TripSender

Release 0.0.1

SanjaySomanath

May 02, 2021

CONTENTS:

CHAPTER

ONE

ABOUT

Trip sender is a tool to generate trips for random agents using the Gothenburg travel survey and the SCB demographic dataset.

1.1 Introduction

1.1.1 About

Trip sender is a tool to generate trips for random agents using the Gothenburg travel survey and the SCB demographic dataset.

1.1.2 What is a trip?

Travel in this survey means all transfers that are made to carry out a case at the goal. Even things that are not normally called trips, such as a walk to a lunch restaurant or a bike ride to the kiosk to buy a newspaper, counts here as travel. Exercise rounds or walking to walk the dog, for example, is not included. Professional traffic, as to example taxi drivers or police do in the service, are also not covered by the survey. If a person carries out two cases in a row, for example leaving children at preschool before traveling on to work, it counts as two trips. The journey home is counted as another journey.

1.1.3 The mode of travel

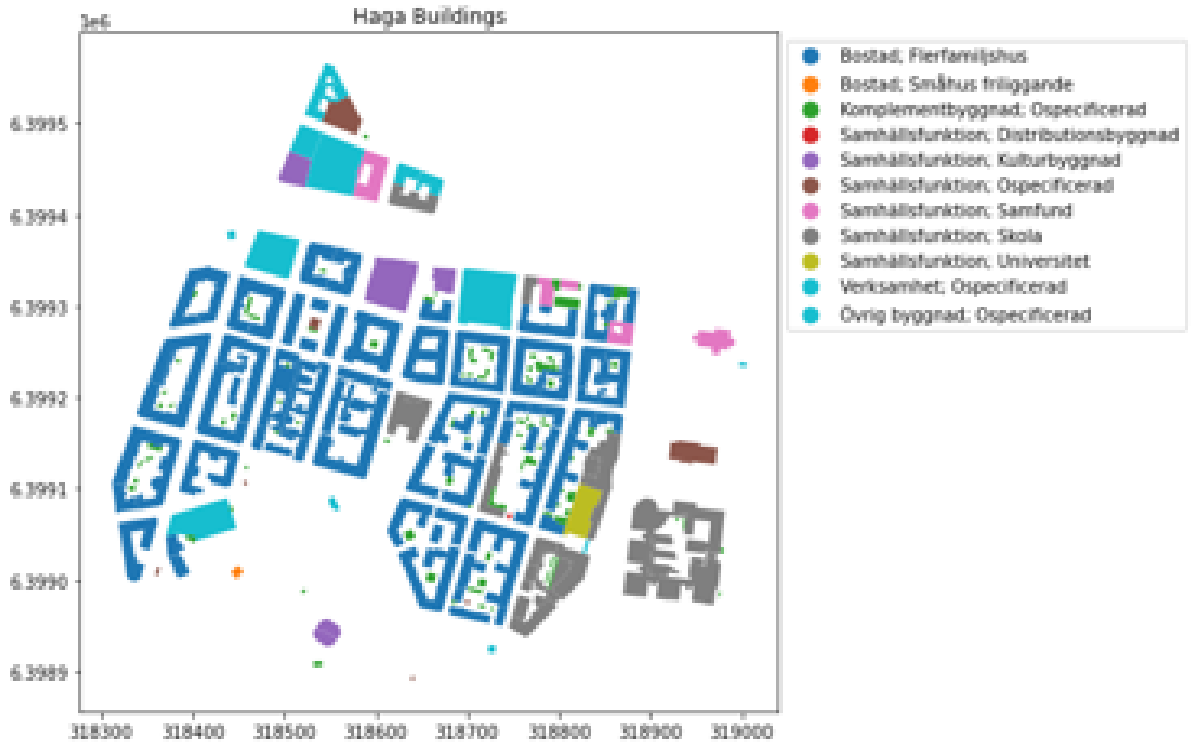
In cases where several modes of transport have been used for one and the same journey, one of the modes of transport has become the main mode of transport and this is the mode of transport that is then shown in the report. To determine which is the main mode of transport a ranking between modes of transport has been made. In this way, the mode of transport that is used the longest usually is the main mode of transport.

