odysseus

...

May 20, 2021

API REFERENCE:

CHAPTER

ONE

INSTALLATION GUIDE

1.1 Setup repository, environment and data

First, let's clone the public git repository and move data into the right folder. For now, we skip explanations about *city_data_manager* functionalities.

```
git clone https://github.com/AleCioc/odysseus my-odysseus-folder
cp -r /home/det_tesi/a.ciociola/input_simulator/data my-odysseus-folder/
↪odysseus/city_data_manager
```

Then, let's install all the necessary libraries.

```
pip install --user -r my-odysseus-folder/requirements.txt
```

1.2 Configuring simulation input

The folder *odysseus/simulator/simulation_input* contains configuration files for simulation.

In particular:

- **sim_configs_target.json**: contains the name of the configuration to run
- **sim_configs_versioned**: contains one folder for each saved configuration e.g. *sim_configs_versioned/turin_iscc_set1* contains the configuration for the first set of simulation used in ISCC paper.

Each configuration folder must contain the following Python files:

- **sim_run_conf.py**: specifies used data source, run mode (single_run or multiple_runs), number of cores to use in case of multiple runs, simulation type (trace-driven or model-driven) and output folder name
- **sim_general_conf.py**: specifies macroscopic parameters about spatial and temporal configuration, as well as fleet load factor policy.
- **single_run_conf.py**: specifies scenario configuration for single run
- **model_validation_conf.py**: special case of single run
- **multiple_runs_conf.py**: specifies scenario configuration for multiple runs
- **vehicle_config.py**: specifies vehicles characteristics
- **cost_conf.py**: specifies cost/revenue configuration

Let's create a new folder for a new configuration:

```
cp -r /home/det_tesi/a.ciociola/input_simulator/ my-odysseus-folder/odysseus/
↪simulator/simulation_input/sim_configs_versioned/
```

Modify configurations as you desire, then run the simulator:

```
cd my-odysseus-folder/
python -m odysseus.simulator
```

Let's wait for simulation to finish and let's check the results folder and the figures folder (figures are created automatically only in single run mode)

```
ls my-odysseus-folder/simulator/results/Torino/single_run/test
ls my-odysseus-folder/simulator/figures/Torino/single_run/test
```

Done! Now we can explore our results and eventually produce other analysis and plots.

CHAPTER

TWO

CITY DATA MANAGER

2.1 City Data Source

2.1.1 Geo Data Source

class GeoDataSource(*city_id*, *data_source_id*)

This abstract class is used only for data sources that represent the place of departure and arrival without GPS coordinates. For example, the Minneapolis-related data they contain are stored in reference to the Centerline MPLS system. In order to correctly interpret these data we use this class that normalizes them and stores them in a shapefile

Parameters

- **city_id** (*str*) – City name. The name also serves to determine the timezone to which the city belongs
- **data_source_id** (*str*) – Data source from which the information is taken. This allows us to have multiple data sources associated with the same city (for example from different operators)

load_raw()

Abstract method that opens the file describing the geometry of the city

Returns nothing

normalise()

Abstract method that normalizes the data and stores the created shapefiles

Returns nothing

check_create_path(*path*)

2.1.2 Trips Data Gatherer

class DataGatherer(*output_path*, *structured_dataset_name*)

Class for automatically downloading data relating to the New York Citi bike sharing operator from a remote database.

Parameters

- **output_path** (*str*) – path in which to store the file
- **structured_dataset_name** –

bulk_download(*standardize=False*)
download all the datasets available at official citi bike website :param standardize: :type standardize: bool, optional :return:

download_data(*year*, *month*)
Download data for a specific month and year.

Parameters

- **year** (*int*) – year expressed as a four-digit number (e.g. 1999)
- **month** (*int*) – month expressed as a number (e.g. for November the method expects to receive 11)

Returns nothing

structured_dataset_name
get from official website the lists of all downloadable csvs dataset_names[yyyymm] = dataset_name_to_attach_to_root_url

2.1.3 Trips Data Source

All the classes of this module are implementations of the abstract class **TripsDataSource** that we report below.

TripsDataSource class

class TripsDataSource(*city_name*, *data_source_id*, *vehicles_type_id*)

TripsDataSource is an abstract class that contains the information needed to describe a trip. This class is implemented by the other classes of this module. The constructor method takes as parameters:

Parameters

- **city_name** (*str*) – City name. The name also serves to determine the timezone to which the city belongs
- **data_source_id** (*str*) – Data source from which the information is taken. This allows us to have multiple data sources associated with the same city (for example from different operators)
- **vehicles_type_id** (*str*) – Type of service represented by the data source (e.g. car sharing or e-scooter)

load_norm(*year*, *month*)
Load a previously created normalized file from memory. It requests month and year as parameters, and checks if the file for that period exists in memory (looking for it with the same format as *save_norm* in the city folder). If it exists, it returns a pandas.DataFrame containing the data read, otherwise it returns an empty DataFrame

Parameters

- **year** (*int*) – year expressed as a four-digit number (e.g. 1999)
- **month** (*int*) – month expressed as a number (e.g. for November the method expects to receive 11)

Returns If the file exists, it returns a pandas.DataFrame containing the data read, otherwise it returns an empty DataFrame

`load_raw()`
Method for loading the data to be preprocessed. Since the data format differs in the various datasets, the method is left abstract. Each city has its own implementation. All implementations will read the data through the pandas readcsv method

Returns nothing

`normalise()`
This method is used to standardize the data format. Again the implementation is highly dependent on the data source and almost all modules override the method.

Returns A normalized pandas.DataFrame

`save_norm()`
It stores normalized data both in a csv file and in a pickle file. The files produced are of the format *<year>_<month number>.csv* (or .pickle). For example *2017_11.csv*.

Returns nothing

The City Data Source module is divided into three submodules that deal with adapting the data format. Data from different sources have different formats: for example, geographic positions can be indicated as GPS coordinates or the city can be divided into a grid and the cell in which you are located can be indicated. Geo Data Source takes care of standardizing geographic information.

