Module Interface Specification for PyERT

Team 17, Track a Trace
Zabrain Ali
Linqi Jiang
Jasper Leung
Mike Li
Mengtong Shi
Hongzhao Tan

April 5, 2023

1 Revision History

Date	Version	Notes
January 16, 2023	1.0	Edited Abbreviations and Acronyms, Introduction and Notation
January 17, 2023	1.1	Edited Module Decomposition
January 18, 2023	1.2	Edited MIS of all Modules
April 5, 2023	2.0	Modified Document according to feedback from Revision 0

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/paezha/PyERT-BLACK/tree/main/docs/SRS.

symbol	description
ArcGIS	Geographic Information System
CRS	Coordinate-Reference System
CSV	Comma-Separated Values
EPSG	European Petroleum Survey Group
GERT	Graphical Evaluation and Review Technique
GPS	Global Positioning System
MIS	Module Interface Specification
MG	Module Guide
OSM	OpenStreetMap: a free, open geographic database
OSMnx	Python package that lets you download geospatial data from OpenStreetMap
PBF format	Protocolbuffer Binary Format
PyERT	Python-based Episode Reconstruction Toolkit
SHP	shapefile: a geospatial vector data format for GIS software
SRS	Software Requirements Specification
URL	Uniform Resource Locator
2D	two-dimensional

Contents

3 Introduction

The following document details the Module Interface Specifications for the project PyERT. The project aims to re-implement the functionalities of GERT (Dalumpines and Scott, 2018) which uses ArcGIS Pro packages, with open-source packages and libraries, and remove any use of ArcGIS in GERT. The same as the original GERT toolkit, the purpose of the product system is to match GPS trajectories to a transportation network for further analysis ~~to~~ of the GPS data.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/paezha/PyERT-BLACK.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types , derived data types and other derived data types from Pandas, GeoPandas, OSMnx and Shapely libraries that are used by PyERT.

4.1 Primitive Data Types

Data Type	**Notation**	**Description**
character	char	A single symbol or digit
integer	$\mathbb{Z}$	A number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	A number without a fractional component in [1, ∞)
real	$\mathbb{R}$	Any number in (-∞, ∞)
boolean	$\mathbb{B}$	A value of either $True$ or $False$

The specification of PyERT uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, PyERT uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

4.2 Data Types From Libraries

Data Type	Notation	Description
DataFrame	df	Tuple that the value for each field is a sequence. Sequences of different fields could be of different data types, but their lengths will be the same. Each element in a sequence of a DataFrame is associated with the elements of other sequences with the same value of index, a row in a DataFrame is a group of elements in the DataFrame (one element from each field) that are associated with the same value of index.
GeoDataFrame	gdf	A special type of df where there will be always a field of 'geometry' which is a sequence of geometric data type (e.g. Point, LineString, etc.)
Point	Point	A geometric data type that describes a point, which will be a sequence of $\mathbb{R}$, the length of the sequence will be always 2
LineString	ls	A geometric data type that describes a line segment, which will be a sequence of sequences of $\mathbb{R}$, the length of the inner sequences will be always 2
Multi-Directed Graph	MultiDiGraph	A directed graph data type from NetworkX library that is returned by many functions of OSMnx library. A directed graph class that can store multiedges. Multiedges are multiple edges between two nodes. Each edge can hold optional data or attributes.
Polygon	Polygon	A geometric data type that describes a shape will be a sequence of sequences of $\mathbb{R}$, where the first and last inner sequences are the same (connecting to make a closed shape), the length of the inner sequences will be always 2

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	GPS Data Preprocessing Module GPS Data Mode Detection Module Trip Segments and Activity Locations Extraction Module Route Choice Set Generator Module Route Choice Analysis Variables Generator Module Activity Locations Identification Module Main Function Module
Software Decision Module	DataFrame Data Structure Module GeoDataFrame Data Structure Module Geometric Object Analysis and Manipulation Module Network Analysis Module OSM Network Dataset Reader Module Plotting Module

Table 1: Module Hierarchy

6 MIS of GPS Data Preprocessing Module

6.1 Module

GPSPreprocess

6.2 Uses

N/A

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

Routine name	In	Out	Exceptions
GPSPreprocess	data: df of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, latitude: seq of $\mathbb{R}$, longitude: seq of $\mathbb{R}$, Speed_kmh: seq of $\mathbb{R}$)		
getData		processedData: gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Speed_kmh: seq of $\mathbb{R}$, geometry: seq of Point)	

Routine name	In	Out	Exceptions
~~filterData~~			
~~smoothData~~	~~data: gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Speed_kmh: seq of $\mathbb{R}$, geometry: seq of Point)~~		

6.4 Semantics

6.4.1 State Variables

processedData: gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Speed_kmh: seq of $\mathbb{R}$, geometry: seq of Point)

6.4.2 Environment Variables

None

6.4.3 Assumptions

None

6.4.4 Access Routine Semantics

GPSPreprocess(data):

- transition: filters and smooths given DataFrame and converts it to a GeoDataFrame
- output: $out := self$
- exception: None

getData():

- output: $out := preprocessedData$
- exception: None

~~filterData(data):~~

- ~~transition: $data := (\forall x.latitude \wedge x.longitude \wedge x.LocalTime, y.latitude \wedge y.longitude \wedge y.LocalTim$~~
- ~~output: a gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Speed_kmh: seq of $\mathbb{R}$, geometry: seq of Point) with redundant GPS points removed~~
- ~~exception: None~~

~~smoothData(data):~~

- ~~transition: $data := (\forall row : data | row.kmh\_speed < 180)$~~
- ~~output: a gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Speed_kmh: seq of $\mathbb{R}$, geometry: seq of Point) with all outliers removed~~
- ~~exception: None~~

6.4.5 Local Functions

filterData: gdf $\rightarrow$ gdf
filterData(data) $\equiv$ $(\forall x.latitude \wedge x.longitude \wedge x.LocalTime, y.latitude \wedge y.longitude \wedge y.LocalTime : data | x \neq y)$

smoothData: gdf $\rightarrow$ gdf
smoothData(data) $\equiv$ $(\forall row : data | row.kmh\_speed < 180)$

7 MIS of GPS Data Mode Detection Module

7.1 Module

ModeDetection

7.2 Uses

N/A

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Routine name	In	Out	Exceptions
detectModes	processedData: gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Speed_kmh: seq of $\mathbb{R}$, geometry: seq of Point)		
getEpisodeData		episodeData: gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Speed_kmh: seq of $\mathbb{R}$, Mode: seq of String, geometry: seq of Point)	

7.4 Semantics

7.4.1 State Variables

episodeData: gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Speed_kmh: seq of $\mathbb{R}$, Mode: seq of String, geometry: seq of Point

7.4.2 Environment Variables

None

7.4.3 Assumptions

None

7.4.4 Access Routine Semantics

detectModes(processedData):

- transition: Read a processedData GeoDataFrame produced by the GPSPreprocsser module, then ~~partition the given points into segments, then classify these episodes by mode (walk, car, bus, etc.).~~ then classify the start of each Mode. A Walk is classified as a segment of points where the speed never exceeds 2.78 meters per second for at least 60 seconds. A Drive is classified as a segment of points where the speed exceeds 2.78 meters per second for at least 120 seconds. A Stop is classified as a segment of points where the speed never exceeds 0.01 meters per second for at least 120 seconds. Store these episodes into the episodeData GeoDataFrame (a state variable).