This report presents the main modes of transport On foot, Bicycle, Public transport, Car and Other. Public transport includes railway (train, commuter train, tram and local train), boat line and bus. Other modes of transport includes transport service, taxi, moped / motorcycle, flight and more.

1.1.4 IPF for creating a synthetic population

Iterative proportional fitting is used to create a synthetic population of a neighbourhood using neighbourhood specific demographic data from SCB. We perform the IPF in three dimensions. Age Group, Gender and Household type. Since a low number of dimensions are used, the accuracy of the fitting is quite good. Demographic profiles are independent of population size, this allows apply the neighbourhood characteristics of one to another and get the right population count. In the next steps, we will also generate tools to allow the user to make their own unique blend of demographics to test their designs in.

1.1.5 Home origins



Home origins are based on Lantmateriet data. Buildings have a land use code that can be used to filter out all homes. The centroid of these geometries are used to assign home origins.

Building heights are determined using the median z-coordinate from the Lantmateriet LiDar data. This allows us to calculate an estimate GFA of the building. The GFA is then used to allocate the population to each home.

1.1.6 Destinations

Destinations can be obtained using two methods. OSM and Google Earth. A concern with both these methods is the completeness of amenity data. Another caveat with the destination data is matching the trip purpose with a destination. Right now, a destination class is assigned at random. (Discussion here)

Visiting relatives/friends (other homes) is another problem. We know roughly the range that they would travel. But calculating unique travel distances might be challenging

Returning home is another challenge. The data needs to be investigated to identify the purpose code that identifies trips back to origin # Right now, the (n-1)th trip is forced home

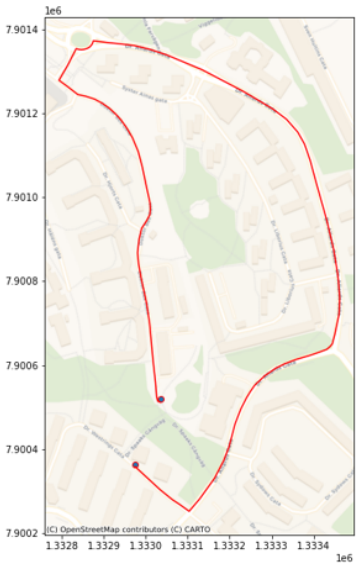
1.1.7 OD Matrix assignment

There are many methods that can be used to assign the destinations. In the Toy Model we use a simple beeline closest distance per Origin – Destination pair. The nodes are first vectorized using the SciPy library for faster calculation of nearest neighbors.

We use cKD trees (Maneewongvatana and Mount 1999) to perform a binary search for the closest node rather than visit every destination for every origin. Alternate methods (cost, amenity rating, random of closest n destinations) could include a destination choice with custom weights to be used in a cost function.

1.1.8 Routing methods

The routing algorithm used is Dijkstra shortest path with edge weights as length. This is again not the fastest or the smartest algorithms (relative to google maps) but it will do the job. There are more algorithms that could be explored (djikstra variants for driving, bellman-ford etc.)



First, PostGIS based PG_Routing was used to perform the routing. This was easy to use, NetworkX was tried next and performance improved. This was further improved with iGraph. The documentation for the iGraph Python binding is sparse and the graph convention is not as user friendly as the previous libraries but saving in computation time is significant.

The network uses the Lantmateriet street and path network. Again, OSM is easier and has more informal paths, but for completeness and scalability, we pick Lantmateriet.

1.2 Documentation

1.2.1 ighelpers module

tripsender.ighelpers.**get_edge_length**(*geoms*)

A function that gets the length of each geometry for the geoms column supplied as a list.

Parameters **geoms** – (GeoDataframe.geoms) The GeoSeries of the geometry column on a GeoDataFrame

Returns (list) List of lengths for each geometry object

tripsender.ighelpers.**profile**(*fnc*)

A decorator that uses cProfile to profile a function

tripsender.ighelpers.**run_tripsender**(*A_car*, *A_ped*, *G_car*, *G_ped*, *SynthPop*, *closest_car*, *closest_ped*, *dict_Personas*)

A function that runs the trip sender function.

