OSMnx Documentation

Release 0.14.1

Geoff Boeing

Jun 13, 2020

REFERENCE

CHAPTER

ONE

OSMNX PACKAGE

1.1 osmnx.bearing module

Calculate graph edge bearings.

osmnx.bearing.**add_edge_bearings**(*G*, *precision=1*)

Add *bearing* attributes to all graph edges.

Calculate the compass bearing from origin node to destination node for each edge in the directed graph then add each bearing as a new edge attribute.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **precision** (*int*) – decimal precision to round bearing

Returns **G** – graph with edge bearing attributes

Return type networkx.MultiDiGraph

osmnx.bearing.**get_bearing**(*origin_point*, *destination_point*)

Calculate the bearing between two lat-lng points.

Each tuple should represent (lat, lng) as decimal degrees.

Parameters

- **origin_point** (*tuple*) – (lat, lng)
- **destination_point** (*tuple*) – (lat, lng)

Returns **bearing** – the compass bearing in decimal degrees from the origin point to the destination point

Return type float

1.2 osmnx.boundaries module

Create GeoDataFrames of place boundaries.

osmnx.boundaries.**gdf_from_place**(*query*, *which_result=1*, *buffer_dist=None*)

Create a GeoDataFrame from a single place name query.

Geocode the query with Nominatim then turn it into a GeoDataFrame with a geometry column.

Parameters

- **query** (*string or dict*) – query string or structured query dict to geocode/download
- **which_result** (*int*) – max number of results to return and which to process upon receipt
- **buffer_dist** (*float*) – distance to buffer around the place geometry, in meters

Returns gdf

Return type geopandas.GeoDataFrame

osmnx.boundaries.**gdf_from_places**(*queries*, *which_results=None*, *buffer_dist=None*)

Create a GeoDataFrame from a list of place name queries.

Geocode the queries with Nominatim then turn result into GeoDataFrame with a geometry column.

Parameters

- **queries** (*list*) – list of query strings or structured query dicts to geocode/download, one at a time
- **which_results** (*list*) – if not None, a list of max number of results to return and which to process upon receipt, for each query in queries
- **buffer_dist** (*float*) – distance to buffer around the place geometry, in meters

Returns gdf

Return type geopandas.GeoDataFrame

1.3 osmnx.distance module

Functions to calculate distances and find nearest node/edge(s) to point(s).

osmnx.distance.**euclidean_dist_vec**(*y1*, *x1*, *y2*, *x2*)

Calculate euclidean distances.

Vectorized function to calculate the euclidean distance between two points or between arrays of points.

Parameters

- **y1** (*float or np.array of float*) – first y coord
- **x1** (*float or np.array of float*) – first x coord
- **y2** (*float or np.array of float*) – second y coord
- **x2** (*float or np.array of float*) – second x coord

Returns **dist** – distance or vector of distances from (x1, y1) to (x2, y2) in graph units

Return type float or np.array of float

osmnx.distance.**get_nearest_edge**(*G*, *point*, *return_geom=False*, *return_dist=False*)

Return the nearest edge to a point, by minimum euclidean distance.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **point** (*tuple*) – the (lat, lng) or (y, x) point for which we will find the nearest edge in the graph
- **return_geom** (*bool*) – Optionally return the geometry of the nearest edge

- **return_dist** (*bool*) – Optionally return the distance in graph's coordinates' units between the point and the nearest edge

Returns Graph edge unique identifier as a tuple of (u, v, key). Or a tuple of (u, v, key, geom) if return_geom is True. Or a tuple of (u, v, key, dist) if return_dist is True. Or a tuple of (u, v, key, geom, dist) if return_geom and return_dist are True.

Return type tuple

osmnx.distance.**get_nearest_edges**(*G*, *X*, *Y*, *method=None*, *dist=0.0001*)

Return the graph edges nearest to a list of points.

Pass in points as separate vectors of X and Y coordinates. The 'kdtree' method is by far the fastest with large data sets, but only finds approximate nearest edges if working in unprojected coordinates like lat-lng (it precisely finds the nearest edge if working in projected coordinates). The 'balltree' method is second fastest with large data sets, but it is precise if working in unprojected coordinates like lat-lng. As a rule of thumb, if you have a small graph just use method=None. If you have a large graph with lat-lng coordinates, use method='balltree'. If you have a large graph with projected coordinates, use method='kdtree'. Note that if you are working in units of lat-lng, the X vector corresponds to longitude and the Y vector corresponds to latitude. The method creates equally distanced points along the edges of the network. Then, these points are used in a kdTree or BallTree search to identify which is nearest.Note that this method will not give the exact perpendicular point along the edge, but the smaller the *dist* parameter, the closer the solution will be.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **X** (*list-like*) – The vector of longitudes or x's for which we will find the nearest edge in the graph. For projected graphs, use the projected coordinates, usually in meters.
- **Y** (*list-like*) – The vector of latitudes or y's for which we will find the nearest edge in the graph. For projected graphs, use the projected coordinates, usually in meters.
- **method** (*string {None, 'kdtree', 'balltree'}*) – Which method to use for finding nearest edge to each point. If None, we manually find each edge one at a time using get_nearest_edge. If 'kdtree' we use scipy.spatial.cKDTree for very fast euclidean search. Recommended for projected graphs. If 'balltree', we use sklearn.neighbors.BallTree for fast haversine search. Recommended for unprojected graphs.
- **dist** (*float*) – spacing length along edges. Units are the same as the geom; Degrees for unprojected geometries and meters for projected geometries. The smaller the value, the more points are created.

Returns **ne** – array of nearest edges represented by u and v (the IDs of the nodes they link) and key

Return type np.array

osmnx.distance.**get_nearest_node**(*G*, *point*, *method='haversine'*, *return_dist=False*)

Find node nearest to a point.

Return the graph node nearest to some specified (lat, lng) or (y, x) point and optionally the distance between the node and the point. This function can use either a haversine or euclidean distance calculator.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **point** (*tuple*) – The (lat, lng) or (y, x) point for which we will find the nearest node in the graph
- **method** (*string {'haversine', 'euclidean'}*) – Which method to use for calculating distances to find nearest node. If 'haversine', graph nodes' coordinates must be in units of decimal degrees. If 'euclidean', graph nodes' coordinates must be projected.

- **return_dist** (*bool*) – Optionally also return the distance (in meters if haversine, or graph node coordinate units if euclidean) between the point and the nearest node

Returns Nearest node ID or optionally a tuple of (node ID, dist), where dist is the distance (in meters if haversine, or graph node coordinate units if euclidean) between the point and nearest node

Return type int or tuple of (int, float)

osmnx.distance.**get_nearest_nodes**(*G*, *X*, *Y*, *method=None*)

Return the graph nodes nearest to a list of points.

Pass in points as separate vectors of X and Y coordinates. The 'kdtree' method is by far the fastest with large data sets, but only finds approximate nearest nodes if working in unprojected coordinates like lat-lng (it precisely finds the nearest node if working in projected coordinates). The 'balltree' method is second fastest with large data sets but it is precise if working in unprojected coordinates like lat-lng.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **X** (*list-like*) – The vector of longitudes or x's for which we will find the nearest node in the graph
- **Y** (*list-like*) – The vector of latitudes or y's for which we will find the nearest node in the graph
- **method** (*string {None, 'kdtree', 'balltree'}*) – Which method to use for finding nearest node to each point. If None, we manually find each node one at a time using utils.get_nearest_node and haversine. If 'kdtree' we use scipy.spatial.cKDTree for very fast euclidean search. If 'balltree', we use sklearn.neighbors.BallTree for fast haversine search.

Returns **nn** – list of nearest node IDs

Return type np.array

osmnx.distance.**great_circle_vec**(*lat1*, *lng1*, *lat2*, *lng2*, *earth_radius=6371009*)

Calculate great-circle distances.

Vectorized function to calculate the great-circle distance between two points or between vectors of points, using haversine.

Parameters

- **lat1** (*float or array of float*) – first lat coord
- **lng1** (*float or array of float*) – first lng coord
- **lat2** (*float or array of float*) – second lat coord
- **lng2** (*float or array of float*) – second lng coord
- **earth_radius** (*numeric*) – radius of earth in units in which distance will be returned (default is meters)

Returns **dist** – distance or array of distances from (lat1, lng1) to (lat2, lng2) in units of earth_radius

Return type float or np.array of floats

1.4 osmnx.downloader module

Interact with the OSM APIs.

osmnx.downloader.**nominatim_request**(*params*, *request_type='search'*, *pause=1*, *error_pause=180*)

Send a request to the Nominatim API via HTTP GET and return JSON response.

Parameters

- **params** (*dict or OrderedDict*) – key-value pairs of parameters
- **request_type** (*string*) – Type of Nominatim query. One of: search, reverse, or lookup
- **pause** (*int*) – how long to pause before requests, in seconds
- **error_pause** (*int*) – how long to pause in seconds before re-trying requests if error

Returns response_json

Return type dict

osmnx.downloader.**overpass_request**(*data*, *pause=None*, *error_pause=None*)

Send a request to the Overpass API via HTTP POST and return JSON response.

Parameters

- **data** (*dict or OrderedDict*) – key-value pairs of parameters to post to the API
- **pause** (*int*) – how long to pause in seconds before requests, if None, will query API status endpoint to find when next slot is available
- **error_pause** (*int*) – how long to pause in seconds before re-trying requests if error

Returns

Return type dict

1.5 osmnx.elevation module

Get node elevations and calculate edge grades.

osmnx.elevation.**add_edge_grades**(*G*, *add_absolute=True*, *precision=3*)

Add *grade* attribute to each graph edge.

Get the directed grade (ie, rise over run) for each edge in the network and add it to the edge as an attribute. Nodes must have elevation attributes to use this function.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **add_absolute** (*bool*) – if True, also add the absolute value of the grade as an edge attribute called grade_abs
- **precision** (*int*) – decimal precision to round grades

Returns **G** – graph with edge grade (and optionally grade_abs) attributes

Return type networkx.MultiDiGraph

osmnx.elevation.**add_node_elevations**(*G*, *api_key*, *max_locations_per_batch=350*, *pause_duration=0.02*, *precision=3*)

Add *elevation* (meters) attribute to each node.

