo szelesseg", "Kezdo hosszusag", "Vegso szelesseg", "Vegso hosszusag", "Tavolsag", "Ido"])
    tavok = []

    for route in tqdm(routes): # progress bart lehet adni a futtatáshoz
        benne_van = False
        for sor in df_2:
            if route[0] == sor[0] and route[1] == sor[1]:
                benne_van = True
                break

        if not benne_van:
            adatok = gmaps.distance_matrix((route[2], route[3]), (route[4], route[5]), mode = 'driving', departure_time = 'now', traffic_model = 'best_guess')

            #távolság és idő kiválasztása
            tavolsag = adatok['rows'][0]['elements'][0]['distance']['text']
            ido = adatok['rows'][0]['elements'][0]['duration']['text']
            tavok.append([route[0], route[1], route[2], route[3], route[4], route[5], tavolsag, ido])
            csvwriter.writerow([route[0], route[1], route[2], route[3], route[4], route[5], tavolsag, ido])

#todo: le kellene csökkenteni a feleslegesen hosszú útvonalakat, mert azok nem lesznek hasznosak
```

```

# ----- IMPORTS -----

import pandas as pd
import numpy as np
from tqdm import tqdm
import csv
import sys
import matplotlib.pyplot as plt

# ----- CLASSES -----

class Csucs:
    def __init__(self, ID, lat, long, tszh, tavolsag_huhatol):
        self.ID = ID
        self.tszh = tszh
        self.lat = lat
        self.long = long
        self.tavolsag_huhatol = tavolsag_huhatol
        self.latogatasok = 0

    def update(self, elvitt_szemet):
        self.tszh = self.tszh - elvitt_szemet
        self.latogatasok = self.latogatasok + 1

class Kukasauto:
    def __init__(self):
        self.kapacitas = 8000
        self.latogatott_csucskok = []
        self.distance = 0
        self.duration = 0
        self.kerulet_beluli_tavolsag = 0
        self.tmp_dist = []
        self.tmp_dur = []

    def update(self, elvitt_szemet, distance, duration, latogatott_csucskok, kerulet_beluli_tavolsag):
        self.kapacitas = self.kapacitas - elvitt_szemet
        self.distance = self.distance + distance
        self.duration = self.duration + duration
        self.latogatott_csucskok.append(latogatott_csucskok)
        self.kerulet_beluli_tavolsag = self.kerulet_beluli_tavolsag + kerulet_beluli_tavolsag
        self.tmp_dist.append(distance)
        self.tmp_dur.append(duration)

# ----- LOAD IN DATA -----

def read_location_data(path):
    data = pd.read_excel(path)
    df = pd.DataFrame(data, columns = ['ID', 'Szelessegi fok', 'Hosszusagi fok', 'TSZH termeles [kg/het]', 'TSZH liearis suruseg [kg/km]']) #definiálom tszh-t is majd később
    df = df[~np.isnan(df).any(axis = 1)]
    return df

def read_route_data(path):
    data = pd.read_excel(path)
    tavok = pd.DataFrame(data, columns = ["Kezdo ID", "Veg ID", "Kezdo szelessegi", "Kezdo hosszusagi", "Vegso szelessegi", "Vegso hosszusagi", "Tavolsag [km]", "Ido [mins]"])
    tavok = np.asarray(tavok, dtype = object)
    return tavok

# ---- GENERATE CSUCSKOK ----

def get_csucskok(df, tavok, huha_befogadas):
    huhabol_indul = []
    for R in tavok:
        if R[0] == 1:
            huhabol_indul.append(R)
    huhabol_indul.sort(key = lambda x: x[6])

    csucskok = []
    i = 0
    while huha_befogadas > 0 and i < len(huhabol_indul):
        tavolsag_huhatol = huhabol_indul[i][6]

        ID = huhabol_indul[i][1]
        terület = df[np.where(df[:,0] == ID)][0]
        tszh = terület[3]

        i = i + 1
        if huha_befogadas - tszh <= 0:
            if i > len(huhabol_indul):
                print('ez soha nem kene kiiradjon :)')
            else:
                huha_befogadas = huha_befogadas - tszh
                if ID != 1:
                    csucs = Csucs(terület[0], terület[1], terület[2], tszh, tavolsag_huhatol)
                    csucskok.append(csucs)

    return csucskok

# ----- DRIVER CODE -----

def distance(HUHA, cel, csucskok):
    p1 = np.array([HUHA.lat, HUHA.long])
    p2 = np.array([cel.lat, cel.long])
    p3 = np.array([csucskok.lat, csucskok.long])
    return np.linalg.norm(np.cross(p2-p1, p1-p3))/np.linalg.norm(p2-p1)

def find_best_routes(HUHA, csucskok, tavok, df, huha_befogadas, r):
    with open("./Eredmenyek/Optimalizalasi_eredmenyek_keruleti_szemettel.csv", 'w') as csvfile:
        csvwriter = csv.writer(csvfile)
        eredmenyek = []
        csvwriter.writerow(['Latogatott IDk', 'Távolság', 'Idő tartam', 'Maradék kapacitás', 'Kerületen belüli távolság'])
        sum_distance = 0
        sum_duration = 0

        while huha_befogadas > 0:
            kukasauto = Kukasauto()
            kukasauto.update(0, 0, 0, HUHA, 0)
            celpontok = sorted(csucskok, key = lambda x: x.tavolsag_huhatol, reverse = True)
            i = 0
            while True:
                cel = celpontok[i]
                if cel.tszh > 0:
                    break
                else:
                    i = i + 1

```

```

elvitt_szemet = min(kukasauto.kapacitas, cel.tszh)
keruleten_beluli_tavolsag = elvitt_szemet / (df[np.where(df[:,0] == cel.ID)][0][4])
idotartam = tavok[np.where(np.logical_and(tavok[:,0] == HUHA.ID, tavok[:, 1] == cel.ID))[0][0]][7]
kukasauto.update(elvitt_szemet, cel.tavolsag_huhatol, idotartam, cel, keruleten_beluli_tavolsag)
cel.update(elvitt_szemet)
huha_befogadas = huha_befogadas - elvitt_szemet

if kukasauto.kapacitas > 0 :
    while kukasauto.kapacitas > 0:
        legkozelebbi_csucs = cel
        region_of_interest = []

        for csucs in csucskok:
            d = distance(HUHA, cel, csucs)

            if d < r and csucs.tszh > 0:

                region_of_interest.append(csucs)

        if len(region_of_interest) > 0:
            minimum = 40
            for csucs in region_of_interest:
                celtol_valo_tavolsag = tavok[np.where(np.logical_and(tavok[:,0] == cel.ID, tavok[:, 1] == csucs.ID))[0][0]][6] # ??? nem tudom, hogy ez Á gy jÃ-e, nem
                if celtol_valo_tavolsag < minimum and csucs != cel:
                    minimum = celtol_valo_tavolsag
                    legkozelebbi_csucs = csucs
            if legkozelebbi_csucs == cel:
                tavolsag_huhatol = tavok[np.where(np.logical_and(tavok[:,0] == cel.ID, tavok[:, 1] == HUHA.ID))[0][0]][6]
                idotartam = tavok[np.where(np.logical_and(tavok[:,0] == cel.ID, tavok[:, 1] == HUHA.ID))[0][0]][7]
                kukasauto.update(0, tavolsag_huhatol, idotartam, HUHA, 0)
                break
            else:
                elvitt_szemet = min(kukasauto.kapacitas, legkozelebbi_csucs.tszh)
                keruleten_beluli_tavolsag = elvitt_szemet / (df[np.where(df[:,0] == cel.ID)][0][4])
                idotartam = tavok[np.where(np.logical_and(tavok[:,0] == cel.ID, tavok[:, 1] == legkozelebbi_csucs.ID))[0][0]][7]

                cel = legkozelebbi_csucs
                kukasauto.update(elvitt_szemet, minimum, idotartam, legkozelebbi_csucs, keruleten_beluli_tavolsag)
                legkozelebbi_csucs.update(elvitt_szemet)
                huha_befogadas = huha_befogadas - elvitt_szemet

        else:
            tavolsag_huhatol = tavok[np.where(np.logical_and(tavok[:,0] == cel.ID, tavok[:, 1] == HUHA.ID))[0][0]][6]
            idotartam = tavok[np.where(np.logical_and(tavok[:,0] == cel.ID, tavok[:, 1] == HUHA.ID))[0][0]][7]
            kukasauto.update(0, tavolsag_huhatol, idotartam, HUHA, 0)
            break

if kukasauto.kapacitas == 0:
    tavolsag_huhatol = tavok[np.where(np.logical_and(tavok[:,0] == cel.ID, tavok[:, 1] == HUHA.ID))[0][0]][6]
    idotartam = tavok[np.where(np.logical_and(tavok[:,0] == cel.ID, tavok[:, 1] == HUHA.ID))[0][0]][7]
    kukasauto.update(0, tavolsag_huhatol, idotartam, HUHA, 0)

latogatott_csucskok = ""
i = 0
for i, latogatott_csucs in enumerate(kukasauto.latogatott_csucskok):
    if i == len(kukasauto.latogatott_csucskok) - 1:
        latogatott_csucskok = latogatott_csucskok + str(latogatott_csucs.ID)
    else:
        latogatott_csucskok = latogatott_csucskok + str(latogatott_csucs.ID) + " -> "

eredmenyek.append([kukasauto.latogatott_csucskok, kukasauto.distance, kukasauto.duration, kukasauto.kapacitas, kukasauto.keruleten_beluli_tavolsag])
csvwriter.writerow([latogatott_csucskok, kukasauto.distance, kukasauto.duration, kukasauto.kapacitas, kukasauto.keruleten_beluli_tavolsag])
x = 0
for csucsi in csucskok:
    if csucsi.tszh == 0:
        x = x + 1

sum_distance = sum_distance + kukasauto.distance + kukasauto.keruleten_beluli_tavolsag
sum_duration = sum_duration + kukasauto.duration

maradek_szemet_sum = 0
for maradek in csucskok:
    maradek_szemet_sum = maradek_szemet_sum + maradek.tszh

if maradek_szemet_sum == 0:
    break

return sum_distance, sum_duration

# ---- VISUALIZATION ----
def visualize_r(all_distances, all_durations, rs):

    plt.plot(rs, all_distances, 'r', label = 'Megtett Ã*t [km] ')
    plt.plot(rs, all_durations, 'b', label = 'SzÃksÃges ido [perc] ')
    plt.xlabel("KeresÃsi tÃvolsÃg")
    plt.legend()
    plt.show()

def visualize_graph(csucskok, df, tavok, HUHA):

    points = []

    served_IDs = []

    for csucs in csucskok:
        served_IDs.append(csucs.ID)

    first_s = True
    first_ns = True
    for area in df:
        if area[0] in served_IDs:
            if first_s:
                plt.scatter(area[2], area[1], c='g', marker='o', label="KiszolgÃlt terÃletek")
                first_s = False
            else:
                plt.scatter(area[2], area[1], c='g', marker='o')
        else :
            if area[0] == HUHA.ID:
                plt.scatter(HUHA.long, HUHA.lat, c='b', s=100, label = 'HUHA', marker='*')
            else:
                if first_ns:
                    plt.scatter(area[2], area[1], c='r', marker='x', label="Nem kiszolgÃlt terÃletek")
                    first_ns = False

```

```

        else:
            plt.scatter(area[2], area[1], c='r', marker='x')
plt.xlabel("latitudinÅlis koordinÅta")
plt.ylabel("longitudinÅlis koordinÅta")
plt.legend()
plt.show()

def main():

    df = read_location_data("./Bemenet/Szallitasi optimalizacio.xlsx")
    tavok = read_route_data("./Bemenet/Google utvonalak.xlsx")

    huha_befogadas = 230000*1000/52

    HUHA = Csucs(df[0][0], df[0][1], df[0][2], huha_befogadas, 0)

    all_distances = []
    all_durations = []
    rs = []

    # Az optimalis r meghatarazashoz szukseges kod
    # (150 szer futatja a utvonaltervezest kulonbozo folyamatosan novekvo r ertekekkel)
    for i in tqdm(range(150)):
        r = i / 1000

        csucskok = get_csucskok(df, tavok, huha_befogadas)

        distance_r, duration_r = find_best_routes(HUHA, csucskok, tavok, df, huha_befogadas, r)

        all_distances.append(distance_r)
        all_durations.append(duration_r)
        rs.append(r)

    visualize_r(all_distances, all_durations, rs)
    visualize_graph(csucskok, df, tavok, HUHA)

    print("Best r:", rs[all_distances.index(min(all_distances))])

    # Ha legjobb eredmÅnyt szeretnÅnk megkapni
    # best_r = 0.045
    # distance, duration = find_best_routes(csucskok, df, tavok, best_r)

if __name__ == '__main__':

    main()

```