Parameters

- **A_car** – (numpy array) Array of all nodes in the graph
- **A_ped** – (numpy array) Array of all nodes in the graph
- **G_car** – (igraph graph object) iGraph Graph object for car network
- **G_ped** – (igraph graph object) iGraph Graph object for ped network
- **SynthPop** – (Pandas DataFrame) DataFrame containing the synthetic profiles
- **closest_car** – (GeoDataFrame) GDF containing origins and closest destinations for each destination class
- **closest_ped** – (GeoDataFrame) GDF containing origins and closest destinations for each destination class
- **dict_Personas** – (List) List of dictionaries containing the 14 synthetic population groups

Returns (GeoDataFrame) GDF of the result with a route column containing a list of nodes for the shortest path

1.2.2 nxhelpers module

tripsender.nxhelpers.**get_closest**(*A*, *pt*)

A function to get the closest point in a numpy array to a point provided.

Parameters

- **A** – (numpy Array)
- **pt** – (tuple) Input coordinates that may not exist in the graph nodes

Returns (tuple) Closest point that exists on the graph nodes.

tripsender.nxhelpers.**get_route**(*A*, *G*, *source*, *target*, *method='dijkstra'*, *plot=False*)

A function to perform routing on the the network from source to target. The source and target points do not have to be on the graph. The function uses the get_closest function to find the closest nodes on the graph and then performs the routing using the routing algorithm specified.

Parameters

- **A** – (numpy Array) Array of all nodes in the graph
- **G** – (networkX Graph) A `networkX` Graph object that represents a street network
- **source** – (tuple) A coordinate of the source point for routing
- **target** – (tuple) A coordinate of the destination point for routing
- **method** – (str: Optional)
- **plot** – (bool: Optional)

Returns (list) A list of tuples representing the nodes along the shortest path from source to target

`tripsender.nxhelpers.`**graph_from_gdf**(*geoms*)
A function to generate a `networkX` Graph object from `GeoSeries` containing street network geometry.

Parameters **geoms** – (GeoPandas GeoSeries) The geometry column of the `GeoDataFrame` containing the road network.

Returns (networkX Graph) A `networkX` Graph object representing the street network

1.2.3 pghelpers module

`tripsender.pghelpers.`**amenity_from_kml**(*mypath*)
A function to fetch all kml data from a folder containing `KML` files.

Parameters **mypath** – (str) A string containing the directory path to a folder with `KML` files

Returns (GeoPandas GeoDataFrame) A `GDF` of amenity lat and lon

`tripsender.pghelpers.`**flatten_gdf**(*gdf*)
A utility function to flatten the nested hierarchy of a `GeoPandas GeoDataFrame`.

Parameters **gdf** – (GeoPandas GeoDataFrame) A GDF to flatten

Returns (GeoPandas GeoDataFrame) A flattened GDF

`tripsender.pghelpers.`**get_amenities**(*gdf*, *amenity*)
A function to fetch amenities using a keyword from the `OpenStreetMaps` Overpass api. (not recommended!) Use overpass turbo to help frame the keywords.

Parameters

- **gdf** – (GeoPandas GeoDataFrame) A *GDF* containing the origins of homes
- **amenity** – (str) A query string formatted for the OSM overpass API. (See overpass turbo)

Returns (GeoPandas GeoDataFrame)

`tripsender.pghelpers.`**get_closest_from_kml**(*kmlPath*, *gdfAgentHomes*)
A function to return a GDF of matched closest amenities from a `KML` directory and a `GDF` of agent origins/homes.

Parameters

- **kmlPath** – (str) A string containing the directory path to a folder with `KML` files
- **gdfAgentHomes** – (GeoPandas GeoDataFrame) A `GeoDataFrame` containing an origin column with point geometry.

Returns (GeoPandas GeoDataFrame, list) A GDF containing the matched closest amenities and a list of amenity names

tripsender.pghelpers.**get_gdf**(*area*, *feature*, *KEY=''*, *VAL=''*, *title='Data from Server'*, *filter=False*, *local=True*, *web=True*, *plot=False*)

A function to fetch spatial geometry from the PostGIS server as GeoPandas GeoDataFrame. This function is simply a wrapper for the psycopg2 module. It constructs an SQL query based on the the params provided and requests data from the server.

Parameters

- **area** – (str) The name of the Primary area in the Model naming format.
- **feature** – (str) The feature name to select from the PostGIS database. (Refer to PostGIS naming convention)
- **KEY** – (str: Optional) An optional attribute to filter from the data at the server level
- **VAL** – (str: Optional) An optional value for a given key to match from the data at the server level
- **title** – (str: Optional) An optional title for the plot
- **filter** – (bool: Optional) An optional input to specify if the data must be filtered at the server level
- **local** – (bool: Optional) Which server to fetch the data from
- **web** – (bool: Optional) If True, the result will be reprojected
- **plot** – (bool: Optional) If True, the result will be plotted

Returns (GeoPandas GeoDataFrame, plot(Optional)) The resulting *GeoPandas GeoDataFrame*.

tripsender.pghelpers.**get_nearest_amenity**(*gdA*, *gdfAmenity*, *amenityList*)

A function to get the nearest amenities from a source and destination GDF.

Parameters

- **gdA** – (GeoPandas GeoDataFrame) A GDF with the origins of homes
- **gdfAmenity** – (GeoPandas GeoDataFrame) A GDF containing the amenities with a unique name for each amenity
- **amenityList** – (list) A list of amenity names as string

Returns

tripsender.pghelpers.**get_pg_query**(*sql*, *local=True*, *web=True*)

A utility function that fetches any data using an SQL query from the PostGIS Server This function is a basic wrapper for the psycopg2 module with secrets included.

Parameters

- **sql** – (str) An SQL query string
- **local** – (bool: Optional) If True, data will be fetched from the local server
- **web** – (bool: Optional) If True, result will be reprojected to EPSG:4326

Returns

tripsender.pghelpers.**get_pg_route**(*source*, *target*, *plot=True*, *local=True*)

A function to perform routing on the PostGIS server using PG_Routing.

Parameters

- **source** – (tuple) A coordinate of the source point for routing
- **target** – (tuple) A coordinate of the destination point for routing

- **plot** – (bool: Optional) If `True`, the result will be plotted
- **local** – (bool: Optional) If `True`, data will be fetched from the local server

Returns (list) A list of tuples representing the nodes along the shortest path from source to target

`tripsender.pghelpers.`**`get_random_source`**(*gdfAgentHomes*)
A function to select a random source from a GeoPandas GeoDataframe.

Parameters **gdfAgentHomes** – (GeoPandas GeoDataFrame) A *Geodataframe* containing an origin column with point geometry.

Returns (tuple) Coordinates as a tuple

`tripsender.pghelpers.`**`get_road`**(*local=True*, *web=True*, *ped=True*)
A function that is hardcoded to only fetch the complete street network as a `GeoPandas GeoDataFrame`. :param local: (bool: Optional) If `True`, data will be fetched from the local server :param web: (bool: Optional) If `True`, result will be reprojected to `EPSG:4326` :param ped: (bool: Optional) If `True`, data will be fetched for the pedestrian network, else car network. :return: (GeoPandas GeoDataFrame) A `GDF` containing the entire street network on the `PostGIS` server.

`tripsender.pghelpers.`**`plot_gdf_pretty`**(*gdf*, *column=None*, *figsize=(10, 10)*, *legend=False*)
A function that plots the `GeoDataFrame` using the `contextily` library against a map background.

Parameters

- **gdf** – (GeoPandas GeoDataFrame) GDF to be plotted
- **column** – (str: Optional) Name of the column to be plotted
- **figsize** – (tuple: Optional)
- **legend** – (bool: Optional)

Returns (Plot)

1.2.4 pshelpers module

`tripsender.pshelpers.`**`clean_travel_data`**(*travelData*)
A function that cleans the raw travel data from the travel survey `csv`.

Parameters **travelData** – (Pandas DataFrame) A DataFrame containing raw data from the Swedish Travel Survey

Returns (Pandas DataFrame) A cleaned DataFrame containing data from the Swedish Travel Survey

`tripsender.pshelpers.`**`create_persona`**(*df_Personas*, *write=True*)
A function to create personas from a list of `Pandas DataFrame` containing data from the Swedish Travel Survey.