2.2 City Geo Trips

2.2.1 CityGeoTrips class

`class CityGeoTrips(`*city_name*, *trips_data_source_id*, *year*, *month*`)`
This abstract class deals with managing geographic travel information (e.g. departure, arrival, distance, etc.). This class is implemented by the other classes of this module. The constructor method takes as parameters:

Parameters

- **city_name** (*str*) – City name. The name also serves to determine the timezone to which the city belongs
- **trips_data_source_id** (*str*) – Data source from which the information is taken. This allows us to have multiple data sources associated with the same city (for example from different operators)
- **year** (*int*) – year expressed as a four-digit number (e.g. 1999)
- **month** (*int*) – month expressed as a number (e.g. for November the method expects to receive 11)

`get_trips_od_gdfs()`
This method is used to store the movements, using the Shapely library. The normalized data is loaded (—-> reference to save_norm magari<——) and the method builds three GeoDataFrame. The trips are encoded using an object of the LineString class from the Shapely library. They are described as a segment having the coordinates of departure and arrival as extremes. In addition, two more GeoDataFrames are created, using objects of the Shapely.Point class to describe departures and arrivals.

Returns nothing

`load()`
Load from memory, using the pickle file created by the save methods, the three GeoDataFrame

Returns nothing

save_points(*points*, *filename*)
Support method to save_data_points. It stores the points passed to it as a parameter both on csv file and on pickle.

Parameters

- **points** (*geopandas.GeoDataFrame*) – A GeoDataFrame describing the information of points to be saved
- **filename** (*str*) – Filename

Returns nothing

save_points_data()
Stores the points representing start and finish on file

Returns nothing

save_trips()
It stores on file the segments that represent the path between start and finish

Returns nothing

The City Data Manager module takes care of data preprocessing. The simulator supports heterogeneous data sources thanks to this module which, starting from a generic input data format, transforms them following the same format adopted by the other simulator modules. The module is divided into two submodules, City Geo Trips and City Data Source.

CHAPTER

THREE

DEMAND MODELLING

CHAPTER

FOUR

SUPPLY MODELLING

CHAPTER

FIVE

ODYSSEUS

5.1 city_data_manager_dashboard module

5.2 odysseus package

5.2.1 Subpackages

odysseus.city_data_manager package

Subpackages

odysseus.city_data_manager.city_data_source package

Subpackages

odysseus.city_data_manager.city_data_source.geo_data_source package

Submodules

odysseus.city_data_manager.city_data_source.geo_data_source.austin_census_tracts module

class AustinCensusTracts

Bases: *odysseus.city_data_manager.city_data_source.geo_data_source.geo_data_source.GeoDataSource*

load_raw()

Abstract method that opens the file describing the geometry of the city

Returns nothing

normalise()

Abstract method that normalizes the data and stores the created shapefiles

Returns nothing

odysseus.city_data_manager.city_data_source.geo_data_source.calgary_hexagonal_grid module

class CalgaryHexagonalGrid
Bases: *odysseus.city_data_manager.city_data_source.geo_data_source.geo_data_source.GeoDataSource*

load_raw()
Abstract method that opens the file describing the geometry of the city

Returns nothing

normalise()
Abstract method that normalizes the data and stores the created shapefiles

Returns nothing

odysseus.city_data_manager.city_data_source.geo_data_source.chicago_census_tracts module

class ChicagoCensusTracts
Bases: *odysseus.city_data_manager.city_data_source.geo_data_source.geo_data_source.GeoDataSource*

load_raw()
Abstract method that opens the file describing the geometry of the city

Returns nothing

normalise()
Abstract method that normalizes the data and stores the created shapefiles

Returns nothing

odysseus.city_data_manager.city_data_source.geo_data_source.chicago_community_areas module

class ChicagoCommunityAreas
Bases: *odysseus.city_data_manager.city_data_source.geo_data_source.geo_data_source.GeoDataSource*

load_raw()
Abstract method that opens the file describing the geometry of the city

Returns nothing

normalise()
Abstract method that normalizes the data and stores the created shapefiles

Returns nothing

odysseus.city_data_manager.city_data_source.geo_data_source.geo_data_source module

class GeoDataSource(*city_id*, *data_source_id*)
Bases: object

This abstract class is used only for data sources that represent the place of departure and arrival without GPS coordinates. For example, the Minneapolis-related data they contain are stored in reference to the Centerline MPLS system. In order to correctly interpret these data we use this class that normalizes them and stores them in a shapefile

Parameters

- **city_id** (*str*) – City name. The name also serves to determine the timezone to which the city belongs
- **data_source_id** (*str*) – Data source from which the information is taken. This allows us to have multiple data sources associated with the same city (for example from different operators)

load_raw()
Abstract method that opens the file describing the geometry of the city

Returns nothing

normalise()
Abstract method that normalizes the data and stores the created shapefiles

Returns nothing

odysseus.city_data_manager.city_data_source.geo_data_source.minneapolis_centerlines module

class MinneapolisCenterlines
Bases: *odysseus.city_data_manager.city_data_source.geo_data_source.geo_data_source.GeoDataSource*

load_raw()
Abstract method that opens the file describing the geometry of the city

Returns nothing

normalise()
Abstract method that normalizes the data and stores the created shapefiles

Returns nothing

odysseus.city_data_manager.city_data_source.geo_data_source.minneapolis_trails_bikes module

class MinneapolisTrailsBikes
Bases: *odysseus.city_data_manager.city_data_source.geo_data_source.geo_data_source.GeoDataSource*

load_raw()
Abstract method that opens the file describing the geometry of the city

Returns nothing

normalise()
Abstract method that normalizes the data and stores the created shapefiles