Uses the google maps elevation API.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **api_key** (*string*) – your google maps elevation API key
- **max_locations_per_batch** (*int*) – max number of coordinate pairs to submit in each API call (if this is too high, the server will reject the request because its character limit exceeds the max)
- **pause_duration** (*float*) – time to pause between API calls
- **precision** (*int*) – decimal precision to round elevation

Returns G – graph with node elevation attributes

Return type networkx.MultiDiGraph

1.6 osmnx.folium module

Create leaflet web maps via folium.

osmnx.folium.**plot_graph_folium**(*G*, *graph_map=None*, *popup_attribute=None*, *tiles='cartodbpositron'*, *zoom=1*, *fit_bounds=True*, *edge_color='#333333'*, *edge_width=5*, *edge_opacity=1*, ***kwargs*)

Plot a graph on an interactive folium web map.

Note that anything larger than a small city can take a long time to plot and create a large web map file that is very slow to load as JavaScript.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **graph_map** (*folium.folium.Map or folium.FeatureGroup*) – if not None, plot the graph on this preexisting folium map object
- **popup_attribute** (*string*) – edge attribute to display in a pop-up when an edge is clicked
- **tiles** (*string*) – name of a folium tileset
- **zoom** (*int*) – initial zoom level for the map
- **fit_bounds** (*bool*) – if True, fit the map to the boundaries of the route's edges
- **edge_color** (*string*) – color of the edge lines
- **edge_width** (*numeric*) – width of the edge lines
- **edge_opacity** (*numeric*) – opacity of the edge lines
- **kwargs** (*dict*) – Extra keyword arguments passed through to folium

Returns graph_map

Return type folium.folium.Map

osmnx.folium.**plot_route_folium**(*G*, *route*, *route_map=None*, *popup_attribute=None*, *tiles='cartodbpositron'*, *zoom=1*, *fit_bounds=True*, *route_color='#cc0000'*, *route_width=5*, *route_opacity=1*, ***kwargs*)

Plot a route on an interactive folium web map.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **route** (*list*) – the route as a list of nodes
- **route_map** (*folium.folium.Map*) – if not None, plot the route on this preexisting folium map object
- **popup_attribute** (*string*) – edge attribute to display in a pop-up when an edge is clicked
- **tiles** (*string*) – name of a folium tileset
- **zoom** (*int*) – initial zoom level for the map
- **fit_bounds** (*bool*) – if True, fit the map to the boundaries of the route's edges
- **route_color** (*string*) – color of the route's line
- **route_width** (*numeric*) – width of the route's line
- **route_opacity** (*numeric*) – opacity of the route lines
- **kwargs** (*dict*) – Extra parameters passed through to folium

Returns route_map

Return type folium.folium.Map

1.7 osmnx.footprints module

Download and plot footprints from OpenStreetMap.

osmnx.footprints.**footprints_from_address**(*address*, *dist=1000*, *footprint_type='building'*, *retain_invalid=False*, *timeout=None*, *memory=None*, *custom_settings=None*)

Get footprints within some distance N, S, E, W of an address.

Parameters

- **address** (*string*) – the address to geocode to a lat-lng point
- **dist** (*numeric*) – distance in meters
- **footprint_type** (*string*) – type of footprint to be downloaded. OSM tag key e.g. 'building', 'landuse', 'place', etc.
- **retain_invalid** (*bool*) – if False discard any footprints with an invalid geometry
- **timeout** (*None*) – deprecated, use ox.config(timeout=value) instead to configure this setting via the settings module
- **memory** (*None*) – deprecated, use ox.config(memory=value) instead to configure this setting via the settings module
- **custom_settings** (*None*) – deprecated, use ox.config(custom_settings=value) instead to configure this setting via the settings module

Returns

Return type geopandas.GeoDataFrame

osmnx.footprints.**footprints_from_place**(*place*, *footprint_type='building'*, *retain_invalid=False*, *which_result=1*, *timeout=None*, *memory=None*, *custom_settings=None*)

Get footprints within the boundaries of some place.

The query must be geocodable and OSM must have polygon boundaries for the geocode result. If OSM does not have a polygon for this place, you can instead get its footprints using the footprints_from_address function, which geocodes the place name to a point and gets the footprints within some distance of that point.

Parameters

- **place** (*string*) – the query to geocode to get geojson boundary polygon
- **footprint_type** (*string*) – type of footprint to be downloaded. OSM tag key e.g. 'building', 'landuse', 'place', etc.
- **retain_invalid** (*bool*) – if False discard any footprints with an invalid geometry
- **which_result** (*int*) – max number of results to return and which to process upon receipt
- **timeout** (*None*) – deprecated, use ox.config(timeout=value) instead to configure this setting via the settings module
- **memory** (*None*) – deprecated, use ox.config(memory=value) instead to configure this setting via the settings module
- **custom_settings** (*None*) – deprecated, use ox.config(custom_settings=value) instead to configure this setting via the settings module

Returns

Return type geopandas.GeoDataFrame

osmnx.footprints.**footprints_from_point**(*point*, *dist=1000*, *footprint_type='building'*, *retain_invalid=False*, *timeout=None*, *memory=None*, *custom_settings=None*)

Get footprints within some distance N, S, E, W of a lat-lng point.

Parameters

- **point** (*tuple*) – a lat-lng point
- **dist** (*numeric*) – distance in meters
- **footprint_type** (*string*) – type of footprint to be downloaded. OSM tag key e.g. 'building', 'landuse', 'place', etc.
- **retain_invalid** (*bool*) – if False discard any footprints with an invalid geometry
- **timeout** (*None*) – deprecated, use ox.config(timeout=value) instead to configure this setting via the settings module
- **memory** (*None*) – deprecated, use ox.config(memory=value) instead to configure this setting via the settings module
- **custom_settings** (*None*) – deprecated, use ox.config(custom_settings=value) instead to configure this setting via the settings module

Returns

Return type geopandas.GeoDataFrame

osmnx.footprints.**footprints_from_polygon**(*polygon*, *footprint_type='building'*, *retain_invalid=False*, *timeout=None*, *memory=None*, *custom_settings=None*)

Get footprints within some polygon.

Parameters

- **polygon** (*shapely.geometry.Polygon or shapely.geometry.MultiPolygon*) – the shape to get data within. coordinates should be in units of latitude-longitude degrees.
- **footprint_type** (*string*) – type of footprint to be downloaded. OSM tag key e.g. 'building', 'landuse', 'place', etc.
- **retain_invalid** (*bool*) – if False discard any footprints with an invalid geometry
- **timeout** (*None*) – deprecated, use ox.config(timeout=value) instead to configure this setting via the settings module
- **memory** (*None*) – deprecated, use ox.config(memory=value) instead to configure this setting via the settings module
- **custom_settings** (*None*) – deprecated, use ox.config(custom_settings=value) instead to configure this setting via the settings module

Returns

Return type geopandas.GeoDataFrame

1.8 osmnx.graph module

Graph creation functions.

osmnx.graph.**graph_from_address**(*address*, *dist=1000*, *dist_type='bbox'*, *network_type='all_private'*, *simplify=True*, *retain_all=False*, *truncate_by_edge=False*, *return_coords=False*, *clean_periphery=True*, *custom_filter=None*, *timeout=None*, *memory=None*, *custom_settings=None*, *max_query_area_size=None*)

Create a graph from OSM within some distance of some address.

Parameters

- **address** (*string*) – the address to geocode and use as the central point around which to construct the graph
- **dist** (*int*) – retain only those nodes within this many meters of the center of the graph
- **dist_type** (*string*) – {'network', 'bbox'} if 'bbox', retain only those nodes within a bounding box of the distance parameter. if 'network', retain only those nodes within some network distance from the center-most node.
- **network_type** (*string*) – what type of street network to get if custom_filter is None. One of 'walk', 'bike', 'drive', 'drive_service', 'all', or 'all_private'.
- **simplify** (*bool*) – if True, simplify the graph topology
- **retain_all** (*bool*) – if True, return the entire graph even if it is not connected
- **truncate_by_edge** (*bool*) – if True, retain node if it's outside bounding box but at least one of node's neighbors are within bounding box

- **return_coords** (*bool*) – optionally also return the geocoded coordinates of the address
- **clean_periphery** (*bool,*) – if True, buffer 500m to get a graph larger than requested, then simplify, then truncate it to requested spatial boundaries
- **custom_filter** (*string*) – a custom network filter to be used instead of the network_type presets, e.g., '["power"~"line"]' or '["highway"~"motorway|trunk"]'. Also pass in a network_type that is in settings.bidirectional_network_types if you want graph to be fully bidirectional.
- **timeout** (*None*) – deprecated, use ox.config(timeout=value) instead to configure this setting via the settings module
- **memory** (*None*) – deprecated, use ox.config(memory=value) instead to configure this setting via the settings module
- **custom_settings** (*None*) – deprecated, use ox.config(custom_settings=value) instead to configure this setting via the settings module
- **max_query_area_size** (*None*) – deprecated, use ox.config(max_query_area_size=value) instead to configure this setting via the settings module

Returns

Return type networkx.MultiDiGraph or optionally (networkx.MultiDiGraph, (lat, lng))

osmnx.graph.**graph_from_bbox**(*north*, *south*, *east*, *west*, *network_type='all_private'*, *simplify=True*, *retain_all=False*, *truncate_by_edge=False*, *clean_periphery=True*, *custom_filter=None*, *timeout=None*, *memory=None*, *custom_settings=None*, *max_query_area_size=None*)

Create a graph from OSM within some bounding box.

Parameters

- **north** (*float*) – northern latitude of bounding box
- **south** (*float*) – southern latitude of bounding box
- **east** (*float*) – eastern longitude of bounding box
- **west** (*float*) – western longitude of bounding box
- **network_type** (*string*) – what type of street network to get if custom_filter is None. One of 'walk', 'bike', 'drive', 'drive_service', 'all', or 'all_private'.
- **simplify** (*bool*) – if True, simplify the graph topology
- **retain_all** (*bool*) – if True, return the entire graph even if it is not connected
- **truncate_by_edge** (*bool*) – if True, retain node if it's outside bounding box but at least one of node's neighbors are within the bounding box
- **clean_periphery** (*bool*) – if True, buffer 500m to get a graph larger than requested, then simplify, then truncate it to requested spatial boundaries
- **custom_filter** (*string*) – a custom network filter to be used instead of the network_type presets, e.g., '["power"~"line"]' or '["highway"~"motorway|trunk"]'. Also pass in a network_type that is in settings.bidirectional_network_types if you want graph to be fully bidirectional.
- **timeout** (*None*) – deprecated, use ox.config(timeout=value) instead to configure this setting via the settings module

- **memory** (*None*) – deprecated, use ox.config(memory=value) instead to configure this setting via the settings module
- **custom_settings** (*None*) – deprecated, use ox.config(custom_settings=value) instead to configure this setting via the settings module
- **max_query_area_size** (*None*) – deprecated, use ox.config(max_query_area_size=value) instead to configure this setting via the settings module

Returns G

Return type networkx.MultiDiGraph

osmnx.graph.**graph_from_place**(*query*, *network_type='all_private'*, *simplify=True*, *retain_all=False*, *truncate_by_edge=False*, *which_result=1*, *buffer_dist=None*, *clean_periphery=True*, *custom_filter=None*, *timeout=None*, *memory=None*, *custom_settings=None*, *max_query_area_size=None*)

Create graph from OSM within the boundaries of some geocodable place(s).

The query must be geocodable and OSM must have polygon boundaries for the geocode result. If OSM does not have a polygon for this place, you can instead get its street network using the graph_from_address function, which geocodes the place name to a point and gets the network within some distance of that point. Alternatively, you might try to vary the which_result parameter to use a different geocode result. For example, the first geocode result (ie, the default) might resolve to a point geometry, but the second geocode result for this query might resolve to a polygon, in which case you can use graph_from_place with which_result=2.

Parameters

- **query** (*string or dict or list*) – the place(s) to geocode/download data for
- **network_type** (*string*) – what type of street network to get if custom_filter is None. One of 'walk', 'bike', 'drive', 'drive_service', 'all', or 'all_private'.
- **simplify** (*bool*) – if True, simplify the graph topology
- **retain_all** (*bool*) – if True, return the entire graph even if it is not connected
- **truncate_by_edge** (*bool*) – if True, retain node if it's outside polygon but at least one of node's neighbors are within bbox
- **which_result** (*int*) – max number of results to return and which to process upon receipt
- **buffer_dist** (*float*) – distance to buffer around the place geometry, in meters
- **clean_periphery** (*bool*) – if True, buffer 500m to get a graph larger than requested, then simplify, then truncate it to requested spatial boundaries
- **custom_filter** (*string*) – a custom network filter to be used instead of the network_type presets, e.g., '["power"~"line"]' or '["highway"~"motorway|trunk"]'. Also pass in a network_type that is in settings.bidirectional_network_types if you want graph to be fully bidirectional.
- **timeout** (*None*) – deprecated, use ox.config(timeout=value) instead to configure this setting via the settings module
- **memory** (*None*) – deprecated, use ox.config(memory=value) instead to configure this setting via the settings module
- **custom_settings** (*None*) – deprecated, use ox.config(custom_settings=value) instead to configure this setting via the settings module

- **max_query_area_size** (*None*) – deprecated, use ox.config(max_query_area_size=value) instead to configure this setting via the settings module

Returns G

Return type networkx.MultiDiGraph

osmnx.graph.**graph_from_point**(*center_point*, *dist=1000*, *dist_type='bbox'*, *network_type='all_private'*, *simplify=True*, *retain_all=False*, *truncate_by_edge=False*, *clean_periphery=True*, *custom_filter=None*, *timeout=None*, *memory=None*, *custom_settings=None*, *max_query_area_size=None*)

Create a graph from OSM within some distance of some (lat, lng) point.

Parameters

- **center_point** (*tuple*) – the (lat, lng) center point around which to construct the graph
- **dist** (*int*) – retain only those nodes within this many meters of the center of the graph, with distance determined according to dist_type argument
- **dist_type** (*string*) – {'network', 'bbox'} if 'bbox', retain only those nodes within a bounding box of the distance parameter. if 'network', retain only those nodes within some network distance from the center-most node.
- **network_type** (*string*) – what type of street network to get if custom_filter is None. One of 'walk', 'bike', 'drive', 'drive_service', 'all', or 'all_private'.
- **simplify** (*bool*) – if True, simplify the graph topology
- **retain_all** (*bool*) – if True, return the entire graph even if it is not connected
- **truncate_by_edge** (*bool*) – if True, retain node if it's outside bounding box but at least one of node's neighbors are within bounding box
- **clean_periphery** (*bool,*) – if True, buffer 500m to get a graph larger than requested, then simplify, then truncate it to requested spatial boundaries
- **custom_filter** (*string*) – a custom network filter to be used instead of the network_type presets, e.g., '["power"~"line"]' or '["highway"~"motorway|trunk"]'. Also pass in a network_type that is in settings.bidirectional_network_types if you want graph to be fully bidirectional.
- **timeout** (*None*) – deprecated, use ox.config(timeout=value) instead to configure this setting via the settings module
- **memory** (*None*) – deprecated, use ox.config(memory=value) instead to configure this setting via the settings module
- **custom_settings** (*None*) – deprecated, use ox.config(custom_settings=value) instead to configure this setting via the settings module
- **max_query_area_size** (*None*) – deprecated, use ox.config(max_query_area_size=value) instead to configure this setting via the settings module

Returns G

Return type networkx.MultiDiGraph

osmnx.graph.**graph_from_polygon**(*polygon*, *network_type='all_private'*, *simplify=True*, *retain_all=False*, *truncate_by_edge=False*, *clean_periphery=True*, *custom_filter=None*, *timeout=None*, *memory=None*, *custom_settings=None*, *max_query_area_size=None*)

Create a graph from OSM within the boundaries of some shapely polygon.

Parameters

- **polygon** (*shapely.geometry.Polygon or shapely.geometry.MultiPolygon*) – the shape to get network data within. coordinates should be in units of latitude-longitude degrees.
- **network_type** (*string*) – what type of street network to get if custom_filter is None. One of 'walk', 'bike', 'drive', 'drive_service', 'all', or 'all_private'.
- **simplify** (*bool*) – if True, simplify the graph topology
- **retain_all** (*bool*) – if True, return the entire graph even if it is not connected
- **truncate_by_edge** (*bool*) – if True, retain node if it's outside polygon but at least one of node's neighbors are within polygon
- **clean_periphery** (*bool*) – if True, buffer 500m to get a graph larger than requested, then simplify, then truncate it to requested spatial boundaries
- **custom_filter** (*string*) – a custom network filter to be used instead of the network_type presets, e.g., '["power"~"line"]' or '["highway"~"motorway|trunk"]'. Also pass in a network_type that is in settings.bidirectional_network_types if you want graph to be fully bidirectional.
- **timeout** (*None*) – deprecated, use ox.config(timeout=value) instead to configure this setting via the settings module
- **memory** (*None*) – deprecated, use ox.config(memory=value) instead to configure this setting via the settings module
- **custom_settings** (*None*) – deprecated, use ox.config(custom_settings=value) instead to configure this setting via the settings module
- **max_query_area_size** (*None*) – deprecated, use ox.config(max_query_area_size=value) instead to configure this setting via the settings module

Returns G

Return type networkx.MultiDiGraph

osmnx.graph.**graph_from_xml**(*filepath*, *bidirectional=False*, *simplify=True*, *retain_all=False*)

Create a graph from data in an OSM-formatted XML file.

Parameters

- **filepath** (*string*) – path to file containing OSM XML data
- **bidirectional** (*bool*) – if True, create bidirectional edges for one-way streets
- **simplify** (*bool*) – if True, simplify the graph topology
- **retain_all** (*bool*) – if True, return the entire graph even if it is not connected

Returns G

Return type networkx.MultiDiGraph

1.9 osmnx.io module

Serialize graphs to/from files on disk.

osmnx.io.**load_graphml**(*filepath*, *node_type=<class 'int'>*)
Load an OSMnx-saved GraphML file from disk.

Converts the node/edge attributes to appropriate data types.

Parameters

- **filepath** (*string*) – path to the GraphML file
- **node_type** (*type*) – convert node ids to this data type

Returns G

Return type networkx.MultiDiGraph

osmnx.io.**save_graph_geopackage**(*G*, *filepath=None*, *encoding='utf-8'*)
Save graph nodes and edges to disk as layers in a GeoPackage file.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **filepath** (*string*) – path to the GeoPackage file including extension. if None, use default data folder + graph.gpkg
- **encoding** (*string*) – the character encoding for the saved file

Returns

Return type None

osmnx.io.**save_graph_shapefile**(*G*, *filepath=None*, *encoding='utf-8'*)
Save graph nodes and edges to disk as ESRI shapefiles.

The shapefile format is proprietary and outdated. Whenever possible, you should use the superior GeoPackage file format instead, for instance, via the save_graph_geopackage function.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **filepath** (*string*) – path to the shapefiles folder (no file extension). if None, use default data folder + graph_shapefile
- **encoding** (*string*) – the character encoding for the saved files

Returns

Return type None

osmnx.io.**save_graph_xml**(*data, filepath=None, node_tags=['highway'], node_attrs=['id', 'timestamp', 'uid', 'user', 'version', 'changeset', 'lat', 'lon'], edge_tags=['highway', 'lanes', 'maxspeed', 'name', 'oneway'], edge_attrs=['id', 'timestamp', 'uid', 'user', 'version', 'changeset'], oneway=False, merge_edges=True, edge_tag_aggs=None*)
Save graph to disk as an OSM-formatted XML .osm file.

Note: for large networks this function can take a long time to run. Before using this function, make sure you configured OSMnx as described in the example below when you created the graph.

Example

```
>>> import osmnx as ox
>>> utn = ox.settings.useful_tags_node
>>> oxna = ox.settings.osm_xml_node_attrs
>>> oxnt = ox.settings.osm_xml_node_tags
>>> utw = ox.settings.useful_tags_way
>>> oxwa = ox.settings.osm_xml_way_attrs
>>> oxwt = ox.settings.osm_xml_way_tags
>>> utn = list(set(utn + oxna + oxnt))
>>> utw = list(set(utw + oxwa + oxwt))
>>> ox.config(all_oneway=True, useful_tags_node=utn, useful_tags_way=utw)
>>> G = ox.graph_from_place('Piedmont, CA, USA', network_type='drive')
>>> ox.save_graph_xml(G, filepath='./data/graph1.osm')
```

Parameters

- **data** (*networkx multi(di)graph OR a length 2 iterable of nodes/edges*) – geopandas GeoDataFrames
- **filepath** (*string*) – path to the .osm file including extension. if None, use default data folder + graph.osm
- **node_tags** (*list*) – osm node tags to include in output OSM XML
- **node_attrs** (*list*) – osm node attributes to include in output OSM XML
- **edge_tags** (*list*) – osm way tags to include in output OSM XML
- **edge_attrs** (*list*) – osm way attributes to include in output OSM XML
- **oneway** (*bool*) – the default oneway value used to fill this tag where missing
- **merge_edges** (*bool*) – if True merges graph edges such that each OSM way has one entry and one entry only in the OSM XML. Otherwise, every OSM way will have a separate entry for each node pair it contains.
- **edge_tag_aggs** (*list of length-2 string tuples*) – useful only if merge_edges is True, this argument allows the user to specify edge attributes to aggregate such that the merged OSM way entry tags accurately represent the sum total of their component edge attributes. For example, if the user wants the OSM way to have a "length" attribute, the user must specify *edge_tag_aggs=[('length', 'sum')]* in order to tell this method to aggregate the lengths of the individual component edges. Otherwise, the length attribute will simply reflect the length of the first edge associated with the way.

Returns

Return type None

osmnx.io.**save_graphml**(*G*, *filepath=None*, *gephi=False*, *encoding='utf-8'*)

Save graph to disk as GraphML file.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **filepath** (*string*) – path to the GraphML file including extension. if None, use default data folder + graph.graphml
- **gephi** (*bool*) – if True, give each edge a unique key to work around Gephi's restrictive interpretation of the GraphML specification

- **encoding** (*string*) – the character encoding for the saved file

Returns

Return type None

1.10 osmnx.plot module

Plot spatial geometries, street networks, and routes.

osmnx.plot.**get_colors**(*n*, *cmap='viridis'*, *start=0.0*, *stop=1.0*, *alpha=1.0*, *return_hex=False*)

Return n-length list of RGBa colors from the passed colormap name and alpha.

Parameters

- **n** (*int*) – number of colors
- **cmap** (*string*) – name of a colormap
- **start** (*float*) – where to start in the colorspace
- **stop** (*float*) – where to end in the colorspace
- **alpha** (*float*) – opacity, the alpha channel for the RGBa colors
- **return_hex** (*bool*) – if True, convert RGBa colors to a hexadecimal string

Returns colors

Return type list

osmnx.plot.**get_edge_colors_by_attr**(*G*, *attr*, *num_bins=5*, *cmap='viridis'*, *start=0*, *stop=1*, *na_color='none'*)

Get a list of edge colors by binning continuous attribute into quantiles.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **attr** (*string*) – the name of the continuous-variable attribute
- **num_bins** (*int*) – how many quantiles
- **cmap** (*string*) – name of a colormap
- **start** (*float*) – where to start in the colorspace
- **stop** (*float*) – where to end in the colorspace
- **na_color** (*string*) – what color to assign nodes with null attribute values

Returns edge_colors

Return type list

osmnx.plot.**get_node_colors_by_attr**(*G*, *attr*, *num_bins=None*, *cmap='viridis'*, *start=0*, *stop=1*, *na_color='none'*)

Get a list of node colors by binning continuous attribute into quantiles.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **attr** (*string*) – the name of the attribute
- **num_bins** (*int*) – how many quantiles (default None assigns each node to its own bin)

- **cmap** (*string*) – name of a colormap
- **start** (*float*) – where to start in the colorspace
- **stop** (*float*) – where to end in the colorspace
- **na_color** (*string*) – what color to assign nodes with null attribute values

Returns node_colors

Return type list

osmnx.plot.**plot_figure_ground**(*G=None*, *address=None*, *point=None*, *dist=805*, *network_type='drive_service'*, *street_widths=None*, *default_width=4*, *fig_length=None*, *edge_color='w'*, *bgcolor='#333333'*, *smooth_joints=True*, *filename=None*, *file_format=None*, *show=False*, *save=True*, *close=True*, *dpi=300*, *figsize=None*, *filepath=None*)

Plot figure-ground diagram of a street network.

Defaults to one square mile.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph, must be unprojected
- **address** (*string*) – the address to geocode as the center point if G is not passed in
- **point** (*tuple*) – the center point if address and G are not passed in
- **dist** (*numeric*) – how many meters to extend north, south, east, and west from the center point
- **network_type** (*string*) – what type of network to get
- **street_widths** (*dict*) – where keys are street types and values are widths to plot in pixels
- **default_width** (*numeric*) – the default street width in pixels for any street type not found in street_widths dict
- **fig_length** (*numeric*) – deprecated, do not use
- **edge_color** (*string*) – the color of the streets
- **bgcolor** (*string*) – the color of the background
- **smooth_joints** (*bool*) – if True, plot nodes same width as streets to smooth line joints and prevent cracks between them from showing
- **filename** (*string*) – deprecated, do not use
- **file_format** (*string*) – deprecated, do not use
- **show** (*bool*) – if True, show the figure
- **save** (*bool*) – if True, save the figure as an image file to disk
- **close** (*bool*) – close the figure (only if show equals False) to prevent display
- **dpi** (*int*) – the resolution of the image file if saving
- **figsize** (*tuple*) – figure width, height (should be equal to each other)
- **filepath** (*string*) – filename.ext to save image in settings.imgs_folder

Returns **fig, ax** – matplotlib figure, axis

Return type tuple

osmnx.plot.**plot_footprints**(*gdf*, *fig=None*, *ax=None*, *figsize=None*, *color='#333333'*, *bgcolor='w'*, *set_bounds=True*, *bbox=None*, *save=False*, *show=True*, *close=False*, *filename=None*, *file_format=None*, *dpi=600*, *filepath=None*)

Plot a GeoDataFrame of footprints.

Parameters

- **gdf** (*geopandas.GeoDataFrame*) – GeoDataFrame of footprints
- **fig** (*figure*) – matplotlib figure
- **ax** (*axis*) – matplotlib axis
- **figsize** (*tuple*) – (width, height) size of matplotlib figure
- **color** (*string*) – the color of the footprints
- **bgcolor** (*string*) – the background color of the plot
- **set_bounds** (*bool*) – if True, set bounds from either passed-in bbox or the spatial extent of the gdf
- **bbox** (*tuple*) – if True and if set_bounds is True, set the display bounds to this bbox
- **save** (*bool*) – whether to save the figure to disk or not
- **show** (*bool*) – whether to display the figure or not
- **close** (*bool*) – close the figure (only if show equals False) to prevent display
- **filename** (*string*) – deprecated, do not use
- **file_format** (*string*) – deprecated, do not use
- **dpi** (*int*) – the resolution of the image file if saving
- **filepath** (*string*) – filename.ext to save image in settings.imgs_folder

Returns **fig, ax** – matplotlib figure, axis

Return type tuple

osmnx.plot.**plot_graph**(*G*, *bbox=None*, *fig_height=None*, *fig_width=None*, *margin=0.02*, *axis_off=None*, *equal_aspect=None*, *bgcolor='w'*, *show=True*, *save=False*, *close=True*, *file_format=None*, *filename=None*, *dpi=300*, *annotate=None*, *node_color='#66ccff'*, *node_size=15*, *node_alpha=1*, *node_edgecolor='none'*, *node_zorder=1*, *edge_color='#999999'*, *edge_linewidth=1*, *edge_alpha=1*, *use_geom=None*, *figsize=None*, *filepath=None*)

Plot a networkx spatial graph.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **bbox** (*tuple*) – bounding box as north,south,east,west - if None will calculate from spatial extents of data. if passing a bbox, you probably also want to pass margin=0 to constrain it.
- **fig_height** (*int*) – deprecated, do not use
- **fig_width** (*int*) – deprecated, do not use
- **margin** (*float*) – relative margin around the figure
- **axis_off** (*bool*) – deprecated, do not use
- **equal_aspect** (*bool*) – deprecated, do not use
- **bgcolor** (*string*) – the background color of the figure and axis

- **show** (*bool*) – if True, show the figure
- **save** (*bool*) – if True, save the figure as an image file to disk
- **close** (*bool*) – close the figure (only if show equals False) to prevent display
- **file_format** (*string*) – deprecated, do not use
- **filename** (*string*) – deprecated, do not use
- **dpi** (*int*) – the resolution of the image file if saving
- **annotate** (*bool*) – deprecated, do not use
- **node_color** (*string*) – the color of the nodes. color is passed to matplotlib
- **node_size** (*int*) – the size of the nodes
- **node_alpha** (*float*) – the opacity of the nodes. if you passed RGBA values to node_color, then set this to None to use the alpha channel in node_color
- **node_edgecolor** (*string*) – the color of the node's marker's border
- **node_zorder** (*int*) – zorder to plot nodes, edges are always 2, so make node_zorder 1 to plot nodes beneath them or 3 to plot nodes atop them
- **edge_color** (*string*) – the color of the edges' lines. color is passed to matplotlib.
- **edge_linewidth** (*float*) – the width of the edges' lines
- **edge_alpha** (*float*) – the opacity of the edges' lines. if you passed RGBA values to edge_color, then set this to None to use the alpha channel in edge_color
- **use_geom** (*bool*) – deprecated, do not use
- **figsize** (*tuple*) – figure (width, height)
- **filepath** (*string*) – filename.ext to save image in settings.imgs_folder

Returns **fig, ax** – matplotlib figure, axis

Return type tuple

osmnx.plot.**plot_graph_route**(*G*, *route*, *bbox=None*, *fig_height=None*, *fig_width=None*, *margin=0.02*, *bgcolor='w'*, *axis_off=None*, *show=True*, *save=False*, *close=True*, *file_format=None*, *filename=None*, *dpi=300*, *annotate=None*, *node_color='#999999'*, *node_size=15*, *node_alpha=1*, *node_edgecolor='none'*, *node_zorder=1*, *edge_color='#999999'*, *edge_linewidth=1*, *edge_alpha=1*, *use_geom=None*, *origin_point=None*, *destination_point=None*, *route_color='r'*, *route_linewidth=4*, *route_alpha=0.5*, *orig_dest_node_alpha=None*, *orig_dest_node_size=None*, *orig_dest_node_color=None*, *orig_dest_point_color=None*, *figsize=None*, *filepath=None*, *orig_dest_size=None*)

Plot a route along a networkx spatial graph.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **route** (*list*) – the route as a list of nodes
- **bbox** (*tuple*) – bounding box as north,south,east,west - if None will calculate from spatial extents of data. if passing a bbox, you probably also want to pass margin=0 to constrain it.
- **fig_height** (*int*) – deprecated, do not use

- **fig_width** (*int*) – deprecated, do not use
- **margin** (*float*) – relative margin around the figure
- **axis_off** (*bool*) – deprecated, do not use
- **bgcolor** (*string*) – the background color of the figure and axis
- **show** (*bool*) – if True, show the figure
- **save** (*bool*) – if True, save the figure as an image file to disk
- **close** (*bool*) – close the figure (only if show equals False) to prevent display
- **file_format** (*string*) – deprecated, do not use
- **filename** (*string*) – deprecated, do not use
- **dpi** (*int*) – the resolution of the image file if saving
- **annotate** (*bool*) – deprecated, do not use
- **node_color** (*string*) – the color of the nodes
- **node_size** (*int*) – the size of the nodes
- **node_alpha** (*float*) – the opacity of the nodes
- **node_edgecolor** (*string*) – the color of the node's marker's border
- **node_zorder** (*int*) – zorder to plot nodes, edges are always 2, so make node_zorder 1 to plot nodes beneath them or 3 to plot nodes atop them
- **edge_color** (*string*) – the color of the edges' lines
- **edge_linewidth** (*float*) – the width of the edges' lines
- **edge_alpha** (*float*) – the opacity of the edges' lines
- **use_geom** (*bool*) – deprecated, do not use
- **origin_point** (*tuple*) – deprecated, do not use
- **destination_point** (*tuple*) – deprecated, do not use
- **route_color** (*string*) – the color of the route
- **route_linewidth** (*int*) – the width of the route line
- **route_alpha** (*float*) – the opacity of the route line
- **orig_dest_node_alpha** (*float*) – deprecated, do not use
- **orig_dest_node_size** (*int*) – deprecated, do not use
- **orig_dest_node_color** (*string*) – deprecated, do not use
- **orig_dest_point_color** (*string*) – deprecated, do not use
- **figsize** (*tuple*) – figure (width, height)
- **filepath** (*string*) – filename.ext to save image in settings.imgs_folder
- **orig_dest_size** (*int*) – the size of the origin and destination nodes

Returns **fig, ax** – matplotlib figure, axis

Return type tuple

osmnx.plot.**plot_graph_routes**(*G*, *routes*, *bbox=None*, *fig_height=None*, *fig_width=None*, *margin=0.02*, *bgcolor='w'*, *axis_off=None*, *show=True*, *save=False*, *close=True*, *file_format=None*, *filename=None*, *dpi=300*, *annotate=None*, *node_color='#999999'*, *node_size=15*, *node_alpha=1*, *node_edgecolor='none'*, *node_zorder=1*, *edge_color='#999999'*, *edge_linewidth=1*, *edge_alpha=1*, *use_geom=None*, *orig_dest_points=None*, *route_color='r'*, *route_linewidth=4*, *route_alpha=0.5*, *orig_dest_node_alpha=None*, *orig_dest_node_size=None*, *orig_dest_node_color=None*, *orig_dest_point_color=None*, *figsize=None*, *filepath=None*, *orig_dest_size=None*)

Plot several routes along a networkx spatial graph.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **routes** (*list*) – the routes as a list of lists of nodes
- **bbox** (*tuple*) – bounding box as north,south,east,west - if None will calculate from spatial extents of data. if passing a bbox, you probably also want to pass margin=0 to constrain it.
- **fig_height** (*int*) – deprecated, do not use
- **fig_width** (*int*) – deprecated, do not use
- **margin** (*float*) – relative margin around the figure
- **axis_off** (*bool*) – if True turn off the matplotlib axis
- **bgcolor** (*string*) – the background color of the figure and axis
- **show** (*bool*) – if True, show the figure
- **save** (*bool*) – if True, save the figure as an image file to disk
- **close** (*bool*) – close the figure (only if show equals False) to prevent display
- **file_format** (*string*) – deprecated, do not use
- **filename** (*string*) – deprecated, do not use
- **dpi** (*int*) – the resolution of the image file if saving
- **annotate** (*bool*) – deprecated, do not use
- **node_color** (*string*) – the color of the nodes
- **node_size** (*int*) – the size of the nodes
- **node_alpha** (*float*) – the opacity of the nodes
- **node_edgecolor** (*string*) – the color of the node's marker's border
- **node_zorder** (*int*) – zorder to plot nodes, edges are always 2, so make node_zorder 1 to plot nodes beneath them or 3 to plot nodes atop them
- **edge_color** (*string*) – the color of the edges' lines
- **edge_linewidth** (*float*) – the width of the edges' lines
- **edge_alpha** (*float*) – the opacity of the edges' lines
- **use_geom** (*bool*) – deprecated, do not use
- **orig_dest_points** (*list of tuples*) – deprecated, do not use

- **route_color** (*string*) – route color (note: will be renamed *route_colors* and take list in next release)
- **route_linewidth** (*int*) – the width of the route line
- **route_alpha** (*float*) – the opacity of the route line
- **orig_dest_node_alpha** (*float*) – deprecated, do not use
- **orig_dest_node_size** (*int*) – deprecated, do not use
- **orig_dest_node_color** (*string*) – deprecated, do not use
- **orig_dest_point_color** (*string*) – deprecated, do not use
- **figsize** (*tuple*) – figure (width, height)
- **filepath** (*string*) – filename.ext to save image in settings.imgs_folder
- **orig_dest_size** (*int*) – the size of the origin and destination nodes

Returns **fig, ax** – matplotlib figure, axis

Return type tuple

osmnx.plot.**plot_shape**(*gdf*, *fc='#cbe0f0'*, *ec='#999999'*, *linewidth=1*, *alpha=1*, *figsize=(6, 6)*, *margin=0.02*, *axis_off=True*)

Plot a GeoDataFrame of place boundary geometries.

Parameters

- **gdf** (*geopandas.GeoDataFrame*) – the gdf containing the geometries to plot
- **fc** (*string or list*) – the facecolor (or list of facecolors) for the polygons
- **ec** (*string or list*) – the edgecolor (or list of edgecolors) for the polygons
- **linewidth** (*numeric*) – the width of the polygon edge lines
- **alpha** (*numeric*) – the opacity
- **figsize** (*tuple*) – the size of the plotting figure
- **margin** (*numeric*) – the size of the figure margins
- **axis_off** (*bool*) – if True, disable the matplotlib axes display

Returns **fig, ax** – matplotlib figure, axis

Return type tuple

1.11 osmnx.pois module

Download points of interests (POIs) from OpenStreetMap.

osmnx.pois.**pois_from_address**(*address*, *tags*, *dist=1000*, *timeout=None*, *memory=None*, *custom_settings=None*)

Get point of interests (POIs) within some distance N, S, E, W of address.

Parameters

- **address** (*string*) – the address to geocode to a lat-lng point

- **tags** (*dict*) – Dict of tags used for finding POIs from the selected area. Results returned are the union, not intersection of each individual tag. Each result matches at least one tag given. The dict keys should be OSM tags, (e.g., *amenity*, *landuse*, *highway*, etc) and the dict values should be either *True* to retrieve all items with the given tag, or a string to get a single tag-value combination, or a list of strings to get multiple values for the given tag. For example, *tags = {'amenity':True, 'landuse':['retail','commercial'], 'highway':'bus_stop'}* would return all amenities, landuse=retail, landuse=commercial, and highway=bus_stop.
- **dist** (*numeric*) – distance in meters
- **timeout** (*None*) – deprecated, use ox.config(timeout=value) instead to configure this setting via the settings module
- **memory** (*None*) – deprecated, use ox.config(memory=value) instead to configure this setting via the settings module
- **custom_settings** (*None*) – deprecated, use ox.config(custom_settings=value) instead to configure this setting via the settings module

Returns gdf

Return type geopandas.GeoDataFrame

osmnx.pois.**pois_from_place**(*place*, *tags*, *which_result=1*, *timeout=None*, *memory=None*, *custom_settings=None*)

Get points of interest (POIs) within the boundaries of some place.

Parameters

- **place** (*string*) – the query to geocode to get boundary polygon
- **tags** (*dict*) – Dict of tags used for finding POIs from the selected area. Results returned are the union, not intersection of each individual tag. Each result matches at least one tag given. The dict keys should be OSM tags, (e.g., *amenity*, *landuse*, *highway*, etc) and the dict values should be either *True* to retrieve all items with the given tag, or a string to get a single tag-value combination, or a list of strings to get multiple values for the given tag. For example, *tags = {'amenity':True, 'landuse':['retail','commercial'], 'highway':'bus_stop'}* would return all amenities, landuse=retail, landuse=commercial, and highway=bus_stop.
- **which_result** (*int*) – max number of geocoding results to return and which to process
- **timeout** (*None*) – deprecated, use ox.config(timeout=value) instead to configure this setting via the settings module
- **memory** (*None*) – deprecated, use ox.config(memory=value) instead to configure this setting via the settings module
- **custom_settings** (*None*) – deprecated, use ox.config(custom_settings=value) instead to configure this setting via the settings module

Returns gdf

Return type geopandas.GeoDataFrame

osmnx.pois.**pois_from_point**(*point*, *tags*, *dist=1000*, *timeout=None*, *memory=None*, *custom_settings=None*)

Get point of interests (POIs) within some distance N, S, E, W of a point.

Parameters

- **point** (*tuple*) – a (lat, lng) point
- **tags** (*dict*) – Dict of tags used for finding POIs from the selected area. Results returned are the union, not intersection of each individual tag. Each result matches at least one tag

given. The dict keys should be OSM tags, (e.g., *amenity*, *landuse*, *highway*, etc) and the dict values should be either *True* to retrieve all items with the given tag, or a string to get a single tag-value combination, or a list of strings to get multiple values for the given tag. For example, *tags = {'amenity':True, 'landuse':['retail','commercial'], 'highway':'bus_stop'}* would return all amenities, landuse=retail, landuse=commercial, and highway=bus_stop.

- **dist** (*numeric*) – distance in meters
- **timeout** (*None*) – deprecated, use ox.config(timeout=value) instead to configure this setting via the settings module
- **memory** (*None*) – deprecated, use ox.config(memory=value) instead to configure this setting via the settings module
- **custom_settings** (*None*) – deprecated, use ox.config(custom_settings=value) instead to configure this setting via the settings module

Returns gdf

Return type geopandas.GeoDataFrame

osmnx.pois.**pois_from_polygon**(*polygon*, *tags*, *timeout=None*, *memory=None*, *custom_settings=None*)

Get point of interests (POIs) within some polygon.

Parameters

- **polygon** (*shapely.geometry.Polygon*) – geographic boundaries to fetch POIs within
- **tags** (*dict*) – Dict of tags used for finding POIs from the selected area. Results returned are the union, not intersection of each individual tag. Each result matches at least one tag given. The dict keys should be OSM tags, (e.g., *amenity*, *landuse*, *highway*, etc) and the dict values should be either *True* to retrieve all items with the given tag, or a string to get a single tag-value combination, or a list of strings to get multiple values for the given tag. For example, *tags = {'amenity':True, 'landuse':['retail','commercial'], 'highway':'bus_stop'}* would return all amenities, landuse=retail, landuse=commercial, and highway=bus_stop.
- **timeout** (*None*) – deprecated, use ox.config(timeout=value) instead to configure this setting via the settings module
- **memory** (*None*) – deprecated, use ox.config(memory=value) instead to configure this setting via the settings module
- **custom_settings** (*None*) – deprecated, use ox.config(custom_settings=value) instead to configure this setting via the settings module

Returns gdf

Return type geopandas.GeoDataFrame

1.12 osmnx.projection module

Project spatial geometries and street networks.

osmnx.projection.**project_gdf**(*gdf*, *to_crs=None*, *to_latlong=False*)
Project a GeoDataFrame from its current CRS to another.

If to_crs is None, project to the UTM CRS for the UTM zone in which the GeoDataFrame's centroid lies. Otherwise project to the CRS defined by to_crs. The simple UTM zone calculation in this function works well for most latitudes, but may not work for some extreme northern locations like Svalbard or far northern Norway.

Parameters

- **gdf** (*geopandas.GeoDataFrame*) – the GeoDataFrame to be projected
- **to_crs** (*dict or string or pyproj.CRS*) – if None, project to UTM zone in which gdf's centroid lies, otherwise project to this CRS
- **to_latlong** (*bool*) – if True, project to settings.default_crs and ignore to_crs

Returns **gdf_proj** – the projected GeoDataFrame

Return type geopandas.GeoDataFrame

osmnx.projection.**project_geometry**(*geometry*, *crs=None*, *to_crs=None*, *to_latlong=False*)
Project a shapely geometry from its current CRS to another.

If to_crs is None, project to the UTM CRS for the UTM zone in which the geometry's centroid lies. Otherwise project to the CRS defined by to_crs.

Parameters

- **geometry** (*shapely.geometry.Polygon or shapely.geometry.MultiPolygon*) – the geometry to project
- **crs** (*dict or string or pyproj.CRS*) – the starting CRS of the passed-in geometry. if None, it will be set to settings.default_crs
- **to_crs** (*dict or string or pyproj.CRS*) – if None, project to UTM zone in which geometry's centroid lies, otherwise project to this CRS
- **to_latlong** (*bool*) – if True, project to settings.default_crs and ignore to_crs

Returns **geometry_proj, crs** – the projected geometry and its new CRS

Return type tuple

osmnx.projection.**project_graph**(*G*, *to_crs=None*)
Project graph from its current CRS to another.

If to_crs is None, project the graph to the UTM CRS for the UTM zone in which the graph's centroid lies. Otherwise, project the graph to the CRS defined by to_crs.

Parameters

- **G** (*networkx.MultiDiGraph*) – the graph to be projected
- **to_crs** (*dict or string or pyproj.CRS*) – if None, project graph to UTM zone in which graph centroid lies, otherwise project graph to this CRS

Returns **G_proj** – the projected graph

Return type networkx.MultiDiGraph

1.13 osmnx.settings module

Global settings, can be configured by user with utils.config().

1.14 osmnx.simplification module

Simplify, correct, and consolidate network topology.

osmnx.simplification.**consolidate_intersections**(*G*, *tolerance=10*, *rebuild_graph=True*, *dead_ends=False*, *update_edge_lengths=True*)

Consolidate intersections comprising clusters of nearby nodes.

Merging nodes and return either their centroids or a rebuilt graph with consolidated intersections and reconnected edge geometries.

The tolerance argument should be adjusted to approximately match street design standards in the specific street network, and you should always use a projected graph to work in meaningful and consistent units like meters.

Divided roads are often represented by separate centerline edges. The intersection of two divided roads thus creates 4 nodes, representing where each edge intersects a perpendicular edge. These 4 nodes represent a single intersection in the real world. This function consolidates them up by buffering them to an arbitrary distance, merging overlapping buffers, and taking their centroid. For best results, the tolerance argument should be adjusted to approximately match street design standards in the specific street network.

Parameters

- **G** (*networkx.MultiDiGraph*) – a projected graph
- **tolerance** (*float*) – nodes within this distance (in graph's geometry's units) will be dissolved into a single intersection
- **rebuild_graph** (*bool*) – if True, use consolidate_intersections_rebuild_graph to consolidate the intersections and rebuild the graph, then return as networkx.MultiDiGraph. if False, just return the consolidated intersection points as a geopandas.GeoSeries (faster than rebuilding graph)
- **dead_ends** (*bool*) – if False, discard dead-end nodes to return only street-intersection points
- **update_edge_lengths** (*bool*) – just passed to consolidate_intersections_rebuild_graph. if True, update the length attribute of edges reconnected to a new merged node; if False, just retain the original edge length.

Returns if rebuild_graph=True, returns MultiDiGraph with consolidated intersections and reconnected edge geometries. if rebuild_graph=False, returns GeoSeries of shapely Points representing the centroids of street intersections

Return type networkx.MultiDiGraph or geopandas.GeoSeries

osmnx.simplification.**simplify_graph**(*G*, *strict=True*, *remove_rings=True*)

Simplify a graph's topology by removing interstitial nodes.

Simplify graph topology by removing all nodes that are not intersections or dead-ends. Create an edge directly between the end points that encapsulate them, but retain the geometry of the original edges, saved as attribute in new edge.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph

- **strict** (*bool*) – if False, allow nodes to be end points even if they fail all other rules but have incident edges with different OSM IDs. Lets you keep nodes at elbow two-way intersections, but sometimes individual blocks have multiple OSM IDs within them too.
- **remove_rings** (*bool*) – if True, remove isolated self-contained rings that have no endpoints

Returns **G** – topologically simplified graph

Return type networkx.MultiDiGraph

1.15 osmnx.speed module

Calculate graph edge speeds and travel times.

osmnx.speed.**add_edge_speeds**(*G*, *hwy_speeds=None*, *fallback=None*, *precision=1*)

Add edge speeds (km per hour) to graph as new *speed_kph* edge attributes.

Imputes free-flow travel speeds for all edges based on mean *maxspeed* value of edges, per highway type. For highway types in graph that have no *maxspeed* value on any edge, function assigns the mean of all *maxspeed* values in graph.

This mean-imputation can obviously be imprecise, and the caller can override it by passing in *hwy_speeds* and/or *fallback* arguments that correspond to local speed limit standards.

If edge *maxspeed* attribute has "mph" in it, value will automatically be converted from miles per hour to km per hour. Any other speed units should be manually converted to km per hour prior to running this function, otherwise there could be unexpected results. If "mph" does not appear in the edge's maxspeed attribute string, then function assumes kph, per OSM guidelines: https://wiki.openstreetmap.org/wiki/Map_Features/Units

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **hwy_speeds** (*dict*) – dict keys = OSM highway types and values = typical speeds (km per hour) to assign to edges of that highway type for any edges missing speed data. Any edges with highway type not in *hwy_speeds* will be assigned the mean preexisting speed value of all edges of that highway type.
- **fallback** (*numeric*) – default speed value (km per hour) to assign to edges whose highway type did not appear in *hwy_speeds* and had no preexisting speed values on any edge
- **precision** (*int*) – decimal precision to round speed_kph

Returns **G** – graph with speed_kph attributes on all edges

Return type networkx.MultiDiGraph

osmnx.speed.**add_edge_travel_times**(*G*, *precision=1*)

Add edge travel time (seconds) to graph as new *travel_time* edge attributes.

Calculates free-flow travel time along each edge, based on *length* and *speed_kph* attributes. Note: run *add_edge_speeds* first to generate the *speed_kph* attribute. All edges must have *length* and *speed_kph* attributes and all their values must be non-null.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **precision** (*int*) – decimal precision to round travel_time

Returns **G** – graph with travel_time attributes on all edges

Return type networkx.MultiDiGraph

1.16 osmnx.stats module

Calculate graph-theoretic network measures.

osmnx.stats.**basic_stats**(*G*, *area=None*, *clean_intersects=False*, *tolerance=15*, *circuity_dist='gc'*)

Calculate basic descriptive metric and topological stats for a graph.

For an unprojected lat-lng graph, tolerance and graph units should be in degrees, and circuity_dist should be 'gc'. For a projected graph, tolerance and graph units should be in meters (or similar) and circuity_dist should be 'euclidean'.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **area** (*numeric*) – the area covered by the street network, in square meters (typically land area); if none, will skip all density-based metrics
- **clean_intersects** (*bool*) – if True, calculate consolidated intersections count (and density, if area is provided) via consolidate_intersections function
- **tolerance** (*numeric*) – tolerance value passed along if clean_intersects=True, see consolidate_intersections function documentation for details and usage
- **circuity_dist** (*string*) – 'gc' or 'euclidean', how to calculate straight-line distances for circuity measurement; use former for lat-lng networks and latter for projected networks

Returns

stats – dictionary of network measures containing the following elements (some keys may not be present, based on the arguments passed into the function):

- n = number of nodes in the graph
- m = number of edges in the graph
- k_avg = average node degree of the graph
- **intersection_count = number of intersections in graph, that is,** nodes with >1 street emanating from them
- **streets_per_node_avg = how many streets (edges in the undirected** representation of the graph) emanate from each node (ie, intersection or dead-end) on average (mean)
- **streets_per_node_counts = dict, with keys of number of streets** emanating from the node, and values of number of nodes with this count
- **streets_per_node_proportion = dict, same as previous, but as a** proportion of the total, rather than counts
- edge_length_total = sum of all edge lengths in the graph, in meters
- edge_length_avg = mean edge length in the graph, in meters
- **street_length_total = sum of all edges in the undirected** representation of the graph
- **street_length_avg = mean edge length in the undirected** representation of the graph, in meters
- **street_segments_count = number of edges in the undirected** representation of the graph

- node_density_km = n divided by area in square kilometers
- **intersection_density_km = intersection_count divided by area in** square kilometers
- **edge_density_km = edge_length_total divided by area in square** kilometers
- **street_density_km = street_length_total divided by area in square** kilometers
- **circuity_avg = edge_length_total divided by the sum of the great** circle distances between the nodes of each edge
- **self_loop_proportion = proportion of edges that have a single node** as its two endpoints (ie, the edge links nodes u and v, and u==v)
- **clean_intersection_count = number of intersections in street** network, merging complex ones into single points
- **clean_intersection_density_km = clean_intersection_count divided by** area in square kilometers

Return type dict

osmnx.stats.**extended_stats**(*G*, *connectivity=False*, *anc=False*, *ecc=False*, *bc=False*, *cc=False*)

Calculate extended topological stats and metrics for a graph.

Many of these algorithms have an inherently high time complexity. Global topological analysis of large complex networks is extremely time consuming and may exhaust computer memory. Consider using function arguments to not run metrics that require computation of a full matrix of paths if they will not be needed.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **connectivity** (*bool*) – if True, calculate node and edge connectivity
- **anc** (*bool*) – if True, calculate average node connectivity
- **ecc** (*bool*) – if True, calculate shortest paths, eccentricity, and topological metrics that use eccentricity
- **bc** (*bool*) – if True, calculate node betweenness centrality
- **cc** (*bool*) – if True, calculate node closeness centrality

Returns

stats – dictionary of network measures containing the following elements (some only calculated/returned optionally, based on passed parameters):

- avg_neighbor_degree
- avg_neighbor_degree_avg
- avg_weighted_neighbor_degree
- avg_weighted_neighbor_degree_avg
- degree_centrality
- degree_centrality_avg
- clustering_coefficient
- clustering_coefficient_avg
- clustering_coefficient_weighted
- clustering_coefficient_weighted_avg

- pagerank
- pagerank_max_node
- pagerank_max
- pagerank_min_node
- pagerank_min
- node_connectivity
- node_connectivity_avg
- edge_connectivity
- eccentricity
- diameter
- radius
- center
- periphery
- closeness_centrality
- closeness_centrality_avg
- betweenness_centrality
- betweenness_centrality_avg

Return type dict

1.17 osmnx.truncate module

Truncate graph by distance, bounding box, or polygon.

osmnx.truncate.**truncate_graph_bbox**(*G*, *north*, *south*, *east*, *west*, *truncate_by_edge=False*, *retain_all=False*, *quadrat_width=0.05*, *min_num=3*)

Remove every node in graph that falls outside a bounding box.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **north** (*float*) – northern latitude of bounding box
- **south** (*float*) – southern latitude of bounding box
- **east** (*float*) – eastern longitude of bounding box
- **west** (*float*) – western longitude of bounding box
- **truncate_by_edge** (*bool*) – if True retain node if it's outside bbox but at least one of node's neighbors are within bbox
- **retain_all** (*bool*) – if True, return the entire graph even if it is not connected
- **quadrat_width** (*numeric*) – passed on to intersect_index_quadrats: the linear length (in degrees) of the quadrats with which to cut up the geometry (default = 0.05, approx 4km at NYC's latitude)

- **min_num** (*int*) – passed on to intersect_index_quadrats: the minimum number of linear quadrat lines (e.g., min_num=3 would produce a quadrat grid of 4 squares)

Returns **G** – the truncated graph

Return type networkx.MultiDiGraph

osmnx.truncate.**truncate_graph_dist**(*G*, *source_node*, *max_dist=1000*, *weight='length'*, *retain_all=False*)

Remove every node farther than some network distance from source_node.

This function can be slow for large graphs, as it must calculate shortest path distances between source_node and every other graph node.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **source_node** (*int*) – the node in the graph from which to measure network distances to other nodes
- **max_dist** (*int*) – remove every node in the graph greater than this distance from the source_node (along the network)
- **weight** (*string*) – how to weight the graph when measuring distance (default 'length' is how many meters long the edge is)
- **retain_all** (*bool*) – if True, return the entire graph even if it is not connected

Returns **G** – the truncated graph

Return type networkx.MultiDiGraph

osmnx.truncate.**truncate_graph_polygon**(*G*, *polygon*, *retain_all=False*, *truncate_by_edge=False*, *quadrat_width=0.05*, *min_num=3*)

Remove every node in graph that falls outside a shapely (Multi)Polygon.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **polygon** (*shapely.geometry.Polygon or shapely.geometry.MultiPolygon*) – only retain nodes in graph that lie within this geometry
- **retain_all** (*bool*) – if True, return the entire graph even if it is not connected
- **truncate_by_edge** (*bool*) – if True retain node if it's outside polygon but at least one of node's neighbors are within polygon
- **quadrat_width** (*numeric*) – passed on to intersect_index_quadrats: the linear length (in degrees) of the quadrats with which to cut up the geometry (default = 0.05, approx 4km at NYC's latitude)
- **min_num** (*int*) – passed on to intersect_index_quadrats: the minimum number of linear quadrat lines (e.g., min_num=3 would produce a quadrat grid of 4 squares)

Returns **G** – the truncated graph

Return type networkx.MultiDiGraph

1.18 osmnx.utils module

General utility functions.

osmnx.utils.**citation**()
Print the OSMnx package's citation information.

Boeing, G. 2017. OSMnx: New Methods for Acquiring, Constructing, Analyzing, and Visualizing Complex Street Networks. Computers, Environment and Urban Systems, 65(126-139). https://doi.org/10.1016/j.compenvurbsys.2017.05.004

Returns

Return type None

osmnx.utils.**config**(*data_folder='data', logs_folder='logs', imgs_folder='images', cache_folder='cache', use_cache=False, log_file=False, log_console=False, log_level=20, log_name='osmnx', log_filename='osmnx', useful_tags_node=['ref', 'highway'], useful_tags_way=['bridge', 'tunnel', 'oneway', 'lanes', 'ref', 'name', 'highway', 'maxspeed', 'service', 'access', 'area', 'landuse', 'width', 'est_width', 'junction'], osm_xml_node_attrs=['id', 'timestamp', 'uid', 'user', 'version', 'changeset', 'lat', 'lon'], osm_xml_node_tags=['highway'], osm_xml_way_attrs=['id', 'timestamp', 'uid', 'user', 'version', 'changeset'], osm_xml_way_tags=['highway', 'lanes', 'maxspeed', 'name', 'oneway'], overpass_settings='[out:json][timeout:{timeout}]{maxsize}', timeout=180, memory=None, max_query_area_size=2500000000, default_access='["access"!~"private"]', default_crs='epsg:4326', default_user_agent='OSMnx Python package (https://github.com/gboeing/osmnx)', default_referer='OSMnx Python package (https://github.com/gboeing/osmnx)', default_accept_language='en', nominatim_endpoint='https://nominatim.openstreetmap.org/', nominatim_key=None, overpass_endpoint='http://overpass-api.de/api', all_oneway=False*)
Configure OSMnx by setting the default global settings' values.

Any parameters not passed by the caller are set to their original default values.

Parameters

- **data_folder** (*string*) – where to save/load data files by default
- **logs_folder** (*string*) – where to save log files
- **imgs_folder** (*string*) – where to save figures by default
- **cache_folder** (*string*) – where to save HTTP response cache
- **use_cache** (*bool*) – if True, cache HTTP responses locally instead of calling API repetitively for the same request
- **log_file** (*bool*) – if True, save log output to a file in logs_folder
- **log_console** (*bool*) – if True, print log output to the console (terminal window)
- **log_level** (*int*) – one of the logger.level constants
- **log_name** (*string*) – name of the logger
- **log_filename** (*string*) – name of the log file
- **useful_tags_node** (*list*) – OSM "node" tags to add as graph node attributes, when present

- **useful_tags_way** (*list*) – OSM "way" tags to add as graph edge attributes, when present
- **osm_xml_node_attrs** (*list*) – node attributes for saving .osm XML files with save_graph_xml function
- **osm_xml_node_tags** (*list*) – node tags for saving .osm XML files with save_graph_xml function
- **osm_xml_way_attrs** (*list*) – edge attributes for saving .osm XML files with save_graph_xml function
- **osm_xml_way_tags** (*list*) – edge tags for for saving .osm XML files with save_graph_xml function
- **overpass_settings** (*string*) – Settings string for overpass queries. For example, to query historical OSM data as of a certain date: *'[out:json][timeout:90][date:"2019-10-28T19:20:00Z"]'*. Use with caution.
- **timeout** (*int*) – the timeout interval for the HTTP request and for API to use while running the query
- **memory** (*int*) – Overpass server memory allocation size for the query, in bytes. If None, server will use its default allocation size. Use with caution.
- **max_query_area_size** (*int*) – maximum area for any part of the geometry in meters: any polygon bigger than this will get divided up for multiple queries to API (default 50km x 50km)
- **default_access** (*string*) – default filter for OSM "access" key
- **default_crs** (*string*) – default coordinate reference system to set when creating graphs
- **default_user_agent** (*string*) – HTTP header user-agent
- **default_referer** (*string*) – HTTP header referer
- **default_accept_language** (*string*) – HTTP header accept-language
- **nominatim_endpoint** (*string*) – the API endpoint to use for nominatim queries
- **nominatim_key** (*string*) – your API key, if you are using an endpoint that requires one
- **overpass_endpoint** (*string*) – the API endpoint to use for overpass queries
- **all_oneway** (*boolean*) – if True, forces all ways to be loaded as oneway ways, preserving the original order of nodes stored in the OSM way XML. Only use if specifically saving to .osm XML file with save_graph_xml function.

Returns

Return type None

osmnx.utils.**log**(*message*, *level=None*, *name=None*, *filename=None*)

Write a message to the logger.

This logs to file and/or prints to the console, depending on the current configuration of settings.log_file and settings.log_console.

Parameters

- **message** (*string*) – the message to log
- **level** (*int*) – one of the logger.level constants

- **name** (*string*) – name of the logger
- **filename** (*string*) – name of the log file

Returns

Return type None

osmnx.utils.**ts**(*style='datetime'*, *template=None*)
Get current timestamp as string.

Parameters

- **style** (*string*) – format the timestamp with this built-in template. must be one of {'datetime', 'date', 'time'}
- **template** (*string*) – if not None, format the timestamp with this template instead of one of the built-in styles

Returns **ts** – the string timestamp

Return type string

1.19 osmnx.utils_geo module

Geospatial utility functions.

osmnx.utils_geo.**bbox_from_point**(*point*, *dist=1000*, *project_utm=False*, *return_crs=False*)
Create a bounding box from a point.

Create a bounding box some distance in each direction (north, south, east, and west) from some (lat, lng) point.

Parameters

- **point** (*tuple*) – the (lat, lng) point to create the bounding box around
- **dist** (*int*) – how many meters the north, south, east, and west sides of the box should each be from the point
- **project_utm** (*bool*) – if True return bbox as UTM coordinates
- **return_crs** (*bool*) – if True and project_utm=True, return the projected CRS

Returns (north, south, east, west) if return_crs=False or (north, south, east, west, crs_proj) if return_crs=True

Return type tuple

osmnx.utils_geo.**bbox_to_poly**(*north*, *south*, *east*, *west*)
Convert bounding box coordinates to shapely Polygon.

Parameters

- **north** (*float*) – northern coordinate
- **south** (*float*) – southern coordinate
- **east** (*float*) – eastern coordinate
- **west** (*float*) – western coordinate

Returns

Return type shapely.geometry.Polygon

osmnx.utils_geo.**geocode**(*query*)
Geocode a query string to (lat, lng) with the Nominatim geocoder.

Parameters **query** (*string*) – the query string to geocode

Returns **point** – the (lat, lng) coordinates returned by the geocoder

Return type tuple

osmnx.utils_geo.**redistribute_vertices**(*geom*, *dist*)
Redistribute the vertices on a projected LineString or MultiLineString.

The distance argument is only approximate since the total distance of the linestring may not be a multiple of the preferred distance. This function works on only (Multi)LineString geometry types.

Parameters

- **geom** (*shapely.geometry.LineString or shapely.geometry.MultiLineString*) – a Shapely geometry (should be projected)
- **dist** (*float*) – spacing length along edges. Units are same as the geom: degrees for unprojected geometries and meters for projected geometries. The smaller the dist value, the more points are created.

Returns the redistributed vertices as a list if geom is a LineString or MultiLineString if geom is a MultiLineString

Return type list or shapely.geometry.MultiLineString

osmnx.utils_geo.**round_geometry_coords**(*shape*, *precision*)
Round the coordinates of a shapely geometry to some decimal precision.

Parameters

- **shape** (*shapely.geometry.geometry, either Point, MultiPoint, LineString,*) – MultiLineString, Polygon, or MultiPolygon the geometry to round the coordinates of
- **precision** (*int*) – decimal precision to round coordinates to

Returns

Return type shapely.geometry.geometry

1.20 osmnx.utils_graph module

Graph utility functions.

osmnx.utils_graph.**add_edge_lengths**(*G*, *precision=3*)
Add *length* (meters) attribute to each edge.

Calculate via great circle distance between nodes u and v.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **precision** (*int*) – decimal precision to round lengths

Returns **G** – graph with edge length attributes

Return type networkx.MultiDiGraph

osmnx.utils_graph.**count_streets_per_node**(*G*, *nodes=None*)
Count how many street segments emanate from each node in this graph.

If nodes is passed, then only count the nodes in the graph with those IDs.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **nodes** (*iterable*) – the set of node IDs to get counts for

Returns **streets_per_node** – counts of how many streets emanate from each node with keys=node id and values=count

Return type dict

osmnx.utils_graph.**get_largest_component**(*G*, *strongly=False*)
Get subgraph of MultiDiGraph's largest weakly/strongly connected component.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **strongly** (*bool*) – if True, return the largest strongly instead of weakly connected component

Returns **G** – the largest connected component subgraph from the original graph

Return type networkx.MultiDiGraph

osmnx.utils_graph.**get_route_edge_attributes**(*G*, *route*, *attribute=None*, *minimize_key='length'*, *retrieve_default=None*)
Get a list of attribute values for each edge in a path.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **route** (*list*) – list of nodes in the path
- **attribute** (*string*) – the name of the attribute to get the value of for each edge. If not specified, the complete data dict is returned for each edge.
- **minimize_key** (*string*) – if there are parallel edges between two nodes, select the one with the lowest value of minimize_key
- **retrieve_default** (*Callable[Tuple[Any, Any], Any]*) – Function called with the edge nodes as parameters to retrieve a default value, if the edge does not contain the given attribute. Per default, a *KeyError* is raised

Returns **attribute_values** – list of edge attribute values

Return type list

osmnx.utils_graph.**get_undirected**(*G*)
Convert MultiDiGraph to MultiGraph.

Maintains parallel edges if their geometries differ.

Parameters **G** (*networkx.MultiDiGraph*) – input graph

Returns **H**

Return type networkx.MultiGraph

osmnx.utils_graph.**graph_from_gdfs**(*gdf_nodes*, *gdf_edges*, *graph_attrs=None*)
Convert node and edge GeoDataFrames to a MultiDiGraph.

This function is the inverse of graph_to_gdfs.

Parameters

- **gdf_nodes** (*geopandas.GeoDataFrame*) – GeoDataFrame of graph nodes
- **gdf_edges** (*geopandas.GeoDataFrame*) – GeoDataFrame of graph edges
- **graph_attrs** (*dict*) – the new G.graph attribute dict; if None, add crs as the only graph-level attribute

Returns G

Return type networkx.MultiDiGraph

osmnx.utils_graph.**graph_to_gdfs**(*G*, *nodes=True*, *edges=True*, *node_geometry=True*, *fill_edge_geometry=True*)
Convert a graph to node and/or edge GeoDataFrames.

This function is the inverse of graph_from_gdfs.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **nodes** (*bool*) – if True, convert graph nodes to a GeoDataFrame and return it
- **edges** (*bool*) – if True, convert graph edges to a GeoDataFrame and return it
- **node_geometry** (*bool*) – if True, create a geometry column from node x and y data
- **fill_edge_geometry** (*bool*) – if True, fill in missing edge geometry fields using nodes u and v

Returns gdf_nodes or gdf_edges or tuple of (gdf_nodes, gdf_edges)

Return type geopandas.GeoDataFrame or tuple

osmnx.utils_graph.**induce_subgraph**(*G*, *node_subset*)
Induce a subgraph of G.

Parameters

- **G** (*networkx.MultiDiGraph*) – input graph
- **node_subset** (*list-like*) – the subset of nodes to induce a subgraph of G

Returns **H** – the subgraph of G induced by node_subset

Return type networkx.MultiDiGraph

osmnx.utils_graph.**remove_isolated_nodes**(*G*)
Remove from a graph all nodes that have no incident edges.

Parameters **G** (*networkx.MultiDiGraph*) – graph from which to remove nodes

Returns **G** – graph with all isolated nodes removed

Return type networkx.MultiDiGraph

OSMnx: retrieve, model, analyze, and visualize street networks from OpenStreetMap. OSMnx is a Python package that lets you download spatial geometries and model, project, visualize, and analyze real-world street networks from OpenStreetMap's APIs. Users can download and model walkable, drivable, or bikeable urban networks with a single line of Python code, and then easily analyze and visualize them. You can just as easily download and work with

amenities/points of interest, building footprints, elevation data, street bearings/orientations, speed/travel time, and network routing.

CHAPTER

TWO

CITATION INFO

If you use OSMnx in your work, please cite the journal article:

Boeing, G. 2017. "OSMnx: New Methods for Acquiring, Constructing, Analyzing, and Visualizing Complex Street Networks." *Computers, Environment and Urban Systems* 65, 126-139. doi:10.1016/j.compenvurbsys.2017.05.004

CHAPTER

THREE

FEATURES

OSMnx is built on top of geopandas, networkx, and matplotlib and interacts with OpenStreetMap's APIs to:

- Download street networks anywhere in the world with a single line of code
- Download other infrastructure types, place boundaries, building footprints, and points of interest
- Download by city name, polygon, bounding box, or point/address + network distance
- Download drivable, walkable, bikeable, or all street networks
- Download node elevations and calculate edge grades (inclines)
- Impute missing speeds and calculate graph edge travel times
- Simplify and correct the network's topology to clean-up nodes and consolidate intersections
- Fast map-matching of points, routes, or trajectories to nearest graph edges or nodes
- Save networks to disk as shapefiles, geopackages, and GraphML
- Save/load street network to/from a local .osm xml file
- Conduct topological and spatial analyses to automatically calculate dozens of indicators
- Calculate and visualize street bearings and orientations
- Calculate and visualize shortest-path routes that minimize distance, travel time, elevation, etc
- Visualize street network as a static map or interactive leaflet web map
- Visualize travel distance and travel time with isoline and isochrone maps
- Plot figure-ground diagrams of street networks and/or building footprints

Examples and demonstrations of these features are in the GitHub repo (see below). More feature development details are in the change log.

CHAPTER

FOUR

INSTALLATION

You can install OSMnx with conda:

```
conda config --prepend channels conda-forge
conda create -n ox --strict-channel-priority osmnx
```

Alternatively, you can run OSMnx + Jupyter directly from its official docker container, or you can install OSMnx via pip if you already have all of its dependencies installed on your system.

CHAPTER

FIVE

EXAMPLES

For code and usage examples/demos, see the examples GitHub repo.

CHAPTER

SIX

SUPPORT

If you've discovered a bug in OSMnx, please open an issue at the OSMnx GitHub repo documenting what is broken in the package. Alternatively, if you have a usage question, please ask it on StackOverflow.

CHAPTER

SEVEN

LICENSE

The project is licensed under the MIT license.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

O

INDEX