Parameters

- **df_Personas** – (list) List of Pandas DataFrame containing data from the Swedish Travel Survey
- **write** – (bool: Optional) If True, write to path 'data/dict' as json

Returns (list) A list of dictionaries containing data from the Swedish Travel Survey

`tripsender.pshelpers.`**`get_mode`**(*randomPersona*, *key*)
A function to get the mode of travel from and individual dict containing a persona.

Parameters

- **randomPersona** – (dict) A dict containing one persona

- **key** – (str) A valid tripID within the random persona to get the mode of travel from

Returns (int,int) Mode index and mode speed. (0 = walk, 1 = bike, 2 = e-bike ... see RVU documentation for details)

tripsender.pshelpers.**get_random_persona**(*age*, *gender*, *dict_Personas*)
A function to fetch a random persona from a list of dictionaries, given the required age and gender. Male is 1 and Female is 2.

Parameters

- **age** – (int)
- **gender** – (int) 1 for Male and 2 for Female
- **dict_Personas** – (list) An ordered list of 14 dicts for each age group and gender

Returns (dict) A randomly fetched persona from the Swedish Travel Survey as a dictionary

tripsender.pshelpers.**retrieve_name**(*var*)
A utility function to retrieve the variable name from a named python element.

Parameters **var** – (python variable element)

Returns (str) The name of the variable element supplied

tripsender.pshelpers.**split_travel_data**(*travelData*)
A function that takes a list of `Pandas DataFrame` representing the travel data for each age group and gender.

Parameters **travelData** – (Pandas DataFrame) A combined DataFrame with all travel data

Returns (list) A list of cleaned DataFrame split from the combined DataFrame

tripsender.pshelpers.**travel_json_to_english**(*swedish_json*)
A function to replace the Swedish keys with English keys on the travel data `JSON`.

Parameters **swedish_json** – (JSON object) The original JSON from the Swedish Travel Survey

Returns (dict) The replaced data as dict with English variable names

1.2.5 scbhelpers module

tripsender.scbhelpers.**check_connection**(*response*)
A utility function to check the response code and print the status.

Parameters **response** – (response object)

Returns

tripsender.scbhelpers.**get_area**(*area*)
Function that searches a list of official area names and ensures that it is spelt correctly. Searches the partial primary area code, non-capitalised partial area names and returns the full string.

Example: Fetching the `pxWeb` compatible area name for Haga:

```
# Importing libraries
import tripsender as ts

# Searching areas with name only
my_area_name = ts.scbhelpers.get_area('Haga')

# Searching areas with area code
my_area_code = ts.scbhelpers.get_area('107')
```

(continues on next page)

(continued from previous page)

```
# Print the results
print(my_area_name==my_area_code, my_area_name)

out:
    True
    107 Haga
```

Parameters **area** – (str) The partial area name or code

Returns (str) Full name string of matched area

tripsender.scbhelpers.**get_area_sp**(*foundArea*)

A function that takes a correctly formatted area string and converts it into the scb special formatting. This special formatting is required for specific queries on the pxWeb platform.

Parameters **foundArea** – The correctly formatted area name as provided by the get_area function

Returns Area name with special pxWeb formatting

tripsender.scbhelpers.**get_distribution**(*df*)

A function that returns the total sum, dataframe of probability distribution, marginals_x and marginals_y.

Parameters **df** – (pandas DataFrame)

Returns (total,probability distribution,marginals on x, marginals on y)

tripsender.scbhelpers.**get_table**(*first*, *second*, *area*, *year*, *showResponse=False*, *plot=True*)

A function to fetch data on two requested variables from the pxWeb server. Note: The server takes instructions using Swedish keywords, hence explicitly typed function.

Parameters

- **first** – (str) First variable
- **second** – (str) Second Variable
- **area** – (str) Partial Primary Area or Primary Area code
- **year** – (str) Year. Note : Some variables are not updated since 2019.
- **showResponse** – (bool: Optional) If True, print the response code
- **plot** – (bool: Optional) If True, Plot the data

Returns (Pandas DataFrame)

tripsender.scbhelpers.**get_trend**(*first*, *second*, *area*, *showResponse=False*)

A function that fetches multi-year data for two valid variables. Currently only age and household are supported.

Parameters

- **first** – (str) First variable
- **second** – (str) Second variable
- **area** – (str) A partial area name or partial area code
- **showResponse** – (bool) If True, show the request response code

Returns (pandas DataFrame, plot)

tripsender.scbhelpers.**prep_data**(*df*, *headings*, *title='Plot of requested data'*, *plot=True*)
A function that prepares the input DataFrame by resetting the index and headings from a list of headings.

Parameters

- **df** – (pandas DataFrame)
- **headings** – (list)
- **title** – (Optional : str) Optional plot title
- **plot** – (Optional : bool) If `True`, plot the prepared DataFrame

Returns (pandas DataFrame)

tripsender.scbhelpers.**prep_data_income**(*df*, *headings*, *title='Plot of requested data'*, *plot=True*)
Note: Use this function for income related data. This is due to the unique age groups that cannot be changed through the query from the `pxWeb` server. A function that prepares the input DataFrame by resetting the index and headings from a list of headings.

Parameters

- **df** – (pandas DataFrame)
- **headings** – (list)
- **title** – (Optional : str) Optional plot title
- **plot** – (Optional : bool) If `True`, plot the prepared DataFrame

Returns (pandas DataFrame)

tripsender.scbhelpers.**query_data**(*url*, *query*, *foundArea*, *showResponse=True*, *timeout=1.5*)
A function to post the json query to the `pxWeb` server for Gothenburg.

Parameters

- **url** – (str) Query URL
- **query** – (str) Query JSON
- **foundArea** – (str) Correctly formatted area name
- **showResponse** – (bool : Optional) If True : Show the request response code
- **timeout** – (float: Optional) Request timeout, default at 1.5 seconds.

Returns (DataFrame, heading (str), title (str))

1.2.6 trhelpers module

tripsender.trhelpers.**alder2age**(*alder*)
A function to take the Swedish age categories from the travel survey and get a random age as int within those bounds.

Parameters **alder** – (str) A string representing a Swedish age category from the Travel Survey

Returns (int) Return a random age as integer from the provided age category

tripsender.trhelpers.**fix_closest**(*closest*, *A*, *amenityName*)
A function to post-process the closest amenities gdf from *nxhelpers.get_closest_from_kml* to a format that *tripsender* can use.

Parameters

- **closest** – (GeoPandas GeoDataFrame) GDF containing origins and closest destinations for each destination class
- **A** – (numpy array)
- **amenityName** – (list) A ordered list of amenity names matching those in the closest GDF

Returns (GeoPandas GeoDataFrame) The post-processed GDF from the *nx-helpers.get_closest_from_kml* function result

tripsender.trhelpers.**generate_synth_pop**(*area*, *year='2019'*)

A function that generates a synthetic population using Iterative proportional fitting. The function uses the `IPFn` python module. The IPF is done in three dimensions by default, age, gender and household type. Population data is fetched on-the-fly from the scb pxWeb database for Gothenburg and uses a simple IPF generator.

Parameters

- **area** – (str) The as string. Note: This should be a valid area name recognised by the `PostGIS` server
- **year** – (str: Optional) Optional String for the year. If not supplied it defaults to 2019

Returns (Pandas DataFrame) Returns a `Dataframe` with age, gender and household attributes for the population.

tripsender.trhelpers.**get_pnbr**(*closest*, *amenityName*)

A function to generate the perceived neighbourhood by matching the closest Origin Destination pairs.

Parameters

- **closest** – (GeoPandas GeoDataFrame) GDF containing origins and closest destinations for each destination class.
- **amenityName** – (list) A ordered list of amenity names matching those in the closest GDF

Returns (GeoPandas GeoDataFrame) A GDF containing the origin and amenity locations

tripsender.trhelpers.**plot_od_lines**(*pNbr*)

A function to visualise the closest amenity with linestring connecting the origin to the destination.

Parameters **pNbr** – (GeoPandas GeoDataFrame) A GDF containing the origin and amenity locations

Returns (contextily plot) A `contextily` plot containing the `Linestring` of the Origin Destination pairs

tripsender.trhelpers.**plot_trip**(*gdf*)

A function to plot the result of the process_trip function on a basemap using `contextily`.

Parameters **gdf** – (GeoPandas GeoDataFrame)

Returns (contextily plot)

tripsender.trhelpers.**process_trip**(*output*)

A function to post-process the output of the *tripsender* function. It reads the list of coordinates on the shortest path and appends the linestring formed by the list of coordinates. The function also processes the length and trip time for each path.

Parameters **output** – (GeoPandas GeoDataFrame) The output GDF of the *tripsender* function

Returns (GeoPandas GeoDataFrame) A fixed output of the *tripsender* function with linestring for shortest path.

tripsender.trhelpers.**process_trip_gdf**(*gdf*)
A Function that combines the step of processing the output of the tripsender function and turns it into a dictionary string that can be read by keplerGL for inline visualisation.

Parameters **gdf** – (GeoPandas GeoDataFrame) The output GDF of the tripsender function

Returns (str) A KeplerGL compatible geojson dictionary string

tripsender.trhelpers.**profile**(*fnc*)
A decorator that uses cProfile to profile a function

tripsender.trhelpers.**read_gdf_to_dict**(*path*)
A function to read the saved geojson output from tripsender as a dict.

Parameters **path** – (str) The string path to the geojson file output from the trip sender function

Returns (dict) The dictionary created from the geojson file

tripsender.trhelpers.**read_synth_pop**(*myPath*)
A function to read a local csv saved from the synthetic population generation step.

Parameters **myPath** – (str) A path to the csv containing results from the synthetic population step

Returns (Pandas DataFrame) Returns a DataFrame with age, gender and household attributes for the total population

tripsender.trhelpers.**run_tripsender**(*A*, *G*, *SynthPop*, *closest*, *dict_Personas*)
A function that runs the tripsender function.

Parameters

- **A** – (numpy array) Array of all nodes in the graph
- **G** – (networkX graph object) networkX Graph object
- **SynthPop** – (Pandas DataFrame) DF containing the synthetic profiles
- **closest** – (GeoPandas GeoDataFrame) GDF containing origins and closest destinations for each destination class
- **dict_Personas** – (List) List of dictionaries containing the 14 synthetic population groups

Returns (GeoPandas GeoDataFrame) GDF of the result with a route column containing nodes for the shortest path.

tripsender.trhelpers.**time2secs**(*t*)
A utility function that converts hh:mm:ss to a unix epoch in seconds. The date always starts from 00 hours on January 1st, 2021.

Parameters **t** – (str) A string of the hh:mm:ss format (Note: Must be above 6 chars)

Returns (None)

tripsender.trhelpers.**write_dict_to_kepler**(*myDict*, *area*, *myString='_0'*)
Write the tripsender dict to a kepler compatible geojson file.

Parameters

- **myDict** – (dict) Dictionary created from the output of the tripsender output
- **area** – (str) The as string. Note: This should be a valid area name recognised by the PostGIS server
- **myString** – (str: Optional) An optional identifier string if you would like to save multiple versions

Returns

tripsender.trhelpers.**write_gdf**(*gdf*, *name='test'*)

A function to re-project, post process and save a geojson from the output of the tripsender function.

Parameters

- **gdf** – (GeoPandas GeoDataFrame) The output GDF from the tripsender Function
- **name** – (str: Optional) An optional name to write the output of the tripsender function to

Returns (None)

1.2.7 tripsender module

1.2.8 agghelpers module

1.3 Project Setup

1.3.1 Introduction

Examples to be added

1.3.2 PostGIS Layer naming Convention

Examples to be added

1.3.3 Downloading Spatial Data

Examples to be added

1.3.4 Git Clone

Examples to be added

1.3.5 Build from Docker

Examples to be added

1.4 Examples

1.4.1 Fetch SCB Data

Examples to be added

1.5 Gallery

1.5.1 Demo example

Examples to be added

1.6 References

1.6.1 Iterative Proportional Fitting

Examples to be added

1.6.2 Routing Algorithms

Examples to be added

1.6.3 K-Nearest neighbours

Examples to be added

1.6.4 GIS Layer naming standards

Examples to be added

1.7 Citation

1.7.1 How to Cite

Examples to be added

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

t

INDEX

S

T

W