Returns nothing

odysseus.city_data_manager.city_data_source.geo_data_source.norfolk_census_tracts module

class NorfolkCensusTracts
Bases: *odysseus.city_data_manager.city_data_source.geo_data_source.geo_data_source.GeoDataSource*

load_raw()
Abstract method that opens the file describing the geometry of the city

Returns nothing

normalise()
Abstract method that normalizes the data and stores the created shapefiles

Returns nothing

Module contents

odysseus.city_data_manager.city_data_source.trips_data_source package

Submodules

odysseus.city_data_manager.city_data_source.trips_data_source.austin_scooter_trips module

class AustinScooterTrips
Bases: *odysseus.city_data_manager.city_data_source.trips_data_source.trips_data_source.TripsDataSource*

load_raw()
Method for loading the data to be preprocessed. Since the data format differs in the various datasets, the method is left abstract. Each city has its own implementation. All implementations will read the data through the pandas readcsv method

Returns nothing

normalise(*year*, *month*)
This method is used to standardize the data format. Again the implementation is highly dependent on the data source and almost all modules override the method.

Returns A normalized pandas.DataFrame

odysseus.city_data_manager.city_data_source.trips_data_source.big_data_db_trips module

class BigDataDBTrips(*city_name*)
Bases: *odysseus.city_data_manager.city_data_source.trips_data_source.trips_data_source.TripsDataSource*

load_raw()
Method for loading the data to be preprocessed. Since the data format differs in the various datasets, the method is left abstract. Each city has its own implementation. All implementations will read the data through the pandas readcsv method

Returns nothing

normalise(*year*, *month*)
This method is used to standardize the data format. Again the implementation is highly dependent on the data source and almost all modules override the method.

Returns A normalized pandas.DataFrame

save_norm(*year*, *month*)
It stores normalized data both in a csv file and in a pickle file. The files produced are of the format <*year*>_<*month number*>*.csv* (or .pickle). For example *2017_11.csv*.

Returns nothing

odysseus.city_data_manager.city_data_source.trips_data_source.calgary_scooter_trips module

class CalgaryScooterTrips
Bases: *odysseus.city_data_manager.city_data_source.trips_data_source.trips_data_source.TripsDataSource*

load_raw()
Method for loading the data to be preprocessed. Since the data format differs in the various datasets, the method is left abstract. Each city has its own implementation. All implementations will read the data through the pandas readcsv method

Returns nothing

normalise(*year*, *month*)
This method is used to standardize the data format. Again the implementation is highly dependent on the data source and almost all modules override the method.

Returns A normalized pandas.DataFrame

odysseus.city_data_manager.city_data_source.trips_data_source.chicago_scooter_trips module

class ChicagoScooterTrips
Bases: *odysseus.city_data_manager.city_data_source.trips_data_source.trips_data_source.TripsDataSource*

load_raw()
Method for loading the data to be preprocessed. Since the data format differs in the various datasets, the method is left abstract. Each city has its own implementation. All implementations will read the data through the pandas readcsv method

Returns nothing

normalise(*year*, *month*)
This method is used to standardize the data format. Again the implementation is highly dependent on the data source and almost all modules override the method.

Returns A normalized pandas.DataFrame

odysseus.city_data_manager.city_data_source.trips_data_source.kansas_city_scooter_trips module

class KansasCityScooterTrips

Bases: *odysseus.city_data_manager.city_data_source.trips_data_source.trips_data_source.TripsDataSource*

load_raw()

Method for loading the data to be preprocessed. Since the data format differs in the various datasets, the method is left abstract. Each city has its own implementation. All implementations will read the data through the pandas readcsv method

Returns nothing

normalise(*year*, *month*)

This method is used to standardize the data format. Again the implementation is highly dependent on the data source and almost all modules override the method.

Returns A normalized pandas.DataFrame

odysseus.city_data_manager.city_data_source.trips_data_source.louisville_scooter_trips module

class LouisvilleScooterTrips

Bases: *odysseus.city_data_manager.city_data_source.trips_data_source.trips_data_source.TripsDataSource*

load_raw()

Method for loading the data to be preprocessed. Since the data format differs in the various datasets, the method is left abstract. Each city has its own implementation. All implementations will read the data through the pandas readcsv method

Returns nothing

normalise(*year*, *month*)

This method is used to standardize the data format. Again the implementation is highly dependent on the data source and almost all modules override the method.

Returns A normalized pandas.DataFrame

odysseus.city_data_manager.city_data_source.trips_data_source.minneapolis_scooter_trips module

class MinneapolisScooterTrips

Bases: *odysseus.city_data_manager.city_data_source.trips_data_source.trips_data_source.TripsDataSource*

load_raw(*year*, *month*)

Method for loading the data to be preprocessed. Since the data format differs in the various datasets, the method is left abstract. Each city has its own implementation. All implementations will read the data through the pandas readcsv method

Returns nothing

normalise(*year*, *month*)

This method is used to standardize the data format. Again the implementation is highly dependent on the data source and almost all modules override the method.

Returns A normalized pandas.DataFrame

odysseus.city_data_manager.city_data_source.trips_data_source.new_york_city_bikes_trips module

class NewYorkCityBikeTrips(*city_name*)

Bases: *odysseus.city_data_manager.city_data_source.trips_data_source.trips_data_source.TripsDataSource*

load_raw(*year*, *month*)

Method for loading the data to be preprocessed. Since the data format differs in the various datasets, the method is left abstract. Each city has its own implementation. All implementations will read the data through the pandas readcsv method

Returns nothing

normalise(*year*, *month*)

This method is used to standardize the data format. Again the implementation is highly dependent on the data source and almost all modules override the method.

Returns A normalized pandas.DataFrame

odysseus.city_data_manager.city_data_source.trips_data_source.new_york_city_taxi_trips module

class NewYorkCityTaxiTrips(*city_name*)

Bases: *odysseus.city_data_manager.city_data_source.trips_data_source.trips_data_source.TripsDataSource*

load_raw(*year*, *month*)

Method for loading the data to be preprocessed. Since the data format differs in the various datasets, the method is left abstract. Each city has its own implementation. All implementations will read the data through the pandas readcsv method

Returns nothing

odysseus.city_data_manager.city_data_source.trips_data_source.norfolk_scooter_trips module

class NorfolkScooterTrips

Bases: *odysseus.city_data_manager.city_data_source.trips_data_source.trips_data_source.TripsDataSource*

load_raw()

Method for loading the data to be preprocessed. Since the data format differs in the various datasets, the method is left abstract. Each city has its own implementation. All implementations will read the data through the pandas readcsv method

Returns nothing

normalise(*year*, *month*)

This method is used to standardize the data format. Again the implementation is highly dependent on the data source and almost all modules override the method.

Returns A normalized pandas.DataFrame

city_data_source.trips_data_source.trips_data_source module

class TripsDataSource(*city_name*, *data_source_id*, *vehicles_type_id*)

Bases: object

TripsDataSource is an abstract class that contains the information needed to describe a trip. This class is implemented by the other classes of this module. The constructor method takes as parameters:

Parameters

- **city_name** (*str*) – City name. The name also serves to determine the timezone to which the city belongs
- **data_source_id** (*str*) – Data source from which the information is taken. This allows us to have multiple data sources associated with the same city (for example from different operators)
- **vehicles_type_id** (*str*) – Type of service represented by the data source (e.g. car sharing or e-scooter)

load_norm(*year*, *month*)

Load a previously created normalized file from memory. It requests month and year as parameters, and checks if the file for that period exists in memory (looking for it with the same format as *save_norm* in the city folder). If it exists, it returns a pandas.DataFrame containing the data read, otherwise it returns an empty DataFrame

Parameters

- **year** (*int*) – year expressed as a four-digit number (e.g. 1999)
- **month** (*int*) – month expressed as a number (e.g. for November the method expects to receive 11)

Returns If the file exists, it returns a pandas.DataFrame containing the data read, otherwise it returns an empty DataFrame

load_raw()

Method for loading the data to be preprocessed. Since the data format differs in the various datasets, the method is left abstract. Each city has its own implementation. All implementations will read the data through the pandas readcsv method

Returns nothing

normalise()

This method is used to standardize the data format. Again the implementation is highly dependent on the data source and almost all modules override the method.

Returns A normalized pandas.DataFrame

save_norm()

It stores normalized data both in a csv file and in a pickle file. The files produced are of the format *<year>_<month number>.csv* (or .pickle). For example *2017_11.csv*.

Returns nothing

Module contents

Module contents

odysseus.city_data_manager.city_geo_trips package

Submodules

odysseus.city_data_manager.city_geo_trips.austin_geo_trips module

class AustinGeoTrips(*city_name='Austin'*, *trips_data_source_id='city_open_data'*, *year=2019*, *month=8*)
Bases: *odysseus.city_data_manager.city_geo_trips.city_geo_trips.CityGeoTrips*

get_trips_od_gdfs()
This method is used to store the movements, using the Shapely library. The normalized data is loaded (—-> reference to save_norm magari<——) and the method builds three GeoDataFrame. The trips are encoded using an object of the LineString class from the Shapely library. They are described as a segment having the coordinates of departure and arrival as extremes. In addition, two more GeoDataFrames are created, using objects of the Shapely.Point class to describe departures and arrivals.

Returns nothing

odysseus.city_data_manager.city_geo_trips.big_data_db_geo_trips module

class BigDataDBGeoTrips(*city_name*, *trips_data_source_id*, *year*, *month*)
Bases: *odysseus.city_data_manager.city_geo_trips.city_geo_trips.CityGeoTrips*

odysseus.city_data_manager.city_geo_trips.calgary_geo_trips module

class CalgaryGeoTrips(*city_name='Calgary'*, *trips_data_source_id='city_open_data'*, *year=2019*, *month=7*)
Bases: *odysseus.city_data_manager.city_geo_trips.city_geo_trips.CityGeoTrips*

get_trips_od_gdfs()
This method is used to store the movements, using the Shapely library. The normalized data is loaded (—-> reference to save_norm magari<——) and the method builds three GeoDataFrame. The trips are encoded using an object of the LineString class from the Shapely library. They are described as a segment having the coordinates of departure and arrival as extremes. In addition, two more GeoDataFrames are created, using objects of the Shapely.Point class to describe departures and arrivals.

Returns nothing

odysseus.city_data_manager.city_geo_trips.chicago_geo_trips module

class ChicagoGeoTrips(*city_name='Chicago'*, *trips_data_source_id='city_open_data'*, *year=2019*, *month=7*)
Bases: *odysseus.city_data_manager.city_geo_trips.city_geo_trips.CityGeoTrips*

get_trips_od_gdfs()
This method is used to store the movements, using the Shapely library. The normalized data is loaded (—-> reference to save_norm magari<——) and the method builds three GeoDataFrame. The trips are encoded using an object of the LineString class from the Shapely library. They are described as a segment having the coordinates of departure and arrival as extremes. In addition, two more GeoDataFrames are created, using objects of the Shapely.Point class to describe departures and arrivals.

Returns nothing

odysseus.city_data_manager.city_geo_trips.city_geo_trips module

class CityGeoTrips(*city_name*, *trips_data_source_id*, *year*, *month*)
Bases: object

This abstract class deals with managing geographic travel information (e.g. departure, arrival, distance, etc.). This class is implemented by the other classes of this module. The constructor method takes as parameters:

Parameters

- **city_name** (*str*) – City name. The name also serves to determine the timezone to which the city belongs
- **trips_data_source_id** (*str*) – Data source from which the information is taken. This allows us to have multiple data sources associated with the same city (for example from different operators)
- **year** (*int*) – year expressed as a four-digit number (e.g. 1999)
- **month** (*int*) – month expressed as a number (e.g. for November the method expects to receive 11)

get_trips_od_gdfs()
This method is used to store the movements, using the Shapely library. The normalized data is loaded (—-> reference to save_norm magari<——) and the method builds three GeoDataFrame. The trips are encoded using an object of the LineString class from the Shapely library. They are described as a segment having the coordinates of departure and arrival as extremes. In addition, two more GeoDataFrames are created, using objects of the Shapely.Point class to describe departures and arrivals.

Returns nothing

load()
Load from memory, using the pickle file created by the save methods, the three GeoDataFrame

Returns nothing

save_points(*points*, *filename*)
Support method to save_data_points. It stores the points passed to it as a parameter both on csv file and on pickle.

Parameters

- **points** (*geopandas.GeoDataFrame*) – A GeoDataFrame describing the information of points to be saved
- **filename** (*str*) – Filename

Returns nothing

save_points_data()
Stores the points representing start and finish on file

Returns nothing

save_trips()
It stores on file the segments that represent the path between start and finish

Returns nothing

odysseus.city_data_manager.city_geo_trips.kansas_city_geo_trips module

class KansasCityGeoTrips(*city_name='Kansas City'*, *trips_data_source_id='city_open_data'*, *year=2019*, *month=7*)
Bases: *odysseus.city_data_manager.city_geo_trips.city_geo_trips.CityGeoTrips*

odysseus.city_data_manager.city_geo_trips.louisville_geo_trips module

class LouisvilleGeoTrips(*city_name='Louisville'*, *trips_data_source_id='city_open_data'*, *year=2019*, *month=7*)
Bases: *odysseus.city_data_manager.city_geo_trips.city_geo_trips.CityGeoTrips*

odysseus.city_data_manager.city_geo_trips.minneapolis_geo_trips module

class MinneapolisGeoTrips(*city_name='Minneapolis'*, *trips_data_source_id='city_open_data'*, *year=2019*, *month=7*)
Bases: *odysseus.city_data_manager.city_geo_trips.city_geo_trips.CityGeoTrips*

get_trips_od_gdfs()
This method is used to store the movements, using the Shapely library. The normalized data is loaded (—-> reference to save_norm magari<——) and the method builds three GeoDataFrame. The trips are encoded using an object of the LineString class from the Shapely library. They are described as a segment having the coordinates of departure and arrival as extremes. In addition, two more GeoDataFrames are created, using objects of the Shapely.Point class to describe departures and arrivals.

Returns nothing

odysseus.city_data_manager.city_geo_trips.norfolk_geo_trips module

class NorfolkGeoTrips(*city_name='Norfolk'*, *trips_data_source_id='city_open_data'*, *year=2019*, *month=8*)
Bases: *odysseus.city_data_manager.city_geo_trips.city_geo_trips.CityGeoTrips*

get_trips_od_gdfs()
This method is used to store the movements, using the Shapely library. The normalized data is loaded (—-> reference to save_norm magari<——) and the method builds three GeoDataFrame. The trips are encoded using an object of the LineString class from the Shapely library. They are described as a segment having the coordinates of departure and arrival as extremes. In addition, two more GeoDataFrames are created, using objects of the Shapely.Point class to describe departures and arrivals.

Returns nothing

odysseus.city_data_manager.city_geo_trips.nyc_citi_bike_geo_trips module

class NewYorkCityBikeGeoTrips(*city_name='New_York_City'*, *trips_data_source_id='citi_bike'*, *year=2017*, *month=1*)
Bases: *odysseus.city_data_manager.city_geo_trips.city_geo_trips.CityGeoTrips*

get_trips_od_gdfs()
This method is used to store the movements, using the Shapely library. The normalized data is loaded (—-> reference to save_norm magari<——) and the method builds three GeoDataFrame. The trips are encoded using an object of the LineString class from the Shapely library. They are described as a segment having

the coordinates of departure and arrival as extremes. In addition, two more GeoDataFrames are created, using objects of the Shapely.Point class to describe departures and arrivals.

Returns nothing

Module contents

odysseus.city_data_manager.config package

Submodules

odysseus.city_data_manager.config.config module

Module contents

Module contents

odysseus.demand_modelling package

Subpackages

odysseus.demand_modelling.demand_model_configs package

Submodules

odysseus.demand_modelling.demand_model_configs.default_config module

Module contents

Submodules

odysseus.demand_modelling.demand_model module

odysseus.demand_modelling.loader module

class Loader(*city*, *data_source_id*, *year*, *month*)

Bases: object

read_data()

Module contents

odysseus.simulator package

Subpackages

odysseus.simulator.demand_model_validation package

Submodules

odysseus.simulator.demand_model_validation.model_validation module

odysseus.simulator.demand_model_validation.model_validation_plot module

odysseus.simulator.demand_model_validation.model_validation_utils module

get_day_moments(*sim_reqs_eventG*, *sim_reqs_traceB*)

get_double_grouped_zones_errs(*sim_reqs_eventG*, *sim_reqs_traceB*, *group_cols*)

get_grouped_reqs_count(*group_col*, *sim_reqs_eventG*, *sim_reqs_traceB*)

get_grouped_zones_errs(*sim_reqs_eventG*, *sim_reqs_traceB*, *group_col*)

get_od_err(*grid*, *sim_reqs_eventG*, *sim_reqs_traceB*)

get_od_err_daymoments(*grid*, *sim_reqs_eventG*, *sim_reqs_traceB*)

get_plot_samples(*ia_threshold*, *sim_reqs_eventG*, *trace_timeouts*)

get_tot_zones_errs(*sim_reqs_eventG*, *sim_reqs_traceB*)

Module contents

odysseus.simulator.multiple_runs package

Submodules

odysseus.simulator.multiple_runs.multiple_runs module

odysseus.simulator.multiple_runs.spark_multiple_runs module

Module contents

odysseus.simulator.simulation package

Submodules

odysseus.simulator.simulation.charging_primitives module

class ChargingPrimitives(*env*, *sim*)
Bases: object

charge_vehicle(*charge_dict*)

check_system_charge(*booking_request*, *vehicle*, *charging_strategy*)

check_user_charge(*booking_request*, *vehicle*)

get_cr_soc_delta(*origin_id*, *destination_id*, *vehicle*)

get_distance(*origin_id*, *destination_id*)

get_timeout(*origin_id*, *destination_id*)

init_charge(*booking_request*, *vehicle*, *beta*)

odysseus.simulator.simulation.charging_strategies module

class ChargingStrategy(*env*, *sim*)
Bases: *odysseus.simulator.simulation.charging_primitives.ChargingPrimitives*

check_charge(*booking_request*, *vehicle*)

get_charge_dict(*vehicle*, *charge*, *booking_request*, *operator*, *charging_relocation_strategy*)

odysseus.simulator.simulation.model_driven_simulator module

odysseus.simulator.simulation.relocation_primitives module

odysseus.simulator.simulation.relocation_strategies module

odysseus.simulator.simulation.scooter_relocation_primitives module

class ScooterRelocationPrimitives(*env*, *sim*)
Bases: object

drop_off_scooter(*zone_id*, *time*, *move_vehicles=False*, *vehicle_ids=None*)

magically_relocate_scooter(*scooter_relocation*)

pick_up_scooter(*zone_id*, *time*, *move_vehicles=False*, *vehicle_ids=None*)

relocate_scooter_multiple_zones(*scheduled_relocation*, *collection_path*, *distribution_path*, *worker*)

relocate_scooter_single_zone(*scooter_relocation*, *move_vehicles=False*, *worker=None*)

reset_current_hour_stats()

update_current_hour_stats(*booking_request*)

update_relocation_stats(*scooter_relocation*)

init_scooter_relocation(*vehicle_ids*, *start_time*, *start_zone_ids*, *end_zone_ids*, *distance*, *duration*, *worker_id='ND'*)

odysseus.simulator.simulation.scooter_relocation_strategies module

odysseus.simulator.simulation.sim_metrics module

class SimMetrics(*metrics_dict*)
Bases: object

metrics_iter()

update_metrics(*metrics*, *value*)

odysseus.simulator.simulation.simulator module

odysseus.simulator.simulation.trace_driven_simulator module

odysseus.simulator.simulation.vehicle_relocation_primitives module

class VehicleRelocationPrimitives(*env*, *sim*)
Bases: object

drop_off_vehicle(*vehicle_relocation*)

get_cr_soc_delta(*origin_id*, *destination_id*, *vehicle*)

get_relocation_distance(*vehicle_relocation*)

get_timeout(*origin_id*, *destination_id*)

pick_up_vehicle(*vehicle_relocation*)

relocate_vehicle(*vehicle_relocation*)

init_vehicle_relocation(*vehicle_ids*, *start_time*, *start_zone_id*, *end_zone_id*, *distance=None*, *duration=0*)

odysseus.simulator.simulation.vehicle_relocation_strategies module

class VehicleRelocationStrategy(*env*, *sim*)
Bases: *odysseus.simulator.simulation.vehicle_relocation_primitives.VehicleRelocationPrimitives*

check_vehicle_relocation(*booking_request*, *vehicles=None*)

choose_ending_zone(*daytype=None*, *hour=None*, *n=1*)

choose_starting_zone(*daytype=None*, *hour=None*, *n=1*)

generate_relocation_schedule(*daytype*, *hour*)

Module contents

odysseus.simulator.simulation_data_structures package

Submodules

odysseus.simulator.simulation_data_structures.charging_station module

class ChargingStation(*env*, *num_poles*, *zone_id*, *station_conf*, *sim_scenario_conf*, *sim_start_time*)
Bases: *odysseus.supply_modelling.charging_station.Pole*

charge(*vehicle*, *start_time*, *soc_delta_charging_trip*, *duration*)

monitor(*data*, *resource*)

odysseus.simulator.simulation_data_structures.vehicle module

class Vehicle(*env*, *plate*, *start_zone*, *start_soc*, *vehicle_config*, *energymix_conf*, *sim_scenario_conf*, *sim_start_time*)
Bases: *odysseus.supply_modelling.vehicle.Vehicle*

booking(*booking_request*)

charge(*percentage*)

odysseus.simulator.simulation_data_structures.zone module

class Zone(*env*, *zone_id*, *sim_start_time*, *vehicles*)
Bases: object

add_vehicle(*t*)

remove_vehicle(*t*)

update_status(*t*)

Module contents

odysseus.simulator.simulation_input package

Subpackages

odysseus.simulator.simulation_input.sim_configs_versioned package

Subpackages

odysseus.simulator.simulation_input.sim_configs_versioned.generalisation package

Subpackages

odysseus.simulator.simulation_input.sim_configs_versioned.generalisation.fleet_size package

Submodules

odysseus.simulator.simulation_input.sim_configs_versioned.generalisation.fleet_size.multiple_runs_conf module

odysseus.simulator.simulation_input.sim_configs_versioned.generalisation.fleet_size.sim_general_conf module

Module contents

Module contents

odysseus.simulator.simulation_input.sim_configs_versioned.isc2 package

Subpackages

odysseus.simulator.simulation_input.sim_configs_versioned.isc2.isc2_set1a package

Submodules

odysseus.simulator.simulation_input.sim_configs_versioned.isc2.isc2_set1a.multiple_runs_conf module

odysseus.simulator.simulation_input.sim_configs_versioned.isc2.isc2_set1a.sim_general_conf module

Module contents

odysseus.simulator.simulation_input.sim_configs_versioned.isc2.isc2_set1b package

Submodules

odysseus.simulator.simulation_input.sim_configs_versioned.isc2.isc2_set1b.multiple_runs_conf module

odysseus.simulator.simulation_input.sim_configs_versioned.isc2.isc2_set1b.sim_general_conf module

Module contents

odysseus.simulator.simulation_input.sim_configs_versioned.isc2.isc2_set1c package

Submodules

odysseus.simulator.simulation_input.sim_configs_versioned.isc2.isc2_set1c.multiple_runs_conf module

odysseus.simulator.simulation_input.sim_configs_versioned.isc2.isc2_set1c.sim_general_conf module

Module contents

odysseus.simulator.simulation_input.sim_configs_versioned.isc2.isc2_set2 package

Submodules

odysseus.simulator.simulation_input.sim_configs_versioned.isc2.isc2_set2.multiple_runs_conf module

odysseus.simulator.simulation_input.sim_configs_versioned.isc2.isc2_set2.sim_general_conf module

Module contents

Module contents

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo package

Subpackages

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.louisville_multiple_runs_eventG_magic_rel package

Submodules

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.louisville_multiple_runs_eventG_magic_rel module

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.louisville_multiple_runs_eventG_magic_rel module

Module contents

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.louisville_multiple_runs_eventG_no_reloca package

Submodules

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.louisville_multiple_runs_eventG_no_reloca module

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.louisville_multiple_runs_eventG_no_reloca
module

Module contents

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.louisville_multiple_runs_eventG_relocation
package

Submodules

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.louisville_multiple_runs_eventG_relocation
module

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.louisville_multiple_runs_eventG_relocation
module

Module contents

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.louisville_multiple_runs_eventG_test
package

Submodules

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.louisville_multiple_runs_eventG_test.multi
module

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.louisville_multiple_runs_eventG_test.sim_g
module

Module contents

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.louisville_multiple_runs_traceB_test
package

Submodules

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.louisville_multiple_runs_traceB_test.multip
module

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.louisville_multiple_runs_traceB_test.sim_g
module

Module contents

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.minneapolis_multiple_runs_eventG_magic_
package

Submodules

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.minneapolis_multiple_runs_eventG_magic
module

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.minneapolis_multiple_runs_eventG_magic
module

Module contents

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.minneapolis_multiple_runs_eventG_no_rel
package

Submodules

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.minneapolis_multiple_runs_eventG_no_rel
module

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.minneapolis_multiple_runs_eventG_no_rel
module

Module contents

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.minneapolis_multiple_runs_eventG_reloca
package

Submodules

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.minneapolis_multiple_runs_eventG_reloca
module

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.minneapolis_multiple_runs_eventG_reloca
module

Module contents

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.minneapolis_multiple_runs_eventG_test
package

Submodules

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.minneapolis_multiple_runs_eventG_test.m
module

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.minneapolis_multiple_runs_eventG_test.si
module

Module contents

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.minneapolis_multiple_runs_traceB_test package

Submodules

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.minneapolis_multiple_runs_traceB_test.mu module

odysseus.simulator.simulation_input.sim_configs_versioned.leonardo.minneapolis_multiple_runs_traceB_test.sim module

Module contents

Module contents

odysseus.simulator.simulation_input.sim_configs_versioned.test package

Subpackages

odysseus.simulator.simulation_input.sim_configs_versioned.test.big_data_db_test package

Submodules

odysseus.simulator.simulation_input.sim_configs_versioned.test.big_data_db_test.multiple_runs_conf module

odysseus.simulator.simulation_input.sim_configs_versioned.test.big_data_db_test.sim_general_conf module

odysseus.simulator.simulation_input.sim_configs_versioned.test.big_data_db_test.single_run_conf module

Module contents

odysseus.simulator.simulation_input.sim_configs_versioned.test.city_single_run_test package

Submodules

odysseus.simulator.simulation_input.sim_configs_versioned.test.city_single_run_test.sim_general_conf module

Module contents

odysseus.simulator.simulation_input.sim_configs_versioned.test.mito_mobility_test package

Submodules

odysseus.simulator.simulation_input.sim_configs_versioned.test.mito_mobility_test.sim_general_conf module

odysseus.simulator.simulation_input.sim_configs_versioned.test.mito_mobility_test.single_run_conf module

Module contents

Module contents

odysseus.simulator.simulation_input.sim_configs_versioned.xian package

Subpackages

odysseus.simulator.simulation_input.sim_configs_versioned.xian.charging_relocation_strategies_test package

Submodules

odysseus.simulator.simulation_input.sim_configs_versioned.xian.charging_relocation_strategies_test.multiple_ru module

odysseus.simulator.simulation_input.sim_configs_versioned.xian.charging_relocation_strategies_test.sim_genera module

Module contents

odysseus.simulator.simulation_input.sim_configs_versioned.xian.turin_test package

Submodules

odysseus.simulator.simulation_input.sim_configs_versioned.xian.turin_test.sim_run_conf module

Module contents

Module contents

Module contents

Submodules

odysseus.simulator.simulation_input.costs_conf module

odysseus.simulator.simulation_input.sim_config_grid module

class EFFCS_SimConfGrid(*conf_grid*)
Bases: object

odysseus.simulator.simulation_input.sim_input module

class SimInput(*conf_tuple*)
Bases: object

get_booking_requests_list()

init_charging_poles()

init_relocation()

init_vehicles()

init_workers()

odysseus.simulator.simulation_input.sim_input_paths module

odysseus.simulator.simulation_input.station_conf module

odysseus.simulator.simulation_input.vehicle_conf module

Module contents

odysseus.simulator.simulation_output package

Submodules

odysseus.simulator.simulation_output.multiple_runs_plotter module

odysseus.simulator.simulation_output.plot_multiple_runs module

odysseus.simulator.simulation_output.sim_output module

class SimOutput(*sim*)
Bases: object

odysseus.simulator.simulation_output.sim_output_plotter module

odysseus.simulator.simulation_output.sim_stats module

class SimStats
Bases: object

get_stats_from_sim(*sim*)

Module contents

odysseus.simulator.single_run package

Submodules

odysseus.simulator.single_run.get_eventG_input module

get_eventG_input(*conf_tuple*)

odysseus.simulator.single_run.get_traceB_input module

get_traceB_input(*conf_tuple*)

odysseus.simulator.single_run.run_eventG_sim module

odysseus.simulator.single_run.run_traceB_sim module

odysseus.simulator.single_run.single_run module

Module contents

Module contents

odysseus.supply_modelling package

Subpackages

odysseus.supply_modelling.supply_model_configs package

Submodules

odysseus.supply_modelling.supply_model_configs.default_config module

Module contents

Submodules

odysseus.supply_modelling.charging_station module

class Pole(*station_config*)
Bases: object

get_charging_time_from_energy(*energy_mj*)

get_energy_from_charging_time(*charging_time*)

get_fuelcost_from_energy(*energy_mj*)

odysseus.supply_modelling.energymix_loader module

class EnergyMix(*city*, *year*)
Bases: object

evaluate_emissions()

evaluate_energy()

open_database()

odysseus.supply_modelling.supply_model module

class SupplyModel(*supply_model_conf*, *year*)
Bases: object

init_charging_poles()

init_relocation()

init_vehicles()

init_workers()

geodataframe_charging_points(*city*, *engine_type*, *station_location*)

odysseus.supply_modelling.vehicle module

class Vehicle(*vehicle_config*, *energy_mix_conf*)
Bases: object

consumption_to_percentage(*consumption*)

distance_to_consumption(*distance*)

distance_to_tanktowheel_emission(*distance*)

distance_to_welltotank_emission(*distance*)

from_kml_to_energyperkm()

from_kml_to_lkm()

get_charging_time_from_perc(*actual_level_perc*, *flow_amount*, *profile*, *beta=100*)

get_percentage_from_charging_time(*charging_time*, *flow_amount*, *profile*)

percentage_to_consumption(*percentage*)

tanktowheel_energy_from_perc(*percentage*)

welltotank_energy_from_perc(*percentage*)

Module contents

odysseus.utils package

Submodules

odysseus.utils.bookings_utils module

update_req_time_info(*booking_request*)

odysseus.utils.cost_utils module

charging_station_lord_cost(*costs*)

get_fuelcost_from_energy(*fuel_type*, *fuel_costs*, *energy_mj*)

insert_scenario_costs(*df*, *sim_scenario_conf*, *vehicles_cost_conf*, *poles_cost_conf*)

insert_sim_costs(*df*, *sim_scenario_conf*, *fuel_costs*, *administrative_cost_conf*, *vehicles_cost_conf*)

odysseus.utils.geospatial_utils module

add_grouped_count_to_grid(*grid*, *trips_locations*, *group_col*, *od_key*, *aggfunc='count'*)

get_city_grid_as_gdf(*total_bounds*, *crs*, *bin_side_length*)

get_city_grid_as_matrix(*total_bounds*, *bin_side_length*)

get_od_distance(*grid*, *origin_id*, *destination_id*)

get_random_point_from_linestring(*linestring*)

get_random_point_from_shape(*shape*)

miles_to_meters(*miles*)

my_haversine(*lon1*, *lat1*, *lon2*, *lat2*)

odysseus.utils.path_utils module

check_create_path(*path*)

odysseus.utils.time_utils module

get_grouped_aggfunc(*df*, *group_cols*, *stats_col*, *aggfuncs*)

get_grouped_resampled_aggfunc(*df*, *group_cols*, *freq*, *stats_col*, *aggfuncs*)

get_grouped_resampled_count(*df*, *group_cols*, *freq*)

get_grouped_resampled_count_aggfunc(*df*, *group_cols*, *freq*, *aggfuncs*)

get_hourly_count_with_time_cols(*trips_df_norm*, *start_or_end*)

get_hourly_mean_with_time_cols(*df_norm*, *start_or_end*, *mean_col*)

get_resampled_aggfunc(*df*, *freq*, *stats_col*, *aggfuncs*)

get_resampled_grouped_aggfunc(*trips_df_norm*, *start_or_end*, *stats_col*, *time_categorical_col*, *freq*, *aggfunc*)

get_resampled_grouped_count_aggfunc(*trips_df_norm*, *start_or_end*, *time_group_col*, *freq*, *aggfunc*)

get_time_group_columns(*trips_df_norm*)

get_time_grouped_hourly_count(*df_norm*, *start_or_end*, *which_df*)

get_time_grouped_hourly_mean(*df_norm*, *start_or_end*, *which_df*, *mean_col*)

get_weekday_int_from_string(*s*)

get_weekday_string_from_int(*i*)

month_year_iter(*start_month*, *start_year*, *end_month*, *end_year*)
MonthYear Iterator. End month is included. :param start_month: :param start_year: :param end_month: :param end_year: :return:

reshape_time_grouped_signature(*time_grouped_signatures*)

update_req_time_info(*booking_request*)

weekday2vec(*weekdays*)
Weekdays to one-hot vector :param weekdays: Array of integer weekdays, where Monday is 0 and Sunday is 6. :return: Array of one-hot vectors, representing weekdays.

Module contents

5.2.2 Module contents

CHAPTER

SIX

INTRODUCTION

ODySSEUS is a data management and simulation software for mobility data, focused mostly on shared fleets in urban environments.

Its goal is to provide a general, easy-to-use framework to simulate shared mobility scenarios across different cities using real-world data.

Internally, it makes use of several open-source Python libraries for geospatial and mobility analysis, such as geopandas (https://geopandas.org/) and scikit-mobility [5] (https://scikit-mobility.github.io/scikit-mobility/).

ODySSEUS is composed by four main functional modules, each one coming with its own API, command line interface and GUI:

- **City Data Manager**
- **Demand Modelling**
- **Supply Modelling**
- **Simulator**

PYTHON MODULE INDEX

O

INDEX

E

F

G

I

K

L

M

N

O

T

U

V

W

Z