OSMnx Documentation

Release

Geoff Boeing

Jun 25, 2017

Contents:

CHAPTER 1

osmnx package

Submodules

osmnx.buildings module

osmnx.buildings.**buildings_from_address**(*address*, *distance*, *retain_invalid=False*)
Get building footprints within some distance north, south, east, and west of an address.

Parameters

- **address** (*string*) – the address to geocode to a lat-long point
- **distance** (*numeric*) – distance in meters
- **retain_invalid** (*bool*) – if False discard any building footprints with an invalid geometry

Returns

Return type GeoDataFrame

osmnx.buildings.**buildings_from_place**(*place*, *retain_invalid=False*)
Get building footprints within the boundaries of some place.

Parameters

- **place** (*string*) – the query to geocode to get geojson boundary polygon
- **retain_invalid** (*bool*) – if False discard any building footprints with an invalid geometry

Returns

Return type GeoDataFrame

osmnx.buildings.**buildings_from_point**(*point*, *distance*, *retain_invalid=False*)
Get building footprints within some distance north, south, east, and west of a lat-long point.

Parameters

- **point** (*tuple*) – a lat-long point
- **distance** (*numeric*) – distance in meters
- **retain_invalid** (*bool*) – if False discard any building footprints with an invalid geometry

Returns

Return type GeoDataFrame

osmnx.buildings.**buildings_from_polygon**(*polygon*, *retain_invalid=False*)

Get building footprints within some polygon.

Parameters

- **polygon** (*Polygon*) –
- **retain_invalid** (*bool*) – if False discard any building footprints with an invalid geometry

Returns

Return type GeoDataFrame

osmnx.buildings.**create_buildings_gdf**(*polygon=None*, *north=None*, *south=None*, *east=None*, *west=None*, *retain_invalid=False*)

Get building footprint data from OSM then assemble it into a GeoDataFrame.

Parameters

- **polygon** (*shapely Polygon or MultiPolygon*) – geographic shape to fetch the building footprints within
- **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
- **retain_invalid** (*bool*) – if False discard any building footprints with an invalid geometry

Returns

Return type GeoDataFrame

osmnx.buildings.**osm_bldg_download**(*polygon=None*, *north=None*, *south=None*, *east=None*, *west=None*, *timeout=180*, *memory=None*, *max_query_area_size=2500000000*)

Download OpenStreetMap building footprint data.

Parameters

- **polygon** (*shapely Polygon or MultiPolygon*) – geographic shape to fetch the building footprints within
- **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

- **timeout** (*int*) – the timeout interval for requests and to pass to API
- **memory** (*int*) – server memory allocation size for the query, in bytes. If none, server will use its default allocation size
- **max_query_area_size** (*float*) – max area for any part of the geometry, in the units the geometry is in: any polygon bigger will get divided up for multiple queries to API (default is 50,000 * 50,000 units (ie, 50km x 50km in area, if units are meters))

Returns list of response_json dicts

Return type list

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

Plot a GeoDataFrame of building footprints.

Parameters

- **gdf** (*GeoDataFrame*) – building footprints
- **fig** (*figure*) –
- **ax** (*axis*) –
- **figsize** (*tuple*) –
- **color** (*string*) – the color of the building 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*) – the name of the file to save
- **file_format** (*string*) – the format of the file to save (e.g., 'jpg', 'png', 'svg')
- **dpi** (*int*) – the resolution of the image file if saving

Returns

Return type GeoDataFrame

osmnx.core module

osmnx.core.**add_edge_lengths**(*G*)

Add length (meters) attribute to each edge by great circle distance between nodes u and v.

Parameters **G** (*networkx multidigraph*) –

Returns **G**

Return type networkx multidigraph

osmnx.core.**add_path**(*G*, *data*, *one_way*)
Add a path to the graph.

Parameters

- **G** (*networkx multidigraph*) –
- **data** (*dict*) – the attributes of the path
- **one_way** (*bool*) – if this path is one-way or if it is bi-directional

Returns

Return type None

osmnx.core.**add_paths**(*G*, *paths*, *network_type*)
Add a collection of paths to the graph.

Parameters

- **G** (*networkx multidigraph*) –
- **paths** (*dict*) – the paths from OSM
- **network_type** (*string*) – {'all', 'walk', 'drive', etc}, what type of network

Returns

Return type None

osmnx.core.**bbox_from_point**(*point*, *distance=1000*, *project_utm=False*)
Create a bounding box some distance in each direction (north, south, east, and west) from some (lat, lng) point.

Parameters

- **point** (*tuple*) – the (lat, lon) point to create the bounding box around
- **distance** (*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

Returns **north, south, east, west**

Return type tuple

osmnx.core.**consolidate_subdivide_geometry**(*geometry*, *max_query_area_size*)
Consolidate a geometry into a convex hull, then subdivide it into smaller sub-polygons if its area exceeds max size (in geometry's units).

Parameters

- **geometry** (*shapely Polygon or MultiPolygon*) – the geometry to consolidate and subdivide
- **max_query_area_size** (*float*) – max area for any part of the geometry, in the units the geometry is in. any polygon bigger will get divided up for multiple queries to API (default is 50,000 * 50,000 units (ie, 50km x 50km in area, if units are meters))

Returns **geometry**

Return type Polygon or MultiPolygon

osmnx.core.**create_graph**(*response_jsons*, *name='unnamed'*, *retain_all=False*, *network_type='all_private'*)
Create a networkx graph from OSM data.

Parameters

- **response_jsons** (*list*) – list of dicts of JSON responses from from the Overpass API
- **name** (*string*) – the name of the graph
- **retain_all** (*bool*) – if True, return the entire graph even if it is not connected
- **network_type** (*string*) – what type of network to create

Returns

Return type networkx multidigraph

osmnx.core.**gdf_from_place**(*query*, *gdf_name=None*, *which_result=1*, *buffer_dist=None*)
Create a GeoDataFrame from a single place name query.

Parameters

- **query** (*string or dict*) – query string or structured query dict to geocode/download
- **gdf_name** (*string*) – name attribute metadata for GeoDataFrame (this is used to save shapefile later)
- **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

Return type GeoDataFrame

osmnx.core.**gdf_from_places**(*queries*, *gdf_name='unnamed'*, *buffer_dist=None*)
Create a GeoDataFrame from a list of place names to query.

Parameters

- **queries** (*list*) – list of query strings or structured query dicts to geocode/download, one at a time
- **gdf_name** (*string*) – name attribute metadata for GeoDataFrame (this is used to save shapefile later)
- **buffer_dist** (*float*) – distance to buffer around the place geometry, in meters

Returns

Return type GeoDataFrame

osmnx.core.**get_from_cache**(*url*)
Retrieve a HTTP response json object from the cache.

Parameters **url** (*string*) – the url of the request

Returns **response_json**

Return type dict

osmnx.core.**get_http_headers**(*user_agent=None*, *referer=None*)
Update the default requests HTTP headers with OSMnx info.

Parameters

- **user_agent** (*str*) – the user agent string, if None will set with OSMnx default
- **referer** (*str*) – the referer string, if None will set with OSMnx default

Returns **headers**

Return type dict

osmnx.core.**get_node**(*element*)
Convert an OSM node element into the format for a networkx node.

Parameters **element** (*dict*) – an OSM node element

Returns

Return type dict

osmnx.core.**get_osm_filter**(*network_type*)
Create a filter to query OSM for the specified network type.

Parameters **network_type** (*string*) – {'walk', 'bike', 'drive', 'drive_service', 'all', 'all_private', 'none'} what type of street or other network to get

Returns

Return type string

osmnx.core.**get_path**(*element*)
Convert an OSM way element into the format for a networkx graph path.

Parameters **element** (*dict*) – an OSM way element

Returns

Return type dict

osmnx.core.**get_pause_duration**(*recursive_delay=5*, *default_duration=10*)
Check the Overpass API status endpoint to determine how long to wait until next slot is available.

Parameters

- **recursive_delay** (*int*) – how long to wait between recursive calls if server is currently running a query
- **default_duration** (*int*) – if fatal error, function falls back on returning this value

Returns

Return type int

osmnx.core.**get_polygons_coordinates**(*geometry*)
Extract exterior coordinates from polygon(s) to pass to OSM in a query by polygon.

Parameters **geometry** (*shapely Polygon or MultiPolygon*) – the geometry to extract exterior coordinates from

Returns **polygon_coord_strs**

Return type list

osmnx.core.**graph_from_address**(*address*, *distance=1000*, *distance_type='bbox'*, *network_type='all_private'*, *simplify=True*, *retain_all=False*, *truncate_by_edge=False*, *return_coords=False*, *name='unnamed'*, *timeout=180*, *memory=None*, *max_query_area_size=2500000000*, *clean_periphery=True*, *infrastructure='way["highway"]'*)
Create a networkx graph from OSM data 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
- **distance** (*int*) – retain only those nodes within this many meters of the center of the graph

- **distance_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
- **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 bbox but at least one of node's neighbors are within bbox
- **return_coords** (*bool*) – optionally also return the geocoded coordinates of the address
- **name** (*string*) – the name of the graph
- **timeout** (*int*) – the timeout interval for requests and to pass to API
- **memory** (*int*) – server memory allocation size for the query, in bytes. If none, server will use its default allocation size
- **max_query_area_size** – float, max size for any part of the geometry, in square degrees: any polygon bigger will get divided up for multiple queries to API
- **clean_periphery** (*bool,*) – if True (and simplify=True), buffer 0.5km to get a graph larger than requested, then simplify, then truncate it to requested spatial extent
- **infrastructure** (*string*) – download infrastructure of given type (default is streets (ie, 'way["highway"]') but other infrastructures may be selected like power grids (ie, 'way["power"~"line"]'))

Returns multidigraph or optionally (multidigraph, tuple)

Return type networkx multidigraph or tuple

osmnx.core.**graph_from_bbox**(*north*, *south*, *east*, *west*, *network_type='all_private'*, *simplify=True*, *retain_all=False*, *truncate_by_edge=False*, *name='unnamed'*, *timeout=180*, *memory=None*, *max_query_area_size=2500000000*, *clean_periphery=True*, *infrastructure='way["highway"]'*)

Create a networkx graph from OSM data 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
- **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 bbox but at least one of node's neighbors are within bbox
- **name** (*string*) – the name of the graph
- **timeout** (*int*) – the timeout interval for requests and to pass to API
- **memory** (*int*) – server memory allocation size for the query, in bytes. If none, server will use its default allocation size

- **max_query_area_size** (*float*) – max size for any part of the geometry, in square degrees: any polygon bigger will get divided up for multiple queries to API
- **clean_periphery** (*bool*) – if True (and simplify=True), buffer 0.5km to get a graph larger than requested, then simplify, then truncate it to requested spatial extent
- **infrastructure** (*string*) – download infrastructure of given type (default is streets (ie, 'way["highway"]') but other infrastructures may be selected like power grids (ie, 'way["power"~"line"]'))

Returns

Return type networkx multidigraph

osmnx.core.**graph_from_place**(*query*, *network_type='all_private'*, *simplify=True*, *retain_all=False*, *truncate_by_edge=False*, *name='unnamed'*, *which_result=1*, *buffer_dist=None*, *timeout=180*, *memory=None*, *max_query_area_size=2500000000*, *clean_periphery=True*, *infrastructure='way["highway"]'*)

Create a networkx graph from OSM data within the spatial 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
- **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 bbox but at least one of node's neighbors are within bbox
- **name** (*string*) – the name of the graph
- **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
- **timeout** (*int*) – the timeout interval for requests and to pass to API
- **memory** (*int*) – server memory allocation size for the query, in bytes. If none, server will use its default allocation size
- **max_query_area_size** (*float*) – max size for any part of the geometry, in square degrees: any polygon bigger will get divided up for multiple queries to API
- **clean_periphery** (*bool*) – if True (and simplify=True), buffer 0.5km to get a graph larger than requested, then simplify, then truncate it to requested spatial extent
- **infrastructure** (*string*) – download infrastructure of given type (default is streets (ie, 'way["highway"]') but other infrastructures may be selected like power grids (ie, 'way["power"~"line"]'))

Returns

Return type networkx multidigraph

osmnx.core.**graph_from_point**(*center_point*, *distance=1000*, *distance_type='bbox'*, *network_type='all_private'*, *simplify=True*, *retain_all=False*, *truncate_by_edge=False*, *name='unnamed'*, *timeout=180*, *memory=None*, *max_query_area_size=2500000000*, *clean_periphery=True*, *infrastructure='way["highway"]'*)

Create a networkx graph from OSM data within some distance of some (lat, lon) center point.

Parameters

- **center_point** (*tuple*) – the (lat, lon) central point around which to construct the graph
- **distance** (*int*) – retain only those nodes within this many meters of the center of the graph
- **distance_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
- **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 bbox but at least one of node's neighbors are within bbox
- **name** (*string*) – the name of the graph
- **timeout** (*int*) – the timeout interval for requests and to pass to API
- **memory** (*int*) – server memory allocation size for the query, in bytes. If none, server will use its default allocation size
- **max_query_area_size** (*float*) – max size for any part of the geometry, in square degrees: any polygon bigger will get divided up for multiple queries to API
- **clean_periphery** (*bool,*) – if True (and simplify=True), buffer 0.5km to get a graph larger than requested, then simplify, then truncate it to requested spatial extent
- **infrastructure** (*string*) – download infrastructure of given type (default is streets (ie, 'way["highway"]') but other infrastructures may be selected like power grids (ie, 'way["power"~"line"]'))

Returns

Return type networkx multidigraph

osmnx.core.**graph_from_polygon**(*polygon*, *network_type='all_private'*, *simplify=True*, *retain_all=False*, *truncate_by_edge=False*, *name='unnamed'*, *timeout=180*, *memory=None*, *max_query_area_size=2500000000*, *clean_periphery=True*, *infrastructure='way["highway"]'*)

Create a networkx graph from OSM data within the spatial boundaries of the passed-in shapely polygon.

Parameters

- **polygon** (*shapely Polygon or MultiPolygon*) – the shape to get network data within
- **network_type** (*string*) – what type of street network to get
- **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 bbox but at least one of node's neighbors are within bbox
- **name** (*string*) – the name of the graph
- **timeout** (*int*) – the timeout interval for requests and to pass to API
- **memory** (*int*) – server memory allocation size for the query, in bytes. If none, server will use its default allocation size
- **max_query_area_size** (*float*) – max size for any part of the geometry, in square degrees: any polygon bigger will get divided up for multiple queries to API
- **clean_periphery** (*bool*) – if True (and simplify=True), buffer 0.5km to get a graph larger than requested, then simplify, then truncate it to requested spatial extent
- **infrastructure** (*string*) – download infrastructure of given type (default is streets (ie, 'way["highway"]') but other infrastructures may be selected like power grids (ie, 'way["power"~"line"]'))

Returns

Return type networkx multidigraph

osmnx.core.**intersect_index_quadrats**(*gdf*, *geometry*, *quadrat_width=0.025*, *min_num=3*, *buffer_amount=1e-09*)

Intersect points with a polygon, using an r-tree spatial index and cutting the polygon up into smaller sub-polygons for r-tree acceleration.

Parameters

- **gdf** (*GeoDataFrame*) – the set of points to intersect
- **geometry** (*shapely Polygon or MultiPolygon*) – the geometry to intersect with the points
- **quadrat_width** (*numeric*) – the linear length (in degrees) of the quadrats with which to cut up the geometry (default = 0.025, approx 2km at NYC's latitude)
- **min_num** (*int*) – the minimum number of linear quadrat lines (e.g., min_num=3 would produce a quadrat grid of 4 squares)
- **buffer_amount** (*numeric*) – buffer the quadrat grid lines by quadrat_width times buffer_amount

Returns

Return type GeoDataFrame

osmnx.core.**nominatim_request**(*params*, *pause_duration=1*, *timeout=30*, *error_pause_duration=180*)

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

Parameters

- **params** (*dict or OrderedDict*) – key-value pairs of parameters
- **pause_duration** (*int*) – how long to pause before requests, in seconds
- **timeout** (*int*) – the timeout interval for the requests library
- **error_pause_duration** (*int*) – how long to pause in seconds before re-trying requests if error

Returns **response_json**

Return type dict

osmnx.core.**osm_net_download**(*polygon=None*, *north=None*, *south=None*, *east=None*, *west=None*, *network_type='all_private'*, *timeout=180*, *memory=None*, *max_query_area_size=2500000000*, *infrastructure='way["highway"]'*)

Download OSM ways and nodes within some bounding box from the Overpass API.

Parameters

- **polygon** (*shapely Polygon or MultiPolygon*) – geographic shape to fetch the street network within
- **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*) – {'walk', 'bike', 'drive', 'drive_service', 'all', 'all_private'} what type of street network to get
- **timeout** (*int*) – the timeout interval for requests and to pass to API
- **memory** (*int*) – server memory allocation size for the query, in bytes. If none, server will use its default allocation size
- **max_query_area_size** (*float*) – max area for any part of the geometry, in the units the geometry is in: any polygon bigger will get divided up for multiple queries to API (default is 50,000 * 50,000 units (ie, 50km x 50km in area, if units are meters))
- **infrastructure** (*string*) – download infrastructure of given type (default is streets (ie, 'way["highway"]') but other infrastructures may be selected like power grids (ie, 'way["power"~"line"]'))

Returns

Return type dict

osmnx.core.**osm_polygon_download**(*query*, *limit=1*, *polygon_geojson=1*, *pause_duration=1*)

Geocode a place and download its boundary geometry from OSM's Nominatim API.

Parameters

- **query** (*string or dict*) – query string or structured query dict to geocode/download
- **limit** (*int*) – max number of results to return
- **polygon_geojson** (*int*) – request the boundary geometry polygon from the API, 0=no, 1=yes
- **pause_duration** (*int*) – time in seconds to pause before API requests

Returns

Return type dict

osmnx.core.**overpass_request**(*data*, *pause_duration=None*, *timeout=180*, *error_pause_duration=None*)

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

Parameters

- **data** (*dict or OrderedDict*) – key-value pairs of parameters to post to the API

- **pause_duration** (*int*) – how long to pause in seconds before requests, if None, will query API status endpoint to find when next slot is available
- **timeout** (*int*) – the timeout interval for the requests library
- **error_pause_duration** (*int*) – how long to pause in seconds before re-trying requests if error

Returns

Return type dict

osmnx.core.**parse_osm_nodes_paths**(*osm_data*)

Construct dicts of nodes and paths with key=osmid and value=dict of attributes.

Parameters **osm_data** (*dict*) – JSON response from from the Overpass API

Returns **nodes, paths**

Return type tuple

osmnx.core.**quadrat_cut_geometry**(*geometry*, *quadrat_width*, *min_num=3*, *buffer_amount=1e-09*)

Split a Polygon or MultiPolygon up into sub-polygons of a specified size, using quadrats.

Parameters

- **geometry** (*shapely Polygon or MultiPolygon*) – the geometry to split up into smaller sub-polygons
- **quadrat_width** (*numeric*) – the linear width of the quadrats with which to cut up the geometry (in the units the geometry is in)
- **min_num** (*int*) – the minimum number of linear quadrat lines (e.g., min_num=3 would produce a quadrat grid of 4 squares)
- **buffer_amount** (*numeric*) – buffer the quadrat grid lines by quadrat_width times buffer_amount

Returns

Return type shapely MultiPolygon

osmnx.core.**remove_isolated_nodes**(*G*)

Remove from a graph all the nodes that have no incident edges (ie, node degree = 0).

Parameters **G** (*networkx multidigraph*) – the graph from which to remove nodes

Returns

Return type networkx multidigraph

osmnx.core.**save_to_cache**(*url*, *response_json*)

Save an HTTP response json object to the cache.

If the request was sent to server via POST instead of GET, then URL should be a GET-style representation of request. Users should always pass OrderedDicts instead of dicts of parameters into request functions, so that the parameters stay in the same order each time, producing the same URL string, and thus the same hash. Otherwise the cache will eventually contain multiple saved responses for the same request because the URL's parameters appeared in a different order each time.

Parameters

- **url** (*string*) – the url of the request
- **response_json** (*dict*) – the json response

Returns

Return type None

osmnx.core.**truncate_graph_bbox**(*G*, *north*, *south*, *east*, *west*, *truncate_by_edge=False*, *retain_all=False*)

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

Needed because overpass returns entire ways that also include nodes outside the bbox if the way (that is, a way with a single OSM ID) has a node inside the bbox at some point.

Parameters

- **G** (*networkx multidigraph*) –
- **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

Returns

Return type networkx multidigraph

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

Remove everything further than some network distance from a specified node in graph.

Parameters

- **G** (*networkx multidigraph*) –
- **source_node** (*int*) – the node in the graph from which to measure network distances to other nodes
- **max_distance** (*int*) – remove every node in the graph greater than this distance from the source_node
- **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

Return type networkx multidigraph

osmnx.core.**truncate_graph_polygon**(*G*, *polygon*, *retain_all=False*, *truncate_by_edge=False*, *quadrat_width=0.025*, *min_num=3*, *buffer_amount=1e-09*)

Remove every node in graph that falls outside some shapely Polygon or MultiPolygon.

Parameters

- **G** (*networkx multidigraph*) –
- **polygon** (*Polygon or 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 (NOT CURRENTLY IMPLEMENTED)

- **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 = 0.025, approx 2km 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)
- **buffer_amount** (*numeric*) – passed on to intersect_index_quadrats: buffer the quadrat grid lines by quadrat_width times buffer_amount

Returns

Return type networkx multidigraph

osmnx.elevation module

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

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*) –
- **add_absolute** (*bool*) – if True, also add the absolute value of the grade as an edge attribute

Returns G

Return type networkx multidigraph

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

Get the elevation (meters) of each node in the network and add it to the node as an attribute.

Parameters

- **G** (*networkx multidigraph*) –
- **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

Returns G

Return type networkx multidigraph

osmnx.globals module

osmnx.plot module

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*)

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

Parameters

- **G** (*networkx multidigraph*) –
- **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

Returns

Return type list

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

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

Parameters

- **G** (*networkx multidigraph*) –
- **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

Returns

Return type list

osmnx.plot.**make_folium_polyline**(*edge*, *edge_color*, *edge_width*, *edge_opacity*, *popup_attribute=None*)

Turn a row from the gdf_edges GeoDataFrame into a folium PolyLine with attributes.

Parameters

- **edge** (*GeoSeries*) – a row from the gdf_edges GeoDataFrame
- **edge_color** (*string*) – color of the edge lines
- **edge_width** (*numeric*) – width of the edge lines

- **edge_opacity** (*numeric*) – opacity of the edge lines
- **popup_attribute** (*string*) – edge attribute to display in a pop-up when an edge is clicked, if None, no popup

Returns pl

Return type folium.PolyLine

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=8*, *edge_color='w'*, *bgcolor='#333333'*, *smooth_joints=True*, *filename=None*, *file_format='png'*, *show=False*, *save=True*, *close=True*, *dpi=300*)

Plot a figure-ground diagram of a street network, defaulting to one square mile.

Parameters

- **G** (*networkx multidigraph*) –
- **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*) – the height and width of this square diagram
- **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*) – filename to save the image as
- **file_format** (*string*) – the format of the file to save (e.g., 'jpg', 'png', 'svg')
- **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

Returns fig, ax

Return type tuple

osmnx.plot.**plot_graph**(*G*, *bbox=None*, *fig_height=6*, *fig_width=None*, *margin=0.02*, *axis_off=True*, *equal_aspect=False*, *bgcolor='w'*, *show=True*, *save=False*, *close=True*, *file_format='png'*, *filename='temp'*, *dpi=300*, *annotate=False*, *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=True*)

Plot a networkx spatial graph.

Parameters

- **G** (*networkx multidigraph*) –
- **bbox** (*tuple*) – bounding box as north,south,east,west - if None will calculate from spatial extents of data
- **fig_height** (*int*) – matplotlib figure height in inches
- **fig_width** (*int*) – matplotlib figure width in inches
- **margin** (*float*) – relative margin around the figure
- **axis_off** (*bool*) – if True turn off the matplotlib axis
- **equal_aspect** (*bool*) – if True set the axis aspect ratio equal
- **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*) – the format of the file to save (e.g., 'jpg', 'png', 'svg')
- **filename** (*string*) – the name of the file if saving
- **dpi** (*int*) – the resolution of the image file if saving
- **annotate** (*bool*) – if True, annotate the nodes in the figure
- **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*) – if True, use the spatial geometry attribute of the edges to draw geographically accurate edges, rather than just lines straight from node to node

Returns fig, ax

Return type tuple

osmnx.plot.**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*)

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*) –

- **graph_map** (*folium.folium.Map*) – 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

Returns graph_map

Return type folium.folium.Map

osmnx.plot.**plot_graph_route**(*G*, *route*, *bbox=None*, *fig_height=6*, *fig_width=None*, *margin=0.02*, *bgcolor='w'*, *axis_off=True*, *show=True*, *save=False*, *close=True*, *file_format='png'*, *filename='temp'*, *dpi=300*, *annotate=False*, *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=True*, *origin_point=None*, *destination_point=None*, *route_color='r'*, *route_linewidth=4*, *route_alpha=0.5*, *orig_dest_node_alpha=0.5*, *orig_dest_node_size=100*, *orig_dest_node_color='r'*, *orig_dest_point_color='b'*)

Plot a route along a networkx spatial graph.

Parameters

- **G** (*networkx multidigraph*) –
- **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
- **fig_height** (*int*) – matplotlib figure height in inches
- **fig_width** (*int*) – matplotlib figure width in inches
- **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*) – the format of the file to save (e.g., 'jpg', 'png', 'svg')
- **filename** (*string*) – the name of the file if saving
- **dpi** (*int*) – the resolution of the image file if saving
- **annotate** (*bool*) – if True, annotate the nodes in the figure
- **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*) – if True, use the spatial geometry attribute of the edges to draw geographically accurate edges, rather than just lines straight from node to node
- **origin_point** (*tuple*) – optional, an origin (lat, lon) point to plot instead of the origin node
- **destination_point** (*tuple*) – optional, a destination (lat, lon) point to plot instead of the destination node
- **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*) – the opacity of the origin and destination nodes
- **orig_dest_node_size** (*int*) – the size of the origin and destination nodes
- **orig_dest_node_color** (*string*) – the color of the origin and destination nodes
- **orig_dest_point_color** (*string*) – the color of the origin and destination points if being plotted instead of nodes

Returns fig, ax

Return type tuple

osmnx.plot.**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*)

Plot a route on an interactive folium web map.

Parameters

- **G** (*networkx multidigraph*) –
- **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

Returns route_map

Return type folium.folium.Map

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** (*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

Return type tuple

osmnx.plot.**rgb_color_list_to_hex**(*color_list*)

Convert a list of RGBa colors to a list of hexadecimal color codes.

Parameters **color_list** (*list*) – the list of RGBa colors

Returns color_list_hex

Return type list

osmnx.plot.**save_and_show**(*fig*, *ax*, *save*, *show*, *close*, *filename*, *file_format*, *dpi*, *axis_off*)

Save a figure to disk and show it, as specified.

Parameters

- **fig** (*figure*) –
- **ax** (*axis*) –
- **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*) – the name of the file to save
- **file_format** (*string*) – the format of the file to save (e.g., 'jpg', 'png', 'svg')
- **dpi** (*int*) – the resolution of the image file if saving
- **axis_off** (*bool*) – if True matplotlib axis was turned off by plot_graph so constrain the saved figure's extent to the interior of the axis

Returns fig, ax

Return type tuple

osmnx.projection module

osmnx.projection.**project_gdf**(*gdf*, *to_crs=None*, *to_latlong=False*)
Project a GeoDataFrame to the UTM zone appropriate for its geometries' centroid.

The simple calculation in this function works well for most latitudes, but won't work for some far northern locations like Svalbard and parts of far northern Norway.

Parameters

- **gdf** (*GeoDataFrame*) – the gdf to be projected
- **to_crs** (*dict*) – if not None, just project to this CRS instead of to UTM
- **to_latlong** (*bool*) – if True, projects to latlong instead of to UTM

Returns

Return type GeoDataFrame

osmnx.projection.**project_geometry**(*geometry*, *crs={'init': 'epsg:4326'}*, *to_crs=None*, *to_latlong=False*)
Project a shapely Polygon or MultiPolygon from lat-long to UTM, or vice-versa

Parameters

- **geometry** (*shapely Polygon or MultiPolygon*) – the geometry to project
- **crs** (*dict*) – the starting coordinate reference system of the passed-in geometry (default is lat-long)
- **to_crs** (*dict*) – if not None, just project to this CRS instead of to UTM
- **to_latlong** (*bool*) – if True, project from crs to lat-long, if False, project from crs to local UTM zone

Returns (geometry_proj, crs), the projected shapely geometry and the crs of the projected geometry

Return type tuple

osmnx.projection.**project_graph**(*G*, *to_crs=None*)
Project a graph from lat-long to the UTM zone appropriate for its geographic location.

Parameters

- **G** (*networkx multidigraph*) – the networkx graph to be projected
- **to_crs** (*dict*) – if not None, just project to this CRS instead of to UTM

Returns

Return type networkx multidigraph

osmnx.save_load module

osmnx.save_load.**gdfs_to_graph**(*gdf_nodes*, *gdf_edges*)
Convert node and edge GeoDataFrames into a graph

Parameters

- **gdf_nodes** (*GeoDataFrame*) –
- **gdf_edges** (*GeoDataFrame*) –

Returns

Return type networkx multidigraph

osmnx.save_load.**get_undirected**(*G*)
Convert a directed graph to an undirected graph that maintains parallel edges if geometries differ.

Parameters **G** (*networkx multidigraph*) –

Returns

Return type networkx multigraph

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

Parameters

- **G** (*networkx multidigraph*) –
- **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 origin and destination nodes

Returns gdf_nodes or gdf_edges or both as a tuple

Return type GeoDataFrame or tuple

osmnx.save_load.**is_duplicate_edge**(*data*, *data_other*)
Check if two edge data dictionaries are the same based on OSM ID and geometry.

Parameters

- **data** (*dict*) – the first edge's data
- **data_other** (*dict*) – the second edge's data

Returns **is_dupe**

Return type bool

osmnx.save_load.**is_same_geometry**(*data*, *data_other*)
Check if LineString geometries in two edge data dicts are the same, in normal or reversed order of points.

Parameters

- **data** (*dict*) – the first edge's data
- **data_other** (*dict*) – the second edge's data

Returns

Return type bool

osmnx.save_load.**load_graphml**(*filename*, *folder=None*)
Load a GraphML file from disk and convert the node/edge attributes to correct data types.

Parameters

- **filename** (*string*) – the name of the graphml file (including file extension)
- **folder** (*string*) – the folder containing the file, if None, use default data folder

Returns

Return type networkx multidigraph

osmnx.save_load.**make_shp_filename**(*place_name*)
Create a filename string in a consistent format from a place name string.

Parameters **place_name** (*string*) – place name to convert into a filename

Returns

Return type string

osmnx.save_load.**save_gdf_shapefile**(*gdf*, *filename=None*, *folder=None*)
Save a GeoDataFrame of place shapes or building footprints as an ESRI shapefile.

Parameters

- **gdf** (*GeoDataFrame*) – the gdf to be saved
- **filename** (*string*) – what to call the shapefile (file extensions are added automatically)
- **folder** (*string*) – where to save the shapefile, if none, then default folder

Returns

Return type None

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

Parameters

- **G** (*networkx multidigraph*) –
- **filename** (*string*) – the name of the shapefiles (not including file extensions)
- **folder** (*string*) – the folder to contain the shapefiles, if None, use default data folder
- **encoding** (*string*) – the character encoding for the saved shapefiles

Returns

Return type None

osmnx.save_load.**save_graphml**(*G*, *filename='graph.graphml'*, *folder=None*)
Save graph as GraphML file to disk.

Parameters

- **G** (*networkx multidigraph*) –
- **filename** (*string*) – the name of the graphml file (including file extension)
- **folder** (*string*) – the folder to contain the file, if None, use default data folder

Returns

Return type None

osmnx.simplify module

osmnx.simplify.**build_path**(*G*, *node*, *endpoints*, *path*)
Recursively build a path of nodes until you hit an endpoint node.

Parameters

- **G** (*networkx multidigraph*) –

- **node** (*int*) – the current node to start from
- **endpoints** (*set*) – the set of all nodes in the graph that are endpoints
- **path** (*list*) – the list of nodes in order in the path so far

Returns **paths_to_simplify**

Return type list

osmnx.simplify.**clean_intersections**(*G*, *tolerance=15*, *dead_ends=False*)

Clean-up intersections comprising clusters of nodes by merging them and returning their centroids.

Divided roads are 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 cleans them up by buffering their points 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*) –
- **tolerance** (*float*) – nodes within this distance (in graph's geometry's units) will be dissolved into a single intersection
- **dead_ends** (*bool*) – if False, discard dead-end nodes to return only street-intersection points

Returns **intersection_centroids** – a GeoSeries of shapely Points representing the centroids of street intersections

Return type geopandas.GeoSeries

osmnx.simplify.**get_paths_to_simplify**(*G*, *strict=True*)

Create a list of all the paths to be simplified between endpoint nodes.

The path is ordered from the first endpoint, through the interstitial nodes, to the second endpoint.

Parameters

- **G** (*networkx multidigraph*) –
- **strict** (*bool*) – if False, allow nodes to be end points even if they fail all other rules but have edges with different OSM IDs

Returns **paths_to_simplify**

Return type list

osmnx.simplify.**is_endpoint**(*G*, *node*, *strict=True*)

Return True if the node is a "real" endpoint of an edge in the network, otherwise False.

OSM data includes lots of nodes that exist only as points to help streets bend around curves. An end point is a node that either: 1) is its own neighbor, ie, it self-loops. 2) or, has no incoming edges or no outgoing edges, ie, all its incident edges point inward or all its incident edges point outward. 3) or, it does not have exactly two neighbors and degree of 2 or 4. 4) or, if strict mode is false, if its edges have different OSM IDs.

Parameters

- **G** (*networkx multidigraph*) –
- **node** (*int*) – the node to examine
- **strict** (*bool*) – if False, allow nodes to be end points even if they fail all other rules but have edges with different OSM IDs

Returns

Return type bool

osmnx.simplify.**is_simplified**(*G*)
Determine if a graph has already had its topology simplified.

If any of its edges have a geometry attribute, we know that it has previously been simplified.

Parameters **G** (*networkx multidigraph*) –

Returns

Return type bool

osmnx.simplify.**simplify_graph**(*G_*, *strict=True*)
Simplify a graph's 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** (*graph*) –
- **strict** (*bool*) – if False, allow nodes to be end points even if they fail all other rules but have edges with different OSM IDs

Returns

Return type networkx multidigraph

osmnx.stats module

osmnx.stats.**basic_stats**(*G*, *area=None*)
Calculate basic descriptive stats and metrics for a graph.

Parameters

- **G** (*networkx multidigraph*) –
- **area** (*numeric*) – the area covered by the street network, in square meters (typically land area); if none, will skip all density-based metrics

Returns

stats –

dictionary of network measures containing the following elements:

- n = number of nodes in the graph
- m = number of edges in the graph
- k_avg = average node degree of the graph
- count_intersections = 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 = count_intersections 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)

Return type dict

osmnx.stats.**count_streets_per_node**(*G*, *nodes=None*)

Count how many street segments emanate from each node (i.e., intersections and dead-ends) in this graph.

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

Parameters

- **G** (*networkx multidigraph*) –
- **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.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*) –
- **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

osmnx.utils module

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_path=['bridge', 'tunnel', 'oneway', 'lanes', 'ref', 'name', 'highway', 'maxspeed', 'service', 'access', 'area', 'landuse', 'width', 'est_width']*)

Configure osmnx by setting the default global vars to desired values.

Parameters

- **data_folder** (*string*) – where to save and load data files
- **logs_folder** (*string*) – where to write the log files
- **imgs_folder** (*string*) – where to save figures
- **cache_folder** (*string*) – where to save the http response cache
- **use_cache** (*bool*) – if True, use a local cache to save/retrieve http responses instead of calling API repetitively for the same request URL
- **log_file** (*bool*) – if true, save log output to a log file in logs_folder
- **log_console** (*bool*) – if true, print log output to the console
- **log_level** (*int*) – one of the logger.level constants
- **log_name** (*string*) – name of the logger
- **useful_tags_node** (*list*) – a list of useful OSM tags to attempt to save from node elements
- **useful_tags_path** (*list*) – a list of useful OSM tags to attempt to save from path elements

Returns

Return type None

osmnx.utils.**geocode**(*query*)

Geocode a query string to (lat, lon) with the Nominatim geocoder.

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

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

Return type tuple

osmnx.utils.**get_largest_component**(*G*, *strongly=False*)

Return the largest weakly or strongly connected component from a directed graph.

Parameters

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

Returns

Return type networkx multidigraph

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

Create a logger or return the current one if already instantiated.

Parameters

- **level** (*int*) – one of the logger.level constants
- **name** (*string*) – name of the logger
- **filename** (*string*) – name of the log file

Returns

Return type logger.logger

osmnx.utils.**get_nearest_node**(*G*, *point*, *return_dist=False*)
Return the graph node nearest to some specified point.

Parameters

- **G** (*networkx multidigraph*) –
- **point** (*tuple*) – the (lat, lon) point for which we will find the nearest node in the graph
- **return_dist** (*bool*) – optionally also return the distance between the point and the nearest node

Returns multidigraph or optionally (multidigraph, float)

Return type networkx multidigraph or tuple

osmnx.utils.**get_route_edge_attributes**(*G*, *route*, *attribute*, *minimize_key='length'*)
Get a list of attribute values for each edge in a path.

Parameters

- **G** (*networkx multidigraph*) –
- **route** (*list*) – list of nodes in the path
- **attribute** (*string*) – the name of the attribute to get the value of for each edge
- **minimize_key** (*string*) – if there are parallel edges between two nodes, select the one with the lowest value of minimize_key

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

Return type list

osmnx.utils.**great_circle_vec**(*lat1*, *lng1*, *lat2*, *lng2*, *earth_radius=6371009*)
Vectorized function to calculate the great-circle distance between two points or between vectors of points.

Parameters

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

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

Return type float or array of float

osmnx.utils.**log**(*message*, *level=None*, *name=None*, *filename=None*)
Write a message to the log file and/or print to the the console.

Parameters

- **message** (*string*) – the content of 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.**make_str**(*value*)

Convert a passed-in value to unicode if Python 2, or string if Python 3.

Parameters **value** (*any*) – the value to convert to unicode/string

Returns

Return type unicode or string

Module contents

OSMnx: retrieve, construct, analyze, and visualize street networks from OpenStreetMap. OSMnx is a Python package that lets you download spatial geometries and construct, project, visualize, and analyze street networks from OpenStreetMap's APIs. Users can download and construct walkable, drivable, or bikable urban networks with a single line of Python code, and then easily analyze and visualize them.

CHAPTER 2

Citation info

Boeing, G. 2017. "OSMnx: New Methods for Acquiring, Constructing, Analyzing, and Visualizing Complex Street Networks." *Computers, Environment and Urban Systems*, forthcoming. doi:10.2139/ssrn.2865501.

CHAPTER 3

Installation

Install OSMnx with pip by running:

```
pip install osmnx
```

or with conda by running:

```
conda install -c conda-forge osmnx
```

If you have any trouble with the installation, try installing OSMnx in a new, clean virtual environment:

```
conda create --yes -c conda-forge -n OSMNX python=3 osmnx
source activate OSMNX
```

CHAPTER 4

Examples

For examples and demos, see the GitHub repo: https://github.com/gboeing/osmnx

CHAPTER 5

Support

Issue tracker: https://github.com/gboeing/osmnx/issues

CHAPTER 6

License

The project is licensed under the MIT license.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

o

Index

P

Q

R

S

T