```
In [ ]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib as mpl
        import pymongo as pm
        import pprint
        from enum import Enum
        from datetime import datetime, timedelta
        import geojson
        import seaborn as sb

        #Connect to the DB
        client = pm.MongoClient('bigdatadb.polito.it',
                                ssl=True,
                                authSource = 'carsharing',
                                username = 'ictts',
                                password ='Ict4SM22!',
                                tlsAllowInvalidCertificates=True)
        db = client['carsharing']

        #Choose the DB to use
        Ictts_enj_p_booking = db['ictts_enjoy_PermanentBookings']
        Ictts_p_booking = db['ictts_PermanentBookings'] #PermanentBookings

        with open("TorinoZonescol.geojson") as f: #GeoJson file with the zones of Turin
            gj = geojson.load(f)

        #Function to get the origin and destination zones of a booking using geoWithin operator of MongoDB
        #weekday - days 2-6 -morning 6-12 - afternoon 12-20
        #weekend - days 1,7 -morning 6-12 - afternoon 12-20
        def weekday_piper(start_hour,end_hour,origin_zone,destination_zone):
          return [
            { "$project":
             {
               "hour":{"$hour":"$init_date"},
               "day":{"$dayOfWeek":"$init_date"},
               "init_loc":1,
               "final_loc":1,
               "init_time":1
               }
            },
            { "$match": {
              "day":{"$gte":2,"$lte":6},
              "hour":{"$gte":start_hour,"$lte":end_hour},
              "init_loc":{"$geoWithin":{"$geometry":{"type":"MultiPolygon","coordinates":origin_zone}}},
              "final_loc":{"$geoWithin":{"$geometry":{"type":"MultiPolygon","coordinates":destination_zone}}}}
            },
            { "$count":"total"}
          ]

        def weekend_piper(start_hour,end_hour,origin_zone,destination_zone):
          return [
            { "$project":
             {
               "hour":{"$hour":"$init_date"},
               "day":{"$dayOfWeek":"$init_date"},
               "init_loc":1,
               "final_loc":1,
               "init_time":1
               }
            },
            { "$match":{
              "day":1 and 7,
              "hour":{"$gte":start_hour,"$lte":end_hour},
              "init_loc":{"$geoWithin":{"$geometry":{"type":"MultiPolygon","coordinates":origin_zone}}},
              "final_loc":{"$geoWithin":{"$geometry":{"type":"MultiPolygon","coordinates":destination_zone}}}}},
            {"$count": "total"}
          ]

        #Function to get the total number of bookings in a specific zone in a specific time interval
        #orining zone and destination zone are retieved from the geojson file
        #origin_zone and destination_zone are the coordinates of the zone as a list of lists which is passed to the geoWithin operat
        def extract_od_matrix(start_hour =1 , end_hour =23, pipeline=[{}]):
          OD_matrix = [([0]*23) for i in range(23) ]
          for i in range(23) :
            orig_zone = gj["features"][i]["geometry"]["coordinates"]
            for j in range(23) :
              dest_zone = gj["features"][j]["geometry"]["coordinates"]
              result = list ( Ictts_p_booking.aggregate(pipeline(start_hour,end_hour,orig_zone,dest_zone)))
              if( len(result ) > 0):
                OD_matrix[i][j] = result[0]["total"]
              else :
                OD_matrix[i][j] = 0
          output_df = pd.DataFrame ( OD_matrix )
          output_df.columns =["Q"+f"{i:03d}" for i in range(1, 24) ]
          output_df['index'] =["Q"+f"{i:03d}" for i in range(1, 24) ]
          output_df = output_df.set_index('index', drop = True ).rename_axis( None )
          return output_df

        #now we can use the function to get the OD matrix for the morning and afternoon of weekdays and weekends
        weekday_morning = extract_od_matrix(6,12,weekday_piper)
        weekday_afternoon = extract_od_matrix(12,20,weekday_piper)
        weekend_morning = extract_od_matrix(6,12,weekend_piper)
        weekend_afternoon = extract_od_matrix(12,20,weekend_piper)
```

```
booking_OD_matrix = []
booking_OD_matrix.append(weekday_morning)
booking_OD_matrix.append(weekday_afternoon)
booking_OD_matrix.append(weekend_morning)
booking_OD_matrix.append(weekend_afternoon)

weekday_morning.to_csv("weekday_morning.csv")
weekday_afternoon.to_csv("weekday_afternoon.csv")
weekend_morning.to_csv("weekend_morning.csv")
weekend_afternoon.to_csv("weekend_afternoon.csv")

bookingFigureTitles = ["Weekday Morning","Weekday Afternoon","Weekend Morning","Weekend Afternoon"]

#filter the IMQ matrix with different parameters
IMQ = pd.read_csv("spostamentiTorino.csv")
copyIMQ = IMQ.copy()
IMQ_OD_matrices = []

filtered_data = copyIMQ[copyIMQ['SESSO']==1]
pivot_table1 = filtered_data.pivot_table( index ='COD_ZONA_PAR',
                                         columns='COD_ZONA_ARR',values='ID_INT',aggfunc =len ,fill_value =0)
pivot_table1.to_csv("IMQ_OD_1"+".csv")

filtered_data = copyIMQ[copyIMQ['SESSO']==2]
pivot_table2 = filtered_data.pivot_table( index ='COD_ZONA_PAR',
                                         columns='COD_ZONA_ARR',values='ID_INT',aggfunc =len ,fill_value =0)
pivot_table2.to_csv("IMQ_OD_2"+".csv")

filtered_data = copyIMQ[copyIMQ['FASCIA_ETA']==1]
pivot_table3 = filtered_data.pivot_table( index ='COD_ZONA_PAR',
                                        columns='COD_ZONA_ARR',values='ID_INT',aggfunc =len ,fill_value =0)
pivot_table3.to_csv("IMQ_OD_3"+".csv")

filtered_data = copyIMQ[copyIMQ['FASCIA_ETA']==2]
pivot_table4 = filtered_data.pivot_table( index ='COD_ZONA_PAR',
                                        columns='COD_ZONA_ARR',values='ID_INT',aggfunc =len ,fill_value =0)
pivot_table4.to_csv("IMQ_OD_4"+".csv")

filtered_data = copyIMQ[copyIMQ['FASCIA_ETA']==3]
pivot_table5 = filtered_data.pivot_table( index ='COD_ZONA_PAR',
                                        columns='COD_ZONA_ARR',values='ID_INT',aggfunc =len ,fill_value =0)
pivot_table5.to_csv("IMQ_OD_5"+".csv")

filtered_data = copyIMQ[copyIMQ['FASCIA_ETA']==4]
pivot_table6 = filtered_data.pivot_table( index ='COD_ZONA_PAR',
                                        columns='COD_ZONA_ARR',values='ID_INT',aggfunc =len ,fill_value =0)
pivot_table6.to_csv("IMQ_OD_6"+".csv")

filtered_data = copyIMQ[copyIMQ['SCOPO']==1]
pivot_table7 = filtered_data.pivot_table( index ='COD_ZONA_PAR',
                                        columns='COD_ZONA_ARR',values='ID_INT',aggfunc =len ,fill_value =0)
pivot_table7.to_csv("IMQ_OD_7"+".csv")

filtered_data = copyIMQ[copyIMQ['SCOPO']==3]
pivot_table8 = filtered_data.pivot_table( index ='COD_ZONA_PAR',
                                        columns='COD_ZONA_ARR',values='ID_INT',aggfunc =len ,fill_value =0)
pivot_table8.to_csv("IMQ_OD_8"+".csv")

filtered_data = copyIMQ[copyIMQ['SCOPO']==4]
pivot_table9 = filtered_data.pivot_table( index ='COD_ZONA_PAR',
                                        columns='COD_ZONA_ARR',values='ID_INT',aggfunc =len ,fill_value =0)
pivot_table9.to_csv("IMQ_OD_9"+".csv")

filtered_data = copyIMQ[copyIMQ['SCOPO']==7]
pivot_table10 = filtered_data.pivot_table( index ='COD_ZONA_PAR',
                                        columns='COD_ZONA_ARR',values='ID_INT',aggfunc =len ,fill_value =0)
pivot_table10.to_csv("IMQ_OD_11"+".csv")

filtered_data = copyIMQ[copyIMQ['SCOPO']==8]
pivot_table11 = filtered_data.pivot_table( index ='COD_ZONA_PAR',
                                        columns='COD_ZONA_ARR',values='ID_INT',aggfunc =len ,fill_value =0)
pivot_table11.to_csv("IMQ_OD_10"+".csv")


#-------------------------------------------------------------------------------------------------------------
#male-work and female-work
filter1 = (copyIMQ['SESSO']==1)
filter2 = (copyIMQ['SCOPO']==1)
men_work =  copyIMQ[filter1 & filter2]
men_work_pivot = men_work.pivot_table( index ='COD_ZONA_PAR',
                                      columns='COD_ZONA_ARR',values='ID_INT',aggfunc =len ,fill_value =0)
men_work_pivot.to_csv("IMQ_OD_11"+".csv")


filter1 = (copyIMQ['SESSO']==2)
filter2 = (copyIMQ['SCOPO']==1)
women_work =  copyIMQ[filter1 & filter2]
women_work_pivot = women_work.pivot_table( index ='COD_ZONA_PAR',
                                          columns='COD_ZONA_ARR',values='ID_INT',aggfunc =len ,fill_value =0)
women_work_pivot.to_csv("IMQ_OD_12"+".csv")
