AUP Documentation

Release 0.1.0

Luis G. Natera

Jul 06, 2020

Chapter 1

AUP package

Users' reference for the AUP api.

Chapter 2

aup.data module

`aup.data.`**`create_polygon`** (*bbox*, *city*, *save=True*)

Create a polygon from a bounding box and save it to a file

Parameters **`{list} -- list containing the coordinates of the bounding box [north, south, east, west]`** (*bbox*) –

Keyword Arguments

`{bool} -- boolean to save or not the polygon to a file as a GeoJSON (default` (*save*) – {True})

Returns polygon – GoeDataFrame with the geometry of the polygon to be used to download the data

`aup.data.`**`df_to_geodf`** (*df*, *x*, *y*, *crs*)

Create a geo data frame from a pandas data frame

Parameters

- **`{pandas.DataFrame} -- pandas data frame with lat, lon or x, y, columns`** (*df*) –
- **`{str} -- Name of the column that contains the x or Longitud values`** (*x*) –
- **`{str} -- Name of the column that contains the y or Latitud values`** (*y*) –
- **`{dict} -- Coordinate reference system to use`** (*crs*) –

Returns geopandas.GeoDataFrame – GeoDataFrame with Points as geometry

`aup.data.`**`download_graph`** (*polygon*, *city*, *network_type='walk'*, *save=True*)

Download a graph from a bounding box, and saves it to disk

Parameters

- **`{polygon} -- polygon to use as boundary to download the network`** (*polygon*) –
- **`{str} -- string with the name of the city`** (*city*) –

Keyword Arguments

- **`{str} -- String with the type of network to download`** (*network_type*) –
- **`{bool} -- Save the graph to disk or not (default`** (*save*) – {True})

Returns nx.MultiDiGraph

`aup.data.`**`load_denue`** (*amenity_name*)

Load the DENUE into a geoDataFrame

Parameters **`{str} -- string with the name of the amenity to load the availables are`** (*`amenity_name`*) – ('farmacias','supermercados','hospitales')

Returns geopandas.geoDataFrame – geoDataFrame with the DENUE

`aup.data.`**`load_mpos`** ( )

Load Mexico's municipal boundaries

Returns geopandas.geoDataFrame – geoDataFrame with all the Mexican municipal boundaries

`aup.data.`**`load_polygon`** (*city*)

Load the polygon of a city from the raw data

Parameters **`{str} -- string with the name of the city/metropolitan area to load`** (*`city`*) –

Returns geopandas.GeoDataFrame – geoDataFrame with the area

`aup.data.`**`load_study_areas`** ( )

Load the study areas json as dict

Returns dict – dictionary with the study areas and attributes

aup.utils module

aup.utils.**create_hexgrid** (*polygon, hex_res, geometry_col='geometry', buffer=0.0*)

Takes in a geopandas geodataframe, the desired resolution, the specified geometry column and some map parameters to create a hexagon grid (and potentially plot the hexgrid

Parameters
- **{geopandas.geoDataFrame} -- geoDataFrame to be used** (*polygon*) –
- **{int} -- Resolution to use** (*hex_res*) –

Keyword Arguments
- **{str} -- column in the geoDataFrame that contains the geometry (default** (*geometry_col*) – {'geometry'})
- **{float} -- buffer to be used (default** (*buffer*) – {0.000})

Returns geopandas.geoDataFrame – geoDataFrame with the hexbins and the hex_id_{resolution} column

aup.utils.**find_nearest** (*G, gdf, amenity_name*)

Find the nearest graph nodes to the points in a GeoDataFrame

Parameters
- **{networkx.Graph} -- Graph created with OSMnx that contains geographic information** (*G*) –
- **{geopandas.GeoDataFrame} -- GeoDataFrame with the points to locate** (*gdf*) –
- **{str} -- string with the name of the amenity that is used as seed** (*amenity_name*) –

Returns geopandas.GeoDataFrame – GeoDataFrame original dataframe with a new column call 'nearest' with the node id closser to the point

aup.utils.**get_seeds** (*gdf, node_mapping, amenity_name*)

Generate the seed to be used to calculate shortest paths for the Voronoi's

Parameters
- **{geopandas.GeoDataFrame} -- GeoDataFrame with 'nearest' column** (*gdf*) –
- **{dict} -- dictionary containing the node mapping from networkx.Graph to igraph.Graph** (*node_mapping*) –

Returns np.array – numpy.array with the set of seeds `aup.utils.`**`haversine`**(*coord1, coord2*)

Calculate distance between two coordinates in meters with the Haversine formula

Parameters
- **`{tuple} -- tuple with coordinates in decimal degrees`** (*`coord2`*) –
- **`{tuple} -- tuple with coordinates in decimal degrees`** –

Returns float – distance between coord1 and coord2 in meters

`aup.utils.`**`to_igraph`**(*G*)

Convert a graph from networkx to igraph

Parameters **`{networkx.Graph} -- networkx Graph to be converted`** (*`G`*) –

Returns igraph.Graph – Graph with the same number of nodes and edges as the original one np.array – With the weight of the graph, if the original graph G is from OSMnx the weights are lengths dict – With the node mapping, index is the node in networkx.Graph, value is the node in igraph.Graph

aup.analysis module

aup.analysis.**calculate_distance_nearest_poi** (*gdf_f*, *G*, *amenity_name*, *city*)

Calculate the distance to the shortest path to the nearest POI (in gdf_f) for all the nodes in the network G

Parameters

- **{geopandas.geoDataFrame} -- geoDataFrame with the Points of Interest the geometry type has to be shapely.Point** (*gdf_f*) –
- **{networkx.MultiDiGraph} -- Graph created with OSMnx** (*G*) –
- **{str} -- string with the name of the amenity that is used as seed** (*amenity_name*) –
- **{str} -- string with the name of the city** (*city*) –

Returns geopandas.GeoDataFrame – geoDataFrame with geometry and distance to the nearest POI

aup.analysis.**get_distances** (*g*, *seeds*, *weights*, *voronoi_assignment*)

Distance for the shortest path for each node to the closest seed

Parameters

- **{[type]} -- [description]** (*voronoi_assignment*) –
- **{[type]} -- [description]** –
- **{[type]} -- [description]** –
- **{[type]} -- [description]** –

Returns [type] – [description]

aup.analysis.**group_by_hex_mean** (*nodes*, *hex_bins*, *resolution*, *amenity_name*)

Group by hexbin the nodes and calculate the mean distance from the hexbin to the closest pharmacy

Parameters

- **{geopandas.geoDataFrame} -- geoDataFrame with the nodes to group** (*nodes*) –
- **{geopandas.geoDataFrame} -- geoDataFrame with the hexbins** (*hex_bins*) –
- **{int} -- resolution of the hexbins, used when doing the group by and to save the column** (*resolution*) –
- **{str} -- string with the name of the amenity that is used as seed** (*amenity_name*) –

Returns geopandas.geoDataFrame – geoDataFrame with the hex_id{resolution}, geometry and average distance to pharmacy for each hexbin

`aup.analysis.`**`voronoi_cpu`** (*g*, *weights*, *seeds*)

Voronoi diagram calculator for undirected graphs

Optimized for computational efficiency

Args:

g (igraph.Graph): graph object with Nodes and Edges weights (numpy.array): array of weights for all edges of length len(V) seeds (numpy.array): generator points as numpy array of indices from the node array

Returns:

[numpy.array]: numpy.array on len(N) where the location (index) of the node refers to the node, the value is the generator (seed) the respective nodes belongs to.

aup.visualization module

`aup.visualization.`**`hex_plot`** (*ax*, *gdf_data*, *gdf_boundary*, *gdf_edges*, *column*, *title*, *save_png=False*, *save_pdf=False*, *show=False*, *name='plot'*, *dpi=300*, *transparent=True*, *close_figure=True*)

Plot hexbin geoDataFrames to create the accesibility plots.

Parameters

- **{matplotlib.axes} -- ax to use in the plot** (*ax*) –
- **{geopandas.GeoDataFrame} -- geoDataFrame with the data to be plotted** (*gdf_data*) –
- **{geopandas.GeoDataFrame} -- geoDataFrame with the boundary to use** (*gdf_boundary*) –
- **{geopandas.GeoDataFrame} -- geoDataFrame with the edges** (*gdf_edges*) –
- **{geopandas.GeoDataFrame} -- column to plot from the gdf_data geoDataFrame** (*column*) –
- **{str} -- string with the title to use in the plot** (*title*) –

Keyword Arguments

- **{bool} -- save the plot in png or not (default** (*save_png*) – {False})
- **{bool} -- save the plot in pdf or not (default** (*save_pdf*) – {False})
- **{bool} -- show the plot or not (default** (*show*) – {False})
- **{str} -- name for the plot to be saved if save=True (default** (*name*) – {plot})
- **{int} -- resolution to use (default** (*dpi*) – {300})
- **{bool} -- save with transparency or not (default** (*transparent*) – {True})

- *Index*
- *Module Index*
- *Search Page*

Python Module Index

a

A

C

D

F

G

H

L

M

T

V