By Abeer Alyasiri

User Guide v1.0

User guide for the yoGERT toolbox

This guide covers usage of all public modules and functions. Every function can be accessed via moduleName.functionName() after having "import moduleName" at the beginning of your Python file.

Table of Contents

PreProcessing module

PreProcessing.ValidateCSV(csvpath, directoryname)

- Confirms that CSV is valid and creates a CSV based on the input and corrects column names while removing invalid data.
- The new column names are a prerequisite before using any of the toolbox's analysis functionality on the user's geo-data.

Parameters	• **csvpath** (*string*): a full path to the input CSV file. • **directoryname** (*string*): string directory that will be created within the current directory to store processed traces.
Returns	**CSVCreated** - boolean to indicate the status of the uploaded CSV.
Return type	boolean
Exceptions	• **InvalidInputDataException** - when the inputted CSV file is invalid because it doesn't have all required columns (latitude, longitude, time).

Example:
bool CSVCreated = PreProcessing.ValidateCSV(“./directoryname/filename.csv”,
“directoryname”)

episodeGeneration module

episodeGeneration.createTrace(csv_path, tracefolder_fullpath)

- Assigns a unique ID to each GPS ping point of the inputted trace file and creates a new CSV file, called trace.csv, for the trace's geo-data in the inputted directory path.
- The function requires that the inputted trace file is for one entity only. This can be done by calling the preprocessing function.

Parameters	• **csv_path** (*string*): a full path to the input CSV file. • **tracefolder_fullpath** (*string*): a directory path where the new CSV file will be created.
Returns	**N/A**

Return type	N/A
Exceptions	N/A

Example:
episodeGeneration.createTrace(“./directoryname/filename.csv”, “./directoryname”)

episodeGeneration.createSegments(tracefolder_fullpath)

- Finds segments of the trace.csv file and creates a CSV file for segment information, called segments.csv, in the inputted directory path.
- The function requires that the inputted directory contains a CSV file for a trace’s geo-data with a unique ID for each row. It can be done by calling the episodeGeneration.createTrace function.

Parameters	• **tracefolder_fullpath** (*string*): a directory path that contains trace.csv and where the new CSV file will be created.
Returns	**N/A**
Return type	N/A
Exceptions	N/A

Example:
episodeGeneration.createSegments(“./directoryname”)

episodeGeneration.findStops(tracefolder_fullpath)

- Analyzes segments in segments.csv file and creates a CSV file for stop points, called stops.csv, in a newly created directory, called stop, within the input directory path.
- The function requires that the input directory contains a CSV file for a trace’s segment information. It can be done by calling the episodeGeneration.createSegments function.

Parameters	• **tracefolder_fullpath** (*string*): a directory path that contains trace.csv and where the new directory and CSV file will be created.
Returns	**N/A**
Return type	N/A
Exceptions	• **FileExistsError** - when stops directory exists within the inputted

	directory.

Example:
episodeGeneration.createStops("./directoryname")

episodeGeneration.cleanStops(tracefolder_fullpath)

- Updates stops.csv file at the stop directory within the inputted directory path with the inputted filtering parameters. It removes any stop points that don't satisfy the filtering tolerances.
- The function requires that the input directory contains a directory called stop with a CSV file for a trace's stop point information. It can be done by calling the episodeGeneration.createStops function.

Parameters	• **tracefolder_fullpath** (*string*): a directory path that contains the directory called stop that has stops.csv.
Returns	**N/A**
Return type	N/A
Exceptions	N/A

Example:
episodeGeneration.cleanStops("./directoryname")

episodeGeneration.createEpisodes(tracefolder_fullpath)

- Generates episodes for the trace.csv file in the inputted directory path and creates a new directory, called episode, that will contain all the episodes' information as CSV files.
- The function requires that the input directory contains a CSV file for a trace's segment information. It can be done by calling the episodeGeneration.createSegments function.

Parameters	• **tracefolder_fullpath** (*string*): a directory path that contains trace.csv and where the new directory and CSV files will be created.
Returns	**N/A**
Return type	N/A
Exceptions	• **FileExistsError** - when episode directory exists within the inputted

	directory.

Example:
episodeGeneration.createEpisodes(“./directoryname”)

episodeGeneration.episodeGenerator(csv_path, tracefolder_fullpath, title)

- Generates episodes for the inputted geo-data and creates new directories and CSV files to store information on segments, stops, and episodes for the inputted trace information.
- The function requires that the inputted trace file is for one entity only. This can be done by calling the preprocessing function..

Parameters	● **csv_path** (*string*): a file path for the trace’s geo-data. ● **tracefolder_fullpath** (*string*): a directory path that contains the user’s geo-data. ● **title** (*string*): a directory name where all the trace’s information should be stored.
Returns	**N/A**
Return type	N/A
Exceptions	● **FileExistsError** - when episode, stop, or trace directory exists within the inputted directory.

Example:
episodeGeneration.createGenerator(“./directoryname/filename.csv”, “./directoryname”,”trace1”)

episodeGeneration.summarymode(tracefilepath)

- Finds the most used travel mode for the inputted trace.csv file and creates a new CSV file containing the summary mode.
- The function requires that the inputted file directory contains the trace’s information including episodes. It can be done by calling the episodeGeneration.episodeGenerator function or calling 4 episodeGeneration functions: createTrace, createSegments, findStops, and createEpisodes.

Parameters	● **tracefilepath** (*string*): a file path that contains trace.csv and where the new CSV file will be created.

Returns	**N/A**
Return type	N/A
Exceptions	N/A

Example:
episodeGeneration.summarymode(“./directoryname/filename.csv”)

episodeGeneration.createStats(tracefolder_fullpath)

- Analyzes the trace’s information and creates a new CSV file, called stats.csv, of ping frequency, mode change count, number of trips, and trace period in the input directory path.
- The function requires that the inputted directory contains the trace’s information including episodes. It can be done by calling the episodeGeneration.episodeGenerator function or calling 4 episodeGeneration functions: createTrace, createSegments, findStops, and createEpisodes.

Parameters	● **tracefolder_fullpath** (*string*): a directory path that contains trace.csv and where the new CSV file will be created.
Returns	**N/A**
Return type	N/A
Exceptions	N/A

Example:
episodeGeneration.createStats(“./directoryname/”)

fetchActivityLocation module

fetchActivityLocations.fetchActivityLocations(inPath, outPath, tol=25)

- Uses the Overpass server to retrieve and create a CSV file for the nearby activity locations to the inputted geo-data.
- Activity locations are amenities that the user might be interested to include in geo-spatial analysis. .

Parameters	● **inPath** (*string*): a full file path to the input CSV file. ● **outPath** (*string*): a file path of where

	the new file for the activity locations should be stored. ● **tol** (integer): tolerance for the search radius of nearby activity locations.
Returns	**0** - nothing is returned when the file is created successfully.
Return type	integer
Exceptions	● **OverpassGatewayTimeout** - when connecting to Overpass server fails because it is at capacity. ● **OverpassTooManyRequests** - when connecting to Overpass server fails because it is at capacity. ● **InvalidInputFileException** - when the inputted CSV file is invalid because it doesn't have all required columns (latitude, longitude, time). ● **WritingFileException** - when the function fails to write to the CSV file.

Example:
fetchActivityLocations.fetchActivityLocaitons("./directoryname/filename.csv", "./directoryname/filename.csv")

NetworkGraph module

NetworkGraph.NetworkGraph(filePath, networkMode=None, episodeAnalysis=True, alternativeAnalysis=False)

- Uses the OSMNX server to create a transportation network graph object for the inputted geo-data.
- NetworkGraph is an object that stores the following information about the network graph: transportation mode, start GPS coordinate, end GPS coordinate, radius distance, and graph of type networkx.MultiDiGraph. The function requires the geo-data to be labeled with unique ideas. This can be done by calling the episodeGeneration.createTrace function.

Parameters	● **filePath** (*string*): a full file path to the input CSV file. ● **networkMode** (*string*): for the entity's mode of transportations for ex: drive or walk. ● **episodeAnalysis** (*boolean*): to know

	the type of inputted geo-data. ● **alternativeAnalysis** (*boolean*): to know the type of analysis required.
Returns	**networkG**
Return type	NetworkGraph
Exceptions	● **InvalidModeException** - when the input value is not a subset of {drive, walk}. ● **EmptyFileException** - when input file path is empty.

Example:
networkG = NetworkGraph.NetworkGraph("./directoryname/filename.csv", "walk", False, False)

NetworkGraph.getNearestNode(self, coord)

- For a NetworkGraph object to find the nearest graph node to a given GPS coordinate.

Parameters	● **coord** (*tuple of integers*): GPS coordinate.
Returns	**node**
Return type	integer
Exceptions	● **OutOfBoundsCoordExceptio** - when the input coordinate is not within the graph area.

Example:
node = networkG.getNearestNode((43.58565864968933, -79.68830703019592))

NetworkGraph.getMode(self)

- For a NetworkGraph object to find the transportation mode of the network.

Parameters	N/A
Returns	**mode**
Return type	string
Exceptions	N/A

Example:
mode = networkG.getMode()

ShortestRouteTrace module

ShortestRouteTrace.ShortestRouteTrace(**networkGraph, filePath, optimizer="time"**)

- Finds the shortest route by some optimizer parameter for a given trace.
- ShortestRouteTrace is an object that stores the following information about the shortest route for a trace: input data, optimizer, graph nodes, and routes.

Parameters	● **networkGraph** (*NetworkGraph*): the network of streets, roads, and walkways for the entire trace. ● **filePath** (*string*): the file path to the CSV file of the trace's geo-data. ● **optimizer** (*string*): the weight type on the graph's edges.
Returns	**traceRoute**
Return type	ShortestRouteTrace
Exceptions	● **InvalidWeightException** - when the inputted optimizer is not a subset of {time, length}. ● **NetworkXNoPath** - when no connection exists between 2 GPS coordinates.

Example:
traceRoute = ShortestRouteTrace.ShortestRouteTrace(networkGraph, "./directoryname/filename.csv", "length")

ShortestRouteEpisode module

ShortestRouteEpisode.ShortestRouteEpisode(**networkGraph, filePath, optimizer="time", sampling=True, samplingDist=50**)

- Finds the shortest route by some optimizer and sampling parameters for a given episode.
- ShortestRouteEpisode is an object that stores the following information about the shortest route for an episode: input data, sampled data, sampling flag, sampling distance, optimizer, graph, graph nodes, and routes.

Parameters	<ul><li>**networkGraph** (*NetworkGraph*): the network of streets, roads, and walkways for the entire trace.</li><li>**filePath** (*string*): the file path to the CSV file of the episode's geo-data.</li><li>**optimizer** (*string*): the weight type on the graph's edges.</li><li>**sampling** (*boolean*): to decide when data sampling is needed to sample GPS coordinates by a specified distance.</li><li>**samplingDist** (*integer*): the sampling distance variable in meters.</li></ul>
Returns	**episodeRoute**
Return type	ShortestRouteEpisode
Exceptions	<ul><li>**InvalidWeightException** - when the inputted optimizer is not a subset of {time, length}.</li><li>**NetworkXNoPath** - when no connection exists between 2 GPS coordinates.</li></ul>

Example:
episodeRoute = ShortestRouteEpisode.ShortestRouteEpisode(networkGraph, "./directoryname/filename.csv", "length")

AlternativeRoute module

AlternativeRoute.AlternativeRoute(filePath, optimizer="time")

- Finds the alternative bus route by some optimizer parameter for a given trace.
- AlternativeRoute is an object that stores the following information about the alternative route for a trace: network graph, and path.

Parameters	<ul><li>**filePath** (*string*): the file path to the CSV file of the trace's geo-data.</li><li>**optimizer** (*string*): the weight type on the graph's edges.</li></ul>
Returns	**alternativeRoute**
Return type	AlternativeRoute
Exceptions	<ul><li>**InvalidWeightException** - when the inputted optimizer is not a subset of</li></ul>

	{time, length}. ● **NetworkXNoPath** - when no connection exists between 2 GPS coordinates.

Example:
alternativeRoute = AlternativeRoute.AlternativeRoute(“./directoryname/filename.csv”, “length”)

Mapping module

Mapping.MapRoute(networkGraph, route, savePath)

- Creates an interactive map for the route and points used for route creation then saves the map as a HTML file.

Parameters	● **networkGraph** (*NetworkGraph*): the network of streets, roads, and walkways for the entire trace. ● **route** (*ShortestRoute or AlternativeRoute*): object that has information of the route and details of how it was created. ● **savePath** (*string*): the full file path where the interactive map will be created.
Returns	**0** - nothing is returned when the file is created successfully.
Return type	integer
Exceptions	N/A

Example:
Mapping.MapRoute(networkGraph, traceRoute, “./directoryname/filename.csv”)

Mapping.MapActivityLocation(activityLocationsFile, stopPointsFile, savePath)

- Creates an interactive map for the activity locations and points used for activity location generation then saves the map as a HTML file.

Parameters	● **activityLocationFile** (*string*): the file path of the trace’s activity locations CSV file. ● **stopPointsFile** (*string*): the file path

	of the trace's stop points CSV file. ● **savePath** (*string*): the full file path where the interactive map will be created.
Returns	**0** - nothing is returned when the file is created successfully.
Return type	integer
Exceptions	N/A

Example:
Mapping.MapActivityLocation("./directoryname/filename.csv", "./directoryname/filename.csv", "./directoryname/filename.html")

Mapping.MapEpisodePoints(GPSCoordFile, savePath)

- Creates an interactive map for the episode points then saves the map as a HTML file.

Parameters	● **activityLocationFile** (*string*): the file path of the episode CSV file. ● **savePath** (*string*): the full file path where the interactive map will be created.
Returns	**0** - nothing is returned when the file is created successfully.
Return type	integer
Exceptions	N/A

Example:
Mapping.MapEpisodePoints("./directoryname/filename.csv", "./directoryname/filename.html")