- exception: None

getEpisodeData():

- output: ~~A~~ The GeoDataFrame ~~object~~ state variable containing the classified GPS Episodes. It consists of recordID (represented by a sequence of integers), serialID (represented by a sequence of integers), LocalTime (represented by a sequence of strings), Speed_kmh (represented by a sequence of real numbers), Mode(represented by a sequence of Strings), and geometry (represented by a sequence of Points).

- exception: None

7.4.5 Local Functions

None

8 MIS of Trip Segments and Activity Locations Extraction Module

8.1 Module

Extractor

8.2 Uses

N/A

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Access Programs

Routine name	In	Out	Exceptions
extractTripSegments	episodeData: gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Mode: seq of String, geometry: seq of Point)		
getTripSegments		tripSegmentsData: gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Mode: seq of string, geometry: seq of Point)	

Routine name	In	Out	Exceptions
extractActivityLocations	episodeData: gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Speed_kmh: seq of $\mathbb{R}$, Mode: seq of String, geometry: seq of Point)		
getActivityLocations		extractedData : gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Mode: seq of string, geometry: seq of Point)	

8.4 Semantics

8.4.1 State Variables

tripSegments: gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Mode: seq of string, geometry: seq of Point
activityLocations: gdf of (recordID: seq of $\mathbb{Z}$, SerialID: seq of $\mathbb{Z}$, LocalTime: seq of string, Mode: seq of string, geometry: seq of Point)

8.4.2 Environment Variables

None

8.4.3 Assumptions

None

8.4.4 Access Routine Semantics

extractTripSegments(episodeData):

- transition: Read the episodeData GeoDataFrame generated by the ModeDetection module, and extract trip segments (sequences of data points in a travel episode) from this GeoDataFrame. Trip segments are defined as points with a mode that is not a Stop. Store these trip segments into the tripSegments GeoDataFrame (a state variable).
- exception: None

getTripSegments():

- output: ~~A~~ The GeoDataFrame ~~object~~ state variable containing the trip segments data. It consists of recordID (represented by a sequence of integers), serialID (represented by a sequence of integers), LocalTime (represented by a sequence of strings), Speed_kmh (represented by a sequence of real numbers), Mode (represented by a sequence of Strings), and geometry (represented by a sequence of Points).
- exception: None

extractActivityLocations(episodeData):

- transition: Read the episodeData GeoDataFrame generated by the ModeDetection module, and extract activity locations (stops or endpoints of trip segments) from this GeoDataFrame. Activity locations are defined as points with a mode of Stop. Store these ~~trip segments~~ activity locations in the ~~tripSegments~~ activityLocations GeoDataFrame (a state variable).
- exception: None

getActivityLocations():

- output: ~~A~~ The GeoDataFrame ~~object~~ state variable containing the activity location data. It consists of recordID (represented by a sequence of integers), serialID (represented by a sequence of integers), LocalTime (represented by a sequence of strings), Speed_kmh (represented by a sequence of real numbers), Mode(represented by a sequence of Strings), and geometry (represented by a sequence of Points).
- exception: None

8.4.5 Local Functions

None

9 MIS of Route Choice Set Generator Module

9.1 Module

~~RCSGenerator~~ RouteSolver

9.2 Uses

N/A

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
~~mapPointToNetwork~~	networkData: gdf of (streetName: sequence of string, geometry: sequence of ls) points: sequence of Point	pointOnNet: gdf of (streetGeometry: sequence of ls, streetName: sequence of string, geometry: sequence of Point)	-
~~detectGap~~	points: sequence of Point	gapPoints: sequence of sequence of Point	-
~~fillInGap~~	gapPoints: sequence of sequence of Point	filledGap: sequence of ls	-
~~findStreetIntersect~~	allExistStreets: sequence of ls	intersections: gdf of (street1Geo: sequence of ls, street2Geo: sequence of ls, geometry: sequence of Point)	-

Name	In	Out	Exceptions
~~connectPoints~~	pointOnNet: gdf of (streetGeometry: sequence of ls, streetName: sequence of string, geometry: sequence of Point]) intersections: gdf of (street1Geo: sequence of ls, street2Geo: sequence of ls, geometry: sequence of Point) filledGap: sequence of ls	route: ls	-
RouteChoiceGen	trip: gdf of (SerialID: sequence of $\mathbb{Z}$, RecordID: sequence of $\mathbb{Z}$, TimeStart: sequence of string, Mode: sequence of string, geometry: sequence of Point) ~~networkData: gdf of (streetName: sequence of string, geometry: sequence of ls)~~ networkGraph:MultiDiGraph networkEdges: gdf of (name: sequence of string oneway: sequence of $\mathbb{B}$ geometry: sequence of ls) networkNodes: gdf of (x: sequence of $\mathbb{R}$ y: sequence of $\mathbb{R}$ geometry: sequence of Point)	routeChoice: gdf of (SerialID: sequence of $\mathbb{Z}$, edgesRoutePassed: sequence of sequence of sequence [3] of $\mathbb{Z}$, geometry: sequence of ls)	-

9.4 Semantics

9.4.1 State Variables

None

9.4.2 Environment Variables

None

9.4.3 Assumptions

None

9.4.4 Access Routine Semantics

~~mapPointToNetwork(networkData, points)~~:

- ~~output: a GeoDataFrame object with the input points mapped onto the input network. It consists of streetGeometry (represented by a sequence of LineString), streetName (represented by a sequence of string), and geometry (represented by a sequence of Point).~~
- ~~exception: none~~

~~detectGap(points)~~:

- ~~output: a sequence of all the gap points among the given points, which is a sequence of sequences [2] of Point.~~
- ~~exception: none~~

~~fillInGap(gapPoints)~~:

- ~~output: a sequence of LineString objects that represent all the filled gaps given the gap points.~~
- ~~exception: none~~

~~findStreetIntersect(allExistStreets)~~:

- ~~output: a GeoDataFrame object consists of street1Geo (represented by a sequence of LineString), street2Geo (represented by a sequence of LineString), and geometry (represented by a sequence of Point), where the Point in geometry is an intersection point between the LineStrings in street1Geo and street2Geo.~~
- ~~exception: none~~

~~connectPoints(pointOnNet, intersections, filledGap)~~:

- ~~output: a LineString object which connects all of the Point objects in pointOnNet, any Point object in intersections that involves, and the LineString objects in filledGap.~~
- ~~exception: none~~

RouteChoiceGen(trip, ~~networkData~~ networkGraph, networkEdges, networkNodes):

- output: $out :=$
  a gdf routeChoice of (SerialID: sequence of $\mathbb{Z}$, edgesRoutePassed: sequence of sequence sequence [3] of $\mathbb{Z}$, geometry: sequence of ls) which:

– routeChoice.SerialID = unique_serials(trip)

– routeChoice.edgesRoutePassed = for every serial_id in unique_serials(trip), unique_edges_passed(map_point_to_network(serial_trip(trip,serial_id), networkGraph, networkEdges), detect_and_fill_gap(map_point_to_network(serial_trip(trip,serial_id), networkGraph, networkEdges), networkGraph, networkEdges, networkNodes))

– routeChoice.geometry = for every serial_id in unique_serials(trip), connect_points_and_filled_gaps(map_point_to_network(serial_trip(trip,serial_id), networkGraph, networkEdges), detect_and_fill_gap(map_point_to_network(serial_trip(trip,serial_id), networkGraph, networkEdges), networkGraph, networkEdges, networkNodes))

~~a GeoDataFrame object consists of SerialID (represented by a sequence of $\mathbb{Z}$), and geometry (represented by a sequence of LineString). The GeoDataFrame object will be generated by using mapPointToNetwork, detectGap, fillInGap, findStreetIntersect and connectPoints functions~~

- exception: none

9.4.5 Local Functions

All local functions' semantics below are added after revision 0 (i.e. Version 1.2)

unique_serials: *trip*: gdf of (SerialID: sequence of $\mathbb{Z}$, ...) $\rightarrow$ sequence of $\mathbb{Z}$
unique_serials $\equiv$ A sequence of $\mathbb{Z}$ where each element in the sequence is a unique value in *trip*.SerialID.

serial_trip: *trip*: gdf of (SerialID: sequence of $\mathbb{Z}$, ...) $\times$ *serial_id*: $\mathbb{Z} \rightarrow$ gdf of (SerialID: sequence of $\mathbb{Z}$, ...)
serial_trip $\equiv$ The rows in *trip* that are with SerialID value equals to *serial_id*

map_point_to_network:
trip: gdf of (SerialID: sequence of $\mathbb{Z}$, RecordID: sequence of $\mathbb{Z}$, TimeStart: sequence of string, Mode: sequence of string, geometry: sequence of Point) $\times$ *networkGraph*: MultiDigraph $\times$ *networkEdges*: gdf of (name: sequence of string, oneway: sequence of $\mathbb{B}$, geometry: sequence of *ls*)
$\rightarrow$
mapped_points: gdf of (SerialID: sequence of $\mathbb{Z}$, RecordID: sequence of $\mathbb{Z}$, nearEdgeID: sequence of sequence [3] of $\mathbb{Z}$, nearEdgeName: sequence of string, nearLeg: sequence of *ls* with outer sequence length equals to 2, geometry: sequence of Point)
map_point_to_network $\equiv$ output a gdf *mapped_points* which,

- *mapped_points*.SerialID = *trip*.SerialID
- *mapped_points*.RecordID = *trip*.RecordID

- $mapped\_points$.nearEdgeID = for each i^{th} element in $trip$.geometry (denoted as $trip$.geometry[i]) where $i \in [0, length\ of\ trip$.geometry), find_nearest_edge($trip$.geometry[i], $networkGraph$)

- $mapped\_points$.nearEdgeName = for each i^{th} element in $trip$.geometry (denoted as $trip$.geometry[i]) where $i \in [0, length\ of\ trip$.geometry), get_edge_name($networkEdge$, find_nearest_edge($trip$.geometry[i], $networkGraph$))

- $mapped\_points$.nearLeg = for each i^{th} element in $trip$.geometry (denoted as $trip$.geometry[i]) where $i \in [0, length\ of\ trip$.geometry), nearest_leg($networkEdge$, find_nearest_edge($trip$.geometry[$i$], $networkGraph$))

- $mapped\_points$.geometry = for each i^{th} element in $trip$.geometry (denoted as $trip$.geometry[i]) where $i \in [0, length\ of\ trip$.geometry), snap_point_to_edge(nearest_leg($networkEdge$, find_nearest_edge($trip$.geometry[i], $networkGraph$)), $trip$.geometry[i])

find_nearest_edge: $point\_to\_be\_mapped$: Point $\times$ $networkGraph$: MultiDiGraph $\rightarrow$ sequence [3] of $\mathbb{Z}$
find_nearest_edge $\equiv$ The index value of the edge that is nearest in terms of distance computed by using the OSMnx.distance.nearest_edges function with $point\_to\_be\_mapped$.x, $point\_to\_be\_mapped$.y and $networkGraph$ as inputs

nearest_leg: $networkEdge$: gdf of (name: sequence of string, oneway: sequence of $\mathbb{B}$, geometry: sequence of ls) $\times$ $nearest\_edge\_id$: sequence [3] of $\mathbb{Z}$ $\times$ $point\_to\_be\_mapped$: Point $\rightarrow$ $nearest\_leg\_geo$: ls
nearest_leg $\equiv$ Denote the element in $networkEdge$.geometry that is associated with index value equals to $nearest\_edge\_id$ as $nearest\_edge\_geo$: ls and the the i^{th} element in the outer sequence of $nearest\_edge\_geo$ as $nearest\_edge\_geo[i]$.
Denote the $\mathbb{Z}$,
$argmin_{i \in [0, length\ of\ nearest_edge_geo-1)}$geo_dist($point\_to\_be\_mapped, nearest\_edge\_geo[i : i{+}1]$) as min_dist_i
then $nearest\_leg\_geo\ =\ nearest\_edge[(min\_dist\_i, min\_dist\_i + 1)]$

geo_dist: geo_1: Point or ls or Polygon $\times$ geo_2: Point or ls or Polygon $\rightarrow \mathbb{R}$
geo_dist $\equiv$ The distance between the two Shapely geometries objects geo_1 and geo_2 that is calculated using Shapely distance function

snap_point_to_edge: leg_snap_to: ls $\times$ $point\_to\_snap$:Point $\rightarrow$ Point
snap_point_to_edge $\equiv$ Point after moving $point\_to\_snap$ to the nearest point to it on leg_snap_to by using Shapley interpolate function along with Shapley project function

get_edge_name: $networkEdge$: gdf of (name: sequence of string, oneway: sequence of $\mathbb{B}$, geometry: sequence of ls) $\times$ $edge\_id$: sequence [3] of $\mathbb{Z} \rightarrow$ $edge\_name$: string
get_edge_name $\equiv$ The value in $networkEdge$.name that is associated with the index value

equals to $edge\_id$.

detect_and_fill_gap:
$mapped\_points$: gdf of (SerialID: sequence of $\mathbb{Z}$, RecordID: sequence of $\mathbb{Z}$, nearEdgeID: sequence of sequence [3] of $\mathbb{Z}$, nearEdgeName: sequence of string, nearLeg: sequence of ls with outer sequence length equals to 2, geometry: sequence of Point) $\times$ $networkGraph$: MultiDigraph $\times$ $networkEdges$: gdf of (name: sequence of string, oneway: sequence of $\mathbb{B}$, geometry: sequence of ls) $\times$ $networkNodes$: gdf of (x: sequence of $\mathbb{R}$ y: sequence of $\mathbb{R}$, geometry: sequence of Point)
$\rightarrow$
gdf of (SerialID: sequence of $\mathbb{Z}$, OrigPointRecordID: sequence of $\mathbb{Z}$, EdgesGapPassed: sequence of sequence of sequence [3] of $\mathbb{Z}$, geometry: sequence of ls)
detect_and_fill_gap $\equiv$ fill_gaps(detect_gaps($mapped\_points$), $mapped\_points$, $networkGraph$, $networkEdges$, $networkNodes$)

detect_gaps:
$mapped\_points$ gdf of (SerialID: sequence of $\mathbb{Z}$, RecordID: sequence of $\mathbb{Z}$, nearEdgeID: sequence of sequence [3] of $\mathbb{Z}$, nearEdgeName: sequence of string, nearLeg: sequence of ls with outer sequence length equals to 2, geometry: sequence of Point) $\rightarrow$ $gaps\_start\_points$: sequence of $\mathbb{Z}$ detect_gaps $\equiv$ The values in $mapped\_points$.RecordID on the i^{th} rows of $mapped\_points$ where,
$((\neg(mapped\_points.\text{nearEdgeID}[i] = mapped\_points.\text{nearEdgeID}[i+1])$
$\wedge\ (\text{geo\_dist}(mapped\_points.\text{geometry}[i], mapped\_points.\text{geometry}[i+1])\ >\ 50))$
$\vee\ \neg(mapped\_points.\text{nearEdgeName}[i]\ =\ mapped\_points.\text{nearEdgeName}[i+1]))\ \equiv\ True$

fill_gaps: $gaps\_start\_points\_id$: sequence of $\mathbb{Z}$ $\times$ $mapped\_points$: gdf of (SerialID: sequence of $\mathbb{Z}$, RecordID: sequence of $\mathbb{Z}$, nearEdgeID: sequence of sequence [3] of $\mathbb{Z}$, nearEdgeName: sequence of string, nearLeg: sequence of ls with outer sequence length equals to 2, geometry: sequence of Point) $\times$ $networkGraph$: MultiDigraph $\times$ $networkEdges$: gdf of (name: sequence of string, oneway: sequence of $\mathbb{B}$, geometry: sequence of ls) $\times$ $networkNodes$: gdf of (x: sequence of $\mathbb{R}$ y: sequence of $\mathbb{R}$, geometry: sequence of Point)
$\rightarrow$
$filled\_gaps\_gdf$: gdf of (SerialID: sequence of $\mathbb{Z}$, OrigPointRecordID: sequence of $\mathbb{Z}$, EdgesGapPassed: sequence of sequence of sequence [3] of $\mathbb{Z}$, geometry: sequence of ls)
fill_gaps $\equiv$ output a gdf $filled\_gaps\_gdf$ which,

- $filled\_gaps\_gdf$.SerialID = a sequence that replicates the values in $mapped\_points$.SerialID that are on the rows with $mapped\_points$.RecordID values that exist in $gaps\_start\_points\_id$
- $filled\_gaps\_gdf$.OrigPointRecordID = $gaps\_start\_points\_id$
- $filled\_gaps\_gdf$.geometry = for every Point in $mapped\_points$.geometry that is associated with the same index value i as a value in $mapped\_points$.RecordID that exist in

$gaps\_start\_points\_id$,

  - Find the indices of the nodes on $networkGraph$ that are nearest to $mapped\_points$.geometry[i] and $mapped\_points$.geometry[i+1] respectively using OSMnx.distance.nearest_nodes function. Denote the two nodes as $start\_node$ and end_node respectively
  - Find the shortest path between $start\_node$ and end_node on $networkGraph$ using OSMnx.distance.shortest_path function which will output a sequence of the indices of the nodes that are on the shortest path in $networkNodes$. Denote these nodes' indices as $shortest\_path\_nodes\_id$.
  - Get the sequence of Point from the values in $networkNodes$.geometry where each of the element Point is associated with an index value that exists in $shortest\_path\_nodes\_id$. Denote the sequence of Point as $shortest\_path\_points$
  - Generate an ls with the x and y values of the elements of $shortest\_path\_points$ using Shapely LineString constructor function

- $filled\_gaps\_gdf$.EdgesGapPassed = sequence of sequence [3] of $\mathbb{Z}$ that are the unique values of the indices of ls in $networkEdges$.geometry that each of the $filled\_gaps\_gdf$.geometry values has been on.

connect_points_and_filled_gaps:
$mapped\_points$: gdf of (SerialID: sequence of $\mathbb{Z}$, RecordID: sequence of $\mathbb{Z}$, nearEdgeID: sequence of sequence [3] of $\mathbb{Z}$, nearEdgeName: sequence of string, nearLeg: sequence of ls with outer sequence length equals to 2, geometry: sequence of Point) $\times$ $filled\_gaps\_gdf$: gdf of (SerialID: sequence of $\mathbb{Z}$, OrigPointRecordID: sequence of $\mathbb{Z}$, EdgesGapPassed: sequence of sequence of sequence [3] of $\mathbb{Z}$, geometry: sequence of ls) $\rightarrow$ $route\_choice$: ls
connect_points_and_filled_gaps $\equiv$ an ls that is constructed with the x and y values of Point objects in $mapped\_points$.geometry and the values in the inner sequences of the ls objects in $filled\_gaps\_gdf$.geometry using Shapely LineString constructor function

unique_edges_passed:
$mapped\_points$: gdf of (SerialID: sequence of $\mathbb{Z}$, RecordID: sequence of $\mathbb{Z}$, nearEdgeID: sequence of sequence [3] of $\mathbb{Z}$, nearEdgeName: sequence of string, nearLeg: sequence of ls with outer sequence length equals to 2, geometry: sequence of Point) $\times$ $filled\_gaps\_gdf$: gdf of (SerialID: sequence of $\mathbb{Z}$, OrigPointRecordID: sequence of $\mathbb{Z}$, EdgesGapPassed: sequence of sequence of sequence [3] of $\mathbb{Z}$, geometry: sequence of ls) $\rightarrow$ $edges\_route\_choice\_passed$: sequence of sequence [3] of $\mathbb{Z}$
unique_edges_passed $\equiv$ A sequence of sequence [3] of $\mathbb{Z}$ where, each of the sequence [3] of $\mathbb{Z}$ is a unique value that exists in $mapped\_points$.nearEdgeID and $filled\_gaps\_gdf$.EdgesGapPassed.

10 MIS of Route Choice Analysis Variables Generator Module

10.1 Module

~~RCAVarGenerator~~ VariableGenerator

10.2 Uses

N/A

10.3 Syntax

10.3.1 Exported Constants

None

10.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
~~routelength~~	~~route: sequence of sequence[2] of $\mathbb{R}$~~	~~length: $\mathbb{R}$~~	~~-~~
countTurns	networkEdges: gdf of (name: sequence of string, oneway: sequence of $\mathbb{B}$, geometry: sequence of ls), edgesRoutePassed: sequence of sequence of sequence [3] of $\mathbb{Z}$, route: sequence of sequence [2] of $\mathbb{R}$	numOfTurnsByType: tuple of (left: $\mathbb{Z}$, right: $\mathbb{Z}$, total: $\mathbb{Z}$)	-

Name	In	Out	Exceptions
~~mapLegToStreet~~ findNearest-Street	networkEdges: gdf of (name: sequence of string, oneway: sequence of $\mathbb{B}$, geometry: sequence of ls), edgesRoutePassed: sequence of sequence of sequence [3] of $\mathbb{Z}$, coord: sequence [2] of $\mathbb{R}$ ~~networkData: gdf of (Name: sequence of *string*, geometry: sequence of ls)~~	~~streetOnRoute: gdf of (streetName: sequence of *string*, geometry: sequence of ls), numOfRoad: $\mathbb{Z}$~~ nearestStreet: string	-
longestLeg	~~streetOnRoute: gdf of (streetName: sequence of *string*, geometry: sequence of ls)~~ networkEdges: gdf of (name: sequence of string, oneway: sequence of $\mathbb{B}$, geometry: sequence of ls), edgesRoutePassed: sequence of sequence of sequence [3] of $\mathbb{Z}$, route: sequence of sequence [2] of $\mathbb{R}$	longestLegInfo: tuple of (legStreet: string, legLength: $\mathbb{Z}$, numOfStreets: $\mathbb{Z}$)	-

Name	In	Out	Exceptions
RCAVarGen	route: (SerialID: sequence of $\mathbb{Z}$, edgesRoutePassed: sequence of sequence of sequence [3] of $\mathbb{Z}$, geometry: sequence of ls) networkEdges: gdf of (name: sequence of string, oneway: sequence of $\mathbb{B}$, geometry: sequence of ls) ~~networkData: gdf of (streetName: sequence of string, geometry: sequence of ls)~~	RCA: (SerialID: sequence of $\mathbb{Z}$, distanceMeter: sequence of $\mathbb{Z}$, numOfLturns: sequence of $\mathbb{Z}$, numOfRturns: sequence of $\mathbb{Z}$, numOfRoads: sequence of $\mathbb{Z}$, streetLongestLeg: sequence of string, lengthLongestLeg: sequence of $\mathbb{Z}$, geometry: sequence of ls)	-

10.4 Semantics

10.4.1 State Variables

None

10.4.2 Environment Variables

None

10.4.3 Assumptions

None

10.4.4 Access Routine Semantics

~~routelength(route):~~

- ~~output: An $\mathbb{Z}$ that represents the length of the input route in meters.~~
- ~~exception: none~~

countTurns(networkEdges, edgesRoutePassed, route):

- output: a tuple of (left: $\mathbb{Z}$, right: $\mathbb{Z}$, total: $\mathbb{Z}$) that contains information about the number of left turns, right turns and the total number of turns in the input route.

- exception: none

~~mapLegToStreet(networkData, route):~~

- ~~output: streetOnRoute := a GeoDataFrame object consists of street name (represented by a sequence of string) and geometry (represented by a sequence of LineStrings. numOfRoad := An $\mathbb{Z}$ that counts the number of unique streets the input route has been on.~~
- ~~exception: none~~

findNearestStreet(networkEdges, edgesRoutePassed, coord):

- output: the name of the nearest street to the input coordinates as a string.
- exception: none

longestLeg(~~streetOnRoute~~ networkEdges, edgesRoutePassed, route):

- output: a tuple of (legStreet: string, legLength: $\mathbb{Z}$, numOfStreets: $\mathbb{Z}$) that contains the street name that the longest leg belongs to, the length of the longest leg in meters, and the number of streets in the route.
- exception: none

RCAVarGen(~~networkData~~ networkEdges, route):

- output: a GeoDataFrame object consists of SerialID (represented by a sequence of $\mathbb{Z}$), the total distance travelled in the route (represented by a sequence of $\mathbb{Z}$), the number of roads (represented by a sequence of $\mathbb{Z}$), the number of left and right turns (each represented by a sequence of $\mathbb{Z}$), the street name that the longest leg belongs to (represented by a sequence of string), the length of the longest leg (represented by $\mathbb{Z}$), and the geometry (represented by a sequence of ls). This GeoDataFrame object will be generated by using countTurns, findNearestStreet and longestLeg functions.

10.4.5 Local Functions

None

11 MIS of Activity Locations Identification Module

11.1 Module

ALIM

11.2 Uses

N/A

11.3 Syntax

11.3.1 Exported Constants

None

11.3.2 Exported Access Programs

Name	In	Out	Exceptions

create_al_info	ALlu_gdf: gdf of (SerialID: sequence of $\mathbb{Z}$, RecordID: sequence of $\mathbb{Z}$, TimeStart: sequence of s, Mode: sequence of string, geometry: sequence of Point, house_number: sequence of $\mathbb{Z}$, street_name: sequence of string, name_of_building: sequence of string, building_geometry: sequence of Polygon, lu_match: sequence of string, lu_code: sequence of $\mathbb{Z}$, lu_classification: sequence of string) ALpal_gdf: gdf of (SerialID: sequence of $\mathbb{Z}$, RecordID: sequence of $\mathbb{Z}$, TimeStart: sequence of s, Mode: sequence of string, geometry: sequence of Point, house_number: sequence of $\mathbb{Z}$, street_name: sequence of string, name_of_building: sequence of string, building_geometry: sequence of Polygon, pal_match: sequence of string, pal_id: sequence of $\mathbb{Z}$, pal_classification: sequence of string)	ALinfo_gdf: gdf of (SerialID: sequence of $\mathbb{Z}$, RecordID: sequence of $\mathbb{Z}$, TimeStart: sequence of s, Mode: sequence of string, geometry: sequence of Point, house_number: sequence of $\mathbb{Z}$, street_name: sequence of string, name_of_building: sequence of string, building_geometry: sequence of Polygon, lu_match: sequence of string, lu_code: sequence of $\mathbb{Z}$, lu_classification: sequence of string, pal_match: sequence of string, pal_id: sequence of $\mathbb{Z}$, pal_classification: sequence of string)	

11.4 Semantics

11.4.1 State Variables

None

11.4.2 Environment Variables

None

11.4.3 Assumptions

None

11.4.4 Access Routine Semantics

create_al_info(al_lu_gdf, al_pal_gdf):

- output: A GeoDataFrame of Activity Location Information with appending information to the object based on the inputted identified Activity Location GeoDataFrame with LU additional information and identified Activity Location GeoDataFrame with PAL additional information
- exception: None

11.4.5 Local Functions

identify_lu:
AL_gdf: gdf of (SerialID: sequence of $\mathbb{Z}$, RecordID: sequence of $\mathbb{Z}$, TimeStart: sequence of s, Mode: sequence of string, geometry: sequence of Point),

LU_gdf: gdf of (house_number: sequence of $\mathbb{Z}$, street_name: sequence of string, name_of_building: sequence of string, building_geometry: sequence of Polygon, lu_code: sequence of $\mathbb{Z}$, lu_classification: sequence of string)
$\rightarrow$
ALlu_gdf: gdf of (SerialID: sequence of $\mathbb{Z}$, RecordID: sequence of $\mathbb{Z}$, TimeStart: sequence of s, Mode: sequence of string, geometry: sequence of Point, house_number: sequence of $\mathbb{Z}$, street_name: sequence of string, name_of_building: sequence of string, building_geometry: sequence of Polygon, lu_match: sequence of string, lu_code: sequence of $\mathbb{Z}$, lu_classification: sequence of string)

A GeoDataFrame of Activity Locations with appending information to the object based on the inputted LU GeoDataFrame that corresponds with an existing SerialID of Activity Locations if it exists. This includes the lu_match which states if the SerialID matches the LU, lu_code and lu_classification

identify_pal:
AL_gdf: gdf of (SerialID: sequence of $\mathbb{Z}$, RecordID: sequence of $\mathbb{Z}$, TimeStart: sequence of s, Mode: sequence of string, geometry: sequence of Point),

PAL_gdf: gdf of (house_number: sequence of $\mathbb{Z}$, street_name: sequence of string, name_of_building:

sequence of string, building_geometry: sequence of Polygon, pal_id: sequence of $\mathbb{Z}$, pal_classification: sequence of string)
$\rightarrow$
ALpal_gdf: gdf of (SerialID: sequence of $\mathbb{Z}$, RecordID: sequence of $\mathbb{Z}$, TimeStart: sequence of s, Mode: sequence of string, geometry: sequence of Point, house_number: sequence of $\mathbb{Z}$, street_name: sequence of string, name_of_building: sequence of string, building_geometry: sequence of Polygon, pal_match: sequence of string, pal_id: sequence of $\mathbb{Z}$, pal_classification: sequence of string)

A GeoDataFrame of Activity Locations with appending information to the object based on the inputted PAL GeoDataFrame that corresponds with an existing SerialID of Activity Locations if it exists. This includes pal_match which states if the SerialID matches the PAL, pal_id and pal_classification

12 MIS of Network Data Utilities Module

12.1 Module

NetworkDataUtils

12.2 Uses

N/A

12.3 Syntax

12.3.1 Exported Constants

NetworkModes = ['drive','walk','all']

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
get_points_boundary	points_gdf: gdf of (any field(s), geometry: sequence of Point)	sequence of [4] $\mathbb{R}$	-
extract_networkdata_pbf	pbf_file_path: string, mode: string, bbox: sequence of [4] $\mathbb{R}$	graph_proj: MultiDigraph, edges_proj: gdf of (name: sequence of string, oneway: sequence of $\mathbb{B}$, geometry: sequence of ls), nodes_proj: gdf of (x: sequence of $\mathbb{R}$, y: sequence of $\mathbb{R}$, geometry: sequence of Point), pbf_boundary: sequence of [4] $\mathbb{R}$	NetworkModeError

Name	In	Out	Exceptions
extract_networkdata_bbox	max_lat: $\mathbb{R}$, min_lat: $\mathbb{R}$, max_lon: $\mathbb{R}$, min_lon: $\mathbb{R}$	graph_proj: MultiDigraph, edges_proj: gdf of (name: sequence of string, oneway: sequence of $\mathbb{B}$, geometry: sequence of ls), nodes_proj: gdf of (x: sequence of $\mathbb{R}$, y: sequence of $\mathbb{R}$, geometry: sequence of Point)	NetworkModeError
extract_ludata_pbf	pbf_file_path: string, bbox: sequence of [4] $\mathbb{R}$	landuse_gdf: gdf of (landuse: sequence of string, geometry: sequence of Polygon)	-
extract_paldata_pbf	string, bbox: sequence of [4] $\mathbb{R}$	pal_info: gdf of (addr:housenumber: sequence of $\mathbb{Z}$, addr:street: sequence of string, building: sequence of string, amenity: sequence of string, addr:city : sequence of string, name: sequence of string, geometry: sequence of Polygon or Point)	-
extract_ludata_bbox	max_lat: $\mathbb{R}$, min_lat: $\mathbb{R}$, max_lon: $\mathbb{R}$, min_lon: $\mathbb{R}$	landuse_gdf: gdf of (landuse: sequence of string, geometry: sequence of Polygon)	-

Name	In	Out	Exceptions
extract_paldata_bbox	max_lat: $\mathbb{R}$, min_lat: $\mathbb{R}$, max_lon: $\mathbb{R}$, min_lon: $\mathbb{R}$	pal_info: gdf of (addr:housenumber: sequence of $\mathbb{Z}$, addr:street: sequence of string, building: sequence of string, amenity: sequence of string, addr:city : sequence of string, name: sequence of string, geometry: sequence of Polygon or Point)	-
get_trip_mode	points_gdf: gdf of (any field(s), modes: sequence of string, geometry: sequence of Point)	trip_mode: string	-

12.4 Semantics

12.4.1 State Variables

None

12.4.2 Environment Variables

None

12.4.3 Assumptions

None

12.4.4 Access Routine Semantics

get_points_boundary(points_gdf):

- output: *out* := *sequence* (max_y(points_gdf.geometry)+0.005, min_y(points_gdf.geometry)-0.005, max_x(points_gdf.geometry)+0.005, min_x(points_gdf.geometry)-0.005)
- exception: None

extract_networkdata_pbf(pbf_file_path, mode, bbox):

- output: out := preprocess_networkdata_pbf(networkdata_from_pbf(pbf_file_path, mode, bbox))
- exception: $exc := \neg(\exists m \in \text{NetworkModes} | (m = \text{mode})) \Rightarrow \text{NetworkModeError}$

extract_networkdata_bbox(max_lat, min_lat, max_lon, min_lon, mode):

- output: out := preprocess_networkdata_osm(networkdata_from_osm(max_lat, min_lat, max_lon, min_lon, mode))
- exception: $exc := \neg(\exists m \in \text{NetworkModes} | (m = \text{mode})) \Rightarrow \text{NetworkModeError}$

extract_ludata_pbf(pbf_file_path, bbox):

- output: out := preprocess_ludata_pbf(ludata_from_pbf(pbf_file_path, bbox))
- exception: None

extract_paldata_pbf(pbf_file_path, bbox):

- output: out := preprocess_paldata_pbf(paldata_from_pbf(pbf_file_path, bbox))
- exception: None

extract_ludata_bbox(max_lat, min_lat, max_lon, min_lon):

- output: out := extract land-use data from OpenStreetMap API using OSMnx.graph.graph_from_bbox function with max_lat, min_lat, max_lon and min_lon as inputs.
- exception: None

extract_paldata_bbox(max_lat, min_lat, max_lon, min_lon):

- output: out := extract buildings and amenities data from OpenStreetMap API using OSMnx.geometries.geometries_from_bbox function with max_lat, min_lat, max_lon and min_lon as inputs.
- exception: None

get_trip_mode(trip_data):

- output:
  $out :=$
  $('Drive' \notin \text{unique\_modes(trip\_data)}) \wedge ('Walk' \in \text{unique\_modes(trip\_data)}) \Rightarrow 'walk'$
  $('Drive' \in \text{unique\_modes(trip\_data)}) \wedge ('Walk' \notin \text{unique\_modes(trip\_data)}) \Rightarrow 'drive'$
  $('Drive' \in \text{unique\_modes(trip\_data)}) \wedge ('Walk' \in \text{unique\_modes(trip\_data)}) \Rightarrow 'all'$
  $('Drive' \notin \text{unique\_modes(trip\_data)}) \wedge ('Walk' \notin \text{unique\_modes(trip\_data)}) \Rightarrow 'no\ mode\ found'$
- exception: None

12.5 Local Functions

max_y: sequence of Point $\rightarrow \mathbb{Z}$
max_y $\equiv$ The maximum y value in a sequence of Point.

min_y: sequence of Point $\rightarrow \mathbb{Z}$
min_y $\equiv$ The minimum y value in a sequence of Point.

max_x: sequence of Point $\rightarrow \mathbb{Z}$
max_x $\equiv$ The maximum x value in a sequence of Point.

min_y: sequence of Point $\rightarrow \mathbb{Z}$
min_y $\equiv$ The minimum y value in a sequence of Point.

networkdata_from_pbf:
pbf_file_path: string $\times$ mode: string $\times$ bbox: sequence of [4] $\mathbb{R} \rightarrow$ network_edges: gdf of (u: sequence of $\mathbb{Z}$, v: sequence of $\mathbb{Z}$, key: sequence of $\mathbb{Z}$, name: sequence of string, oneway: sequence of $\mathbb{B}$, geometry: sequence of ls) $\times$ network_nodes: gdf of (osmid: sequence of $\mathbb{Z}$, x: sequence of $\mathbb{R}$, y: sequence of $\mathbb{R}$, geometry: sequence of Point)
networkdata_from_pbf $\equiv$ Two gdf that are generated by reading the transportation network data from an OSM PBF file located at pbf_file_path using functions in Pyrosm library with pbf_file_path, mode and bbox as inputs

preprocess_networkdata_pbf:
network_edges: gdf of (u: sequence of $\mathbb{Z}$, v: sequence of $\mathbb{Z}$, key: sequence of $\mathbb{Z}$, name: sequence of string, oneway: sequence of $\mathbb{B}$, geometry: sequence of ls) $\times$ network_nodes: gdf of (osmid: sequence of $\mathbb{Z}$, x: sequence of $\mathbb{R}$, y: sequence of $\mathbb{R}$, geometry: sequence of Point)
$\rightarrow$
edges_proj: gdf of (name: sequence of string, oneway: sequence of $\mathbb{B}$, geometry: sequence of ls) $\times$ nodes_proj: gdf of (x: sequence of $\mathbb{R}$, y: sequence of $\mathbb{R}$, geometry: sequence of Point), graph_proj: MultiDiGraph
preprocess_networkdata_pbf $\equiv$

- Set network_edges.u, network_edges.v and network_edges.key as multi-level index of network_edges and network_nodes.osmid as index of network_nodes.
- Generate a MultiDigraph graph_proj using OSMnx.utils_graph.graph_from_gdfs function with network_edges and network_nodes as inputs, project the generated MultiDigraph using OSMnx.projection.project_graph function.
- Generate two gdf nodes_proj and edges_proj using OSMnx.utils_graph.graph_to_gdfs function with graph_proj as input.

networkdata_from_osm: max_lat: $\mathbb{R}$ $\times$ min_lat: $\mathbb{R}$ $\times$ max_lon: $\mathbb{R}$ $\times$ min_lon: $\mathbb{R}$ $\times$ mode: string $\rightarrow$ network_graph: MultiDigraph networkdata_from_osm $\equiv$ A MultiDiGraph generated using OSMnx.graph.graph_from_bbox function with max_lat, min_lat, max_lon, min_lon and mode as inputs

preprocess_networkdata_osm: network_graph: MultiDigraph $\rightarrow$ edges_proj: gdf of (name: sequence of string, oneway: sequence of $\mathbb{B}$, geometry: sequence of ls) $\times$ nodes_proj: gdf of (x: sequence of $\mathbb{R}$, y: sequence of $\mathbb{R}$, geometry: sequence of Point), graph_proj: MultiDiGraph
preprocess_networkdata_osm $\equiv$

- Projecting network_graph using OSMnx.projection.project_graph function to get graph_proj.
- Generate two gdf nodes_proj and edges_proj using OSMnx.utils_graph.graph_to_gdfs function with graph_proj as input.

ludata_from_pbf: pbf_file_path: string $\times$ bbox: sequence of [4] $\mathbb{R}$ $\rightarrow$ lu_gdf: gdf of (element_type: sequence of $\mathbb{Z}$, osmid: sequence of $\mathbb{Z}$, landuse: sequence of string, geometry: sequence of Polygon)
ludata_from_pbf $\equiv$ A gdf that is generated by reading the land-use data from an OSM PBF file located at pbf_file_path using functions in Pyrosm library using pbf_file_path and bbox as inputs

preprocess_ludata_pbf:
lu_gdf: gdf of (element_type: sequence of $\mathbb{Z}$, osmid: sequence of $\mathbb{Z}$, landuse: sequence of string, geometry: sequence of Polygon)
$\rightarrow$
lu_gdf: gdf of (landuse: sequence of string, geometry: sequence of Polygon)
preprocess_ludata_pbf $\equiv$ Setting lu_gdf.element_type and lu_gdf.osmid as multi-level index of lu_gdf.

paldata_from_pbf: pbf_file_path: string $\times$ bbox: sequence of [4] $\mathbb{R}$ $\rightarrow$
buildings_gdf: gdf of (element_type: sequence of $\mathbb{Z}$, osmid: sequence of $\mathbb{Z}$, addr:housenumber: sequence of $\mathbb{Z}$, addr:street: sequence of string, building: sequence of string, addr:city : sequence of string, name: sequence of string, geometry: sequence of Polygon) $\times$
amenities_gdf: gdf of (element_type: sequence of $\mathbb{Z}$, osmid: sequence of $\mathbb{Z}$, addr:housenumber: sequence of $\mathbb{Z}$, addr:street: sequence of string, amenity: sequence of string, addr:city : sequence of string, name: sequence of string, geometry: sequence of Polygon or Point)
paldata_from_pbf $\equiv$

- A gdf buildings_gdf that is generated by reading the buildings' data from an OSM PBF file located at pbf_file_path using functions in Pyrosm library using pbf_file_path and bbox as inputs
- A gdf amenities_gdf that is generated by reading the amenities' data from an OSM PBF file located at pbf_file_path using functions in Pyrosm library using pbf_file_path and bbox as inputs

preprocess_paldata_pbf:
buildings_gdf: gdf of (element_type: sequence of $\mathbb{Z}$, osmid: sequence of $\mathbb{Z}$, addr:housenumber: sequence of $\mathbb{Z}$, addr:street: sequence of string, building: sequence of string, addr:city : sequence of string, name: sequence of string, geometry: sequence of Polygon) $\times$
amenities_gdf: gdf of (element_type: sequence of $\mathbb{Z}$, osmid: sequence of $\mathbb{Z}$, addr:housenumber: sequence of $\mathbb{Z}$, addr:street: sequence of string, amenity: sequence of string, addr:city : sequence of string, name: sequence of string, geometry: sequence of Polygon or Point)
$\rightarrow$
pal_info: gdf of (addr:housenumber: sequence of $\mathbb{Z}$, addr:street: sequence of string, building: sequence of string, amenity: sequence of string, addr:city : sequence of string, name: sequence of string, geometry: sequence of Polygon or Point)
paldata_from_pbf $\equiv$

- Set buildings_gdf.element_type and buildings_gdf.osmid as multi-level index of buidings_gdf and amenities_gdf.element_type and amenities_gdf.osmid as multi-level index of amenities_gdf.
- Generate a gdf pal_info by concatenating buildings_gdf and amenities_gdf using Pandas.concat function.
- Remove rows in pal_info with duplicated indeices (keep the first row for each of the unique duplicated indices)
- Sort rows of pal_info by their index values using pandas.DataFrame.sort_index function

unique_modes: trip_data: gdf of (modes: sequence of string, geometry: sequence of Point) $\rightarrow$ sequence of string
unique_modes $\equiv$ A sequence of string where each element in the sequence is a unique value in trip_data.modes.

13 MIS of Main Function Module

13.1 Module

Main

13.2 Uses

GPSPreprocess (Section 6), ModeDetection (Section 7), Extractor (Section 8), ~~RCSGenerator~~RouteSolver (Section 9), ~~RCAVarGenerator~~VariableGenerator (Section 10), ALIM (Section 11) NetworkDataUtils (Section 12)

13.3 Syntax

13.3.1 Exported Constants

EPSGNumEarth: $\mathbb{N}$

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
main	-	-	InvalidFilePathException, InvalidFileFormatException, ~~InvalidEPSGNum~~ InvalidDataException, InvalidGPSDataException, NetworkDataExtractionError OutofBoundException, InvalidInputException

13.4 Semantics

13.4.1 State Variables

programProgress: $\mathbb{R}$

13.4.2 Environment Variables

consoleWin: 2D sequence of pixels displayed on the screen
gpsDataFile: A text file in CSV format
networkDatasetFile: A file/folder that contains the data for the transportation network dataset
outputFolder: A virtual location in computer for files or other folders/directories

13.4.3 Assumptions

None

13.4.4 Access Routine Semantics

13.4.5 Local Functions

main():

- transition: ~~Modify consoleWin to display text prompts asking user to input the file paths for the gpsDataFile, networkDatasetFile and outputFolder, asking user to input the EPSG CRS number (ℕ), informing the user which stage of map matching the running program is currently on, showing error/warning to user if exception appears, display the current progress based on the programProgress state variable, and displaying URLs that user can copy to visualize the map-matching result on browser.~~
~~Modify outputFolder to generate files that the program will output under the file path of outputFolder.~~
Modify and process the state of environment variables by the following steps:
  - Modify consoleWin to display text prompts asking user to input the file paths for the gpsDataFile, networkDatasetFile and outputFolder, asking user to input the radius around the activity locations that the activity locations' information will be gathered from and the number of GPS points in the trip segment that the user wants PyERT to match and process.
  - Modify consoleWin to display text prompts to inform the user that PyERT is now preprocessing the GPS data. Read and preprocess the raw GPS data from gpsDataFile using exported access programs of GPS Data Preprocessing Module(Section 6).
  - Modify consoleWin to display text prompts to inform the user PyERT is now detecting modes of the GPS data. Detect Modes of preprocessed GPS data using exported access programs of GPS Data Mode Detection Module(Section 7).
  - Modify consoleWin to display text prompts to inform the user PyERT is now extracting trip and stop segments from the GPS data. Extract trip and stop segments from mode-detected GPS data using exported access programs of Trip Segments and Activity Locations Extraction Module(Section 8).
  - Modify consoleWin to display text prompts to inform the user PyERT is now extracting transportation network, landuse, amenities and buildings data. Extract transportation network, landuse, amenities and buildings by first getting the longitudes and latitudes boundaries of trip and stop segments and then extract data from networkDatasetFile or from OSM API based on the boundaries using exported access programs of Network Data Untilities Module(Section 12).
  - Modify consoleWin to display text prompts to inform the user PyERT is now generating route choices for trip segments. Generate route choices for the extracted trip segments using exported access program RouteChoiceGen of Route Choice Set Generator Module(Section 9) by matching the trip segments onto the transportation network data set extracted.

- Modify consoleWin to display text prompts to inform the user PyERT is now generating values of route choice analysis variables for the generated route choices. Generate values of route choice analysis variables for the route choices using exported access program RCAVarGen of Route Choice Analysis Variables Genrator Module(Section 10).
- Generate URL for the generated route choices using functions from geojsonio library and open it in web browser for the user to to visualize the map-matching result
- Generate output files that contains the geographic information of route choices and values of route choice analysis variables into outputFolder as shapefile and CSV file type respectively into outputFolder.
- Modify consoleWin to display text prompts to inform the user PyERT is now identifying activity locations inforamation for stop segments. Identify activity locations inforamation for stop segments using exported access program createActLocInfo of Activity Locations Identification Module(Section 11).
- Generate URL for the identified activity locations around the stop segments using functions from geojsonio library and open it in web browser for the user to to visualize the activity locations.
- Generate output files that contains the information of identified activity locations as shapefile file type into outputFolder.

• output: None

• exception: exc := a file path gpsDataFile OR networkDatasetFile cannot be found ⇒ ~~FileNotFoundError~~ InvalidFilePathException
exc := the format of gpsDataFile OR networkDatasetFile is incorrect ⇒ ~~FileTypeError~~ InvalidFileFormatException
~~exc := the EPSG CRS number from user input cannot be used ⇒ InvalidEPSGNum~~
exc := the raw GPS data does not have necessary attributes ['RecordID', 'SerialID', 'LocalTime', 'latitude', 'longitude', 'Fix_Status', 'DOP', 'Speed_kmh', 'Limit_kmh'] ⇒ InvalidDataException
exc := the preprocessed GPS data does not have necessary 'geometry' attribute ⇒ InvalidGPSDataException
exc := None is output from extracting transporation network data from networkDatasetFile⇒ NetworkDataExtractionError
exc := The longitudes and latitudes boundaries of trip segments are not within the boundaries of the extracted transportation network data⇒ OutofBoundException
exc := The input path to outputFolder is not to a folder ⇒ OutofBoundException

13.4.6 Local Functions

None

References

Ron Dalumpines and Darren M. Scott. Gis-based episode reconstruction toolkit (gert): A transferable, modular, and scalable framework for automated extraction of activity episodes from gps data. Travel Behaviour and Society, 11:121–130, 2018. ISSN 2214-367X. doi: https://doi.org/10.1016/j.tbs.2017.04.001. URL https://www.sciencedirect.com/science/article/pii/S2214367X17300406.

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. Fundamentals of Software Engineering. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. Software Design, Automated Testing, and Maintenance: A Practical Approach. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

14 Appendix

N/A