#-------------------------------------------------------------------------------------------------------------

IMQ_OD_matrices.append(pivot_table1)
IMQ_OD_matrices.append(pivot_table2)
IMQ_OD_matrices.append(pivot_table3)
IMQ_OD_matrices.append(pivot_table4)
IMQ_OD_matrices.append(pivot_table5)
IMQ_OD_matrices.append(pivot_table6)
IMQ_OD_matrices.append(pivot_table7)
IMQ_OD_matrices.append(pivot_table8)
IMQ_OD_matrices.append(pivot_table9)
IMQ_OD_matrices.append(pivot_table10)
IMQ_OD_matrices.append(pivot_table11)
#---------------------------------------------------------------
#male-work and female-work
IMQ_OD_matrices.append(men_work_pivot)
IMQ_OD_matrices.append(women_work_pivot)
#---------------------------------------------------------------
# to be used in the heatmap as labels also in meshgrid titles
figureTitles = ["Filter on males", "Filter on Females",
                "Filter on 11-19 Age Range", "Filter on 20-29 Age Range",
                "Filter on 50-64 Age Range", "Filter on +65 Age Range",
                "Filter on Work (Motivation)", "Filter on Study (Motivation)",
                "Filter on Shopping (Motivation)", "Filter on Sport or Leisure (Motivation)",
                "Filter on Go Back Home (Motivation)",
                "Filter on Men going to work", "Filter on Women going to work"]


#---------------------------------------------------------------
#calculating the distance between each two matrices
# • Comparison between the OD Matrices
# • L2 distance between the OD matrices formula is given by:
# L2_distance = sqrt( sum( ( normalized_matrix1 - normalized_matrix2 )^2 ) )
def L2_distance( matrix1 , matrix2 ):
 # Ensure matrices have the same dimensions
 # assert is used to check if a condition is true, if not, the program will raise an error
 assert matrix1.shape == matrix2.shape , "Matrices must have the same dimensions"
 normalized_matrix1 = matrix1 / matrix1.sum( axis =1, keepdims = True )
 normalized_matrix2 = matrix2 / matrix2.sum( axis =1, keepdims = True )

 # Calculate the squared differences between corresponding cells
 squared_diff = np.square( normalized_matrix1 - normalized_matrix2 )
 # Sum the squared differences
 sum_squared_diff = np.sum( squared_diff )
 l2_distance = np.sqrt( sum_squared_diff )
 return l2_distance

#creating a list of lists to store the distances between each two matrices
distances = [([0]* len( booking_OD_matrix )) for i in range(len( IMQ_OD_matrices ))]

#calculating the distance between each two matrices
for i, imq in enumerate( IMQ_OD_matrices ):
  for j, rental in enumerate( booking_OD_matrix ):
    distance = L2_distance(imq.values , rental.values )
    distances[i][j] = distance

#  plotting the heatmap of the distances
hm = sb.heatmap( data =np.array( distances ), annot = True, fmt=".3f",
                cmap="inferno" ,xticklabels=bookingFigureTitles, yticklabels=figureTitles)#rainbow
hm.set_xticklabels(hm.get_xticklabels(), rotation=0, fontsize=6)
hm.set_yticklabels(hm.get_yticklabels(), rotation=0, fontsize=6)
plt.title("L2 distance between the OD matrices")
plt.savefig("L2_distance.png", dpi=300)
plt.show()
plt.close()

#---------------------------------------------------------------
# Plotting the OD Matrices using 3D surface plots and meshgrids
def plot_matrix( od_matrix , title ):
 x, y = np.meshgrid( range(od_matrix.shape[0]), range(od_matrix.shape[1]))

 fig = plt.figure(dpi =300 , figsize =(8 , 8))
 ax = fig.add_subplot(111 , projection ='3d')
 ax.plot_surface(x, y, od_matrix , cmap ='inferno')#rainbow
 ax.set_xlabel('Origin ')
 ax.set_ylabel('Destination ')
 ax.set_zlabel('# of Trips')
 ax.set_title( title )

# Customize tick positions for both x and y axes
 ax.set_xticks(np.arange(1,24))
 ax.set_yticks(np.arange(1,24))
 ax.tick_params(axis='both', which='major', labelsize=4)
 ax. view_init ( elev =45 , azim =70)

 plt.savefig(title+".png", dpi=300)
#  plt.show()

#plotting the OD matrices
# calling the function plot_matrix for each matrix of the booking_OD_matrix and IMQ_OD_matrices
for i, element in enumerate( booking_OD_matrix):
  plot_matrix( booking_OD_matrix[i].values , bookingFigureTitles[i])

for i, od_matrix in enumerate( IMQ_OD_matrices ):
  plot_matrix( IMQ_OD_matrices[i].values , figureTitles[i])
```

In [ ]: