LRTP DI/DB Threshold Creation and Process

# Introduction

This project was an assignment from Betsy Harvey to calculate the average travel times to four categories of destinations for the MPO in order to support EJ analysis based on the MPO’s regional travel demand model. Work started in late January 2021 with a deadline of August 2021. Several directions were attempted - please see previous documentation for those attempts - and the process documented here was agreed upon by Betsy Harvey and Marty Milkovits.

## Project Purpose

To determine an estimate the average travel time to work, healthcare, higher education, and essential services for the Boston Region MPO and to flag which TAZs have access above or below the average travel time for the MPO.

## Types of Destinations

To get the average travel time for the MPO of important destinations in order to determine access, four types of destinations were identified to be analysed:

* **Work**
* **Health Care**
  + Community Health Centers (Non Administrative) - MassGIS
  + Hospitals - MassGIS
  + Medical Clinics - MA Department of Health
* **Higher Education**
  + Location: MassGIS
  + Enrollment: NCES
  + Percent commuters: school websites, College Board, US News and World Report
* **Essential Services**
  + Grocery Stores
    - as defined by ReferenceUSA, via MAPC, as Meat Markets, Fish and Seafood Markets, All Other Specialty Food Stores, Supermarkets/Other Grocery, Fruit & Vegetable Markets, Warehouse Clubs & Supercenters, Department Stores
  + Farmers Markets - MassGIS
  + Police Stations - MassGIS
  + Fire Stations - MassGIS
  + City/Town Halls - MassGIS
  + Public Libraries - MassGIS
  + Licensed Retail Pharmacies - Massachusetts Department of Health
  + Post Offices - USPS
  + Licensed Medical Clinics - Massachusetts Department of Health
  + Acute Care Hospitals - MassGIS
  + Community Health Centers - MassGIS

# Calculating Average Travel Time to Work

Unlike the other categories of destinations (healthcare, higher education, and essential services), travel time to work is included in Census data. Therefore, the process for calculating average travel time to work differs from calculating the average travel time to the other categories of destination. See below for the methodology to calculate the average travel time to work for the MPO.

**Average Travel Time to Work**

Notes:

* Paul said to use Towns but there is some data suppression in that geography level so must use PUMAs. This means that some fineagling must be done to make sure that the user only uses the parts of the PUMAs that overlap with the MPO.

Datasets:

1. 2019 5YR est ACS - B08136 & B08301
   1. B08136 - sum of all travel times of those counted (aggregate)
   2. B08301 - sum of all people per means of transportation to work

Methodology:

1. Sum B08136 and B08301 by PUMA for the MPO
   1. Categories of Summation: Vehicles, Transit, Rapid Transit, Bus.
2. Then divide: B08136 MPO Sum / B08301 MPO Sum which will give the average travel time.

# Calculating Average Travel Time to Other Destination Categories

* Create a feature class for each destination type (healthcare, higher education, essential services) that contains the point locations, types, and capacity data.
* For each TAZ, find out how many (and ideally which) destinations of each type are within a buffer of the TAZ centroid.
* Flag each TAZ with whether it meets the threshold for the number of destinations within a buffer around the TAZ centroid for the type of destination.
* Use model skims for travel times from TAZ centroid to TAZ centroid for all of the following modes: Drive (SOV as proxy), Transit, Rapid Transit, Bus, Walk.
* Then pick the minimum time to reach a flagged TAZ and weight by TAZ population and use that for calculation of the average travel time for MPO.
* Then flag origin TAZs as whether their minimum distance to a flagged TAZ is within the average threshold for MPO.

## Getting and Creating Destination Data

While the data and its sources are above, this section is intended to explain what the end state of the destination data is. A few notes about retrieving data - while all of the MassGIS and MAPC data can be directly downloaded from the web (and filtered if needed), the USPS data, MDH, and Higher Education data was more complicated.

For the Post Office location data - a web scraper was used to clean downloaded pages of the USPS website’s Post Office Locator web app. In addition to downloading each page of data/locations one by one, several starting point locations were needed in order to get the full range of the MPO given that there is no place in the MPO where the whole rest of the MPO is within 100 miles (as the crow flies). The web app is located here: <https://tools.usps.com/find-location.htm>

The web scraper script used can be found here: M:\LRTP\Post\_Offices\bS\_loop\_PO.py, and in the appendix of this document. The output ends up being a CSV that has the names and addresses of the entire MPO. If repeated post offices have not been deleted, this is the time to do that. At this point, the post offices addresses from the CSV were loaded into ArcMap and the MassGIS Address Geocoder was used to locate each post office to a spatial location (or nearby if the address was not included in the MassGIS Master Address File (MAF)).

The data from MDH (clinics and pharmacies) had to be geocoded as well as the downloads were tables of addresses.

Higher Education data was prepared by [Betsy Harvey](mailto:eharvey@ctps.org)- who manually found the commuter percentage for all institutions that MassGIS includes in its Colleges and Universities layer. She also obtained the enrollment data from NCES which was joined to the MassGIS data but some updates were conducted manually to make sure all the institutions that had a passable commuter percentage threshold were given NCES enrollment data.

Lastly, for the healthcare data, especially for the clinics and community health centers there is some overlap between layers. Compare the layers to make sure that the same clinic or CHC isn’t being counted twice, and if it is, remove the offending record from one point layer. Note - hospitals often have complexes with clinics or CHCs or pharmacies, these do not need to be removed.

Since the point is to assign the destination data to TAZs or TAZ buffers - you need to make sure that the TAZs are filtered to just be those in the MPO and any TAZs that are split (like for example 1495 where the TAZ crosses a town boundary) are merged back together before doing any buffering or assignment of destinations.

Once all the data has been retrieved and cleaned, move onto the next section.

## Assigning Destination Data to TAZs

Create centroids of the TAZs and buffer the centroids in two different distances:

* For Higher Ed - 5 mile radius circular buffer
* For Healthcare and Essential Services - 1 mile radius circular buffer

Calculate the number of destinations per TAZ and per TAZ buffer (radius of buffer depending on destination type).

**Getting Points per Polygon**

This article walks you through how to get the number of points within a polygon.

<https://support.esri.com/en/technical-article/000008599>

This method was used for getting the number of destinations per TAZ. If you want to use a circular buffer, you can use the buffers as the polygons instead of TAZs.

## Determining what TAZs Count as Destinations

If the area of a TAZ is greater than the area of a 1 mile (or 5 mile in the Higher Ed scenario) buffer around its centroid, then instead of using the number of destinations within the buffer, use the number of destinations within the actual TAZ. Essentially this creates a minimum area of Pi square miles.

For Higher Ed, if one or more Higher Ed points are within the 5 mile buffer of a TAZs centroid, that TAZ counts as a destination TAZ for the Higher Ed category. E.g. The threshold is 1 Higher Ed location within a 5 mile buffer of a TAZ centroid.

For Healthcare, if one or more Healthcare points are within the 1 mile buffer of a TAZs centroid, that TAZ counts as a destination TAZ for the Healthcare category. E.g. The threshold is 1 Healthcare location within a 1 mile buffer of a TAZ centroid.

For Essential Services, if two different categories (Civil, Healthcare, Food) and four plus total destinations are within the TAZ or the TAZ buffer, it counts as an essential services TAZ. We are aiming for diverse essential service access for each destination TAZ.

((Food\_Count > 0 And HealthCare\_Count > 0) OR (Food\_Count > 0 AND Civil\_Count > 0) OR (Civil\_Count > 0 AND HealthCare\_Count > 0)) AND (Food\_Count + HealthCare\_Count + Civil\_Count) >= 4

## Calculating Average Travel Time

### Get the Data for Calculation

Export skims from the Model into OMXs to use in the calculation script. The following skim matrices: (note: in model data, all .omx is replaced with an .mtx before converted to OMX)

/Out:

* Walk\_skim.omx

/AM:

* A\_DAT\_for\_Boat\_tr\_skim.omx
* A\_DAT\_for\_CommRail\_tr\_skim.omx
* A\_DAT\_for\_LocalBus\_tr\_skim.omx
* A\_DAT\_for\_RapidTransit\_tr\_skim.omx
* SOV\_skim.omx
* WAT\_for\_All\_tr\_skim.omx

/PM:

* A\_DAT\_for\_Boat\_tr\_skim.omx
* A\_DAT\_for\_CommRail\_tr\_skim.omx
* A\_DAT\_for\_LocalBus\_tr\_skim.omx
* A\_DAT\_for\_RapidTransit\_tr\_skim.omx
* SOV\_skim.omx
* WAT\_for\_All\_tr\_skim.omx

Determined for usage based on the following website:

<https://sites.google.com/ctps.org/ask-ed/tdm/input-data/data-dictionaries>

Export CSV of the attribute table from the TAZ shapefile that has the counts of every destination type within each TAZ centroid’s buffers (5 or 1 mile depending on the type) and save it to the Out folder. This way you can create a list of what TAZ’s pass the destination threshold (as seen above in Determining what TAZs Count as Destinations) through the python script that calculates the average travel time of the MPO for each type of destination. Need the following fields: ['taz', 'ES\_FLAG\_ALL','HIED\_Count5', 'HLTH\_COUNT1m'].

Export CSV with the total population in each TAZ.

Export the OMX for trip tables - AfterSC\_AM\_FinalTables.mtx and AfterSC\_MD\_FinalTables.mtx to use for weighting the DAT mode skims by trips to create a DAT all transit skim.

### Calculate Script

Calculate MPO Average Travel Time for Driving, Transit, Bus, Rapid Transit, and Walking by taking for each TAZ origin the shortest distance to the closest flagged destination TAZ (not including itself), weighting by population, summing per mode, and dividing by the total population of the MPO.

* The model skims do not change day to day and are calculated by time period (e.g. AM, Midday, PM etc.) not hour. Therefore the average is simple - just making sure the skim is appropriate for the type of destination.
  + Healthcare and Essential Services: AM - 6:00 AM to 9:00 AM
  + Higher Ed: Midday - 9:00 AM to 3:00 PM

Please - see <https://github.com/mmatkinson7/LRTP_DIDB_AVGTT>

(or M:\JupyterHome\JupyterNotebooks\LRTP\_DIDB\_AVGTT\LRTP\_DIDB\_AVGTT)

**July 13th, 2021 Averages Results:**

WAT Avgs

{'ES\_AM\_TR\_Avg': 23.055239813307843,

'ES\_SOV\_Avg': 1.4344942460072214,

'ES\_Walk\_Avg': 23.06080420530682,

'HLTH\_AM\_TR\_Avg': 26.14402159508965,

'HLTH\_SOV\_Avg': 2.263966118767715,

'HLTH\_Walk\_Avg': 18.334103021228177,

'HIED\_MD\_TR\_Avg': 25.4199342019939,

'HIED\_SOV\_Avg': 2.2477777843488687,

'HIED\_Walk\_Avg': 75.45288245336253}

WAT\_DAT Avgs

{'ES\_AM\_TR\_Avg': 22.33236239153218,

'ES\_SOV\_Avg': 1.4344942460072214,

'ES\_Walk\_Avg': 23.06080420530682,

'HLTH\_AM\_TR\_Avg': 23.15659744492253,

'HLTH\_SOV\_Avg': 2.263966118767715,

'HLTH\_Walk\_Avg': 18.334103021228177,

'HIED\_MD\_TR\_Avg': 23.92563767015071,

'HIED\_SOV\_Avg': 2.2477777843488687,

'HIED\_Walk\_Avg': 75.45288245336253}

Please Note: For this draft - because the model needs to be rerun to get Walk Access Transit for different modes (like local bus, rapid transit, and commuter rail), I only did an All-Transit average. Additionally, the reason for having both WAT and DAT/WAT versions is due to Betsy wanting to include all access types to transit and Marty’s preference to not use the DAT data due to inaccuracy of representing actual patterns of transit usage (e.g. driving all the way in to the city to take transit one stop).

## Flagging TAZs with Access to Destination TAZs Below Average MPO Travel Time

If the TAZ has a travel time to a Destination TAZ that is below the average for the MPO, it counts as a TAZ with Access for that destination type for that mode.

# Appendix

## Beautiful Soup Web Scraper

#beautiful soup

from bs4 import BeautifulSoup

import pandas as pd

root = r'//lilliput/matkinson/LRTP/'

fn = 'query\_page\_1.html'

files = ["Ipswich\_Center.html","Ipswich\_Center2.html","Ipswich\_Center3.html", "NS\_Center1.html",

"NS\_Center2.html","NS\_Center3.html","NS\_Center4.html", "Westborough\_Center1.html",

"Westborough\_Center2.html","Westborough\_Center3.html","Westborough\_Center4.html","Westborough\_Center5.html"]

files = ["Ipswich\_Center.html"]

names = []

addresses = []

for fn in files:

full\_fn = root + fn

print(full\_fn)

fp = open(full\_fn)

html = fp.read()

soup = BeautifulSoup(html)

results = soup.findAll("p", class\_="address")

for r in results:

# Grab the post office address

the\_address = r.contents[0]

# Grab the post office name

prev\_sib = r.find\_previous\_sibling()

# Filter out those pesky "Special hours are currently in effect..." records.

if prev\_sib != None:

po\_name = prev\_sib.find("strong").contents[0]

# Here: compose a string consisting of:

# 1. the PO name

# 2. a comma

# 3.the PO's address

# and add both to lists

outstr = po\_name + ',' + the\_address

names.append(po\_name)

addresses.append(the\_address)

#print(outstr)

# end\_if

# end\_for

#end large for

#make dictionary from lists

dict = {'names': names, 'addresses': addresses}

#make the dictionary into a csv table

dfToCSV = pd.DataFrame(dict)

dfToCSV.to\_csv('PO\_round3.csv')

## 

## Calculate the Averages and Flag Script

#!/usr/bin/env python

# coding: utf-8

# # Calculating Average Travel Time to Other Destination Categories

# - Create a feature class for each destination type (healthcare, higher education, essential services) that contains the point locations, types, and capacity data.

# - For each TAZ, find out how many (and ideally which) destinations of each type are within a buffer of the TAZ centroid.

# - Flag each TAZ with whether it meets the threshold for the number of destinations within a buffer around the TAZ centroid for the type of destination.

# - Use model skims for travel times from TAZ centroid to TAZ centroid for all of the following modes: Drive (SOV as proxy), Transit, Rapid Transit, Bus, Walk.

# - Then pick the minimum time to reach a flagged TAZ and weight by TAZ population and use that for calculation of the average travel time for MPO.

# - Then flag origin TAZs as whether their minimum distance to a flagged TAZ is within the average threshold for MPO.

#

#

# Environment: base\_py\_37\_omx\_geop

#

# In[1]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import geopandas

import openmatrix as omx

# In[2]:

##Import all the CSVs that contain skims

Walk\_skim= omx.open\_file('M:/LRTP/LRTP\_AvgTT/out/Walk\_skim.omx','r')

#Import AM

DAT\_Boat\_skim\_AM = omx.open\_file('M:/LRTP/LRTP\_AvgTT/out/AM/A\_DAT\_for\_Boat\_tr\_skim.omx','r')

DAT\_CommRail\_skim\_AM = omx.open\_file('M:/LRTP/LRTP\_AvgTT/out/AM/A\_DAT\_for\_CommRail\_tr\_skim.omx','r')

DAT\_LocalBus\_skim\_AM = omx.open\_file('M:/LRTP/LRTP\_AvgTT/out/AM/A\_DAT\_for\_LocalBus\_tr\_skim.omx','r')

DAT\_RapidTransit\_skim\_AM = omx.open\_file('M:/LRTP/LRTP\_AvgTT/out/AM/A\_DAT\_for\_RapidTransit\_tr\_skim2.omx','r')

WAT\_Transit\_skim\_AM = omx.open\_file('M:/LRTP/LRTP\_AvgTT/out/AM/WAT\_for\_All\_tr\_skim.omx','r')

SOV\_skim\_AM = omx.open\_file('M:/LRTP/LRTP\_AvgTT/out/AM/SOV\_skim.omx','r')

#put AM CSVs into Dictionary

skims\_AM = {'DAT\_Boat':DAT\_Boat\_skim\_AM, 'DAT\_CR':DAT\_CommRail\_skim\_AM,

'DAT\_LB':DAT\_LocalBus\_skim\_AM, 'DAT\_RT':DAT\_RapidTransit\_skim\_AM,

'WAT\_TR':WAT\_Transit\_skim\_AM, 'SOV':SOV\_skim\_AM, 'Walk': Walk\_skim}

#Import MD

DAT\_Boat\_skim\_MD = omx.open\_file('M:\LRTP\LRTP\_AvgTT\out\MD\A\_DAT\_for\_Boat\_tr\_skim.omx','r')

DAT\_CommRail\_skim\_MD = omx.open\_file('M:\LRTP\LRTP\_AvgTT\out\MD\A\_DAT\_for\_CommRail\_tr\_skim.omx','r')

DAT\_LocalBus\_skim\_MD = omx.open\_file('M:\LRTP\LRTP\_AvgTT\out\MD\A\_DAT\_for\_LocalBus\_tr\_skim.omx','r')

DAT\_RapidTransit\_skim\_MD = omx.open\_file('M:\LRTP\LRTP\_AvgTT\out\MD\A\_DAT\_for\_Rapid\_Transit\_tr\_skim.omx','r')

WAT\_Transit\_skim\_MD = omx.open\_file('M:\LRTP\LRTP\_AvgTT\out\MD\WAT\_for\_All\_tr\_skim.omx','r')

SOV\_skim\_MD = omx.open\_file('M:\LRTP\LRTP\_AvgTT\out\MD\SOV\_skim.omx','r')

#put MD CSVs into Dictionary

skims\_MD = {'DAT\_Boat':DAT\_Boat\_skim\_MD, 'DAT\_CR':DAT\_CommRail\_skim\_MD,

'DAT\_LB':DAT\_LocalBus\_skim\_MD, 'DAT\_RT':DAT\_RapidTransit\_skim\_MD,

'WAT\_TR':WAT\_Transit\_skim\_MD, 'SOV':SOV\_skim\_MD, 'Walk': Walk\_skim}

# In[3]:

##Import TAZ tables for Essential Services, Healthcare, and Higher Ed.

dest\_TAZs = pd.read\_csv(r'M:/LRTP/LRTP\_AvgTT/MPO\_TAZ\_ES\_1m2.csv',header=0, sep=',',

usecols=['taz', 'ES\_FLAG\_ALL','HIED\_Count5', 'HLTH\_COUNT1m'])

#create table with column for each destination type that flags (0,1) which TAZs pass the destination threshold

dest\_TAZs['HIED\_FLAG']=[1 if x >=1 else 0 for x in dest\_TAZs['HIED\_Count5']]

dest\_TAZs['HLTH\_FLAG']=[1 if x >=1 else 0 for x in dest\_TAZs['HLTH\_COUNT1m']]

#bring in population data for later

tot\_pop = pd.read\_csv(r'M:/LRTP/LRTP\_AvgTT/TAZ\_Pop\_CTPS.csv', header=0, sep=',', usecols=['TAZ\_ID', 'Tot\_Pop'])

dest\_TAZs = dest\_TAZs.merge(tot\_pop, how='left', left\_on='taz', right\_on ='TAZ\_ID')

#dest\_TAZs

# In[4]:

#make list of total TAZs (this is for row and column names for matrices)

tazs = dest\_TAZs['taz'].tolist()

tazs.sort()

#make lists of only destination TAZs (this is for averaging)

ES\_list = dest\_TAZs[dest\_TAZs['ES\_FLAG\_ALL']==1]['taz'].tolist()

HLTH\_list = dest\_TAZs[dest\_TAZs['HLTH\_FLAG']==1]['taz'].tolist()

HIED\_list = dest\_TAZs[dest\_TAZs['HIED\_FLAG']==1]['taz'].tolist()

len(HIED\_list)

# In[5]:

#import trips matrices

AM\_trips = omx.open\_file('M:/LRTP/LRTP\_AvgTT/out/AM/AfterSC\_Final\_AM\_Tables.omx','r')

MD\_trips = omx.open\_file('M:/LRTP/LRTP\_AvgTT/out/MD/AfterSC\_Final\_MD\_Tables.omx','r')

#turn into dataframes

AM\_trips\_DAT = pd.DataFrame((np.array(AM\_trips['DAT\_Boat'])+np.array(AM\_trips['DAT\_CR'])+

np.array(AM\_trips['DAT\_LB']) +np.array(AM\_trips['DAT\_RT']))[1:1902, 1:1902],

index=tazs, columns=tazs).replace(-np.inf, np.nan)

AM\_trips\_WAT = pd.DataFrame(np.array(AM\_trips['WAT']))

MD\_trips\_DAT = pd.DataFrame((np.array(MD\_trips['DAT\_Boat'])+np.array(MD\_trips['DAT\_CR'])+

np.array(MD\_trips['DAT\_LB']) +np.array(MD\_trips['DAT\_RT']))[1:1902, 1:1902],

index=tazs, columns=tazs).replace(-np.inf, np.nan)

MD\_trips\_WAT = pd.DataFrame(np.array(MD\_trips['WAT']))

#close omx

AM\_trips.close()

MD\_trips.close()

#make trip total tables for transit so can do weighted average

AM\_trips\_AT = AM\_trips\_DAT + AM\_trips\_WAT

MD\_trips\_AT = MD\_trips\_DAT + MD\_trips\_WAT

# In[6]:

##Sum the Skims to get Travel Time per Mode by Time of Day (loop through both AM and MD dictionaries)

AM\_Tables = {}

MD\_Tables = {}

for x in skims\_AM.keys():

if x != 'SOV' and x != 'Walk' and x != 'WAT\_TR':

AM\_Tables[x] = pd.DataFrame((np.array(skims\_AM[x]['Access Drive Time'])+np.array(skims\_AM[x]['Access Walk Time'])

+np.array(skims\_AM[x]['Dwelling Time'])+np.array(skims\_AM[x]['Egress Drive Time'])

+np.array(skims\_AM[x]['Egress Walk Time'])+np.array(skims\_AM[x]['In-Vehicle Time'])

+np.array(skims\_AM[x]['Initial Wait Time'])+np.array(skims\_AM[x]['Transfer Penalty Time'])

+np.array(skims\_AM[x]['Transfer Wait Time'])+np.array(skims\_AM[x]['Transfer Walk Time']))[1:1902, 1:1902],

index=tazs, columns=tazs).replace(-np.inf, np.nan)

if x == 'WAT\_TR':

AM\_Tables[x] = pd.DataFrame((np.array(skims\_AM[x]['Access Walk Time'])

+np.array(skims\_AM[x]['Dwelling Time'])

+np.array(skims\_AM[x]['Egress Walk Time'])+np.array(skims\_AM[x]['In-Vehicle Time'])

+np.array(skims\_AM[x]['Initial Wait Time'])+np.array(skims\_AM[x]['Transfer Penalty Time'])

+np.array(skims\_AM[x]['Transfer Wait Time'])+np.array(skims\_AM[x]['Transfer Walk Time']))[1:1902, 1:1902],

index=tazs, columns=tazs).replace(-np.inf, np.nan)

if x == 'SOV':

AM\_Tables[x] = pd.DataFrame(np.array(skims\_AM[x]['CongTime'])[1:1902, 1:1902],

index=tazs, columns=tazs).replace(-np.inf, np.nan)

if x == 'Walk':

AM\_Tables[x] = pd.DataFrame((np.array(skims\_AM[x]['WalkTime']))[1:1902, 1:1902],

index=tazs, columns=tazs).replace(-np.inf, np.nan)

for x in skims\_MD.keys():

if x != 'SOV' and x != 'Walk' and x != 'WAT\_TR':

MD\_Tables[x] = pd.DataFrame((np.array(skims\_MD[x]['Access Drive Time'])+np.array(skims\_MD[x]['Access Walk Time'])

+np.array(skims\_MD[x]['Dwelling Time'])+np.array(skims\_MD[x]['Egress Drive Time'])

+np.array(skims\_MD[x]['Egress Walk Time'])+np.array(skims\_MD[x]['In-Vehicle Time'])

+np.array(skims\_MD[x]['Initial Wait Time'])+np.array(skims\_MD[x]['Transfer Penalty Time'])

+np.array(skims\_MD[x]['Transfer Wait Time'])+np.array(skims\_MD[x]['Transfer Walk Time']))[1:1902, 1:1902],

index=tazs, columns=tazs).replace(-np.inf, np.nan)

if x == 'WAT\_TR':

MD\_Tables[x] = pd.DataFrame((np.array(skims\_MD[x]['Access Walk Time'])

+np.array(skims\_MD[x]['Dwelling Time'])

+np.array(skims\_MD[x]['Egress Walk Time'])+np.array(skims\_MD[x]['In-Vehicle Time'])

+np.array(skims\_MD[x]['Initial Wait Time'])+np.array(skims\_MD[x]['Transfer Penalty Time'])

+np.array(skims\_MD[x]['Transfer Wait Time'])+np.array(skims\_MD[x]['Transfer Walk Time']))[1:1902, 1:1902],

index=tazs, columns=tazs).replace(-np.inf, np.nan)

if x == 'SOV':

MD\_Tables[x] = pd.DataFrame(np.array(skims\_MD[x]['CongTime'])[1:1902, 1:1902],

index=tazs, columns=tazs).replace(-np.inf, np.nan)

if x == 'Walk':

MD\_Tables[x] = pd.DataFrame((np.array(skims\_MD[x]['WalkTime']))[1:1902, 1:1902],

index=tazs, columns=tazs).replace(-np.inf, np.nan)

# In[7]:

#close everything!

Walk\_skim.close()

DAT\_Boat\_skim\_AM.close()

DAT\_CommRail\_skim\_AM.close()

DAT\_LocalBus\_skim\_AM.close()

DAT\_RapidTransit\_skim\_AM.close()

WAT\_Transit\_skim\_AM.close()

SOV\_skim\_AM.close()

DAT\_Boat\_skim\_MD.close()

DAT\_CommRail\_skim\_MD.close()

DAT\_LocalBus\_skim\_MD.close()

DAT\_RapidTransit\_skim\_MD.close()

WAT\_Transit\_skim\_MD.close()

SOV\_skim\_MD.close()

# In[8]:

AM\_Tables['DAT\_RT']

# In[35]:

#make DAT tables (get min for each cell!)

AM\_Tables['AM\_DAT'] = np.fmin(np.fmin(AM\_Tables['DAT\_Boat'],AM\_Tables['DAT\_CR']), np.fmin(AM\_Tables['DAT\_LB'],AM\_Tables['DAT\_RT']))

MD\_Tables['MD\_DAT'] = np.fmin(np.fmin(MD\_Tables['DAT\_Boat'],MD\_Tables['DAT\_CR']), np.fmin(MD\_Tables['DAT\_LB'],MD\_Tables['DAT\_RT']))

#Create new table in dictionary: All Transit

#take the average - weight it by # of trips for each type of transit

#AM\_Tables['AM\_Transit'] = ((AM\_Tables['AM\_DAT']\*AM\_trips\_DAT)+(AM\_Tables['AM\_WAT']\*AM\_trips\_WAT))/(AM\_trips\_AT)

#MD\_Tables['MD\_Transit'] = ((MD\_Tables['MD\_DAT']\*MD\_trips\_DAT)+(MD\_Tables['MD\_WAT']\*MD\_trips\_WAT))/(MD\_trips\_AT)

#take the lowest

#AM\_Tables['AM\_TR'] = np.fmin(AM\_Tables['AM\_DAT'], AM\_Tables['WAT\_TR'])

#MD\_Tables['MD\_TR'] = np.fmin(MD\_Tables['MD\_DAT'], MD\_Tables['WAT\_TR'])

#use these if no drive access transit

AM\_Tables['AM\_TR'] = AM\_Tables['WAT\_TR']

MD\_Tables['MD\_TR'] = MD\_Tables['WAT\_TR']

# In[36]:

##Calculate Averages

#weighted average of all the lowest travel times from each TAZ to closest (timewise) destination TAZ

#(ones that pass the destination threshold (flagged)), weighted by population of the origin TAZ.

#do this for all modes and TOD - should end up with a number each for SOV, All Transit, Rapid Transit, Local Bus, Commuter Rail, Boat, and Walking for AM and MD.

#want to end up with a table for each mode for each dest type with these columns: Origin TAZ, Dest TAZ, Time, Tot\_Pop

#need to do AM for Healthcare and Essential Services, do MD for HiEd. (three tables per mode)

Avgs = {}

#Transit - ES

#SOV - ES

#Walking - ES

for x in ['AM\_TR', 'SOV', 'Walk']:

Avgs['ES\_'+x] = AM\_Tables[x].loc[:,ES\_list] #restrict columns to just what is an ES destination TAZ

Avgs['ES\_'+x]['Min\_Dest\_TAZ'] = Avgs['ES\_'+x].idxmin(axis=1) #new column with the column name of the smallest time in each row

Avgs['ES\_'+x]['TravelTime'] = Avgs['ES\_'+x].min(1) #new column with the min val in each row (corresponds with dest TAZ above)

Avgs['ES\_'+x] = Avgs['ES\_'+x].reset\_index() #turn index into origin taz field

Avgs['ES\_'+x] = Avgs['ES\_'+x].rename(columns = {'index':'origin taz'})

Avgs['ES\_'+x] = Avgs['ES\_'+x][['origin taz', 'Min\_Dest\_TAZ', 'TravelTime']] #restrict to a manageable table

#Transit - Healthcare

#Walking - Healthcare

#SOV - Healthcare

for x in ['AM\_TR', 'SOV', 'Walk']:

Avgs['HLTH\_'+x] = AM\_Tables[x].loc[:,HLTH\_list] #restrict columns to just what is an HLTH destination TAZ

Avgs['HLTH\_'+x]['Min\_Dest\_TAZ'] = Avgs['HLTH\_'+x].idxmin(axis=1) #new column with the column name of the smallest time in each row

Avgs['HLTH\_'+x]['TravelTime'] = Avgs['HLTH\_'+x].min(1) #new column with the min val in each row (corresponds with dest TAZ above)

Avgs['HLTH\_'+x] = Avgs['HLTH\_'+x].reset\_index() #turn index into origin taz field

Avgs['HLTH\_'+x] = Avgs['HLTH\_'+x].rename(columns = {'index':'origin taz'})

Avgs['HLTH\_'+x] = Avgs['HLTH\_'+x][['origin taz', 'Min\_Dest\_TAZ', 'TravelTime']] #restrict to a manageable table

#SOV - HiEd(MD)

#Transit - HiEd (MD)

#Walking - HiEd

for x in ['MD\_TR', 'SOV', 'Walk']:

Avgs['HIED\_'+x] = MD\_Tables[x].loc[:,HIED\_list] #restrict columns to just what is an HiED destination TAZ

Avgs['HIED\_'+x]['Min\_Dest\_TAZ'] = Avgs['HIED\_'+x].idxmin(axis=1) #new column with the column name of the smallest time in each row

Avgs['HIED\_'+x]['TravelTime'] = Avgs['HIED\_'+x].min(1) #new column with the min val in each row (corresponds with dest TAZ above)

Avgs['HIED\_'+x] = Avgs['HIED\_'+x].reset\_index() #turn index into origin taz field

Avgs['HIED\_'+x] = Avgs['HIED\_'+x].rename(columns = {'index':'origin taz'})

Avgs['HIED\_'+x] = Avgs['HIED\_'+x][['origin taz', 'Min\_Dest\_TAZ', 'TravelTime']] #restrict to a manageable table

#if want to not restrict to just transit (e.g. if O and D are same TAZ)

#ES\_RT.loc[ES\_RT['origin taz'].isin(ES\_list), 'Min\_Dest\_TAZ'] = ES\_RT['origin taz'] #if taz is already a dest taz, set D taz to O taz

#ES\_RT.loc[ES\_RT['origin taz'] == ES\_RT['Min\_Dest\_TAZ'], 'TravelTime'] = None #if the taz is a dest taz, set TT to null

#note here: so there will never be a transit value if the dest taz is the same as the origin taz

#will need to replace with Walk or SOV here - depends on DAT or WAT?

Avgs['HIED\_MD\_TR']

# In[37]:

##Calculate Averages PART 2

#weighted average of all the lowest travel times from each TAZ to closest (timewise) destination TAZ

#(ones that pass the destination threshold (flagged)), weighted by population of the origin TAZ.

#do this for all modes and TOD - should end up with a number each for SOV, All Transit, Rapid Transit, Local Bus, Commuter Rail, Boat, and Walking for AM and MD.

#actually calculate averages

AvgNums = {}

for x in Avgs.keys():

Avgs[x] = Avgs[x].merge(dest\_TAZs, how = 'left', left\_on = 'origin taz', right\_on = 'taz') #merge to have all data in one table

Avgs[x]['TT\_Pop'] = Avgs[x]['TravelTime']\*Avgs[x]['Tot\_Pop'] #calculate time \* pop (weight)

avg = np.nansum(Avgs[x]['TT\_Pop'])/np.nansum(Avgs[x]['Tot\_Pop']) #make the actual avg, excluding nan values

AvgNums[x+'\_Avg'] = avg #add avg to dictionary with id

Avgs[x]['Avg'] = avg #add a avg field for calc later

AvgNums

# In[38]:

##Make Flag Tables

#if the TAZ has access to a destination TAZ within the MPO average for that mode, flag.

#How do I make this into a table? - maybe copy the TAZ to TAZ table and clear its contents, then just go down the column of all the destination TAZs and query the skim table for that mode and TOD.

#where column name is in list (aka field in the table that flags destination TAZs for each type)

for x in Avgs.keys():

Avgs[x]['AvgTT\_Flag'] = np.where(Avgs[x]['TravelTime'] <= Avgs[x]['Avg'], 1, 0)

# In[39]:

Avgs[x]

# In[40]:

for x in Avgs.keys():

Avgs[x].to\_csv('M:\LRTP\LRTP\_AvgTT\Avgs\_WAT\Avg\_'+x+'.csv',sep = ',')

# In[ ]: